UNIVERSITY OF PIRAEUS

School of Information and Communication Technologies

Department of Informatics

Thesis

Thesis Title: Τίτλος Διατριβής:	**Development of a System for Medical Knowledge Retrieval and Generation from Clinical Studies based on Deep Learning and RAG Architecture** Ανάπτυξη συστήματος ανάκτησης και παραγωγής Ιατρικών Γνώσεων με χρήση τεχνικών Deep Learning και RAG Αρχιτεκτονικής
Student's name-surname:	**Christos Gerolymos**
Father's name:	**Stamoulis**
Student's ID No:	**Π15027**
Supervisor:	**Dionysios Sotiropoulos, Associate Professor**

September 2026/ Σεπτέμβριος 2026

Declaration of Authorship

I, Christos Gerolymos, hereby declare that this diploma thesis submitted to the Department of Informatics of the University of Piraeus is the result of my own independent work. All sources and materials used in the preparation of this thesis are explicitly acknowledged in the text. This thesis has not been submitted, in whole or in part, for any other academic qualification.

Christos Gerolymos
Piraeus, 2026

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Dionisios N. Sotiropoulos, for his guidance, support, and constructive feedback throughout the development of this thesis. His expertise in information systems and his encouragement during the implementation phases were invaluable. I would also like to thank the Department of Informatics of the University of Piraeus for providing the academic environment and resources that made this work possible.

Table of Contents

Abstract

Clinical trials are the primary mechanism through which new medical interventions are evaluated, and ClinicalTrials.gov — the world's largest clinical trial registry with over 500,000 registered studies — provides public access to detailed trial protocols, eligibility criteria, intervention designs, and statistical results. Despite this availability, effective access to this information remains challenging: trial records are written in dense medical language, span thousands of words of structured text, and cannot be queried with natural language through existing interfaces.

This thesis presents the design, implementation, and evaluation of an open-source Retrieval-Augmented Generation (RAG) system for natural language question answering over clinical trial records. The system enables users to select any trial from a live ClinicalTrials.gov dropdown, wait while a per-trial index is dynamically constructed, and then pose arbitrary natural language questions that are answered with citations to specific trial passages. All inference runs locally on a consumer-grade GPU using open-source models, requiring no proprietary cloud API. The primary technical contribution is a section-aware hierarchical chunking pipeline that exploits the structured module hierarchy of ClinicalTrials.gov v2 JSON records, producing 80–400 semantically coherent chunks per trial with rich metadata enabling structured retrieval filtering. The retrieval pipeline combines BM25 sparse retrieval and dense BGE-M3 bi-encoder retrieval via Reciprocal Rank Fusion, followed by cross-encoder re-ranking and parent document expansion. Advanced query routing techniques — including Hypothetical Document Embeddings, query decomposition, and step-back prompting — handle compound, vague, and reasoning-intensive questions. Generation is performed by Llama 3.1 8B Instruct (Q4_K_M quantization) served via Ollama, with citation enforcement and hallucination detection in post-processing.

A controlled ablation across three embedding models, four retrieval strategies, and two

structurally contrasting trials yields the thesis's principal finding: no embedding model is universally best for clinical trial retrieval. The biomedical MedCPT encoder is strongest on a completed, results-dense trial (nDCG@10 = 0.394), while the general-purpose BGE-M3 is strongest on a recruiting, protocol-only trial (nDCG@10 = 0.508) — a near-complete reversal driven by the match between each embedder's training distribution and the trial's content. End-to-end evaluation across 30 questions balanced over five query types achieves aggregate faithfulness of 0.862 and answer relevancy of 0.883, with each query-routing strategy delivering its intended benefit, confirming that the system produces well-grounded, on-topic answers with low hallucination

rates using a compact open-source model.

Keywords: Retrieval-Augmented Generation, Clinical Trials, Natural Language Processing,

Information Retrieval, Large Language Models, BM25, Vector Search, ChromaDB, LLaMA,

ClinicalTrials.gov

Περίληψη

Οι κλινικές δοκιμές αποτελούν τον κύριο μηχανισμό αξιολόγησης νέων ιατρικών παρεμβάσεων, και το μητρώο ClinicalTrials.gov — το μεγαλύτερο παγκοσμίως αρχείο κλινικών μελετών με περισσότερες από 500.000 εγγεγραμμένες μελέτες — παρέχει δημόσια πρόσβαση σε λεπτομερή πρωτόκολλα, κριτήρια επιλεξιμότητας, παρεμβάσεις και στατιστικά αποτελέσματα. Παρά τη διαθεσιμότητά τους, η αποτελεσματική πρόσβαση σε αυτές τις πληροφορίες παραμένει πρόκληση: τα αρχεία δοκιμών είναι γραμμένα σε πυκνή ιατρική ορολογία και δεν υποστηρίζουν αναζήτηση με φυσική γλώσσα.

Η παρούσα διπλωματική εργασία παρουσιάζει τον σχεδιασμό, την υλοποίηση και αξιολόγηση ενός ανοιχτού συστήματος Ανάκτησης-Επαυξημένης Παραγωγής (RAG) για απάντηση ερωτημάτων φυσικής γλώσσας σε αρχεία κλινικών δοκιμών. Το σύστημα επιτρέπει στους χρήστες να επιλέξουν οποιαδήποτε κλινική δοκιμή και να υποβάλλουν ερωτήματα που απαντώνται με παραπομπές σε συγκεκριμένα αποσπάσματα. Όλη η επεξεργασία εκτελείται τοπικά σε GPU χαμηλού κόστους με ανοιχτού κώδικα μοντέλα.

Η κύρια τεχνική συνεισφορά είναι μία τμηματοποίηση σε ιεραