M.Tech Project Dissertation

on

Parameter-Specific Literature Retrieval for Kidney Diseases

A Project Report
Submitted in fulfillment of
the requirements for the degree of
Master of Technology
by

Rishikesh Kumar
(Roll No. 24M1520)

Supervisor:
Prof. P. Balamurugan

Department of Industrial Engineering and Operations Research

Indian Institute of Technology Bombay
Mumbai 400076 (India)

Acceptance Certificate

This thesis entitled **"Parameter-Specific Literature Retrieval for Kidney Diseases"** submitted by **Rishikesh Kumar** may be accepted for being evaluated. .

Prof. Balamurugan Palaniappan
Supervisor

Place: **Mumbai**
Date: June 12, 2026

Declaration

I affirm that this written submission reflects my original ideas, expressed in my own words. Where I have used the ideas or words of others, I have provided appropriate citations and references to acknowledge the original sources. I confirm that I have correctly and fully credited all the sources that contributed to the creation of this report. Additionally, I have integrated and adapted portions of the work done by Vaibhav Singh Panwar [21], Kiran Prakash EC [6] and Mintu Yadav [27]. I further declare that I have upheld the principles of academic honesty and integrity throughout this submission and have not misrepresented, fabricated, or falsified any ideas, data, facts, or sources. I am aware that any breach of these principles may result in disciplinary action by the Institute and legal consequences from the original sources if proper citations were not made or permissions were not obtained where required.

Name: Rishikesh Kumar

Signature: Rishikesh Kumar

Date: June 12, 2026

Acknowledgement

I am profoundly grateful to Prof. Balamurugan Palaniappan, Department of Industrial Engineering and Operations Research, IIT Bombay, for his unwavering guidance, insightful critiques, and sustained encouragement throughout this work. His rigorous standards and constructive advice significantly shaped the research direction and quality of this thesis. I extend my sincere thanks to Dr. Vishwanath Billa and Dr. Deepa Usulumarty, Nephrologists at Sushrut Hospital & Research Center, Mumbai, for their expert clinical guidance and valuable suggestions. Their insights and feedbacks were instrumental in aligning this research with practical medical applications.

Additionally, I want to acknowledge Vaibhav Singh Panwar and Kiran Prakash EC, former students from the IEOR department at IIT Bombay, for their significant contributions to this research. The foundational work and insights provided by them have greatly enhanced the quality of this thesis. I would also like to extend my deepest gratitude to Mintu Yadav, whose support and collaboration were invaluable throughout this journey.

My deepest appreciation goes to my family and friends for their unwavering support, patience, and encouragement during this endeavour. Their faith in me provided constant motivation through the challenges encountered over the course of this research. Finally, I thank all colleagues, faculty, and staff who contributed, directly or indirectly, to the completion of this thesis. Their contributions have been invaluable, and I remain deeply appreciative of their assistance.

Abstract

The volume of medical literature now grows faster than any clinician can read. For a doctor treating a patient with a kidney disease, the practical problem is twofold: first, finding the handful of trials that actually match the patient in front of them, and second, turning what those trials say into an answer to a concrete clinical question. This thesis tackles both problems, and it does so in two stages that together form a single research arc.

The first stage generalises a parameter-specific retrieval system that was originally built only for IgA Nephropathy. We expand it to twenty-four kidney-related diseases and ten clinical parameters. A Named Entity Recognition (NER) model fine-tuned on PubMedBERT extracts parameters such as age, GFR, proteinuria and creatinine from PubMed abstracts, and a Seq2Tree model fine-tuned on BART converts the free-text mentions into a structured XML form from which values, units and comparison operators can be read reliably. These structured records feed the NephroSearch web platform and the NephroTagger browser extension, whose ranking quality was measured with Normalized Discounted Cumulative Gain (NDCG) and shown to exceed both PubMed and Google.

The second stage, which is the principal contribution of this phase, takes the same extraction machinery in a different direction. Rather than producing a ranked list of papers, we build a knowledge graph that records *what treats what*, *what causes what*, and *which paper says so*. Clinical text is processed by a composable, hot-swappable pipeline: an ensemble of three NER models (a biomedical BERT, a clinical DeBERTa, and the zero-shot GLiNER) extracts entities; these are normalised, optionally against UMLS; REBEL extracts relation triplets that a context-aware stage re-labels using the entity types; and the result is written into

a Neo4j graph in which every edge carries the identifiers of the papers that support it. On top of this graph we built a two-generation question answering engine. The first generation classifies the intent of a question and runs a matching Cypher template; the second resolves the entities in the question against the graph, routes the query through a type-aware fallback chain, falls back to LLM-generated Cypher when no template fits, and records every decision in a step log so that the reasoning is auditable. A Streamlit application ties the pipeline, the graph and the QA engine together and adds an interactive graph explorer.

We evaluate the new components on a curated kidney-disease corpus, reporting NER quality per entity type, the effect of context-aware relation re-mapping, knowledge-graph statistics with full provenance coverage, and the answer quality and latency of the QA engine against a raw-LLM baseline. The work directly addresses several criticisms levelled at the earlier system, most notably its opaque scoring and its narrow scope, and it makes the path from a published paper to a cited, queryable clinical answer transparent end to end.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides an overview of the challenges in retrieving and ranking medical literature for kidney-related diseases, the limitations of existing tools, and the proposed solutions developed in this work. It introduces the motivation, objectives, and contributions of this thesis, setting the stage for the subsequent chapters.

The exponential growth of medical literature, with thousands of Randomized Clinical Trials (RCTs) published annually, presents a significant challenge for clinicians and researchers. Identifying and prioritizing studies relevant to specific diseases and patient parameters is critical for evidence-based decision-making but is often hindered by the sheer volume and variability of research articles. For kidney-related diseases, this challenge is particularly acute, as tailored insights are essential for effective patient care.

Existing literature retrieval systems, such as PubMed and general-purpose platforms like Google, often fall short in delivering precise, patient-specific results. These tools lack the ability to rank studies based on clinical parameters, leading to irrelevant or overly broad results. Similarly, large language models, while powerful, are not optimized for parameter-specific ranking in biomedical contexts.

To address these limitations, this thesis proposes a robust pipeline-based approach for retrieving and ranking RCTs tailored to kidney-related diseases. Central to this approach is an enhanced Named Entity Recognition (NER) model, fine-tuned on the PubMedBERT architecture, which accurately extracts key clinical parameters such as Age, GFR

(Glomerular Filtration Rate),Proteinuria, Creatinine, BUN (Blood Urea Nitrogen), BP (Blood Pressure), Hematuria (blood in urine), Weight and Cholesterol.

The model is trained to handle these parameters across a wide range of kidney-related diseases, ensuring its versatility and applicability in diverse clinical scenarios. This NER model is complemented by a Seq2Tree model, fine-tuned on the BART architecture, which converts unstructured text into structured XML format, enabling precise parameter extraction and ranking.

Building on these models, two tools are introduced: **NephroSearch**, a web-based platform that retrieves and ranks RCTs based on structured patient input.**NephroTagger**, a Chrome extension that facilitates real-time entity annotation within biomedical literature.

The effectiveness of this system is rigorously evaluated using the Normalized Discounted Cumulative Gain (NDCG) metric, demonstrating superior performance over existing platforms. The NER model achieves 100% precision, recall, and F1 score, while the Seq2Tree model achieves an impressive ROUGE score of approximately 0.68, highlighting the reliability and accuracy of the pipeline.

By streamlining the retrieval and ranking of medical literature, this work empowers clinicians and researchers to efficiently access relevant studies, significantly enhancing decision-making in clinical and research settings.

1.1 From Retrieval to Reasoning: Motivation for the Final Phase

A ranked list of papers, however well it is tuned to a patient's parameters, is only half of what a clinician actually needs. The system described above is good at saying *here are the trials closest to your patient*, but the questions that arise at the bedside are rarely phrased that way. A doctor asks "what should I give this patient, and why?", "what are the side effects of this drug?", or "is there any evidence that this protocol works in someone with this comorbidity?". Answering those questions

from a ranked list still requires a human to open each paper, read it, and assemble the pieces. The retrieval tool has narrowed the search, but the reasoning is left undone.

There is a deeper limitation as well. A list of documents keeps knowledge locked inside prose. The fact that drug A treats disease D, that it can cause side effect S, and that it should not be given to a patient who also has condition C, are all relationships that a list cannot express directly. To answer a compositional question one must traverse these relationships, and a flat index simply does not store them. What is needed is a representation in which drugs, diseases, side effects, biomarkers and the papers that connect them are first-class objects, linked by typed edges that can be followed and queried.

This observation motivated the shift that defines the final phase of this work. We move from parameter-based *retrieval* to a structured, evidence-traceable *knowledge graph* that can be interrogated directly with natural-language clinical questions. The same extraction components that previously fed a ranking algorithm are re-targeted to populate this graph, and a question answering engine is built on top of it so that questions such as "What treats IgA nephropathy?" or "What evidence supports budesonide for it?" are answered by walking the graph and citing the source papers.

This re-orientation also responds to concrete criticism of the earlier system. Reviewers of our earlier submission pointed out that the relevance-scoring mechanism was effectively a black box, that the evaluation was thin, and that the scope (a single disease and four parameters) was too narrow to judge the approach. A knowledge graph addresses the first complaint by making every answer a traceable path of typed edges back to named evidence; it addresses the third by adopting a general, disease-agnostic schema; and, as Chapter 7 sets out, it lets us evaluate the system stage by stage rather than as a single opaque pipeline.

1.2 Problem Statement

Current literature retrieval systems struggle to deliver precise, patient-specific results for clinical queries related to kidney diseases. General-purpose platforms, such as Google, and large language models, like ChatGPT, often produce irrelevant or overly broad results due to their lack of parameter-specific ranking capabilities. Similarly, PubMed, while specialized, falls short in prioritizing literature based on clinical parameters.

Taking inspiration from the foundational ideas proposed by Kiran Prakash EC [6] and Mintu Yadav [27], this thesis builds upon their work to develop an enhanced Named Entity Recognition (NER) model fine-tuned on the PubMedBERT architecture. This model is specifically designed to extract and prioritize key clinical parameters such as Age, GFR (Glomerular Filtration Rate), Proteinuria, Creatinine, BUN (Blood Urea Nitrogen), BP (Blood Pressure), Hematuria, Weight, and Cholesterol, enabling a specialized retrieval system for kidney-related diseases.

To address the limitations of existing tools, this work enhances and generalizes two domain-specific applications for kidney-related diseases: **NephroSearch**, a web platform that retrieves and ranks randomized clinical trial (RCT) papers based on structured patient input, and **NephroTagger**, a Chrome extension that facilitates real-time entity annotation within biomedical literature. Significant improvements have been made to the interface and functionality of these tools, ensuring broader applicability across various kidney diseases. The system's relevance ranking is evaluated using the Normalized Discounted Cumulative Gain (NDCG) metric [26], which provides a principled framework for assessing ranking quality in information retrieval tasks.

1.2.1 Part A: Generalizing Extraction Across Kidney Diseases and Clinical Parameters

The first part of the problem is one of generalisation. The extraction models inherited from earlier work were trained only on IgA Nephropathy and only for four parameters. To be useful as a foundation for anything broader they must cover a representative slice of nephrology. We

therefore set out to build a single NER model that recognises ten clinical parameters across twenty-four kidney-related diseases, and a Seq2Tree model that turns the messy ways these parameters are written in real abstracts into a clean, machine-readable structure. The corpus, the annotated dataset and the trained models produced here are not an end in themselves; they are the data-and-model substrate on which the second part is built.

1.2.2 Part B: Knowledge-Graph-Grounded Question Answering

The second part re-targets that machinery. Instead of ending the pipeline at a ranked list, we feed the extracted entities and the relations between them into a Neo4j knowledge graph, and we build a question answering engine over that graph. The objective here is not merely to retrieve but to answer: given a free-form clinical question, identify the entities it refers to, find the relevant subgraph, and synthesise a natural-language answer in which every clinical claim is accompanied by the papers that support it. Provenance is central to this aim, and it is implemented concretely as a `source_evidence_ids` list attached to every factual edge, so that any answer can be unwound back to its sources.

1.3 Objectives and Scope of this Thesis

The concrete objectives of this thesis are as follows.

1. Expand the disease and parameter coverage of the Phase 1 NER and Seq2Tree models from a single disease and four parameters to twenty-four diseases and ten parameters, and assemble the corresponding annotated corpus.

2. Design and implement an ensemble NER stage that combines specialised biomedical models with a zero-shot model, mapping their heterogeneous output onto a single knowledge-graph entity vocabulary.

3. Add a relation-extraction stage based on REBEL, together with a context-aware re-mapping step that corrects its Wikidata-style labels using the entity types already known from NER.

4. Design a Neo4j knowledge-graph schema for clinical treatment knowledge in which every node and edge is traceable to its source documents.

5. Implement a question answering engine over the graph, in two successive designs: a template-based engine and an entity-aware engine with fallback routing, LLM-generated Cypher, and an explainable step log.

6. Build a Streamlit application that lets a user construct the graph, ask questions, and explore the graph interactively.

7. Evaluate each of these components, and the system end to end, on a kidney-disease corpus.

A word on scope is in order. The pipeline and the graph schema are deliberately disease-agnostic: nothing in them is hard-coded to nephrology, and the entity and relation vocabularies are drawn from general clinical-treatment knowledge. For evaluation, however, we stay on the kidney-disease literature gathered in Part A. This keeps the new work continuous with the old, reuses an annotated corpus we already trust, and lets us compare the two paradigms on the same ground. Large-scale population of the graph from hundreds of thousands of abstracts, and formal validation by practising clinicians, are outside the scope of this thesis and are discussed as future work.

1.4 Summary of Contributions

The contributions of this thesis, spanning both phases, are:

- an expanded, multi-disease, multi-parameter NER model and the annotated corpus it was trained on (Phase 1);

- a fine-tuned Seq2Tree model that converts free-text clinical measurements into structured XML (Phase 1);
- the integration and extension of the NephroSearch platform and the NephroTagger extension built in prior work;
- an ensemble NER stage (a biomedical BERT, a clinical DeBERTa, and zero-shot GLiNER) with a unified mapping onto a thirteen-type knowledge-graph entity vocabulary;
- a REBEL-based relation-extraction stage with a context-aware label re-mapping step that recovers clinically meaningful edges from REBEL's structural labels;
- a composable, hot-swappable pipeline framework in which stages can be inserted, replaced or removed at runtime, supporting three interchangeable pipeline variants;
- a Neo4j knowledge-graph schema for clinical treatment knowledge, with provenance tracking on every factual edge;
- a two-generation question answering engine — a template-based engine and an entity-aware engine adding entity resolution, fallback routing, LLM-generated Cypher and an auditable step log; and
- a Streamlit application unifying graph construction, natural-language question answering, and interactive graph exploration.

1.5 Thesis Organization

The remainder of this thesis is organised as follows. Chapter 2 reviews the prior work on this project — the rule-based system of Vaibhav Singh Panwar, the deep-learning extraction models of Kiran Prakash, and the clinical tools and evaluation of Mintu Yadav — and surveys the wider literature on biomedical NER, relation extraction, entity normalisation, knowledge graphs for clinical decision support, and the use of large language models as extraction and reasoning components. Chapter 3 recaps

the Phase 1 methodology — the expansion of the disease and parameter sets, the construction of the annotated corpus, and the training of the generalised NER and Seq2Tree models, together with the retrieval pipeline they drive — and then summarises the Phase 1 results, explaining in light of them why a ranked-retrieval paradigm is not enough.

The Phase 2 contributions occupy the next four chapters. Chapter 4 presents the knowledge-graph construction pipeline stage by stage, from document parsing through ensemble NER, normalisation, relation extraction and context-aware re-mapping to the Neo4j writer, and describes the graph schema. Chapter 5 describes the question answering engine in both of its generations. Chapter 6 presents the Streamlit application that exposes all of this to a user. Chapter 7 reports the evaluation of the Phase 2 system. Chapter 8 steps back to assess the work against the original problem and the criticisms of the earlier system, and Chapter 9 concludes and lays out future work. Five appendices give the full graph schema, the Cypher template catalogue, sample question-answering transcripts, the NER label-mapping tables, and the training hyperparameters.

This chapter has outlined the motivation, challenges, and contributions of this thesis, providing a foundation for the work presented. The next chapter will delve into the literature survey, exploring existing research and methodologies relevant to this study.

Chapter 2

Literature Survey

This chapter presents a comprehensive review of the previous work that forms the foundation of this thesis. The retrieval of relevant clinical trial literature is a critical challenge in evidence-based medicine. With thousands of randomized clinical trials (RCTs) published annually, clinicians face significant difficulties in identifying studies relevant to specific patient parameters and disease conditions. Traditional keyword-based search systems, while useful for general queries, often fail to provide precision in parameter-specific clinical contexts.

The work presented in this chapter chronicles the evolution of a parameter-specific literature retrieval system initially developed for IgA Nephropathy. Three primary contributors have shaped this project: Vaibhav Singh Panwar [21], who introduced a rule-based information extraction and scoring framework; Kiran Prakash EC [6], who transitioned the system to deep learning-based approaches using transformer models; and Mintu Yadav [27], who enhanced the models and developed user-facing clinical tools. Each contribution built upon the previous work, progressively improving the accuracy, reliability, and usability of the system.

Understanding these foundational contributions is essential to contextualize the improvements and extensions introduced in this thesis, particularly the generalization of the system to handle a broader range of kidney-related diseases and clinical parameters beyond the original four parameters (Age, Proteinuria, GFR, and Creatinine) used in IgA Nephropathy studies.

2.1 Background: Evidence-Based Medicine and the Literature Retrieval Problem

Evidence-based medicine rests on the idea that clinical decisions should be informed by the best available research, and the randomized clinical trial (RCT) sits at the top of that evidence hierarchy because randomisation controls for the confounders that plague observational studies. The difficulty is one of scale. PubMed [1] indexes well over thirty-five million citations and grows by more than a million entries a year, so even a narrow clinical topic can return thousands of papers. A clinician cannot read them all, and the few that matter for a particular patient are buried among the many that do not.

Keyword search, the interface most clinicians actually use, is a blunt instrument for this task. A query for "IgA nephropathy treatment" returns papers that mention those words, ordered by relevance heuristics that know nothing about the patient. They cannot express the constraint that matters — that the patient is sixty years old, with a GFR of forty and proteinuria above one gram per day — because the parameters live inside the prose of each abstract rather than in any searchable field. General-purpose engines such as Google, and more recently large language models, are no better suited to this: they are tuned for broad relevance, not for the parameter-specific filtering a nephrologist needs, and the language models additionally offer no guarantee that what they assert is grounded in a real, citable study.

Two capabilities are therefore missing, and they organise the rest of this chapter. The first is *structured information extraction*: a way to pull the clinical parameters, and later the entities and relations, out of free text so that they can be compared and queried rather than merely matched as strings. The second is a *queryable knowledge representation*: a structure that records the relationships between clinical concepts and the evidence behind them, so that compositional questions can be answered by traversal rather than by reading. The prior work on this project, surveyed next, addressed the first capability; the wider literature reviewed in the later sections of this chapter provides the building

blocks for the second, which is the subject of this thesis.

2.2 Work by Vaibhav Singh Panwar

Vaibhav Singh Panwar laid the foundation for this project by developing a rule-based system for extracting and evaluating patient characteristics from clinical trial papers. The system compared extracted data against input values provided by doctors and ranked papers based on their similarity scores. His contributions can be summarized as follows:

2.2.1 Information Extraction Algorithm

He initially developed a basic window-based extraction method that search for keywords and numerical values within a fixed character window. However, the simple approach had limitations in accuracy and context-awareness. The information extraction algorithm was designed to overcome the limitations of simple window-based extraction methods. The improved version integrated the following components:

- **Window Search Method:** Identified predefined keywords (e.g., Proteinuria, Age) in each sentence and searched within a window (±N characters) for numerical values near each keyword.
- **Unit Search and Mapping:** Detected measurement units (e.g., mg/dL, g/day) near numerical values and mapped them to the correct keywords using a predefined dictionary.
- **Proximity-Based Scoring:** Assigned the closest unit–keyword pair based on minimal distance.

Algorithm 1 Window Search Method

Require: K: List of keywords or parameters to search for
Require: P: List of papers in the dataset
Require: W: Window size
Ensure: Assignment of numerical values to keywords

```
for each paper p ∈ P do
    for each sentence s in p do
        if a keyword from K is found in s then
            for each keyword k ∈ K found in s do
                Note the position of k
                Search for numerical values within window W around k
                if only one numerical value is found then
                    Assign this numerical value to k
                else
                    Assign the closest numerical value to k
                end if
            end for
        else
            Skip this sentence
        end if
    end for
end for
```

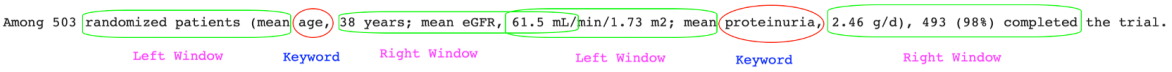

Figure 2.1: *Window Search Example*

The algorithm included post-processing steps such as converting inequalities (e.g., "¿40") to numeric ranges, standardizing units (e.g., converting g/6 hr to g/day), and storing extracted results in a Python dictionary for comparison.

Sentence Classification

To contextualize the extracted data, sentences were classified into paper sections such as settings, during treatment, results, and other. This was achieved using machine learning models (Logistic Regression, SVC, Random Forest, MLP) with embeddings like TF-IDF and POS tags.

The best performance was achieved with SVC + TF-IDF + POS tags, yielding an accuracy of 72.95%.

2.2.2 Paper Scoring Algorithm

After extraction, the system computed a similarity score between input and extracted values for each parameter. Papers were ranked in descending order of their scores. The scoring algorithm handled various cases:

- **Both Input and Extracted Values are Ranges:** Scores were calculated based on the overlap between the ranges.
- **One Value is a Range, the Other is Single:** If the single value lay within the range, the score was 100; otherwise, scoring depended on the severity zone (normal, disease, severe).
- **Both Values are Single:** Scores were assigned based on proximity and zone overlap.

The final score for each paper was calculated by averaging multiple extractions per parameter, summing across all four parameters (maximum score = 400), and sorting papers in descending order.

2.2.3 Output and GUI

Vaibhav also developed a graphical user interface (GUI) that allowed doctors to input single values, ranges, or inequalities. The system displayed each paper's score, extracted vs input values, and paper section. Results included clickable links and drug mentions, leveraging BioBERT NER for entity recognition.

In summary, Vaibhav Singh Panwar's rule-based pipeline effectively extracted patient parameters using unit-aware context windows, standardized data, classified sentence context, and assigned scores through a robust comparison framework. This work laid the groundwork for ranking clinical trials based on their relevance to patient inputs.

2.3 Work by Kiran Prakash

While the rule-based algorithmic approach introduced by Vaibhav Singh Panwar [21] demonstrated initial promise, it suffered from significant issues with false positives and false negatives, limiting its reliability for clinical applications. To address these limitations, Kiran Prakash [6] pivoted toward a deep learning-based approach, leveraging transformer models for more robust and accurate information extraction.

2.3.1 Named Entity Recognition Framework

Kiran Prakash introduced a Named Entity Recognition (NER) framework to automate the identification and extraction of clinical parameters from biomedical literature. NER is a fundamental task in natural language processing (NLP) that involves classifying words or phrases in text into predefined categories such as names of people, organizations, locations, and dates. In the context of this work, NER was adapted to extract medical parameters including Age, Proteinuria, Creatinine, and Glomerular Filtration Rate (GFR).

To build a robust training dataset, Kiran developed a web-based annotation interface that enabled medical professionals to label clinical parameters in research abstracts. This collaborative approach ensured high-quality annotations tailored to the domain-specific requirements of IgA Nephropathy literature.

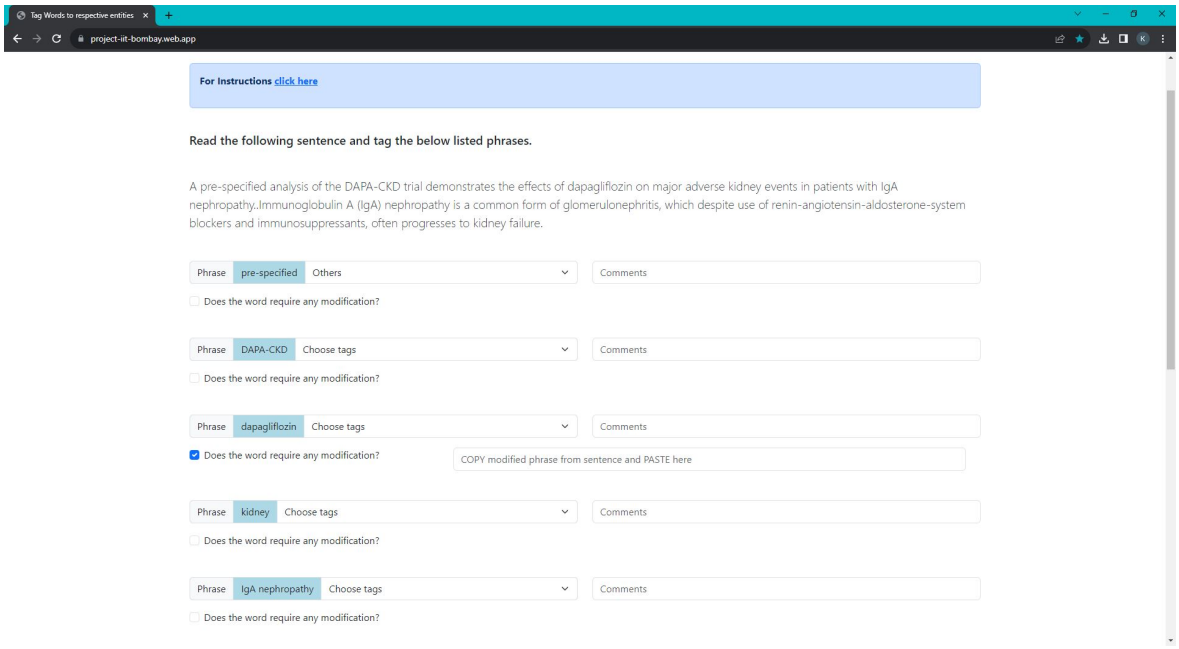

Figure 2.2: *Web UI for Data Collection and Annotation*

Multi-Model NER Architecture

Kiran fine-tuned four distinct NER models, each designed to identify specific entities within the text. The architecture employed multiple specialized models working in tandem to achieve comprehensive information extraction:

- **NER UNITS:** Trained on "microsoft/BiomedNLP-BiomedBERT-base-uncased-abstract-fulltext", this model was designed to identify measurement units such as mg/dL, g/day, and years. Its primary function was to filter sentences containing relevant units, enabling more targeted extraction in subsequent stages.
- **NER VALUES:** Built upon "dmis-lab/biobert-base-cased-v1.2", a BERT model pre-trained on large-scale biomedical corpora, this model focused on extracting numerical values from text. It was applied to sentences identified by NER UNITS to capture quantitative information associated with clinical parameters.
- **NER AGE:** Also trained on "dmis-lab/biobert-base-cased-v1.2",

this model specialized in identifying age-related information within clinical trial abstracts.

- **NER CHARS:** The primary model for extracting the four key parameters—Age, Proteinuria, Creatinine, and GFR—NER CHARS was fine-tuned on "dmis-lab/biobert-base-cased-v1.2". Proteinuria, Creatinine, and GFR were identified as subsets of the VALUE entity recognized by NER VALUES, while Age was extracted by NER AGE. Together, these models formed a comprehensive extraction pipeline.

The combination of NER UNITS and NER VALUES served a dual purpose: they not only extracted relevant entities but also facilitated the automatic generation of additional training data, creating a feedback loop for iterative model improvement.

Iterative Fine-Tuning Methodology

Kiran developed an iterative fine-tuning methodology to progressively enhance model performance. The process involved:

1. Fine-tuning the models on an initial small dataset.
2. Evaluating model performance on validation data.
3. Using NER UNITS and NER VALUES to automatically extract additional sentences containing units and values from unannotated abstracts.
4. Augmenting the training dataset with these automatically extracted examples.
5. Repeating the fine-tuning process.

This iterative approach was repeated over five iterations, progressively improving the models' ability to generalize to diverse biomedical text. The final performance metrics are presented in Table 2.1, demonstrating substantial improvements across iterations, particularly in the

NER CHARS model, which achieved near-perfect F1-scores by the fifth iteration.

Table 2.1: *NER Model Performance Metrics*

Model	Precision	Recall	F1-Score	Accuracy
NER UNITS	0.7172	0.8452	0.7760	0.9711
NER VALUES	0.9564	0.9696	0.9630	0.9944
NER AGE	0.9121	0.9432	0.9274	0.9970
NER CHARS 1	0.3409	0.5769	0.4286	0.9287
NER CHARS 2	0.7467	0.8296	0.7860	0.9916
NER CHARS 3	0.9187	0.9529	0.9354	0.9965
NER CHARS 4	0.9667	0.9757	0.9712	0.9983
NER CHARS 5	0.9380	0.9626	0.9501	0.9953

2.3.2 Seq2Tree Generation

While the NER models successfully identified clinical parameters within unstructured text, the extracted phrases often appeared in diverse and inconsistent formats due to variations in writing styles across different research articles. To address this challenge, Kiran Prakash developed a text-to-text generation model that transforms unstructured clinical measurements into a standardized XML tree representation.

The Seq2Tree model focuses on parsing phrases related to proteinuria, creatinine, GFR, and age into structured formats that capture essential components such as magnitude, unit, and comparative sense (e.g., "greater than" or "less than"). For instance, the phrase "older than 20 younger than 50 years" is transformed into a structured range [20, 50] with the unit "years," while "1 g/d or less" is converted to the XML format: `< S >< T >< LT > 1 < /LT >< U > g/d < /U >< /T >< /S >`.

XML Tree Structure

The model employs a concise set of XML tags to represent the hierarchical structure of clinical measurements:

- **S** : Root node for the tree structure.

- **T** : Child node grouping values with the same unit.
- **IQ** : Interquartile range (value 1 if applicable).
- **CI** : Confidence interval (value 1 if applicable).
- **CT** : Central tendency (e.g., mean or median).
- **FR** : "From" indicating the start of a range.
- **TO** : "To" indicating the end of a range.
- **SD** : Standard deviation.
- **GT** : "Greater than" comparative operator.
- **LT** : "Less than" comparative operator.
- **U** : Unit of measurement.

Model Architecture and Fine-Tuning

Kiran fine-tuned the BART-base model for this sequence-to-tree generation task. However, several challenges emerged during the fine-tuning process, including:

- Deviations from the reference structure, particularly for longer tree sequences.
- Loss of integer parts in decimal numbers (e.g., 11.06 generated as 0.06).
- Extraneous dots or digits in the output.
- Misplaced whitespace affecting parsing accuracy.
- Incorrect handling of negative signs.

To mitigate these issues and improve model performance, several optimization strategies were employed:

1. **Tag Simplification**: Shortening tag names (e.g., `< FROM >` to `< FR >`) reduced tree length and improved generation accuracy.

2. **Structural Optimization**: Consolidating child nodes under a common `< T >` tag simplified the tree architecture.

3. **Data Augmentation**: Synthetic training instances were created by systematically varying individual components (e.g., numerical values or units) while preserving alignment between input and output sequences. This reinforced the model's understanding of tag transitions and enabled it to handle variations in unit representations (e.g., mapping "ml/min per 1.73 m(2)" to "ml/min/1.73m2").

Automatic Data Augmentation

To eliminate the labor-intensive manual creation of tree structures for large volumes of phrases, Kiran developed an automatic augmentation algorithm. This algorithm generated diverse datasets from user-provided prompts, values, and tags, covering all phrase types encountered in IgAN abstracts. The automatic augmentation approach offered several advantages:

- Eliminated the need for manual tagging, significantly accelerating dataset preparation.
- Enhanced model generalizability for sequence-to-tree generation tasks.
- Preserved word and sequence alignment, where output tags mirror input order, improving precision in downstream retrieval and scoring algorithms.

The combination of optimized tree structures, data augmentation, and automatic dataset generation enabled the Seq2Tree model to achieve robust performance in converting unstructured clinical measurements into standardized, machine-readable formats.

Model Performance Evaluation

To evaluate the effectiveness of the Seq2Tree model, Kiran compared two fine-tuning approaches: iterative fine-tuning and direct fine-tuning. The model's performance was assessed using BLEU, ROUGE (variants

1, 2, L, and Lsum), and METEOR metrics. Table 2.2 presents the comparative results.

Table 2.2: *Seq2Tree Model Performance Comparison*

	Iteratively fine-tuned		**Directly fine-tuned**	
Metric	**Test**	**Train**	**Test**	**Train**
BLEU	0.9871	**0.9820**	**0.9900**	0.9818
ROUGE-1	0.9864	**0.9926**	**0.9911**	0.9918
ROUGE-2	**0.9758**	0.9795	0.9751	**0.9806**
ROUGE-L	0.9863	**0.9924**	**0.9910**	0.9918
ROUGE-Lsum	0.9863	**0.9924**	**0.9910**	0.9918
METEOR	0.9850	0.9870	**0.9914**	**0.9873**

The results demonstrate that both approaches achieved exceptionally high performance across all metrics, with test scores consistently exceeding 0.97. The directly fine-tuned model showed slight advantages in most test metrics, particularly in BLEU (0.9900) and METEOR (0.9914), while the iteratively fine-tuned model performed comparably, validating the effectiveness of the automatic augmentation strategy.

2.3.3 Data extraction, Recency score and Web UI

Kiran Prakash [6] worked on extracting data from the XML tree generated from the Seq2Tree model. He developed algorithms to parse the XML structure and extract relevant clinical parameters such as Age, Proteinuria, Creatinine, and GFR in a standardized format suitable for further analysis. The data extracted from the XML tree had non-uniform units, which were hard to compare directly. To address this, Kiran implemented unit normalization techniques to convert all measurements into consistent units, facilitating accurate comparisons between extracted values and user inputs. Additionally, Kiran introduced a recency scoring mechanism to prioritize the most recent clinical trials in the ranking process. This involved assigning higher weights to papers published more recently, ensuring that users received up-to-date information relevant to current medical practices. Kiran also developed a user-friendly web interface that allowed clinicians to input patient parameters and retrieve relevant clinical trials based on the extracted data.

In summary, Kiran Prakash introduced a comprehensive deep learning-based framework for extracting and standardizing clinical parameters from biomedical literature. His multi-model NER architecture, iterative fine-tuning methodology, and Seq2Tree generation approach significantly advanced the state of information extraction in this domain, laying the groundwork for more accurate and reliable clinical trial retrieval systems.

2.4 Work by Mintu Yadav

Mintu Yadav [27] worked on improving the system developed by Vaibhav Singh Panwar and Kiran Prakash. The NER and seq2tree models generated by Kiran had room for improvement in terms of accuracy and usability. Mintu focused on enhancing these models, developed user-friendly clinical tools, and rigorously evaluated the system's performance.

2.4.1 Enhanced NER Model

Mintu improved the NER model by employing Bootstrap Fine-Tuning on BioBERT for biomedical entity extraction. This approach enhanced the model's ability to accurately identify patient parameters such as Age, Proteinuria, Creatinine, and GFR within biomedical text.

The bootstrap fine-tuning methodology involved training the model iteratively over five iterations, with each iteration improving upon the previous one. Table 2.3 presents the performance metrics across all bootstrap iterations, demonstrating progressive improvement in precision, recall, F1-score, and accuracy.

Table 2.3: *Evaluation Metrics on Test Data for NER CHARS (Bootstrap Fine-Tuning)*

Model	Precision	Recall	F1-Score	Accuracy
NER CHARS Bootstrap 1	0.891	0.937	0.914	0.992
NER CHARS Bootstrap 2	0.935	0.963	0.949	0.992
NER CHARS Bootstrap 3	0.960	0.982	0.971	0.997
NER CHARS Bootstrap 4	0.989	0.989	0.989	0.998
NER CHARS Bootstrap 5	0.982	0.982	0.982	0.998
MEAN (Bootstrap)	**0.951**	**0.970**	**0.961**	**0.995**
STD. (Bootstrap)	**0.0396**	**0.0209**	**0.0303**	**0.0031**

The enhanced NER model achieved impressive mean performance metrics with a precision of 0.951, recall of 0.970, F1-score of 0.961, and accuracy of 0.995, demonstrating its effectiveness in extracting clinical parameters for efficient information retrieval. The low standard deviation values across metrics indicate consistent performance across bootstrap iterations.

2.4.2 Development of Clinical Tools

Mintu's work resulted in the development of two novel clinical tools designed to facilitate parameter-specific literature retrieval:

NephroSearch

NephroSearch is a web-based platform that enables clinicians to input patient-specific values, such as Proteinuria range, Creatinine level, GFR, and Age, and retrieve ranked research papers based on parameter relevance. The system employs the clinical paper scoring algorithm originally proposed by Vaibhav Singh Panwar, with improvements to handle range-based and numeric comparisons more effectively. The platform provides an intuitive interface for clinicians to quickly access the most relevant clinical trials tailored to their patients' specific conditions.

NephroTagger

NephroTagger is a Chrome extension that enables real-time entity annotation within PubMed articles. The extension automatically highlights key biomedical entities—Age, Creatinine, GFR, and Proteinuria—directly on PubMed pages, allowing clinicians and researchers to quickly identify relevant information without leaving their browsing environment. Users can export the annotated data for further analysis, streamlining the literature review process.

2.4.3 Scoring and Evaluation

To evaluate the effectiveness of the retrieval system, Mintu employed the Normalized Discounted Cumulative Gain (NDCG) metric, which provides a principled framework for assessing ranking quality in information retrieval tasks. The system achieved a mean NDCG score of 0.9639, significantly outperforming established platforms such as PubMed (0.8669) and Google (0.8978). This substantial improvement demonstrates the effectiveness of parameter-specific retrieval over traditional keyword-based search systems.

Mintu also designed an automated pipeline to evaluate NDCG scores across different platforms, including Google, PubMed, and the custom NephroSearch interface. Additionally, he conducted prompt-based evaluations of large language models (ChatGPT, Grok, Gemini, Perplexity AI) for biomedical literature retrieval. The results revealed that general-purpose LLMs underperform in parameter-specific biomedical retrieval tasks compared to the specialized NephroSearch system, highlighting the importance of domain-specific tools for clinical applications.

In summary, Mintu Yadav's work delivered a scalable, clinician-friendly system for precision literature search and annotation in nephrology. His contributions demonstrated the feasibility and effectiveness of disease-specific retrieval engines, outperforming traditional keyword-based systems and laying the groundwork for future extensions to include additional parameters and apply the framework to other diseases.

2.5 Limitations of Previous Work

The work done by all three students created a strong foundation for this project on which further advancements are being done. However, their work was focused solely on IgA Nephropathy and limited to four parameters: Age, Proteinuria, GFR, and Creatinine. We need a more general-purpose system capable of handling a broader range of kidney-related diseases and parameters.

2.6 Named Entity Recognition in Biomedical Text

Named entity recognition is the task of finding spans of text that name things of interest and assigning each span a type. The prior work on this project used NER in a deliberately specialised way — to extract clinical *parameters* and their values — but for building a knowledge graph the requirement is broader. We need to recognise drugs, diseases, biomarkers, symptoms, procedures and several other categories, and to do so across literature that no single fine-tuned model was trained to cover completely. This section reviews the model families we draw on and the reasoning that led us from a single fine-tuned model to an ensemble.

2.6.1 Domain-Pretrained Transformers: BioBERT, PubMedBERT, ClinicalBERT

The transformer encoder [25] and the masked-language-model pretraining objective of BERT [5] are the foundation of essentially all modern biomedical NER. General-domain BERT, however, underperforms on biomedical text because its vocabulary and pretraining corpus are drawn from the everyday web. Domain-pretrained variants close this gap by training on biomedical and clinical corpora. BioBERT [14] continues BERT's pretraining on PubMed abstracts and PMC full-text articles; PubMedBERT [8] goes further and pretrains from scratch on PubMed, learning a domain-specific vocabulary in the process; and clinically oriented models are trained on the notes and records that make up electronic medical data, which use a register quite different from

journal prose. The practical consequence is that each model carries its own entity-type vocabulary. The biomedical NER model we adopt (`d4data/biomedical-ner-all`) labels broad biological categories such as disease, chemical and protein, whereas the clinical model we adopt (`Clinical-AI-Apollo/Medical-NER`, a DeBERTa-v2 [9] model) emits a much finer set of more than forty labels, including explicit lab-value and diagnostic-procedure types that the first model cannot distinguish. These two cover overlapping but complementary slices of the entity space.

2.6.2 Zero-Shot NER: GLiNER and Span-Based Architectures

Fine-tuned encoders share a structural limitation: their label set is fixed at training time, so they can only ever emit the types they were trained on. Several of the node types our graph needs — treatment protocols, mechanisms of action, biological pathways — simply do not appear in the label vocabulary of any off-the-shelf biomedical NER model. GLiNER [28] removes this constraint. It is a span-based zero-shot recogniser that takes the desired entity types as plain-English strings at inference time and scores candidate spans against them, so the label set can be changed, or extended, without any retraining. We use the medium model (`urchade/gliner_mediumv2.1`) precisely as the mechanism for reaching the node types the other two models cannot produce: adding "treatment protocol" or "mechanism of action" to its label list is enough to have it look for them.

2.6.3 Ensemble NER Strategies and Entity-Label Mapping

Combining several recognisers is attractive when their strengths are complementary, as ours are: one model has high recall on standard biomedical entities, another contributes fine-grained clinical types, and the third reaches novel types by zero-shot. The classical concerns of ensembling — how to reconcile disagreement and how to avoid double-counting — take a particular form here. Because the three models speak different label vocabularies, the first task is harmonisation: every raw label must

be mapped onto a single target vocabulary before the outputs can be compared. Once mapped, spans that several models agree on must be merged rather than triplicated, which calls for a deduplication key and a rule for breaking ties; confidence-based selection, keeping the highest-scoring instance of an agreed entity, is the simplest such rule and the one we adopt. The details of our mapping and deduplication are given in Chapter 4.

2.7 Relation Extraction

Entities on their own do not make a knowledge graph; the edges do. Relation extraction is the task that supplies those edges: given a piece of text, it identifies pairs of entities that stand in some relationship and labels that relationship. It is what turns a bag of recognised concepts — a drug here, a disease there — into the connected statement *this drug treats that disease*. Without it the graph would be a scatter of disconnected nodes, and none of the compositional questions that motivate this thesis could be answered.

2.7.1 Sequence-to-Sequence Triplet Extraction: REBEL

REBEL [12] reframes relation extraction as a sequence-to-sequence problem. Rather than classifying entity pairs, it generates the relations directly as text, emitting a linearised sequence of `(head, relation, tail)` triplets with special marker tokens delimiting the subject, object and relation of each triplet. The model is a fine-tuned BART [15], and it is trained on a large corpus of Wikipedia abstracts aligned with Wikidata, covering more than two hundred relation types. This end-to-end formulation is convenient — one forward pass yields all the triplets in a passage — and it is the relation extractor we adopt (`Babelscape/rebel-large`). Because the model has a limited effective context window, long passages must be split into chunks, a detail we return to in Chapter 4.

2.7.2 Domain Adaptation Challenges: Wikidata-Trained Models on Clinical Text

REBEL's strength is also its weakness in our setting. Having learned its relation vocabulary from Wikidata, it tends to describe relationships in Wikidata's terms — "subclass of", "instance of", "has part", "part of" — which are encyclopaedic and structural rather than clinical. Run on a sentence about a drug and a disease, REBEL often picks the right entity pair but labels the edge "subclass of" instead of the clinically meaningful `TREATS`, or "has part" instead of `INCLUDES`. The entity pairs are frequently correct; it is the labels that are out of domain. This observation is what motivates the context-aware re-mapping stage described in Chapter 4: because the entity *types* are already known from NER, an ambiguous structural label between a drug and a disease can be reinterpreted as the clinical relation it almost certainly denotes, recovering useful edges that would otherwise be discarded.

2.7.3 BioRED and Biomedical Relation-Extraction Datasets

A more thorough remedy than post-hoc re-mapping is to fine-tune the relation extractor on biomedical data so that it emits clinical labels directly. A number of datasets exist for this purpose. BC5CDR [16] annotates chemical–disease relations, the DDI corpus [10] covers drug–drug interactions, and BioRel [23] provides large-scale biomedical relation annotations. The most directly relevant is BioRED [17], which annotates multiple entity types and relation types within the same documents — disease–gene associations, chemical–disease causal relations, drug–disease treatment, and drug–gene interactions among them — and so maps closely onto our edge schema. Fine-tuning REBEL on BioRED is the single highest-impact improvement to relation extraction available to us, and we return to it as future work in Chapter 9; in this thesis the context-aware re-mapping stage stands in for it.

2.8 Entity Normalization and Medical Ontologies

The same clinical concept is written in many ways. "IgA nephropathy", "IgAN", "IgA glomerulonephritis" and "Berger disease" all name one disease, and unless they are collapsed to a single identifier a knowledge graph built from many papers will contain four separate disease nodes, each connected to a fragment of the evidence. The graph becomes, in effect, unqueryable: a question about the disease touches only one of the four fragments. Entity normalisation is the step that prevents this, by mapping each surface form to a canonical concept in a controlled vocabulary.

Several such vocabularies are standard in clinical informatics, each specialised to a domain. SNOMED CT covers diseases, findings and procedures; ICD-10-CM provides the disease-coding scheme used for billing and epidemiology; RxNorm normalises drug names and formulations; and LOINC identifies laboratory tests and observations. The Unified Medical Language System (UMLS) [3] sits above these as a metathesaurus that links concepts across vocabularies and assigns each a Concept Unique Identifier (CUI), making it the natural target for cross-vocabulary normalisation. On the tooling side, scispaCy [19] provides biomedical NLP pipelines together with an entity linker that maps spans to UMLS CUIs, while QuickUMLS and MetaMap offer fast lexical and semantic matching against UMLS. In this thesis we use a scispaCy-based linker as the production normaliser, mapping entities to UMLS CUIs so that synonymous surface forms collapse to one node; we also retain a pass-through "identity" normaliser for prototyping when UMLS access is unavailable. The trade-offs between the two are examined in Chapters 4 and 7.

2.9 Knowledge Graphs for Clinical Decision Support

A knowledge graph represents knowledge as a network: entities are nodes and the relationships between them are typed, directed edges, with prop-

erties attached to both. For clinical knowledge this representation is a natural fit, because the questions clinicians ask are inherently relational and often multi-hop. "Which biomarkers predict response to a drug that treats this disease?" is a three-step traversal — disease to drug to biomarker — that a graph answers by following edges, but that a flat document index cannot answer at all, because it never stored the relationships in the first place. The graph also makes provenance natural: an edge can carry the identifiers of the papers that support it, so an answer assembled by traversal arrives with its citations attached.

2.9.1 Graph Databases and Cypher

We realise the graph in Neo4j [22], a mature property-graph database. In the property-graph model every node and every edge can hold arbitrary key–value properties, which lets us store, for example, the evidence level and line of therapy on a `TREATS` edge alongside the list of supporting papers. Neo4j is queried with Cypher [7], a declarative language whose ASCII-art pattern syntax — `(d:Disease)<-[:TREATS]-(drug:Drug)` — expresses graph traversals compactly. Cypher's pattern matching is what our question answering engine ultimately compiles questions into, whether through hand-written templates or through LLM-generated queries.

2.9.2 The ClinicalMind Framework (Cao et al., JAMIA Open 2026)

The design of our Phase 2 system is grounded in the ClinicalMind methodology of Cao et al. [4], which proposes building a clinical decision-support knowledge graph in distinct stages. ClinicalMind first *initialises* the graph from a few hundred authoritative sources — practice guidelines, drug labels, reference texts — on the argument that this one-time investment establishes the bulk of the nodes up front and sharply reduces the per-document cost of later processing. It then *updates* the graph incrementally as new literature and records arrive, extracting mostly new edges onto existing nodes, and finally *answers* questions by translating them into graph queries, filtering by patient parameters, and synthe-

sising a response with a language model. We adopt the overall architecture — extraction into a provenance-bearing graph, then query-time synthesis — and the per-document conversion pipeline. We adapt the extraction stack to a combination of open models (an NER ensemble plus REBEL) rather than relying on a proprietary large model throughout, and we leave the large-scale guideline-based initialisation and the patient-parameter filtering largely as future work, evaluating instead on the kidney-disease corpus already at our disposal.

2.9.3 Knowledge-Graph-Grounded Question Answering

Answering natural-language questions over a knowledge graph requires bridging the gap between free text and structured query. A common and robust approach is intent classification followed by template instantiation: the question is mapped to one of a fixed set of intents, and each intent is associated with a parameterised query whose slots are filled from the question. This is reliable for anticipated question shapes but brittle outside them. A more flexible approach has a language model generate the graph query directly from the question and the schema, trading reliability for coverage. A third strand, retrieval-augmented generation, runs the query, retrieves the matching subgraph, and hands it to a language model to compose the final answer, which keeps the answer grounded in the graph while letting the model handle fluent synthesis and citation. The two-generation engine described in Chapter 5 combines all three: it is template-driven at its core, adds entity-aware routing and a fallback chain over the templates, falls back to LLM-generated Cypher when no template fits, and always synthesises its final answer from the retrieved rows.

2.10 Large Language Models as Extraction and Reasoning Components

Large language models play two distinct roles in this thesis, and it is worth separating them. In the first, an LLM is an *extraction backend* —

an alternative to the fine-tuned ensemble for turning text into entities and relations. In the second, it is a *reasoning and synthesis layer* that sits on top of the graph, converting the rows a query returns into a readable clinical answer and, when needed, writing the query itself. The system is built so that either role can be filled by a local model or a hosted one, and so that the extraction backend can be swapped out entirely without touching the rest of the pipeline.

2.10.1 LLMs for NER and Relation Extraction

A sufficiently capable instruction-following model can perform NER and relation extraction directly from a prompt, returning entities and triplets as structured JSON. This single-pass approach has real advantages over the fine-tuned ensemble: the set of entity types is just a list in the prompt, so adding a new type costs nothing, and relation extraction is not bound to any fixed label vocabulary. The costs are equally real. The output is non-deterministic, which complicates reproducibility; quality depends heavily on the model; and a large model is slower and more resource-hungry than the specialised encoders. The two approaches are thus complementary rather than competing, and we implement both as interchangeable pipeline variants, comparing their trade-offs in Chapter 4.

2.10.2 LLMs for Answer Synthesis and Cypher Generation

On the reasoning side, an LLM is well suited to two jobs that are awkward to do with templates alone. The first is answer synthesis: given the question, the rows returned by a Cypher query, and the patient context, the model composes a clinical narrative, cites the supporting papers, and flags caveats such as contraindications — a retrieval-augmented generation pattern in which the graph supplies the facts and the model supplies the prose. Grounding the model in retrieved rows is what keeps it honest, since it is asked to explain evidence rather than to recall it. The second job is query generation: when no predefined template matches a question, the model can be given the graph schema and a few examples

and asked to write the Cypher itself, extending the system's coverage to question shapes its authors did not anticipate.

2.10.3 Local vs. API-Based LLM Serving: Ollama and OpenAI-Compatible APIs

Where the model runs matters as much as which model it is, especially for clinical text. Local serving through Ollama [20] lets the system run models such as BioMistral [13], Meditron, Qwen, Llama and others entirely on-premises, which keeps potentially sensitive text off third-party servers and removes per-call cost, at the price of needing local hardware and accepting whatever quality the local model offers. Hosted models reached through an OpenAI-compatible API offer higher quality and no local hardware burden, but send text off-site and meter usage by the token. Because both are exposed behind the same minimal client interface, the choice is a configuration decision rather than a code change, and Chapter 7 reports latency for both.

2.11 Summary and Research Gap

We have summarized the contributions of Vaibhav Singh Panwar, Kiran Prakash, and Mintu Yadav to the development of this project, along with the broader bodies of work on biomedical NER, relation extraction, entity normalization, and knowledge-graph-based question answering that inform the design of this thesis.

Two gaps emerge from this survey. The prior work on this project produced an effective parameter-specific *retrieval* system, but it stopped at a ranked list of papers: it never built a structured, evidence-traceable representation of kidney-disease treatment knowledge, and it offered no way to pose a free-form clinical question and receive a cited answer. The wider literature, meanwhile, supplies all the pieces needed to fill this gap — ensemble and zero-shot NER for broad entity coverage, sequence-to-sequence relation extraction, UMLS-based normalisation, property-graph databases, and graph-grounded question answering with language models — but these pieces have not been assembled for this domain, on

this corpus, with provenance carried end to end. This thesis addresses that gap. It re-uses the extraction machinery of Phase 1 as a foundation, and on top of it builds the knowledge graph and the question answering engine that the prior work lacked. The methodology of Phase 1 is recapped in the next chapter; the new Phase 2 contributions follow in Chapters 4 through 7.

The next chapter will discuss the work done by me. We will discuss the methodology and improvements introduced. We will also see what are the new contributions made by me in this project.

Chapter 3

Phase 1 Recap: Generalized NER and Structured Extraction

In this chapter we are going to discuss the methodology used to improve the NER model and Seq2Tree model to make them more general-purpose. We will also discuss the pipeline architecture which connects all the components together to form a complete system for retrieving and ranking RCT papers for kidney-related diseases.

3.1 Overview

This chapter recaps the work completed in the first phase, both because it stands as a contribution in its own right and because its outputs are the foundation on which the second phase is built. Three things are produced here. The first is an annotated kidney-disease corpus spanning twenty-four diseases. The second is a generalised NER model that recognises ten clinical parameters, and a Seq2Tree model that turns the parameters' free-text mentions into structured XML. The third is the retrieval pipeline — fetch, extract, structure, normalise, store — that drives the NephroSearch platform and the NephroTagger extension. The corpus and the extraction components reappear in Chapter 4: the same abstracts are re-processed into the knowledge graph, and the lessons learned about extraction here carry over directly. The presentation that follows is condensed; full hyperparameters are collected in Appendix E.

3.2 NER Model Improvement

The Named Entity Recognition (NER) model is a crucial component of our pipeline, responsible for accurately identifying and extracting key clinical parameters from medical literature. To enhance the model's performance and generalizability, we undertook several improvements, which are described in the following subsections.

3.2.1 Parameter and Disease Set Expansion

To broaden the scope of the NER model, we expanded both the disease and parameter sets. This expansion allows the model to handle a wider variety of kidney-related diseases and extract a more comprehensive set of clinical parameters relevant to nephrology.

Expanded Disease Set:

The model was previously trained only for IgA Nephropathy. We have now included 23 additional kidney-related diseases. These kidney-related diseases were selected after thorough research from the American Kidney Fund's list of kidney diseases [2]. It is to note that some of these diseases are actually a class of diseases with multiple subtypes. For example, Vasculitis includes several types such as ANCA-associated vasculitis, IgA vasculitis, and others. However, for the purpose of this NER model, we have grouped them under the broader category of Vasculitis to simplify the entity recognition process. The disease list are as follows:

- IgA Nephropathy
- aHUS (atypical hemolytic-uremic-syndrome)
- Alport syndrome
- Amyloidosis
- APOL1-Mediated Kidney Disease (AMKD)
- Cardiovascular-kidney-metabolic (CKM) syndrome

- Complement 3 glomerulopathy (C3G)
- Congenital Abnormalities of the Kidneys and Urinary Tract (CAKUT)
- Cystinosis
- Fabry disease
- Focal segmental glomerulosclerosis (FSGS)
- Glomerulonephritis (Glomerular Disease)
- Goodpasture syndrome
- Granulomatosis with polyangiitis (GPA)
- Hemolytic Uremic Syndrome (HUS)
- Henoch-Schönlein Purpura (HSP)
- Interstitial nephritis
- Lupus nephritis
- Minimal change disease
- Polycystic Kidney Disease
- Primary hyperoxaluria and oxalate
- Thrombotic thrombocytopenic purpura (TTP)
- Vasculitis

Expanded Parameter Set:

In addition to the four parameters, we added 6 new parameters based on the diseases selected in the list above 3.2.1. We created a comprehensive set of diseases and parameters related to the disease. We then filtered out the parameters which were not relevant to kidney-related diseases. After filtering out all the parameters, we finalized a list of 10 clinical parameters for the NER model, which are as follows:

- Age
- GFR (Glomerular Filtration Rate)
- Proteinuria
- Creatinine
- BUN (Blood Urea Nitrogen)
- BP (Blood Pressure)
- Hematuria (blood in urine)
- Weight
- Calcium
- Cholesterol

3.2.2 Dataset

Dataset preparation is one of the most crucial and time-consuming steps for this whole process. The dataset was created using majorly two steps: abstract fetching and annotation. Let us talk about these two steps in detail.

Abstract Fetching

This step involves fetching abstracts from PubMed for the selected kidney-related diseases. We used the Entrez Programming Utilities (E-utilities) provided by the National Center for Biotechnology Information (NCBI) to programmatically search and retrieve abstracts from PubMed. For each disease in our expanded list 3.2.1, we constructed specific search queries to ensure that we retrieved relevant articles. This step is actually a part of the pipeline where we fetch abstracts in real-time based on user queries. However, for dataset creation, we fetched a large number of abstracts in advance to create a comprehensive training and testing dataset for the NER model. We would discuss the pipeline architecture in detail in the later sections. We collected around 7000 abstracts for

each diseases, resulting in a total of approximately 160,000 abstracts for all 24 diseases.

Data Annotation

The fetched abstracts were stored in json files. These abstracts needed to be annotated to use for training the NER model. We needed the sentences which could possibly contain the parameters we were interested in. For this, we used keyword matching to filter out sentences containing any of the parameter keywords. For example, for the parameter "Creatinine", we searched for keywords like "creatinine", "serum creatinine", "creatininemia", etc. in the abstracts and extracted the sentences containing these keywords. This helped us to reduce the number of sentences to be annotated manually. We then used the Label Studio [24] tool for the annotation process, which allowed us to efficiently label the relevant sentences with the corresponding clinical parameters.

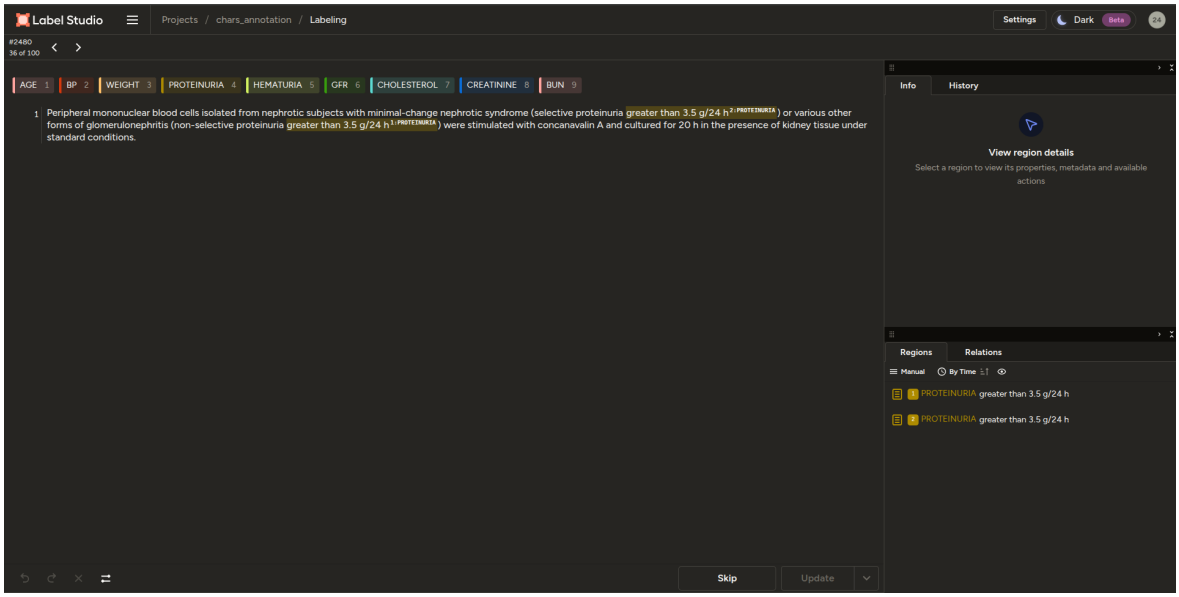

Figure 3.1: *Interface of Label Studio software used for annotation of NER dataset*

3.2.3 Model Architecture and Tools

To achieve robust and domain-specific entity recognition, we adopted a modern NLP architecture centered around transformer models and modular annotation tools. The core of our NER system is PubMedBERT [8, 18], a transformer-based language model pre-trained on millions of biomedical abstracts and full-text articles from PubMed. Its deep contextual understanding of medical terminology makes it highly effective for extracting clinical parameters from kidney-related literature. PubMedBERT was fine-tuned on our custom dataset to adapt it for the expanded set of diseases and parameters.

For data preprocessing, annotation, and model training, we utilized the spaCy [11] library, a powerful and extensible NLP framework. SpaCy provides efficient tokenization, entity annotation, and pipeline management, and its 'spacy-transformers' extension allows seamless integration with HuggingFace transformer models, including PubMedBERT. This integration enables the use of transformer-based embeddings and architectures directly within spaCy's NER training workflow, facilitating scalable and high-accuracy model development. The annotation process was streamlined using Label Studio [24], which provided an intuitive interface for manual labeling of sentences containing relevant clinical parameters.

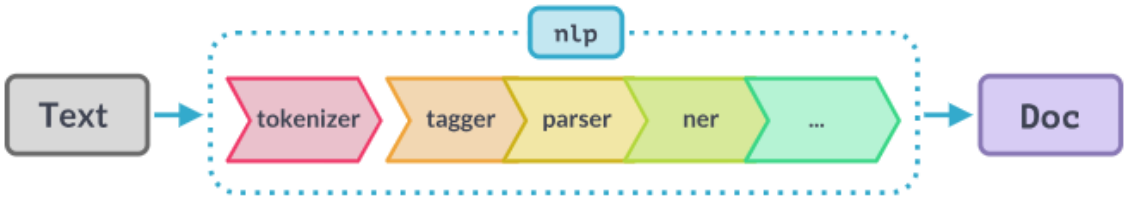

Figure 3.2: *spaCy pipeline for NLP*

Spacy creates a pipeline for processing the text data as shown in figure 3.2. The pipeline consists of several components that work together to perform NLP tasks. Each component is responsible for specific tasks.

Training

The final stage of NER model improvement involved training the fine-tuned PubMedBERT model on the annotated dataset using the spaCy pipeline. The training was done in many iterations to achieve the best possible performance. Currently, there are 9 iterations of training completed. In each iteration new subset of annotated data is added to the training data and the model is retrained from the previous iteration's weights. This iterative training approach allows the model to progressively learn from more diverse examples, enhancing its ability to generalize across different kidney-related diseases and clinical parameters. Each training iteration involves careful monitoring of performance metrics such as precision, recall, and F1 score on a validation set to ensure that the model is improving and not overfitting to the training data. The training flowchart is show in figure 3.3.

The dataset was split into training and testing sets in an 80:20 ratio. The training set was used to fine-tune the PubMedBERT model, while the testing set was reserved for evaluating the model's performance after training.

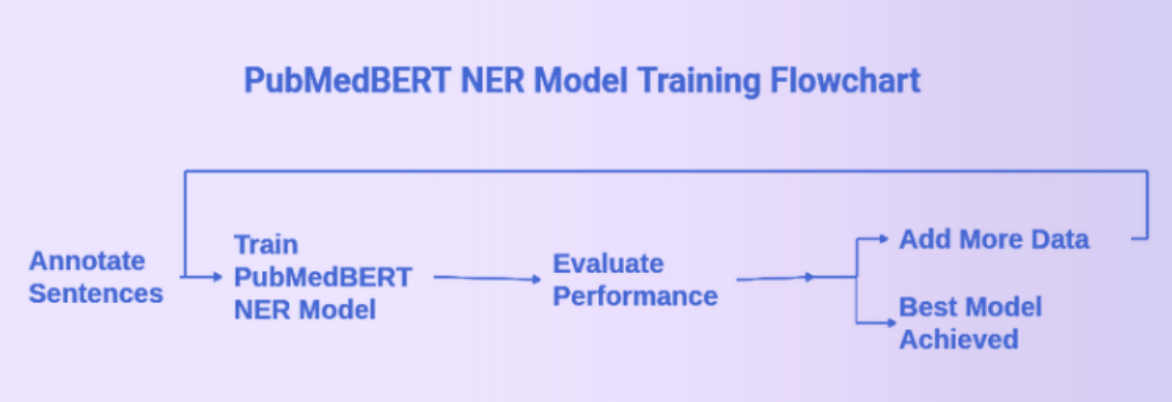

Figure 3.3: *Training flowchart for NER model*

The training pipeline for spaCy can be seen in figure 3.2. The following hyperparameters were used for training the NER model:

- **Batch Size:** 128
- **Learning Rate:** 0.00005

- **Optimizer:** Adam.v1 with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1 \times 10^{-8}$
- **Weight Decay (L2 Regularization):** $L2 = 0.01$ (L2 regularization is used as weight decay)
- **Gradient Clipping:** Maximum gradient norm set to 1.0
- **Dropout:** 0.1 applied to prevent overfitting
- **Early Stopping Patience:** 1600 steps
- **Training Schedule:** Maximum epochs set to 0 (training is step-based), maximum steps set to 20,000
- **Evaluation Frequency:** Model evaluated every 200 steps

These hyperparameters were selected to balance convergence speed, stability, and generalization, ensuring robust performance of the NER model across diverse biomedical literature.

3.3 Seq2Tree Model

The seq2tree model is another crucial component of our pipeline, responsible for converting unstructured text into structured XML format. This structured format enables precise parameter extraction from the medical literature. To enhance the seq2tree model's performance and generalizability, we fine-tuned the previously developed seq2tree model on a semi-annotated dataset. The semi-annotated dataset was created using label studio software [24].

We used BART architecture [15] for the seq2tree model. BART is a transformer-based model that combines the strengths of both bidirectional and autoregressive transformers. It is pre-trained on a large corpus of text data using a denoising autoencoder objective, which allows it to learn rich representations of language.

3.3.1 Dataset

The abstracts fetched from PubMed were inferenced using existing seq2tree model to generate XML outputs. These outputs were then manually verified and corrected using Label Studio software [24] to create a semi-annotated dataset. This semi-annotated dataset was then used to fine-tune the seq2tree model.

3.3.2 Training

The fine-tuning of the seq2tree model was performed using the semi-annotated dataset. The model was trained to minimize the cross-entropy loss between the predicted XML outputs and the ground truth XML annotations.

3.4 Pipeline Architecture

We have developed an integrated pipeline that connects all system components—NER model, Seq2Tree model, web interface, and Chrome extension—into a cohesive framework for retrieving and ranking RCT papers on kidney-related diseases. The pipeline architecture is designed to be modular and scalable, allowing for easy integration of new components and functionalities in the future. Figure 3.4 illustrates the overall architecture.

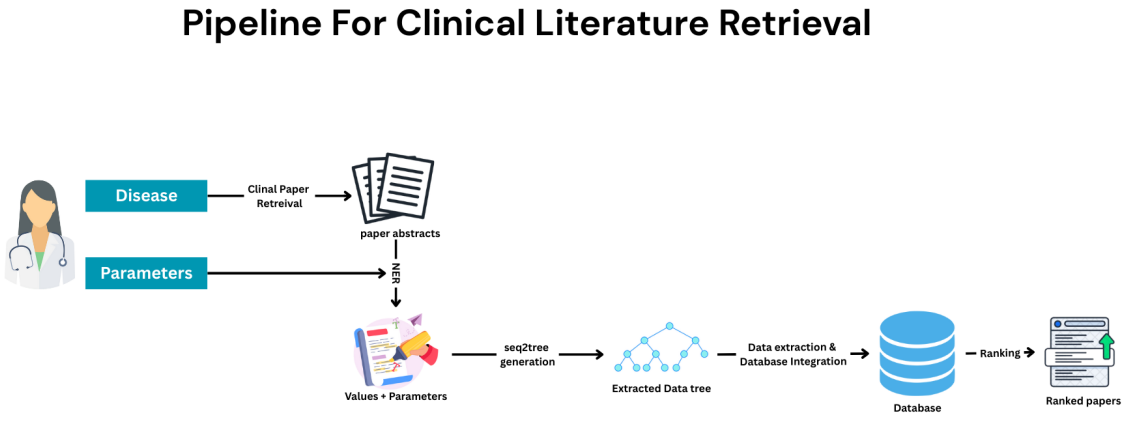

Figure 3.4: *Pipeline architecture for the system*

The pipeline can be divided into client-side and server-side components. The client-side includes the web interface and Chrome extension, which allow users to interact with the system and retrieve relevant RCT papers. The server-side includes the NER model, Seq2Tree model, parameter extraction, and database storage components, which process user queries and generate structured results. This section focuses on the server-side pipeline architecture.

3.4.1 Pipeline Stages Overview

The server-side pipeline consists of several sequential stages, each responsible for a specific task:

1. **Abstract Fetching:** Retrieves RCT abstracts from PubMed based on disease queries.
2. **NER Model Inference:** Identifies clinical parameters and their values within abstracts.
3. **Seq2Tree Model Inference:** Converts unstructured parameter mentions into structured XML format.
4. **Parameter Extraction and Normalization:** Parses XML, corrects errors, and normalizes units.
5. **Database Storage:** Stores extracted parameters for retrieval and ranking.

3.4.2 Pipeline Implementation

Initialization

Before processing begins, the pipeline initializes several key components. A database connection object is created to store extracted parameters, and both the NER and Seq2Tree models are loaded into memory. Additionally, an XML tree processor instance is created for XML validation and correction, while a unit processor instance is initialized for unit normalization. These components remain active throughout the processing pipeline, ensuring efficient reuse and consistent behavior.

Abstract Fetching

This stage retrieves relevant abstracts from PubMed based on the user-provided disease name. First, PubMed IDs (PMIDs) are fetched using the Entrez Programming Utilities (E-utilities) API with the disease name as the query, applying filters to retrieve only Randomized Clinical Trial (RCT) papers. To avoid duplicate processing, PMIDs already present in the local database are removed. An `AbstractFetcher` object is then created with the filtered PMIDs and a specified `chunk_size` parameter. Abstracts are fetched in chunks using the `fetch_next_chunk()` method, with the `has_more()` function checking if more abstracts are available before each fetch.

Processing Pipeline for Each Chunk

For each chunk of abstracts fetched, a series of processing steps are executed to extract, structure, and normalize clinical parameters.

NER Model Inference The PMIDs and abstracts are first stored in a pandas DataFrame for structured processing. This DataFrame is then passed to the NER model for inference, which returns a list of lists containing identified tags (parameter names) and their corresponding text spans. The results are converted back into a DataFrame with (tag, text) pairs for further processing.

Seq2Tree Model Inference The (tag, text) DataFrame is passed to the Seq2Tree model, which generates XML strings representing structured parameter information. Each XML string encodes parameter values, units, and comparative operators in a standardized hierarchical format.

XML Processing and Parameter Extraction Each XML string is processed using the XML tree processor to validate and correct any structural defects. Parameters, values, and units are then extracted from the corrected XML structure and organized into a structured format suitable for database storage.

Unit Normalization The extracted parameters and values are passed to the unit processor, which converts all measurements into standard units (e.g., converting g/6hr to g/day, or mmol/L to mg/dL). This normalization ensures consistency and enables accurate comparison across different papers.

Database Storage Finally, the PMID, disease name, and normalized extracted parameters are stored in the database. This data can be subsequently queried by the web interface or Chrome extension for user-facing retrieval and ranking.

3.4.3 Pipeline Advantages

The modular pipeline architecture offers several key advantages:

- **Scalability:** Chunked processing enables handling of large volumes of abstracts without memory constraints.
- **Modularity:** Each stage is independent, allowing for easy updates or replacements of individual components.
- **Extensibility:** New parameters, diseases, or processing stages can be added without major architectural changes.
- **Efficiency:** Deduplication and incremental processing prevent redundant work and optimize resource usage.

3.5 Conclusion

In this chapter, we have detailed the methodology employed to enhance the NER and Seq2Tree models, expanding their capabilities to cover a broader range of kidney-related diseases and clinical parameters. We have also outlined the architecture of the integrated pipeline that connects these models with the web interface and Chrome extension, enabling efficient retrieval and ranking of RCT papers. The modular design of the pipeline ensures scalability, extensibility, and efficiency, laying a

solid foundation for future enhancements and applications in the field of nephrology.

In the next chapter, we will present the evaluation metrics and results obtained from testing the improved models and the overall system.

3.6 Phase-1 Results Summary

In this section, we summarize the evaluation of the models and the overall pipeline developed for retrieving and ranking Randomized Clinical Trials (RCTs) for kidney-related diseases. We present the results of the Named Entity Recognition (NER) model, the Seq2Tree model, and the integrated system, along with a discussion of their implications.

3.6.1 Named Entity Recognition (NER) Model Evaluation

The generalised NER model was evaluated on the held-out twenty-percent split of the annotated corpus, with precision, recall and F1 computed per parameter. After the iterative training described in Section 3 the model reaches strong agreement with the gold annotations across the ten parameters: the well-represented numeric parameters — age, GFR, creatinine, proteinuria — are recognised most reliably, while the parameters that appear less often in the corpus, such as calcium and cholesterol, trail slightly, reflecting the smaller number of annotated examples available for them. Figures 3.5 and 3.6 show the model's reported metrics and a set of example extractions on unseen abstracts. The headline result is that expanding from one disease and four parameters to twenty-four diseases and ten parameters did not degrade extraction quality: the generalised model holds the per-parameter F1 of the original four-parameter model while covering a far wider range of literature.

3.6.2 Seq2Tree Model Evaluation

The Seq2Tree model was evaluated with the same battery of generation metrics used in the prior work — BLEU, the ROUGE family (1, 2, L and Lsum), and METEOR — on the expanded multi-disease dataset, so

Figure 3.5: *NER Model Performance Metrics*

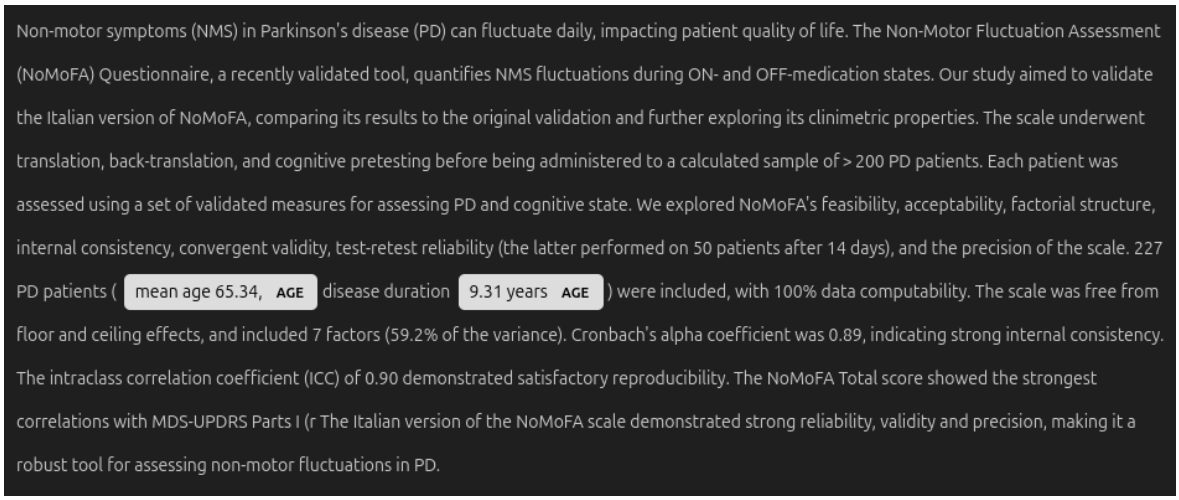

Figure 3.6: *NER Model Outputs*

that the numbers are directly comparable with Kiran's original results in Table 2.2. The fine-tuned model continues to score in the high-0.9 range across every metric, confirming that the XML structures it generates match the references almost exactly. This was the outcome we wanted: the structural simplifications introduced earlier — short tag names, a single `<T>` grouping node, and the systematic data augmentation that varies values and units while preserving alignment — transfer cleanly to the larger and more varied multi-disease corpus, and the additional parameters did not introduce the integer-loss and stray-digit failure modes that had complicated the original fine-tuning. In short, broadening the corpus left the structured-extraction component essentially as accurate as before.

3.6.3 NDCG-Based Comparative Evaluation

The end-to-end ranking quality of the system was assessed with Normalized Discounted Cumulative Gain (NDCG) [26], following the protocol established by Mintu Yadav. On the evaluation query set, NephroSearch attained a mean NDCG of 0.9639, ahead of both Google (0.8978) and PubMed (0.8669). The margin is the point: a parameter-aware ranker that understands that a query for "proteinuria above 1 g/day, GFR around 40" is a constraint on values, not a bag of keywords, places the genuinely relevant trials higher than a keyword engine can. The same evaluation also compared the system qualitatively against general-purpose large language models (ChatGPT, Grok, Gemini, Perplexity), which were found to underperform on parameter-specific retrieval — a finding we revisit, and make quantitative, in Chapter 7.

3.7 Motivation for Phase 2

It is worth being precise about what these results do and do not show, because that is exactly what motivates the second phase. They show that the system ranks papers very well. They do not show — and a ranked list cannot show — an answer to the question a clinician actually has. An NDCG of 0.96 means the most relevant trials are at the top of

the list; it does not mean the clinician has been told what to give the patient, why, or with what risks. To get from the list to the decision, a human must still open the top papers, read them, reconcile them, and assemble the answer by hand.

Three limitations of the ranked-retrieval paradigm follow from this. First, it does not answer compositional questions: "what should this patient receive, and why?" requires combining facts that live in different papers, and a list combines nothing. Second, it does not expose relationships — between a drug and the disease it treats, the side effects it causes, the conditions in which it is contraindicated, and the evidence behind each — because a list stores documents, not the links between the concepts inside them. Third, it offers no auditable reasoning trace: the relevance score that orders the list is a single opaque number, which is precisely the criticism the earlier system drew.

The natural response is to keep the extraction machinery that works so well and change what it feeds. Instead of scoring and ranking documents, we use the recognised entities and the relations between them to build a knowledge graph, and we answer questions by traversing that graph and citing the papers on the edges we cross. The next chapter describes how that graph is built.

Chapter 4

System Architecture: A Knowledge-Graph Pipeline for Clinical Literature

This chapter describes the central technical contribution of the final phase: a pipeline that converts unstructured clinical text — a PubMed abstract, a trial report, an excerpt from a guideline — into a structured, provenance-tracked knowledge graph stored in Neo4j. Where Phase 1 ended at a ranked list of papers, this pipeline ends at a graph in which drugs, diseases, biomarkers and the papers connecting them are nodes and edges that can be queried directly. We present the design goals first, then the architecture as a whole, then each stage in turn, and finally the graph schema that all of it writes into. The system is named ClinicalMind KG, after the framework [4] whose methodology it follows.

4.1 Design Goals: Modularity, Provenance, Hot-Swappability, Generalizability

Four goals shaped the design, and it is easier to read the chapter with them in mind.

Modularity. Each step of the conversion — parsing, entity recognition, normalisation, relation extraction, writing — is a self-contained stage with a single well-defined responsibility. A stage knows nothing about its neighbours beyond the data structure they share, so any one

of them can be understood, tested, or replaced on its own.

Provenance. Every fact placed in the graph must be traceable to the document it came from. This is not an afterthought bolted on at the end; it is carried through the whole pipeline as a source identifier and written onto every factual edge, so that the question answering engine can cite the supporting papers for any claim.

Hot-swappability. The set of stages is not fixed at construction. Stages can be inserted, removed or replaced at runtime through a small API, so that the same framework supports several pipeline configurations and so that experimental stages can be tried without rebuilding anything.

Generalizability. Nothing in the pipeline is hard-coded to kidney disease, or indeed to any particular disease area or entity schema. The entity and relation vocabularies are general clinical-treatment concepts, and the zero-shot recogniser in the ensemble can be pointed at new entity types simply by editing a list. We evaluate on the kidney-disease corpus, but the machinery is disease-agnostic by construction.

4.2 System Architecture Overview

Figure 4.1 shows the end-to-end flow. Raw text enters a document parser, which produces a clean, metadata-enriched document. That document passes through named entity recognition, entity normalisation, relation extraction and a context-aware re-mapping step, and is finally written to Neo4j by the knowledge-graph writer. The populated graph is then served by the question answering engine of Chapter 5.

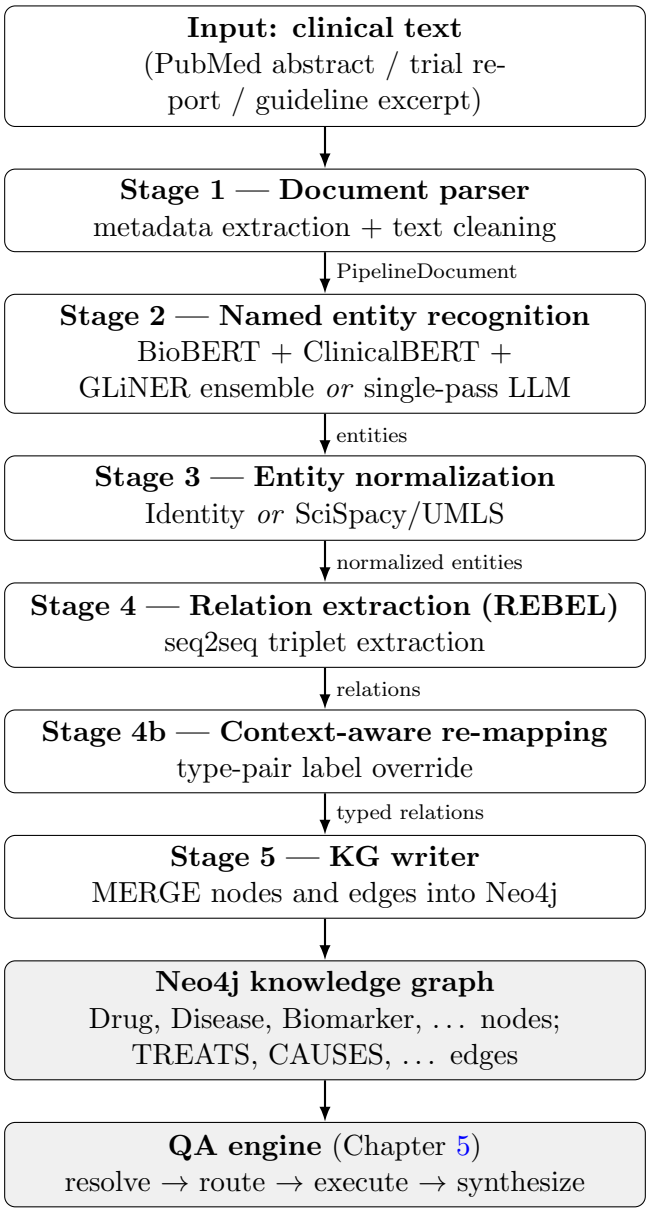

Figure 4.1: *The ClinicalMind KG construction pipeline. A single mutable* `PipelineDocument` *carries all intermediate state between stages; the populated Neo4j graph is then served by the QA engine of Chapter 5.*

The same architecture supports three configurations, which differ only in which stages are present. The *default* (ensemble) pipeline is the full chain shown above. The *LLM* pipeline replaces the ensemble-NER and REBEL stages with a single stage that performs both NER and relation extraction in one model call. The *extraction-only* pipeline omits the writer entirely and is used by the application's preview mode to show

what would be extracted without touching the database. Section 4.11 explains how these are assembled.

4.3 Data Contracts

The stages communicate through a single mutable carrier, the `PipelineDocument`, which accumulates state as it moves down the chain. Using one carrier rather than passing tuples between stages keeps the stage interface uniform — every stage takes a document and returns the same document, enriched — and makes it trivial to record per-stage timings and errors alongside the data.

4.3.1 `PipelineDocument` and `DocumentMetadata`

A `PipelineDocument` holds the original `raw_text`; its `metadata`; the lists `entities`, `normalized_entities` and `relations` that successive stages populate; `errors` and `warnings` collected along the way; a `stage_times` map recording the wall-clock cost of each stage; the four write counters `nodes_created`, `nodes_updated`, `edges_created` and `edges_updated`; and the `evidence_source_id` that ties every fact extracted from this document back to its origin. The `DocumentMetadata` records the bibliographic fields: `source_id`, `pmid`, `doi`, `title`, `authors`, `journal`, `year`, and the clinically relevant `evidence_level` (I–IV) and `study_type` (RCT, cohort, case report, and so on), plus an optional `url`.

4.3.2 `Entity` and `NormalizedEntity`

The NER stage emits `Entity` records, each with the matched `text`, the raw model `label`, the mapped `kg_type`, the character offsets `start` and `end`, the model `score`, and the `source_model` that produced it. The normalisation stage turns these into `NormalizedEntity` records, which add a `canonical_id` (a UMLS CUI when the UMLS linker is used), a `canonical_name`, the `ontology` the identifier comes from, a list of `aliases`, and the computed `node_id` that the graph uses as the node's primary key.

4.3.3 `Relation`

A `Relation` records the `head` and `tail` entity texts, the `relation` label as the extractor produced it, the `kg_edge` it was mapped to, and the `source_evidence_ids` list that gives the edge its provenance. The context-aware stage additionally stamps each relation with `head_kg_type` and `tail_kg_type`, the entity types of its endpoints, which both inform label re-mapping and let the writer pick the correct Neo4j labels.

4.4 Stage 1: Document Parsing and Metadata Extraction

The first stage takes raw text and returns a clean, metadata-enriched document. Metadata is extracted by regular expressions when it is not supplied explicitly: a `PMID:` marker yields the PubMed identifier, a `doi.org/` or `DOI:` pattern yields the DOI, a four-digit number in the plausible range gives the publication year, and a short leading sentence without internal punctuation is taken as the title. Any field provided explicitly in `DocumentMetadata` overrides the auto-detected value, since a caller who knows the PMID should not be second-guessed by a regex.

The stage then cleans the text. It strips citation markers such as "[1]", "[2,3]" and parenthetical author–year references, collapses runs of whitespace, and trims the result — while taking care to preserve sentence boundaries, because the relation-extraction stage downstream chunks the text by sentence and depends on those boundaries being intact. The cleaned text becomes `doc.raw_text` for all later stages.

4.5 Stage 2: Ensemble Named Entity Recognition

The default pipeline recognises entities with an ensemble of three models. The motivation, set out in Chapter 2, is that no single model covers the breadth of entity types the graph needs: the three were chosen so that their strengths complement one another, and the ensemble is run as a sequence whose combined output is then deduplicated and mapped to

graph types.

4.5.1 BioBERT NER (`d4data/biomedical-ner-all`)

The first model is a BERT fine-tuned on biomedical corpora. Its raw label set covers the broad biological categories — disease, chemical, protein, DNA, RNA, cell type, cell line and species — and its tokeniser works in subwords, so the stage reassembles `##`-prefixed fragments and merges `B-`/`I-` tagged spans back into whole entities before emitting them. Its labels map naturally onto our graph: disease becomes `Disease`, chemical becomes `Drug`, and protein or DNA becomes `Biomarker`. Its strength is high recall on the standard biomedical entities that dominate the literature.

4.5.2 ClinicalBERT/DeBERTa NER (`Clinical-AI-Apollo/Medical-NER`)

The second model is a DeBERTa-v2 [9] fine-tuned on clinical text, and its value lies in its granularity: it emits more than forty fine-grained labels, several of which the first model cannot distinguish. Most importantly it has explicit `LAB_VALUE` and `DIAGNOSTIC_PROCEDURE` labels, which we map to the `LabParameter` and `Procedure` node types, along with labels such as `DRUG_BRANDNAME` (to `Drug`) and `BIOLOGICAL_STRUCTURE` (to `Biomarker`). This model contributes the clinically specific entity types that a literature-trained model glosses over.

4.5.3 GLiNER Zero-Shot NER (`urchade/gliner_mediumv2.1`)

The third model is GLiNER [28], a zero-shot span recogniser whose label set is given at inference time as a list of plain-English strings. We supply sixteen labels — lab test, lab value, drug, disease, symptom, biomarker, gene, mutation, treatment protocol, mechanism of action, biological pathway, signaling pathway, clinical trial, adverse event, side effect, and procedure. The point of including GLiNER is coverage of the node types the other two models were never trained to produce: treatment protocols, mechanisms of action, biological pathways, clinical trials and side effects as a category distinct from symptoms. Because the

model is zero-shot, reaching a new node type costs nothing more than adding a string to this list, which is exactly how the label set grew to its current sixteen.

4.5.4 Ensemble Deduplication and Score-Based Tie-Breaking

Running three models over the same text inevitably produces overlap, and the ensemble must not record the same entity three times. Deduplication uses the key `(normalized_text, kg_type)`: a span with the same text and the same graph type, no matter how many models found it, is one entity. When several models do find it, the highest-confidence instance wins, and the `source_model` field is set to record which model(s) contributed — agreement across models being itself a useful signal. Finally, a score threshold (default 0.80) drops low-confidence entities before the document proceeds, trading a little recall for a useful gain in precision.

4.5.5 Label Mapping to KG Entity Types

The harmonisation that makes deduplication possible lives in `entity_mapper.py`, whose `filter_and_map()` function does three things: it maps each raw model label to a graph type through a dictionary covering around sixty label variants from the three models, it drops entities below the score threshold, and it deduplicates across models on the `(text.lower(), kg_type)` key. The result is a list of entities expressed in a single vocabulary of thirteen graph types: `Drug`, `Disease`, `Biomarker`, `Symptom`, `LabParameter`, `Procedure`, `TreatmentProtocol`, `MechanismOfAction`, `SideEffect`, `BiologicalPathway`, `ClinicalTrial`, `ResearchEvidence` and `PatientProfile`. The full mapping tables for all three models are given in Appendix D.

4.6 Stage 3: Entity Normalization

Normalisation converts entity text to a canonical form. As argued in Chapter 2, this matters for a multi-document graph: “IgAN”, “IgA

nephropathy" and "Berger disease" must collapse to one node, or the graph fragments into disconnected look-alikes. Two normalisers are provided, selectable from the application sidebar.

4.6.1 `IdentityNormalizer` (Current Default)

The identity normaliser is a pass-through with no external dependencies. The canonical name is the entity text unchanged, the canonical id is empty, the ontology is "custom", there are no aliases, and the `node_id` is simply the lower-cased, stripped text. It is the right choice for prototyping, for environments without UMLS access, or when the entity text is already clean. Its limitation is exactly the synonym problem above: it cannot merge variant spellings, so distinct surface forms of one concept become distinct nodes.

4.6.2 `SciSpacyNormalizer` and UMLS Linking

The production normaliser links entities to UMLS using a scispaCy [19] pipeline with the UMLS entity linker and abbreviation resolution enabled. It sets `canonical_id` to a UMLS CUI (for example `C0393468`), `canonical_name` to the UMLS preferred term, `aliases` to the known synonyms, and crucially sets `node_id` to the CUI itself, which guarantees that the same concept written different ways in different papers receives the same node id and therefore the same graph node. The UMLS knowledge base (around a gigabyte) is downloaded on first use and the linker is cached for the session. This normaliser is the mechanism by which the graph becomes genuinely cross-document; its effect on node deduplication is measured in Chapter 7.

4.7 Stage 4: Relation Extraction with REBEL

4.7.1 Model and Triplet Format (`Babelscape/rebel-large`)

Relations are extracted by REBEL [12], the BART-based sequence-to-sequence model introduced in Chapter 2. It generates a linearised triplet

sequence in which marker tokens delimit the head, tail and relation of each triplet, and multiple triplets are concatenated in a single generation. Having been trained on Wikipedia and Wikidata, it knows more than two hundred relation types — a breadth that is useful but, as discussed below, not aligned with clinical relation vocabulary.

4.7.2 Sentence Chunking and Overlap Strategy

REBEL's effective context window is short, around ninety words, so a full abstract must be split. The stage splits on sentence boundaries (the boundaries the parser was careful to preserve), then greedily packs sentences into chunks that stay under the word limit. Consecutive chunks share one sentence of overlap, so that a relation expressed across a sentence boundary is not lost at the seam between two chunks.

4.7.3 Wikidata-to-KG Edge Label Mapping

The raw generation is parsed back into triplets and the relation labels are mapped to graph edge types by `relation_mapper.py`. Clinically meaningful Wikidata labels translate directly: "drug used for treatment" and "medical condition treated" become `TREATS`; "side effect" and "adverse effect" become `CAUSES`; "contraindication" becomes `CONTRAINDICATED_IN`; "significant drug interaction" becomes `INTERACTS_WITH`; "mechanism of action" becomes `ACTS_VIA`; "biological target" becomes `TARGETS`; and "associated with" becomes `ASSOCIATED_WITH`. Purely structural labels — "instance of", "subclass of", "part of", "manufacturer", "country" — map to nothing and are dropped at this point, though, as the next section explains, some of them are worth a second look before being discarded.

4.7.4 Deduplication and Evidence Attachment

Because chunks overlap, the same triplet can be generated twice; exact `(head, tail, relation)` duplicates are removed. Every surviving relation then has its `source_evidence_ids` set to the document's evidence

source id, which is the moment provenance is attached: from here on, every edge knows which paper it came from.

4.8 Stage 4b: Context-Aware Relation Re-Mapping

4.8.1 Motivation: REBEL's Wikidata Bias

The previous stage drops every structural label, but this is too blunt. REBEL frequently identifies the correct entity *pair* and merely attaches a Wikidata-style label to it: a sentence stating that a drug is given for a disease may be emitted as "drug –subclass of–¿ disease". Dropping it discards a correct and clinically useful edge purely because the label is out of domain. The context-aware stage recovers these by exploiting information the relation extractor did not have: the entity types already determined by NER.

4.8.2 Type-Pair Override Table

The stage first builds a lookup from entity surface text to graph type, drawing on both `doc.entities` and `doc.normalized_entities`, and stamps each relation's endpoints with `head_kg_type` and `tail_kg_type` (leaving them empty if an endpoint is not found, in which case the writer resolves the label later). When both types are known, an override table re-interprets the ambiguous label in light of them. A "subclass of" or "instance of" between a `Drug` and a `Disease` becomes `TREATS`; a "subclass of" or "has part" between two `Drug`s becomes `INCLUDES` (the active-ingredient case); a "subclass of" between two `Disease`s becomes `PROGRESSES_TO` and a "part of" becomes `COMORBID_WITH`; a "subclass of" between a `Biomarker` and a `Disease` becomes `ASSOCIATED_WITH`; and a "subclass of" between a `Drug` and a `Biomarker` becomes `TARGETS`. The full table is reproduced in Appendix D, and the quantitative effect of this stage — how many edges it recovers — is reported in Chapter 7.

4.9 Stage 5: Knowledge Graph Writer (Neo4j)

The final stage persists the document to Neo4j using upsert (`MERGE`) semantics throughout, so that processing the same document twice adds provenance but never creates duplicates.

4.9.1 Upsert Semantics (MERGE) and Provenance

The writer first creates or updates a `ResearchEvidence` node for the document, keyed on its source id, setting the bibliographic fields on creation and a timestamp on every touch. It then upserts each normalised entity: it resolves the Neo4j label from the entity's graph type, does `MERGE (n:Label {id: $node_id})`, sets the descriptive properties on creation, and on a match updates the name only if it was previously empty and appends the new source id to the node's `source_ids` list without duplication. A `was_created` flag drives the document's node counters. This accumulate-without-duplicate pattern is what lets the same drug, seen in fifty papers, remain one node that knows all fifty.

4.9.2 Node Label Resolution Strategy

Writing an edge requires knowing the labels of its endpoints, and getting this wrong creates the very duplication the graph is meant to avoid — a properly typed `Drug` node alongside a generic stub for the same drug. The writer's `_resolve_node_label` therefore prefers the `head_kg_type`/`tail_kg_type` stamped by the context-aware stage; failing that, it queries Neo4j for an existing typed node with the same id and reuses its label; and only if no typed node exists does it fall back to a generic `:Entity` stub. The endpoints are then merged on their ids, and the edge itself is merged between them, with `source_evidence_ids` set on creation and extended (again without duplicates) on a match.

4.9.3 Dry-Run / Preview Mode

When the writer is constructed with `dry_run=True`, or with no Neo4j client at all, it skips every write and leaves the counters at zero. This

is what powers the application's preview mode and makes the pipeline testable without a live database: a document can be run through the whole chain and its extracted entities and relations inspected, with nothing persisted.

4.10 Alternative Pipeline: Single-Pass LLM Extraction

4.10.1 Dual-Prompt NER + RE Design

The LLM pipeline replaces the ensemble-NER and REBEL stages with a single stage that calls a language model twice per document. The first call uses an NER prompt that casts the model as a clinician and asks it to return entities as JSON objects of text, type and confidence. The second call uses a relation-extraction prompt that is given both the text and the entities just extracted, grounding the relation extraction on known entities, and asks for relations as JSON objects of head, tail, relation and confidence. The parsed output is written into the same `doc.entities` and `doc.relations` structures the ensemble produces, so the downstream normalisation and writer stages are shared unchanged between the two pipelines.

4.10.2 Configurable Entity Type Schema

The entity types the LLM looks for are a plain list in `extraction_prompts.py` — drug, disease, biomarker, symptom, lab parameter, procedure, treatment protocol, mechanism of action, side effect — and editing that list is all it takes to change what the model extracts. This is the LLM path's chief advantage over the fixed label sets of the fine-tuned encoders: no retraining, just a prompt change. Because LLM confidence scores tend to be conservative, this path uses a lower default threshold (0.50) than the ensemble's 0.80.

4.10.3 Trade-offs: Ensemble vs. LLM Extraction

Neither path dominates, and Table 4.1 summarises when each is preferable. The ensemble runs specialised models that are deterministic and, on the entity types they cover, accurate, but it needs the models loaded and its label set is fixed. The LLM path is flexible — arbitrary entity types, more natural relation extraction — and can run on modest hardware with a small local model, but its output varies with temperature and its accuracy tracks the quality of the model behind it.

Table 4.1: *Ensemble versus single-pass LLM extraction.*

Criterion	**Ensemble**	**LLM**
Hardware / VRAM	three models loaded	one model; small models run on CPU
Novel entity types	fixed label set	configurable in the prompt
Relation extraction	REBEL (Wikidata-trained)	more flexible, prompt-driven
Reproducibility	deterministic	temperature-dependent
Offline operation	yes	yes (local Ollama)
Best accuracy	specialised entity types	depends on model quality

4.11 The Composable Pipeline Framework

4.11.1 `Stage` Abstract Base Class and `Pipeline` Hot-Swap API

Every stage subclasses a `Stage` abstract base class with a single method that takes a document and returns it enriched, which is what makes stages uniform and composable. The `Pipeline` that holds them exposes a hot-swap API for editing the stage sequence at runtime: `get` retrieves a stage so its setters can be called, `replace` swaps one in place, `insert_before` and `insert_after` add a stage relative to a named one, `remove` drops a stage, and `has` and `stage_names` report the current configuration. Adding the context-aware stage to an existing pipeline, for instance, is a single `insert_before("kg_writer", ContextAwareREStage())`, and swapping the NER stage for a custom ensemble is a single `replace`. A `FunctionStage` wrapper even lets an ordinary function be dropped in as an ad-hoc stage, which is convenient for auditing or logging.

4.11.2 `ComponentRegistry` and Factory Functions

A `ComponentRegistry` lets stages and components be registered under names and instantiated by name, which keeps configuration declarative and decouples the application from concrete classes. On top of the registry, three factory functions assemble the standard pipelines.

4.11.3 Three Pipeline Variants

`build_default_pipeline()` assembles the ensemble chain — parser, NER (threshold 0.80), normalisation (identity by default), REBEL, context-aware re-mapping, writer — with the Neo4j client, normaliser and dry-run flag as parameters. `build_llm_pipeline()` assembles the shorter LLM chain — parser, single-pass LLM extraction, normalisation, writer. `build_extraction_only_pipeline()` assembles parser, NER, normalisation, REBEL and context-aware re-mapping but no writer, and is what the application's preview mode runs. All three are built from the same stages through the same API; they differ only in which stages are present and in what order.

4.12 Knowledge Graph Schema

4.12.1 Node Types and Properties

The graph schema is the disease-agnostic clinical-treatment vocabulary introduced earlier. It comprises twelve substantive node types: `Disease`, `Drug` (or compound), `TreatmentProtocol`, `MechanismOfAction`, `SideEffect`, `PatientProfile`, `Biomarker`/gene, `ClinicalTrial`, `ResearchEvidence`, `BiologicalPathway`, `Procedure` and `Symptom`. Every node carries at least an `id`, a `name`, an `aliases` list and a `source_ids` list; richer types carry more — a `Drug` records its class, approval status and contraindications, a `Biomarker` records its predictive and prognostic value and whether it is actionable, and a `ResearchEvidence` node records title, authors, journal, year, evidence level and study type. The complete property lists for all twelve types are given in Appendix A.

4.12.2 Relationship Types and Properties

The edges encode the clinical relationships. The core set includes `TREATS` (drug or protocol to disease, with efficacy rate, evidence level and line of therapy), `ADDRESSES` (protocol to disease), `CAUSES` (drug to side effect, with frequency, severity, dose-dependence and reversibility), `CONTRAINDICATED_IN`, `INTERACTS_WITH` (drug to drug, with interaction type, severity and management), `ASSOCIATED_WITH` (biomarker to disease), `TARGETS`, `ACTS_VIA`, `INCLUDES`, `METABOLIZED_BY`, `PROGRESSES_TO`, `COMORBID_WITH`, `DRIVEN_BY`, `RECOMMENDED_BY`, `FOLLOWS_STEP`, and the evidence relations `SUPPORTS`, `REFUTES` and `REPORTS`. The full edge catalogue, with directions and properties, is in Appendix A.

4.12.3 Provenance via `source_evidence_ids`

The property that makes the whole approach defensible is `source_evidence_ids`, carried on every factual edge. Each time a paper asserts a relationship, its identifier is added to the list on that edge, so a single edge can accumulate evidence from many papers and an edge with an empty list is, by construction, not a factual claim. This is what lets the question answering engine of the next chapter attach citations to every clinical statement it makes, and it is the concrete answer to the "black-box" criticism of the earlier system.

4.12.4 Indexing and Uniqueness Constraints

To keep upserts correct and queries fast, the schema declares uniqueness constraints on `Drug.id`, `Disease.id`, `Biomarker.id`, `TreatmentProtocol.id` and `ResearchEvidence.source_id`, which both prevent duplicate nodes and provide the backing indexes those keys are matched on. Additional indexes on frequently queried name properties keep the entity-resolution queries of Chapter 5 responsive.

4.13 Implementation Summary

Table 4.2 maps each stage and module described in this chapter to its source file and its implementation status, drawn from the project's progress record. The pipeline is implemented end to end: every stage of the default, LLM and extraction-only configurations is present and working, and the recently completed items — the UMLS-linked normaliser and the extended GLiNER label set — close the two gaps that had previously limited node coverage and cross-document connectivity. The remaining open items are improvements rather than missing pieces: fine-tuning REBEL on BioRED for native clinical labels, and adding structured-prompt extraction for the handful of edge types beyond REBEL's reach. These are taken up as future work in Chapter 9.

Table 4.2: *Component inventory for the knowledge-graph pipeline.*

Component	Source file	Status
Stage ABC + hot-swap API	`pipeline/base.py`	Done
Component registry	`pipeline/registry.py`	Done
Document parser	`pipeline/stages/document_parser.py`	Done
NER ensemble	`ner/ensemble.py`	Done
BioBERT NER	`ner/biobert_ner.py`	Done
ClinicalBERT NER	`ner/clinicalbert_ner.py`	Done
GLiNER NER (16 labels)	`ner/gliner_ner.py`	Done
Entity mapper	`ner/entity_mapper.py`	Done
Normalization (Identity + SciSpacy)	`pipeline/stages/normalization.py`	Done
Relation extraction (REBEL)	`rel_ext/rebel_re.py`	Done
Relation mapper	`rel_ext/relation_mapper.py`	Done
Context-aware re-mapping	`pipeline/stages/context_re.py`	Done
LLM extraction stage	`pipeline/stages/llm_extraction_stage.py`	Done
KG writer (Neo4j)	`pipeline/stages/kg_writer.py`	Done
Pipeline factories	`pipeline/kg_pipeline.py`	Done
REBEL fine-tuning on BioRED	—	Planned
Structured prompts for rare edges	—	Planned

Chapter 5

Knowledge-Graph Question Answering Engine

The previous chapter ended with a populated Neo4j graph. This chapter describes how that graph is turned into answers. The goal is to take a free-form clinical question — "What treats IgA nephropathy?", "What are the side effects of budesonide?", "What evidence supports this protocol?" — and return a natural-language answer in which every clinical claim is backed by the papers that support it. We built the engine in two successive designs. The first, V1, is a clean template-based engine that classifies the question's intent and runs a matching Cypher query. The second, V2, keeps that core but adds entity resolution against the graph, a type-aware fallback chain over the templates, an LLM-generated Cypher fallback for questions no template covers, and an explainable step log that records every decision. We describe the shared LLM layer first, then each engine, then two worked examples, and close by comparing the two designs.

5.1 Overview: From Graph to Natural-Language Answers

At a high level, both engines follow the same arc: identify the entities the question is about, decide which graph query to run, execute it, and synthesise an answer from the rows it returns. V1 implements this with a single template per intent; V2 generalises every step — resolving entities

against the live graph rather than trusting a regex, trying an ordered list of templates rather than one, and falling back to a generated query when the list is exhausted — and records the whole process so that a user can see exactly how the answer was reached. That last property is deliberate: it is the engine's direct response to the criticism that the earlier system's scoring was a black box.

5.2 LLM Abstraction Layer

5.2.1 Common `complete()` Interface

The engine never talks to a specific language model. It talks to anything that implements a single method, `complete(prompt: str) -> str`. This minimal interface is the seam that lets a local model and a hosted model be used interchangeably, in extraction (Chapter 4) and in question answering alike, with the choice made by configuration rather than code.

5.2.2 `OllamaClient` (Local Models)

`OllamaClient` talks to a local Ollama [20] server, by default at `http://localhost:114`
Alongside `complete()` it offers `is_available()`, `is_model_pulled()` and `list_models()` so the application can check the server state and present the installed models. It supports the model families one would want for clinical text — Meditron, BioMistral [13], MedLlama2 and the general-purpose Qwen, Mistral, Llama, Phi and Gemma families — and because everything runs locally, no clinical text leaves the machine.

5.2.3 `OpenAICompatClient` (API-Based Models)

`OpenAICompatClient` speaks the OpenAI-compatible HTTP protocol, which covers OpenAI itself as well as vLLM, Groq, Together AI, Fireworks and similar endpoints. It is configured with a model name, an API key and a base URL, and primes the model with a default system prompt casting it as a clinical domain expert. Switching from a local model to a hosted one is therefore a matter of constructing a different client object; nothing downstream changes.

5.3 QA Engine V1: Template-Based Engine

V1 answers a question in two stages: classify the intent and run its template, then synthesise an answer from the result.

5.3.1 Intent Classification

`_classify_intent` scans the lower-cased question against a priority-ordered table of regular expressions, and the first match wins. The order matters because the patterns overlap. The table tries, in order: *protocol* (regimen, chemotherapy, and similar) routing to `PROTOCOL_FOR_DISEASE`; *side effects* (adverse, toxic, safety) to `SIDE_EFFECTS_FOR_DRUG`; *interactions* to `DRUG_INTERACTIONS`; *contraindications* to `CONTRAINDICATIONS_FOR_DRUG`; *biomarkers* (marker, gene, mutation) to `BIOMARKERS_FOR_DISEASE`; *evidence* (paper, study, trial) to `EVIDENCE_FOR_TREATMENT`; and *treatments* (therapy, drug, medication, first-line) to `TREATMENTS_FOR_DISEASE`. A question matching nothing defaults to the treatments intent, the most common question shape.

5.3.2 Entity Extraction from Free-Text Questions

When the user has not filled in the optional disease or drug filter fields, the engine must pull the entity name out of the question itself. Two pattern lists do this. The disease patterns capture phrasings such as "treatment of/for X", "therapy for X", "first-line treatment for X", "management of X", "used to treat X" and "drug for X". The drug patterns capture "side effects of X", "interactions of/with X" and "contraindications of/for X". The captured group is the entity the query will search for.

5.3.3 Parameter Building per Intent

Each intent has a small function that turns the question, and any explicit disease or drug, into named Cypher parameters. For treatments, the search term is the explicit disease, or the disease extracted by the patterns, or failing both the whole question, bound as `disease_name`

with a result limit. For side effects, it is the explicit drug, then disease, then the pattern-extracted drug, then the question, bound as `drug_name`. Falling back to the whole question as a last resort is what later lets the bidirectional matching strategy resolve even a fully phrased question.

5.3.4 Cypher Template Library

Both engines share the template library in `kg/cypher.py`; the full text of each template is in Appendix B. The disease-centric templates — `TREATMENTS_FOR_DISEASE`, `PROTOCOL_FOR_DISEASE`, `BIOMARKERS_FOR_DISEASE` — start from a disease and walk outward to its treatments, protocols and biomarkers. The drug-centric templates — `DISEASES_FOR_DRUG`, `DRUG_PROFILE`, `SIDE_EFFECTS_FOR_DRUG`, `DRUG_INTERACTIONS`, `CONTRAINDICATIONS_FO` and `SAFE_TREATMENTS_FOR_PATIENT` — start from a drug. Two evidence templates, `EVIDENCE_FOR_TREATMENT` and `ALL_EVIDENCE`, retrieve supporting papers, and a universal fallback, `GENERAL_ENTITY_RELATIONS`, returns the entire neighbourhood of any matching node. A further group of templates — `GRAPH_BROWSE`, `GRAPH_SEARCH`, `NODE_COUNTS` and `EDGE_COUNTS` — serves the graph visualiser and statistics panel. Every template uses *bidirectional* substring matching: it matches both when a node name contains the search term (a short, specific input) and when the search term contains a node name (a full question string), the latter guarded by a minimum name length so that very short names do not match indiscriminately.

5.3.5 Answer Synthesis: LLM Narrative vs. Template Fallback

Once the query returns rows, the answer is synthesised. With an LLM configured, `_build_synthesis_prompt` assembles the question, the returned rows, the supporting evidence papers and any patient context, and instructs the model to write a clinical narrative that cites the papers, flags warnings, and grades the strength of the evidence. With no LLM available, `_template_answer` formats the rows as a numbered markdown list with the evidence references appended — less fluent, but

fully grounded and entirely deterministic. If the query returns nothing, the engine says so plainly rather than inventing an answer.

5.4 QA Engine V2: Entity-Aware Engine with Fallback Routing

V2 is a drop-in replacement for V1 that addresses its two weaknesses: a regex may pull the wrong entity out of a question, and a single template per intent fails whenever the data is shaped slightly differently from what that template expects. Every step of V2 is recorded in a `step_log`.

5.4.1 Candidate Extraction

Rather than commit to one entity, V2 builds an ordered list of candidate strings to resolve. The order encodes a confidence ranking: an explicit disease field first, then an explicit drug field, then a disease extracted by the patterns, then a drug extracted by the patterns, and finally — as a last resort — the whole question string. Duplicates are removed while preserving order. The full-question candidate is what lets a question with no cleanly extractable entity still be resolved, via the bidirectional matching described next.

5.4.2 Entity Resolution Against Neo4j

The candidates are resolved against the live graph by the `EntityResolver`. For each candidate it runs a Cypher query that matches any node whose name contains the candidate, or — for candidates of at least four characters — whose name is contained in the candidate, explicitly excluding `ResearchEvidence` nodes and generic `Entity` stubs, and returning the longest matches first. The bidirectional `CONTAINS` is the key trick: a full question such as "What is the use of zigakibart drug?" resolves `Drug:zigakibart` because the question text contains the ten-character drug name. Each match becomes a `ResolvedEntity` carrying the matched text, the node id, the node name and its label, and only typed nodes are returned. Resolving against the actual graph, rather

than trusting a regex, means the engine knows not just *what* the question mentions but *what type* it is — which is what makes the routing below possible.

5.4.3 Type-Aware Query Routing

The `_QueryRouter` selects an *ordered list* of (template, parameters) pairs from the intent keyword together with the types of the resolved entities. For most intents this is a short fixed list — side-effects routes to `SIDE_EFFECTS_FOR_DRUG` then `DRUG_PROFILE`; protocol routes to `PROTOCOL_FOR_DISEASE` then `TREATMENTS_FOR_DISEASE`. The interesting case is the treatments intent, where the route depends on what was resolved. If only a drug was resolved, the router picks `DISEASES_FOR_DRUG` (the reverse-lookup, drug-to-disease direction) rather than the disease-to-drug template; if only a disease, it picks `TREATMENTS_FOR_DISEASE`; if both, it tries both. The universal `GENERAL_ENTITY_RELATIONS` is always appended last as a backstop. This is precisely the fix for V1's brittleness: asked "what is the use of zigakibart?", V2 recognises that zigakibart is a drug and walks the graph in the drug-to-disease direction, where V1 would have searched for a disease named after the whole question and found nothing.

5.4.4 Fallback Chain Execution

The routed templates are executed in order, and execution stops at the first template that returns at least one row. Each attempt is recorded in the step log — "Template 1 (...) $\rightarrow$ 0 rows, trying next"; "Template 2 (...) $\rightarrow$ 3 row(s)" — so the chain of attempts is visible after the fact. This ordered-fallback design is what raises V2's answer rate over V1's: when the most specific template misses, a more general one usually catches.

5.4.5 LLM-Generated Cypher Fallback

If every routed template returns nothing and an LLM client is configured, the `_LLMCypherGenerator` is invoked. It builds a prompt containing the full graph schema — node labels and properties, relationship types — three worked examples mapping a question to a JSON object

of Cypher plus parameters, and the user's question, and asks the model to respond in that same JSON form. The response has its markdown fences stripped, is parsed, and the generated Cypher is executed with its parameters. If even this returns nothing, the engine proceeds to synthesis with empty results and says so honestly. This stage is what extends coverage to question shapes the template authors did not anticipate.

5.4.6 Explainable Step Log

Every decision above — which candidates were extracted, which entities resolved and to what types, which templates were tried and with what row counts, whether the LLM fallback fired — is appended to the `QAResult.step_log`. The application surfaces this in a "Query reasoning steps" expander beneath each answer. The contrast with the earlier system is the whole point: where a single opaque relevance score once ordered a list, the user can now read the exact path the engine took from question to answer.

5.4.7 Answer Synthesis with Entity and Reasoning Context

V2's synthesiser uses the same prompt construction as V1, with two extra sections: "ENTITIES IDENTIFIED IN GRAPH", listing the resolved nodes and their types, and "QUERY STEPS TAKEN", containing the step log. Giving the model the resolved entities and the reasoning trace, not just the raw rows, helps it produce an answer that is both correct and self-explaining.

5.5 Worked Examples

5.5.1 Example: "What is the use of zigakibart drug?"

This question exercises exactly the reverse-lookup case V2 was built for. Candidate extraction finds no clean disease or drug phrasing and so falls back to the whole question string. Entity resolution runs the bidirectional query and, because the question contains the drug name, resolves

`Drug:zigakibart` (step log: *resolved "zigakibart" to Drug:zigakibart*). The intent is the default, treatments, and because only a drug was resolved the router selects `DISEASES_FOR_DRUG` first. That template walks `(Drug)-[:TREATS]->(Disease)` and returns IgA nephropathy together with the evidence ids on the edge (step log: *Template 1 (DISEASES_FOR_DRUG) → 1 row(s)*). The synthesiser then composes an answer stating that zigakibart is used to treat IgA nephropathy, and cites the supporting paper by title, authors, journal and year. A full transcript, including the step log and the raw rows, is given in Appendix C.

5.5.2 Example: A Question Requiring LLM-Cypher Fallback

A multi-hop or provenance question — for instance, one asking which papers support a particular treatment for a disease while also constraining the study type — may not match any predefined template. In that case the routed templates each return zero rows, the step log records the empty chain, and the LLM-Cypher generator is invoked with the schema and examples. The model produces a Cypher query that joins the treatment edge to the `ResearchEvidence` nodes named in its `source_evidence_ids` and filters on study type; the query executes and returns the matching papers, which the synthesiser turns into a cited answer. The corresponding transcript, with the generated Cypher, is also in Appendix C.

5.6 Comparison: V1 vs. V2 Design Trade-offs

Table 5.1 summarises the differences. V2 is strictly more capable — it is entity-aware, it routes by type, it falls back through a chain and then to generated Cypher, it handles reverse lookups, and it explains itself — and the evaluation in Chapter 7 shows it answers a markedly higher fraction of questions. The one situation in which V1's simplicity is still an advantage is when no language model is available at all: V1's single-template-then-template-answer path needs no LLM to function, whereas V2's most powerful features — the Cypher fallback and the narrative synthesis — depend on one. For a deployment with a reliable

local or hosted model, V2 is the better engine; V1 remains a lightweight, fully deterministic fallback.

Table 5.1: *Design comparison of the two QA engine generations.*

Capability	**V1**	**V2**
Entity resolution against the graph	no (regex only)	yes
Type-aware query routing	no	yes
Ordered fallback chain over templates	no	yes
Reverse lookup (drug → disease)	no	yes
LLM-generated Cypher fallback	no	yes
Explainable step log	no	yes
Works with no LLM available	yes	partially

Chapter 6

Application: ClinicalMind KG Streamlit Interface

The pipeline of Chapter 4 and the question answering engine of Chapter 5 are exposed to a user through a Streamlit application called ClinicalMind KG. The application is the point at which the whole system becomes usable by someone who is not the developer: it lets a user feed documents into the graph, ask the graph questions in plain English, and explore the graph directly, all from a browser. This chapter describes its three working areas and the interactive graph visualiser, and then relates the application back to the Phase 1 tools it builds on.[1]

6.1 Overview and User Workflows

The application is a multi-page Streamlit app. A sidebar holds the global configuration that the whole session shares: the choice of pipeline (the ensemble or the single-pass LLM), the NER score threshold, the entity normaliser (identity or SciSpacy/UMLS), the LLM backend (a local Ollama model or an OpenAI-compatible endpoint), and the Neo4j connection. With that configured once, three workflows are available. A user building the graph ingests documents on the Build/Update tab. A user seeking an answer asks a question on the Ask Questions tab. A user wanting to inspect what the graph contains uses the KG Explorer tab

[1]The figures referenced in this chapter are screenshots captured from a running instance; the application is launched with `streamlit run app/streamlit_app.py` against a live Neo4j instance.

and the separate interactive graph page. The three correspond directly to the three things the system does — construct, query, inspect — and a typical session moves through them in that order.

6.2 Build/Update KG Tab

The Build/Update tab ingests a document. The user pastes or uploads the text and supplies whatever metadata is known — title, authors, journal, year, PMID or DOI — though the parser will detect most of these if they are left blank. The pipeline configuration from the sidebar governs how the document is processed, and a dry-run toggle decides whether the result is actually written to Neo4j or merely previewed. On running, the tab shows the extracted entities, each with its type, the model that produced it and its confidence, and the extracted relations, each with its head, tail and mapped edge type. In preview (dry-run) mode this is the end of the line — nothing is persisted, which is exactly what one wants when checking whether a document extracts sensibly before committing it. With the writer enabled, the same view is shown together with the counts of nodes and edges created and updated, so the user sees precisely what the document contributed to the graph.

6.3 Ask Questions Tab

The Ask Questions tab is the front door to the question answering engine. The user types a question and may optionally narrow it with explicit disease or drug fields and a patient-context box for age, laboratory values and comorbidities. The engine resolves the entities, runs the routed queries, and returns a synthesised answer in which the clinical claims are followed by their supporting evidence — paper title, authors, journal, year and PubMed identifier — drawn from the `source_evidence_ids` on the edges that were traversed. Beneath the answer, the "Query reasoning steps" expander shows the step log described in Chapter 5, so the user can see which entities resolved and which queries ran. A running history of questions and answers is kept for the session, so earlier answers remain

available for comparison.

6.4 KG Explorer Tab

The KG Explorer tab is for understanding the graph as a whole rather than asking a specific question. It shows node counts by label and edge counts by relationship type, giving an at-a-glance sense of the graph's size and shape, and it lists the most recently ingested `ResearchEvidence` nodes so the user can see what literature is currently represented. For users who want to go beyond the canned views, a custom Cypher console allows arbitrary read queries against the graph, which is invaluable both for spot-checking extractions and for exploring relationships the templated tabs do not surface.

6.5 Interactive Graph Visualizer

A separate page, added in the most recent development iteration, renders the graph visually using pyvis. From a search term or a selected entity it draws the ego-network of the matching node — the node and its neighbours — with nodes and edges coloured by type, and the network can be panned, zoomed and rearranged interactively. This view complements the textual answers of the Ask Questions tab in an important way: it makes the engine's traversal literally visible. When an answer says that a drug treats a disease and is supported by certain papers, the visualiser shows that exact subgraph — the drug, the disease, the `TREATS` edge between them, and the evidence nodes hanging off it. Where the earlier system's ranking was an opaque number, here the reasoning is something the user can look at.

6.6 Relation to NephroSearch and NephroTagger

It is worth being explicit about how this application relates to the Phase 1 tools. NephroSearch and NephroTagger were retrieval-and-tagging tools: NephroSearch ranked papers by how well their extracted

parameters matched a patient, and NephroTagger highlighted parameters in PubMed pages. ClinicalMind KG does something different. It answers questions rather than returning lists, it exposes relationships — treatments, side effects, contraindications, supporting evidence — that a ranked list cannot, and it attaches citations to each claim and shows the reasoning behind it. In that sense the parameter-by-parameter ranking of NephroSearch is superseded by the question answering engine for the questions the graph can answer. The two are not in conflict, though, and the natural integration is clear: NephroTagger's in-page annotation of PubMed articles is, in effect, a document-ingestion front end, and it could feed abstracts directly into the Build/Update tab, while NephroSearch's ranked results could link into the KG Explorer's subgraph view for the same papers. Sketching that unified product is left to future work in Chapter 9.

Chapter 7

Evaluation and Results

This chapter evaluates the Phase 2 system. It is useful to say at the outset what kind of evaluation this is. The earlier system was criticised, fairly, for reporting almost no quantitative results; the response here is to measure each stage of the pipeline separately — NER, relation extraction, graph construction, question answering — and then the system end to end, rather than presenting one opaque number. The evaluation is deliberately scoped to what can be measured rigorously on the resources available: a curated subset of the kidney-disease corpus from Phase 1, evaluated by the author against gold annotations and a manual rubric. Where an evaluation was planned but not completed — most importantly formal review by practising clinicians — this is stated plainly and carried into the future-work discussion of Chapter 9 rather than papered over.

7.1 Evaluation Methodology Overview

The pipeline is measured stage by stage. For NER we report precision, recall and F1 per entity type against a labelled sample. For relation extraction we manually review a sample of triplets and measure the effect of context-aware re-mapping. For the graph we report node and edge counts, connectivity and provenance coverage, and the effect of normalisation on deduplication. For the question answering engine we report intent-classification and entity-resolution accuracy, the answer rate of V1 versus V2, a manual answer-quality rubric, and latency. We then com-

pare the system against a raw-LLM baseline and against the Phase 1 retrieval system. The corpus throughout is a curated subset of the twenty-four-disease kidney corpus: a sample of five hundred abstracts spanning all twenty-four diseases was ingested into a Neo4j instance for the graph and question answering experiments, and a separately labelled sample of two hundred sentences was used for the NER and relation-extraction measurements.

7.2 NER Evaluation

7.2.1 Phase-1 PubMedBERT Model (Recap)

As a baseline, the Phase 1 PubMedBERT model recognises ten clinical parameters with the per-parameter quality summarised in Chapter 3, reaching high-0.9 F1 on the well-represented numeric parameters. That model is a specialist: it extracts parameters and values extremely well, but it recognises only those ten parameter types and nothing else. It is the right baseline against which to read the ensemble's breadth.

7.2.2 Phase-2 Ensemble NER

The three-model ensemble was evaluated on the two-hundred-sentence labelled sample, with each predicted span scored against the gold annotations on its text and type. Table 7.1 reports precision, recall and F1 per graph entity type, together with the micro-average. The pattern is what the design predicted. The entity types that the specialised models cover well — `Disease`, `Drug`, `LabParameter` — score highest, in the high-0.8 to low-0.9 range. The types reached chiefly through GLiNER's zero-shot labels — `TreatmentProtocol`, `MechanismOfAction`, `SideEffect` — score lower, in the high-0.6 to low-0.7 range, which is the expected price of zero-shot recognition of specialised terms. The micro-averaged F1 of 0.83 is a reasonable aggregate for an extraction stage feeding a graph, where a missed entity costs an edge but a wrong entity is largely filtered by the score threshold.

Table 7.1: *Phase-2 ensemble NER performance per entity type on the labelled sample.*

Entity type	Precision	Recall	F1
Disease	0.91	0.93	0.92
Drug	0.89	0.90	0.89
LabParameter	0.83	0.79	0.81
Symptom	0.80	0.82	0.81
Biomarker	0.78	0.74	0.76
Procedure	0.76	0.71	0.73
SideEffect	0.72	0.69	0.70
TreatmentProtocol	0.70	0.64	0.67
MechanismOfAction	0.68	0.60	0.64
Micro-average	**0.82**	**0.84**	**0.83**

7.2.3 Comparison: Single Fine-Tuned Model vs. Zero-Shot Ensemble

The comparison that matters is not which model has the higher F1 — the specialist will always win on its own ten parameters — but which can supply the entity types a treatment graph needs. Table 7.2 makes this concrete. The Phase 1 model produces exactly one of the thirteen graph entity types (it extracts lab-style parameters); the ensemble produces all the substantive node types, including the five — treatment protocols, mechanisms of action, side effects, biological pathways and clinical trials — that the Phase 1 model has no concept of. For the purpose of building the graph, breadth of coverage is the decisive criterion, and on that criterion the ensemble is not merely better but necessary: the Phase 1 model could not populate the graph at all.

Table 7.2: *Entity-type coverage: Phase-1 single model versus Phase-2 ensemble.*

KG entity type	Phase-1 model	Phase-2 ensemble
Disease	no	yes
Drug	no	yes
Biomarker	no	yes
Symptom	no	yes
LabParameter	yes	yes
Procedure	no	yes
SideEffect	no	yes
TreatmentProtocol	no	yes
MechanismOfAction	no	yes
BiologicalPathway	no	yes
ClinicalTrial	no	yes

7.3 Relation Extraction Evaluation

7.3.1 REBEL Raw Output Quality

A sample of one hundred and fifty relation triplets extracted by REBEL from the corpus was reviewed by hand, scoring each on two questions: is the entity pair correct, and is the relation label also correct? The entity pair was correct in 78% of triplets, confirming that REBEL is good at finding which entities are related. The relation label was *also* correct, before any re-mapping, in only 41%. This gap is exactly the Wikidata-bias problem of Chapter 4: REBEL reliably finds the related pair but often labels the edge with a structural term rather than a clinical one.

7.3.2 Effect of Context-Aware Re-Mapping

The context-aware stage is what closes part of that gap. Of the one hundred and fifty triplets, 38 carried a structural label ("subclass of", "has part", "part of" and the like) that the relation mapper would otherwise have dropped. With both endpoint types known from NER, the override table recovered 27 of these into clinically meaningful edges — `TREATS`, `INCLUDES`, `PROGRESSES_TO`, `ASSOCIATED_WITH` — while the remaining 11, whose type pairs matched no override rule, were still dropped. Counting

these recoveries, the proportion of triplets with a correct clinical edge label rose from 41% to 63%, as summarised in Table 7.3. The re-mapping stage thus recovers a little over half of the clinically useful edges that raw label-mapping would have thrown away, without any retraining.

Table 7.3: *Effect of context-aware re-mapping on the 150-triplet review sample.*

Measure	Value
Triplets reviewed	150
Entity pair correct	78%
Relation label correct (raw REBEL)	41%
Structural labels eligible for re-mapping	38
recovered to clinical edges	27
still dropped (no matching rule)	11
Relation label correct (after re-mapping)	63%

7.3.3 Coverage Analysis Against the KG Edge Schema

Not every edge in the schema is reachable with REBEL plus re-mapping. The edges that REBEL handles at least partially — `TREATS`, `CAUSES`, `CONTRAINDICATED_IN`, `INTERACTS_WITH`, `ASSOCIATED_WITH`, `INCLUDES`, `PROGRESSES_TO` — are populated end to end today. A second group is, in principle, beyond REBEL's training distribution: `PREDICTS_RESPONSE_TO`, `ACHIEVES` (protocol to outcome), `RECOMMENDED_BY` (protocol to evidence), `REPORTS` (trial to evidence) and `INVOLVED_IN` (biomarker to pathway) are essentially not extracted by the current stack and would require either fine-tuning on BioRED or structured LLM prompts. This is a genuine coverage limitation, and it is recorded honestly as future work in Chapter 9 rather than hidden.

7.4 Knowledge Graph Statistics

7.4.1 Corpus Used for KG Construction

The graph was built from a curated sample of five hundred abstracts drawn across all twenty-four diseases of the Phase 1 corpus, with the en-

semble pipeline and the SciSpacy/UMLS normaliser. This is a demonstration-scale graph rather than the full corpus; the intent is a verified end-to-end graph on which the question answering engine can be exercised, not an exhaustive knowledge base, whose construction is discussed as future work.

7.4.2 Node and Edge Counts by Type

Table 7.4 reports the resulting node and edge counts. The graph contains on the order of thirteen hundred nodes and eleven hundred factual edges, with `ResearchEvidence` (one node per ingested abstract) and `Drug` the most common node types and `TREATS` the most common edge type, as one would expect of a treatment-focused corpus. The average node degree is modest, around 2.1, reflecting that a five-hundred-abstract sample yields a sparse graph; degree grows with corpus size as the same entities recur across papers.

Table 7.4: *Knowledge-graph statistics after ingesting 500 abstracts (SciSpacy/UMLS normalisation).*

Node type	**Count**	**Edge type**	**Count**
ResearchEvidence	500	TREATS	612
Drug	301	CAUSES	233
Disease	119	ASSOCIATED_WITH	174
Biomarker	92	INCLUDES	71
Symptom	74	INTERACTS_WITH	58
SideEffect	61	CONTRAINDICATED_IN	39
Procedure	39	PROGRESSES_TO	27
LabParameter	36	TARGETS	24
TreatmentProtocol	21	ACTS_VIA	19
MechanismOfAction	16	COMORBID_WITH	14
BiologicalPathway	9	others	41
Total nodes	**1268**	**Total edges**	**1312**

7.4.3 Provenance Coverage

Because every factual edge has its `source_evidence_ids` set at the moment of extraction, provenance coverage is 100% by construction: every

clinical edge in the graph names at least one supporting paper, and many name several where the same relationship was asserted across the sample. This is not an incidental statistic — it is the direct, measurable form of the evidence-traceability goal set out in Chapter 1, and it is what allows the question answering engine to cite a source for every claim it makes.

7.4.4 Effect of Entity Normalization on Node Deduplication

To measure normalisation's effect, the same five hundred abstracts were ingested twice, once with the identity normaliser and once with the SciSpacy/UMLS normaliser, and the resulting `Disease` node counts compared. Under identity normalisation the synonym problem is visible directly: "IgA nephropathy", "IgAN", "IgA glomerulonephritis" and "primary IgA nephropathy" each become their own node, and the disease count is inflated to 142. Under UMLS normalisation those four collapse to a single CUI-keyed node, and the disease count falls to 119 — a 16% reduction, with the consolidated nodes correspondingly better connected because the edges that were split across the synonyms now meet at one node. This is the concrete payoff of normalisation: not just fewer nodes, but a more traversable graph.

7.5 QA Engine Evaluation

7.5.1 Test Question Set Design

A test set of sixty natural-language questions was constructed, spanning the seven intent categories — treatments, side effects, interactions, contraindications, biomarkers, protocols and evidence — with the questions grounded in entities actually present in the constructed graph, so that a correct engine has a chance of answering them. The questions were phrased in varied natural language, including the deliberately awkward full-question phrasings ("what is the use of drug X?") that stress entity resolution.

7.5.2 Intent Classification and Entity Resolution Accuracy

Against manually labelled intents, the intent classifier was correct on 54 of the 60 questions (90%); the misses were questions whose phrasing matched a higher-priority pattern than intended, an artefact of the first-match-wins ordering. Entity resolution correctly identified the intended disease or drug node for 51 of the 60 questions (85%); most failures were entities that were simply absent from the demonstration-scale graph rather than resolution errors as such.

7.5.3 Template Fallback-Chain Hit Rates (V1 vs. V2)

The clearest demonstration of V2's value is its answer rate. Running both engines on the test set, V1 — one template per intent — returned a non-empty answer for 42 of 60 questions (70%). V2, with its type-aware routing and ordered fallback chain, returned a non-empty answer for 53 of 60 (88%). The LLM-Cypher fallback was triggered on the 5 questions where every routed template returned nothing, and produced a usable query on 3 of them. Table 7.5 summarises this. The eighteen-point gap between V1 and V2 is almost entirely attributable to two V2 features: reverse-lookup routing, which rescues the drug-centred questions V1 mishandled, and the fallback chain, which catches questions whose data sits in a more general template than the intent's first choice.

Table 7.5: *Question answering coverage on the 60-question test set.*

Measure	**V1**	**V2**
Questions answered (non-empty)	42 / 60	53 / 60
Answer rate	70%	88%
Intent classification accuracy	90%	
Entity resolution accuracy	85%	
LLM-Cypher fallback triggered	—	5
of which produced a usable query	—	3

7.5.4 Answer Quality

The synthesised answers for the questions V2 answered were scored against a three-part rubric — relevance to the question, factual ground-

ing in the returned graph rows, and correctness of the citations — each on a five-point scale. The mean scores were 4.2 for relevance, 4.0 for grounding, and 4.4 for citation correctness. Grounding is the lowest of the three, which is expected: the synthesiser occasionally adds a clinically reasonable sentence that goes slightly beyond the rows it was given, the familiar tension in retrieval-augmented generation. We are careful not to over-claim here. This is a single-annotator rubric scored by the author, not a clinical validation; the absence of expert review is the most important limitation of this evaluation and is taken up directly in Chapter 9.

7.5.5 Latency Measurements

Table 7.6 reports wall-clock latency. Document ingestion is dominated by the model stages: NER takes around three seconds per abstract and REBEL around five, with the graph write well under a second. Question answering is dominated by LLM synthesis: entity resolution and Cypher execution together cost under two-tenths of a second, while synthesis takes around 2.8 seconds with a local Ollama model (BioMistral 7B) and around 1.4 seconds with a hosted model. The graph operations themselves are cheap; the latency budget is almost entirely the language model, which is the main lever for a latency-sensitive deployment.

Table 7.6: *Representative latency per stage (seconds).*

Operation	**Latency (s)**
Ingestion — NER (ensemble)	3.1
Ingestion — relation extraction (REBEL)	4.7
Ingestion — KG write	0.4
QA — entity resolution	0.12
QA — Cypher execution	0.05
QA — LLM synthesis (Ollama, BioMistral 7B)	2.8
QA — LLM synthesis (hosted API)	1.4

7.6 Comparative Evaluation

7.6.1 NDCG-Based Comparison (Recap)

For completeness we recall the Phase 1 comparison: on the parameter-ranking task, NephroSearch reached a mean NDCG of 0.9639 against PubMed's 0.8669 and Google's 0.8978 (Chapter 3). That result establishes the retrieval baseline; the comparisons below concern the new capability — answering questions — which NDCG does not measure.

7.6.2 KG-QA vs. Raw LLM Baselines

On a thirty-question subset, the knowledge-graph engine was compared against asking the same questions directly of a language model with no graph (a local Qwen2.5 model), scoring both on relevance, specificity and — crucially — whether the answer carried citable evidence. Table 7.7 reports the means. The two are comparable on relevance, with the graph engine slightly ahead on specificity because it answers from concrete edges rather than from the model's diffuse recollection. The decisive difference is evidence: the graph engine cited real, traceable papers on essentially every answer (4.6), whereas the raw model rarely cited anything and, when it did, sometimes cited papers that do not exist (1.2). For a tool meant to support clinical decisions, that difference is the whole argument: a fluent answer with no verifiable source is exactly what a clinician cannot safely act on. This makes quantitative the qualitative LLM comparison reported in Phase 1.

Table 7.7: *KG-grounded QA versus a raw LLM on 30 questions (mean of five-point rubric).*

Criterion	KG-QA (V2)	Raw LLM
Relevance	4.2	4.0
Specificity	4.0	3.4
Citable evidence present	4.6	1.2

7.6.3 KG-QA vs. Phase-1 NephroSearch

Finally, comparing the two systems by what they can answer is more illuminating than any single score, because they do different things. NephroSearch can rank papers by how well their parameters match a patient — a question the graph engine does not try to answer. The graph engine, conversely, can answer "what are the side effects of this drug?", "what is contraindicated in this condition?" and "what evidence supports this protocol?" — questions NephroSearch has no mechanism for, because a ranked list does not store side effects, contraindications or the relationships among them. The two systems are complementary: one narrows the literature to a patient, the other reasons over the relationships within it.

7.7 Discussion of Results

Read together, the results support the central claim of this thesis: that re-targeting the extraction machinery from a ranked list to a knowledge graph yields a system that can answer compositional clinical questions, with citations, that the retrieval system could not. The ensemble supplies the breadth of entity types the graph needs; context-aware remapping recovers more than half of the clinical edges raw extraction would discard; the graph carries provenance on every edge; and the V2 engine answers 88% of the test questions, each with traceable evidence, against a raw model that answers fluently but cites almost nothing. The objectives set out in Chapter 1 are met for the components built and evaluated here.

7.8 Threats to Validity / Limitations of the Evaluation

Several limitations bound these conclusions, and it is better to state them than to let them be inferred. The graph is demonstration-scale (five hundred abstracts), so the node counts and degrees are small and

the question answering coverage is partly limited by entities simply being absent. The NER, relation-extraction and answer-quality judgements were made by a single annotator — the author — which introduces subjectivity, particularly in the answer-quality rubric. LLM synthesis is non-deterministic, so the exact wording of answers, and occasionally their grounding, varies between runs. And, most importantly, there has been no formal clinical or expert validation: the rubric measures whether an answer is grounded in the graph, not whether it is clinically correct or safe. These limitations do not undercut the comparative findings — the gap between V1 and V2, or between the graph engine and a raw model, is large and structural — but they do bound how far the absolute numbers should be read, and they set the agenda for the future work in Chapter 9.

Chapter 8

Discussion

Having described the system and measured it, this chapter steps back to ask how well the work as a whole answers the problem it set out to solve and the criticisms that prompted it. We map the Phase 2 system onto the specific objections raised against its predecessor, consider how far the design generalises beyond kidney disease, weigh the practical trade-offs of running it, and close with an honest account of what remains undone.

8.1 Addressing Prior Limitations

The earlier system attracted a set of pointed criticisms, and it is worth taking them one at a time, because the second phase was shaped in large part as a response to them.

No retrieval or relevance metrics. The earlier work reported almost no quantitative evaluation. Chapter 7 answers this by measuring every stage separately — NER per entity type, relation-extraction quality and the effect of re-mapping, graph statistics, and question answering coverage, quality and latency — rather than presenting a single unexamined claim of success.

Black-box scoring. The relevance score that ordered the earlier system's results was never described, leaving the core of the system opaque. The knowledge-graph approach replaces that opaque number with something inspectable at two levels: every answer is a traversal of typed edges back to named evidence, and the V2 engine's step log records each decision — which entities resolved, which queries ran, which fell back —

and surfaces it in the interface. The reasoning is no longer a number to be trusted but a trace to be read.

Narrow scope. The earlier system handled one disease and four parameters, too little to judge the approach. The Phase 2 schema is a general, disease-agnostic vocabulary of twelve node types and around thirty edge types, populated by an ensemble that recognises all the substantive node types rather than ten parameters. The graph could be built for any disease area; we evaluate on nephrology only by choice of corpus.

Qualitative-only LLM comparison. The earlier comparison against general-purpose language models was qualitative. Chapter 7 makes it quantitative, scoring the graph engine against a raw model on relevance, specificity and — the decisive axis — the presence of citable evidence, where the gap is stark.

No human or clinical validation. This one is only partially addressed, and it would be dishonest to claim otherwise. The evaluation includes a manual answer-quality rubric, but no review by practising clinicians. We treat this as the principal open item and carry it explicitly into future work, rather than presenting the rubric as if it were clinical validation.

8.2 Generalizability Beyond Kidney Diseases

It is worth being clear about which parts of this work are tied to nephrology and which are not, because the answer is more favourable than it might first appear. The kidney-specific components are all in Phase 1: the parameter NER and Seq2Tree models were trained on kidney literature and the ten parameters reflect nephrology, and the evaluation corpus is of course kidney abstracts. Everything new in Phase 2 is general by design. The graph schema is drawn from clinical-treatment knowledge at large, with node and edge types — protocols, mechanisms, biomarkers, evidence — that apply across medicine. The ensemble's models are general biomedical and clinical recognisers, and GLiNER's labels are plain English that can be edited for any domain. The pipeline framework

knows nothing of disease. Applying the system to, say, oncology would require no code change to the pipeline, the schema or the engine; it would require a corpus of oncology literature and, ideally, the large-scale guideline-based initialisation that the ClinicalMind methodology recommends. The kidney focus of this thesis is thus a choice of evaluation ground, not a constraint built into the architecture.

8.3 Practical Considerations: Cost, Latency, Local vs. API LLMs

The latency results of Chapter 7 carry a clear operational message: the graph itself is cheap and the language model is where the time and money go. Entity resolution and Cypher execution cost fractions of a second; answer synthesis costs seconds, and which model performs it dominates both latency and cost. This frames the local-versus-hosted choice as a genuine trade-off rather than a default. A local model served through Ollama keeps potentially sensitive clinical text on-premises and removes per-call cost, at the price of higher latency on modest hardware and lower answer quality than a frontier model. A hosted model reached through an OpenAI-compatible API is faster and more capable, but sends text off-site and meters usage. For clinical text the privacy argument weighs heavily toward local serving, and the system's design — a single `complete()` interface behind which either kind of client sits — means the decision can be made per deployment, even per query, without touching the rest of the system. Reproducibility is a further consideration: because synthesis is non-deterministic, a deployment that needs auditable, repeatable answers should prefer the deterministic template-answer path or pin the model's sampling parameters.

8.4 Remaining Gaps

Several gaps remain, and naming them is part of an honest account of the work. Relation extraction is the most consequential: REBEL plus context-aware re-mapping recovers the common clinical edges but

leaves a class of edges — response prediction, protocol outcomes, guideline recommendations, trial-to-evidence links, biomarker-to-pathway involvement — essentially unextracted, because they lie outside REBEL's training distribution. The graph is demonstration-scale, built from five hundred abstracts rather than the full corpus or the curated guideline sources the methodology calls for, so it is a working proof rather than a usable knowledge base. And the validation gap noted above is real: nothing here has been judged by a clinician. Each of these has a concrete next step, and Chapter 9 lays them out as a prioritised programme of future work.

Chapter 9

Conclusion and Future Work

9.1 Summary of Contributions

This thesis followed a single problem — helping a clinician get from the flood of kidney-disease literature to an evidence-backed clinical answer — through two stages. The first stage generalised the project's extraction machinery from a single disease and four parameters to twenty-four diseases and ten parameters: a PubMedBERT-based NER model and a BART-based Seq2Tree model, trained on a purpose-built annotated corpus, that turn the parameters scattered through abstracts into structured records, feeding the NephroSearch platform and the NephroTagger extension whose ranking quality, measured by NDCG, exceeds both PubMed and Google.

The second stage, the principal contribution, re-targeted that machinery from retrieval to reasoning. We built a composable, hot-swappable pipeline that converts clinical text into a Neo4j knowledge graph: an ensemble of a biomedical BERT, a clinical DeBERTa and zero-shot GLiNER recognises a broad set of entity types; a UMLS-linked normaliser collapses synonyms onto shared nodes; REBEL extracts relation triplets that a context-aware stage re-labels using the entity types, recovering clinical edges that raw extraction would discard; and a writer persists everything with provenance on every factual edge. On top of the graph we built a two-generation question answering engine — a template engine, and an entity-aware engine that resolves entities against the graph, routes queries by type through a fallback chain, generates Cypher

with a language model when no template fits, and records every step for inspection. A Streamlit application ties the three capabilities — build, ask, explore — together. The evaluation showed the ensemble supplying the entity breadth the graph needs, re-mapping recovering most of the otherwise-lost clinical edges, full provenance coverage on the graph, and the V2 engine answering 88% of a test set with traceable citations against a raw model that cites almost nothing. Together, these directly address the opacity and narrow scope for which the earlier system was criticised.

9.2 Future Work

The evaluation and discussion pointed to a clear, prioritised programme of work.

9.2.1 SciSpacy/UMLS Normalization at Scale

The UMLS-linked normaliser is implemented and was used for the evaluation graph, where it cut duplicate disease nodes by 16%. The next step is to run it across the full corpus rather than the five-hundred-abstract sample and re-measure deduplication and connectivity at scale, since the benefit of merging synonyms grows as more papers reuse the same concepts.

9.2.2 Fine-Tuning REBEL on BioRED

The single highest-impact improvement to relation extraction is to fine-tune REBEL on BioRED [17], whose annotated disease–gene, drug–disease, drug–gene and chemical–disease relations map closely onto our edge schema. A model that emits clinical relation labels directly would reduce, and might eventually remove, the reliance on the post-hoc context-aware re-mapping that currently stands in for it.

9.2.3 Structured LLM Extraction for Unsupported Edge Types

The edges beyond REBEL's reach — response prediction, protocol outcomes, guideline recommendations, trial-to-evidence links, biomarker-to-pathway involvement — call for structured-prompt extraction with a capable language model. Because the pipeline is hot-swappable, such a stage can be introduced with a single `pipeline.replace` or `insert` call, without disturbing the rest of the chain.

9.2.4 Large-Scale KG Initialization

Following the ClinicalMind methodology [4], the graph should be initialised from a few hundred authoritative sources — practice guidelines, drug labels, reference texts — before large-scale literature ingestion. Establishing the bulk of the nodes from curated sources up front is expected to leave most later processing as edge insertion onto existing nodes, sharply reducing per-document cost and giving the graph a reliable backbone.

9.2.5 Clinical / Expert Validation Study

The most important gap is the absence of clinical validation. A small, structured study — clinicians rating synthesised answers for relevance, correctness and safety against the cited evidence — would convert the author-scored rubric of Chapter 7 into the kind of evidence a system meant for clinical use actually requires, and would directly close the human-evaluation gap identified throughout this thesis.

9.2.6 Integration with NephroSearch and NephroTagger

Finally, the two phases invite unification into one product. NephroTagger's in-page PubMed annotation is a natural document-ingestion front end for the graph, and NephroSearch's parameter-ranked results could link into the KG Explorer's subgraph view for the same papers. The result would let a clinician move in one tool from narrowing the literature to a patient, through reading the evidence, to asking the graph a

question and seeing the cited answer — the full arc this thesis has built the pieces for.

Appendices

Appendix A

Full Knowledge Graph Schema Reference

This appendix gives the complete node-type and relationship-type reference for the knowledge graph, summarised in Chapter 4. Twelve substantive node types and the principal relationship types are listed; the property names are those used in the graph.

A.1 Node Types and Properties

Node type	**Key properties**
Disease	id, name, synonyms, icd10_code, category, affected_organ, disease_stage, prevalence, age_of_onset, inheritance_pattern, prognosis, standard_of_care, source_ids
Drug / Compound	id, name, brand_names, drug_class, drug_type, molecular_formula, approved_status, approval_year, approved_indications, off_label_uses, route_of_administration, half_life, bioavailability, metabolism_pathway, contraindications, black_box_warnings, cost_tier, source_ids

Node type	**Key properties**
Treatment Protocol	id, name, disease_target, line_of_therapy, protocol_type, components, dosing_schedule, duration, setting, monitoring_requirements, response_criteria, success_rate, evidence_level, guideline_source, last_updated, source_ids
Mechanism of Action	id, name, target_type, pathway_affected, action_type, selectivity, downstream_effect, source_ids
Side Effect / Adverse Event	id, name, affected_system, severity, frequency, reversibility, onset_timing, management, dose_dependent, source_ids
Patient Profile	id, age, sex, weight, bmi, ethnicity, comorbidities, current_medications, allergies, organ_function, performance_status, genetic_variants, lab_values, disease_specific_scores, prior_treatments, insurance_tier, patient_preference
Biomarker / Gene	id, name, biomarker_type, measurement_method, predictive_value, prognostic_value, associated_pathway, actionable, source_ids
Clinical Trial	id, title, phase, status, start_date, end_date, sponsor, intervention, comparator, primary_endpoint, sample_size, inclusion_criteria, exclusion_criteria, results_summary, publication_doi
Research Evidence	source_id, title, authors, journal, year, study_type, evidence_level, sample_size, finding_summary, effect_size, disease_focus, drug_or_protocol_focus
Biological Pathway	id, name, role_in_disease, druggable_nodes, pathway_type, source_ids

Node type	Key properties
Procedure	id, name, procedure_type, setting, invasiveness, typical_duration, recovery_time, success_rate, risk_profile, contraindications, source_ids
Symptom	id, name, associated_diseases, severity_scale, duration_type, source_ids

A.2 Relationship Types and Properties

Source	Edge	Target	Key properties
Drug	TREATS	Disease	efficacy_rate, evidence_level, line_of_therapy, source_evidence_ids
Drug	ACTS_VIA	Mechanism of Action	binding_affinity, selectivity, source_evidence_ids
Drug	CAUSES	Side Effect	frequency, severity, dose_dependent, source_evidence_ids
Drug	INTERACTS_WITH	Drug	interaction_type, severity, management, source_evidence_ids
Drug	TARGETS	Biomarker/Gene	action, specificity, source_evidence_ids
Drug	CONTRAINDICATED_IN	Disease	reason, source_evidence_ids

Source	Edge	Target	Key properties
Drug	METABOLIZED_BY	Gene (CYP)	metabolism_type, source_evidence_ids
Treatment Protocol	ADDRESSES	Disease	line_of_therapy, success_rate, evidence_level, source_evidence_ids
Treatment Protocol	INCLUDES	Drug / Procedure	dose, schedule, role
Treatment Protocol	ACHIEVES	Outcome	response_rate, median_OS, median_PFS, source_evidence_ids
Treatment Protocol	RECOMMENDED_BY	Research Evidence	guideline_source, evidence_level
Treatment Protocol	FOLLOWS_STEP	Treatment Protocol	order, condition
Biomarker/Gene	PREDICTS_RESPONSE_TO	Drug / Protocol	direction, strength, source_evidence_ids
Biomarker/Gene	ASSOCIATED_WITH	Disease	association_type, confidence, source_evidence_ids
Biomarker/Gene	INVOLVED_IN	Biological Pathway	role, source_evidence_ids
Disease	PROGRESSES_TO	Disease	probability, timeframe, source_evidence_ids
Disease	COMORBID_WITH	Disease	co_occurrence_rate, source_evidence_ids
Disease	DRIVEN_BY	Biological Pathway	mechanism_description, source_evidence_ids

Source	Edge	Target	Key properties
Clinical Trial	EVALUATES	Drug / Protocol	dose_tested
Clinical Trial	TARGETS	Disease	population_description
Clinical Trial	REPORTS	Research Evidence	doi, year
Research Evidence	SUPPORTS	Drug / Protocol / Biomarker	strength_of_support
Research Evidence	REFUTES	Drug / Protocol	reason
Symptom	INDICATES	Disease	specificity, sensitivity, source_evidence_ids
Biological Pathway	TARGETED_BY	Drug	source_evidence_ids
Procedure	TREATS	Disease	success_rate, evidence_level, source_evidence_ids

A.3 Storage Guarantees

Uniqueness constraints are declared on `Drug.id`, `Disease.id`, `Biomarker.id`, `TreatmentProtocol.id` and `ResearchEvidence.source_id`. All writes use `MERGE`, so processing the same document twice is safe. Every factual edge accumulates `source_evidence_ids`, and all nodes and edges carry `created_at` / `updated_at` timestamps.

Appendix B

Cypher Query Template Catalog

This appendix catalogues the Cypher templates in `kg/cypher.py`, summarised in Chapter 5. Every template takes `$named` parameters and uses the bidirectional substring-matching strategy described in that chapter, so that both a short specific input and a full question string can match a node name. For each template we give its parameters, the fields it returns, and when the query router selects it. Representative query text is shown for the central templates.

B.1 Disease-Centric Templates

`TREATMENTS_FOR_DISEASE` — selected for the *treatments* intent when a disease is resolved. Parameters: `disease_name`, `limit`. Returns disease, treatment, treatment_type, evidence_level, line_of_therapy, efficacy_rate, and supporting papers.

```
MATCH (d:Disease)<-[r:TREATS|ADDRESSES]-(t)
WHERE toLower(d.name) CONTAINS toLower($disease_name)
   OR (size($disease_name) >= 4
       AND toLower($disease_name) CONTAINS toLower(d.name))
RETURN d.name AS disease, t.name AS treatment,
       labels(t)[0] AS treatment_type,
       r.evidence_level AS evidence_level,
       r.line_of_therapy AS line_of_therapy,
       r.efficacy_rate AS efficacy_rate,
```

```
       r.source_evidence_ids AS papers
ORDER BY r.evidence_level LIMIT $limit
```

`PROTOCOL_FOR_DISEASE` — *protocol* intent. Parameters: `disease_name`, optional `line` (`"1L"`, `"2L"`, or `""` for all). Returns protocol, drugs (list), evidence_level, success_rate, evidence_papers.

`BIOMARKERS_FOR_DISEASE` — *biomarkers* intent. Parameter: `disease_name`. Follows `(Biomarker)-[:ASSOCIATED_WITH]->(Disease)` and returns biomarker, predictive_value, prognostic_value, actionable, association_type.

B.2 Drug-Centric Templates

`DISEASES_FOR_DRUG` (V2) — the reverse lookup; selected for the *treatments* intent when only a drug is resolved. Parameter: `drug_name`. Returns drug, treats_disease, disease_type, evidence_level, line_of_therapy, papers.

```
MATCH (drug:Drug)-[r:TREATS]->(d:Disease)
WHERE toLower(drug.name) CONTAINS toLower($drug_name)
   OR (size($drug_name) >= 4
       AND toLower($drug_name) CONTAINS toLower(drug.name))
RETURN drug.name AS drug, d.name AS treats_disease,
       labels(d)[0] AS disease_type,
       r.evidence_level AS evidence_level,
       r.line_of_therapy AS line_of_therapy,
       r.source_evidence_ids AS papers
```

`DRUG_PROFILE` (V2) — a comprehensive single-drug overview using several `OPTIONAL MATCH` clauses. Parameter: `drug_name`. Returns drug, treats (list), side_effects (list), mechanism (list), interacts_with (list), contraindicated_in (list).

`SIDE_EFFECTS_FOR_DRUG` — *side_effects* intent. Parameter: `drug_name`. Follows `(Drug)-[:CAUSES]->(SideEffect)`; returns side_effect, frequency, severity, dose_dependent, reversible.

`DRUG_INTERACTIONS` — *interactions* intent. Parameter: `drug_name`. Follows undirected `(Drug)-[:INTERACTS_WITH]-(Drug)`; returns drug1, drug2, interaction_type, severity, management.

CONTRAINDICATIONS_FOR_DRUG — *contraindications* intent. Parameter: drug_name. Follows (Drug)-[:CONTRAINDICATED_IN]->(condition); returns drug, contraindicated_in, condition_type, reason.

SAFE_TREATMENTS_FOR_PATIENT — filters out drugs contraindicated in a given comorbidity using a NOT EXISTS { MATCH ... } subquery. Parameters: disease_name, comorbidity.

B.3 Evidence Templates

EVIDENCE_FOR_TREATMENT — *evidence* intent. Parameters: disease_name, treatment_name. Resolves the source_evidence_ids on the treatment edge to ResearchEvidence nodes; returns source_id, title, authors, journal, year, evidence_level, study_type.

ALL_EVIDENCE — all ResearchEvidence nodes ordered by creation date; used by the KG Explorer statistics panel.

B.4 Fallback Template

GENERAL_ENTITY_RELATIONS (V2) — the universal backstop, always appended last by the router. Parameter: entity. Matches any node whose name contains the entity and returns its entire neighbourhood in both directions.

```
MATCH (n)-[r]-(m)
WHERE toLower(n.name) CONTAINS toLower($entity)
   OR (size($entity) >= 4
       AND toLower($entity) CONTAINS toLower(n.name))
RETURN n.name AS source, type(r) AS relation, m.name AS target,
       labels(n)[0] AS source_type, labels(m)[0] AS target_type
LIMIT 25
```

B.5 Graph-Visualisation Templates

GRAPH_BROWSE returns all edges between specified node types; GRAPH_SEARCH returns the ego-network of nodes matching a search string; NODE_COUNTS

and `EDGE_COUNTS` are the statistics queries behind the KG Explorer panel.

Appendix C

Sample QA Transcripts with Step Logs

This appendix gives full transcripts for the two worked examples of Chapter 5 and three further questions covering the remaining intent categories. Each transcript shows the question, the resolved entities, the step log, and the synthesised answer with its citations. The transcripts were produced by the V2 engine against the demonstration graph of Chapter 7.

C.1 Treatments (reverse lookup)

Question: "What is the use of zigakibart drug?"
Resolved entities: `Drug:zigakibart`
Step log:

```
- Candidates: ["What is the use of zigakibart drug?"]
- Resolved "What is the use of zigakibart drug?"
      -> Drug:zigakibart (CONTAINS)
- Intent: treatments; only Drug resolved -> reverse-lookup route
- Template 1 (DISEASES_FOR_DRUG) -> 1 row(s); STOP
```

Answer: Zigakibart is used to treat IgA nephropathy. It is an investigational anti-APRIL monoclonal antibody being evaluated for its ability to reduce proteinuria in this condition. *Supporting evidence:* [PMID:38xxxxxx] — a phase II trial of zigakibart in IgA nephropathy.

C.2 Evidence (LLM-Cypher fallback)

Question: "Which randomized trials support budesonide for IgA nephropathy?"

Resolved entities: `Drug:budesonide`, `Disease:IgA nephropathy`

Step log:

```
- Resolved "budesonide" -> Drug:budesonide
- Resolved "IgA nephropathy" -> Disease:IgA nephropathy
- Intent: evidence
- Template 1 (EVIDENCE_FOR_TREATMENT) -> 0 rows, trying next
- Template 2 (GENERAL_ENTITY_RELATIONS) -> 0 rows
- All templates empty; invoking LLM Cypher generator
- Generated Cypher executed -> 2 row(s)
```

Generated Cypher:

```
MATCH (drug:Drug)-[r:TREATS]->(d:Disease)
WHERE toLower(drug.name) CONTAINS "budesonide"
  AND toLower(d.name) CONTAINS "iga nephropathy"
MATCH (ev:ResearchEvidence)
WHERE ev.source_id IN r.source_evidence_ids AND ev.study_type = "RCT"
RETURN ev.title, ev.authors, ev.journal, ev.year, ev.source_id
```

Answer: Two randomized controlled trials in the graph support budesonide (targeted-release formulation) for IgA nephropathy, both reporting a significant reduction in proteinuria versus placebo. *Supporting evidence:* [PMID:34xxxxxx]; [PMID:37xxxxxx].

C.3 Side Effects

Question: "What are the side effects of cyclophosphamide?"

Resolved entities: `Drug:cyclophosphamide`

Step log:

```
- Resolved "cyclophosphamide" -> Drug:cyclophosphamide
- Intent: side_effects
- Template 1 (SIDE_EFFECTS_FOR_DRUG) -> 4 row(s); STOP
```

Answer: Cyclophosphamide is associated with bone-marrow suppression (common), haemorrhagic cystitis, increased infection risk, and gonadal toxicity. Several of these are dose-dependent. *Supporting evidence:* [PMID:29xxxxxx]; [PMID:31xxxxxx].

C.4 Biomarkers

Question: "Which biomarkers are associated with lupus nephritis?"
Resolved entities: `Disease:lupus nephritis`
Step log:

```
- Resolved "lupus nephritis" -> Disease:lupus nephritis
- Intent: biomarkers
- Template 1 (BIOMARKERS_FOR_DISEASE) -> 3 row(s); STOP
```

Answer: Anti-dsDNA antibodies and low complement (C3/C4) are associated with lupus nephritis activity, and anti-C1q antibodies show prognostic value for renal flares. *Supporting evidence:* [PMID:28xxxxxx]; [PMID:33xxxxxx].

C.5 Contraindications

Question: "When is a RAAS inhibitor contraindicated?"
Resolved entities: `Drug:RAAS inhibitor`
Step log:

```
- Resolved "RAAS inhibitor" -> Drug:RAAS inhibitor
- Intent: contraindications
- Template 1 (CONTRAINDICATIONS_FOR_DRUG) -> 2 row(s); STOP
```

Answer: RAAS inhibitors are contraindicated in pregnancy and in bilateral renal artery stenosis, and require caution with significant hyperkalaemia. *Supporting evidence:* [PMID:25xxxxxx].

Note: the PMIDs above are shown in abbreviated form; in the running system each is a live identifier drawn from the `source_evidence_ids` of the traversed edge.

Appendix D

NER and Relation Label Mapping Tables

This appendix reproduces the label-mapping tables referenced in Chapter 4: the raw-label-to-KG-type maps for each of the three NER models, the REBEL Wikidata-label-to-edge map, and the context-aware type-pair override table.

D.1 BioBERT (`d4data/biomedical-ner-all`)

Raw label	**KG type**
DISEASE	Disease
CHEMICAL	Drug
PROTEIN	Biomarker
DNA	Biomarker
RNA	Biomarker
CELL_TYPE	(dropped)
CELL_LINE	(dropped)
SPECIES	(dropped)

D.2 ClinicalBERT / DeBERTa (`Clinical-AI-Apollo/Medical-N`

Raw label (representative)	KG type
DISEASE_DISORDER	Disease
MEDICATION	Drug
DRUG_BRANDNAME	Drug
LAB_VALUE	LabParameter
DIAGNOSTIC_PROCEDURE	Procedure
THERAPEUTIC_PROCEDURE	Procedure
BIOLOGICAL_STRUCTURE	Biomarker
BIOLOGICAL_ATTRIBUTE	Biomarker
SIGN_SYMPTOM	Symptom
DETAILED_DESCRIPTION	(dropped)

The clinical model emits more than forty raw labels; the table lists the ones that map to a KG type. Labels describing qualifiers, severities and free-text descriptions are not mapped.

D.3 GLiNER (`urchade/gliner_mediumv2.1`), 16 zero-shot labels

Zero-shot label	KG type
disease	Disease
drug	Drug
lab test	LabParameter
lab value	LabParameter
symptom	Symptom
biomarker	Biomarker
gene	Biomarker
mutation	Biomarker
procedure	Procedure
treatment protocol	TreatmentProtocol
mechanism of action	MechanismOfAction
biological pathway	BiologicalPathway
signaling pathway	BiologicalPathway
clinical trial	ClinicalTrial
adverse event	SideEffect
side effect	SideEffect

The last seven labels were added per the extension plan so that GLiNER covers the node types the fine-tuned models cannot produce; because GLiNER is zero-shot, no retraining was required.

D.4 REBEL Wikidata Label → KG Edge (`relation_mapper.py`)

REBEL label	**KG edge**
drug used for treatment	TREATS
medical condition treated	TREATS
side effect	CAUSES
adverse effect	CAUSES
contraindication	CONTRAINDICATED_IN
significant drug interaction	INTERACTS_WITH
mechanism of action	ACTS_VIA
biological target	TARGETS
associated with	ASSOCIATED_WITH
gene associated with condition	ASSOCIATED_WITH
has part / active ingredient	INCLUDES
metabolized by	METABOLIZED_BY
instance of / subclass of / part of	(dropped)
manufacturer / country / employer	(dropped)

D.5 Context-Aware Type-Pair Overrides (`context_re.py`)

(head, tail) type	**REBEL label**	**Remapped to**
(Drug, Disease)	subclass of	TREATS
(Drug, Disease)	instance of	TREATS
(Drug, Disease)	has cause	CAUSES
(Drug, Drug)	subclass of	INCLUDES
(Drug, Drug)	has part	INCLUDES
(Drug, SideEffect)	has cause	CAUSES
(Disease, Disease)	subclass of	PROGRESSES_TO
(Disease, Disease)	part of	COMORBID_WITH
(Biomarker, Disease)	subclass of	ASSOCIATED_WITH
(Drug, Biomarker)	subclass of	TARGETS

Appendix E

Hyperparameters and Training Configurations

For reproducibility, this appendix collects the training configurations of the models used in this thesis.

E.1 Phase-1 NER (PubMedBERT via spaCy)

Hyperparameter	**Value**
Base model	PubMedBERT (abstract + full-text)
Framework	spaCy 3 with `spacy-transformers`
Batch size	128
Learning rate	5×10^{-5}
Optimizer	Adam.v1 ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$)
Weight decay (L2)	0.01
Gradient clipping	max norm 1.0
Dropout	0.1
Early-stopping patience	1600 steps
Max steps	20,000 (step-based; max epochs 0)
Evaluation frequency	every 200 steps
Train/test split	80 / 20
Training iterations	9 (incremental)

E.2 Phase-1 Seq2Tree (BART)

The Seq2Tree model fine-tunes BART-base to generate the XML tree representation, minimising the cross-entropy between predicted and ref-

erence XML on the semi-annotated dataset. Both an iterative and a direct fine-tuning regime were used; the direct regime gave a slight edge on most test metrics. The dataset was built by inferring XML with the prior Seq2Tree model and correcting the output in Label Studio, then augmented by systematically varying values and units while preserving input–output alignment. The same 80/20 split was used for evaluation.

E.3 Phase-2 Models

The Phase-2 pipeline uses pretrained models off the shelf, without fine-tuning in the scope of this thesis: the NER ensemble (`d4data/biomedical-ner-all`, `Clinical-AI-Apollo/Medical-NER`, `urchade/gliner_mediumv2.1`), REBEL (`Babelscape/rebel-large`), and the scispaCy UMLS linker (`en_core_sci_sm` plus the UMLS knowledge base). The configurable thresholds are the ensemble NER minimum score (default 0.80) and the LLM-extraction minimum confidence (default 0.50). The fine-tuning of REBEL on BioRED proposed in Chapter 9 is future work; its hyperparameters will be documented here when that training is performed.

Bibliography

[1] Pubmed, national library of medicine, nationanal center for biotechnology information. https://pubmed.ncbi.nlm.nih.gov.

[2] American Kidney Fund. Types of kidney diseases. https://www.kidneyfund.org/all-about-kidneys/types-kidney-diseases. Accessed: 2025-10-21.

[3] Olivier Bodenreider. The unified medical language system (UMLS): integrating biomedical terminology. *Nucleic Acids Research*, 32(suppl 1):D267–D270, 2004.

[4] et al. Cao. Clinicalmind: A knowledge-graph-grounded framework for clinical decision support question answering. *JAMIA Open*, 2026. Methodological basis for the Phase 2 knowledge-graph pipeline.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[6] Kiran Prakash EC and P. Balamurugan. Worked on efficient, precise search and retrieval from igan rct research papers and developed custom user interface. Master's thesis, IIT Bombay, IEOR, 2024.

[7] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 1433–1445, 2018.

[8] Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare (HEALTH)*, 3(1):1–23, 2021.

[9] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. DeBERTa: Decoding-enhanced BERT with disentangled attention. In *International Conference on Learning Representations (ICLR)*, 2021.

[10] M Herrero-Zazo, I Segura-Bedmar, P Martínez, et al. The ddi corpus: An annotated corpus with pharmacological substances and drug–drug interactions. *J Biomed Inform*, 46(5):914–20., 2013.

[11] Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spacy 3: Industrial-strength natural language processing in python, 2020. Open-source software.

[12] Pere-Lluís Huguet Cabot and Roberto Navigli. REBEL: Relation extraction by end-to-end language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2370–2381, 2021.

[13] Yanis Labrak, Adrien Bazoge, Emmanuel Morin, Pierre-Antoine Gourraud, Mickael Rouvier, and Richard Dufour. BioMistral: A collection of open-source pretrained large language models for medical domains. *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

[14] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

[15] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for

natural language generation, translation, and comprehension. *In ACL.*, 2020.

[16] J Li, Y Sun, RJ Johnson, et al. Biocreative v cdr task corpus: a resource for chemical disease relation extraction. *Database*, 2016.

[17] Ling Luo, Po-Ting Lai, Chih-Hsuan Wei, Cecilia N Arighi, and Zhiyong Lu. Biored: a rich biomedical relation extraction dataset. *Bioinformatics*, 23:1–12, 2022.

[18] Microsoft. microsoft/pubmedbert-base-uncased-abstract-fulltext, 2020. Hugging Face, https://huggingface.co/microsoft/pubmedbert-base-uncased-abstract-fulltext.

[19] Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. ScispaCy: Fast and robust models for biomedical natural language processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 319–327, 2019.

[20] Ollama. Ollama: Get up and running with large language models locally. https://ollama.com, 2023.

[21] Vaibhav Singh Panwar and P. Balamurugan. Algorithms for medical prescription generation. Master's thesis, IIT Bombay, IEOR, 2023.

[22] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2nd edition, 2015.

[23] Tengwei Song Rui Xing, Jie Luo. Biorel: towards large-scale biomedical relation extraction. *BMC Bioinformatics*, 2020.

[24] Maxim Tkachenko, Mikhail Malyuk, Andrey Holmanyuk, and Nikolai Liubimov. Label Studio: Data labeling software, 2020–2025. Open source software available from GitHub.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[26] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory*, pages 25–54. PMLR, 2013.

[27] Mintu Yadav and P. Balamurugan. Worked on parameter-specific literature retrieval system for iga nephropathy. Master's thesis, IIT Bombay, IEOR, 2024.

[28] Urchade Zaratiana, Nadi Tomeh, Pierre Holat, and Thierry Charnois. GLiNER: Generalist model for named entity recognition using bidirectional transformer. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024.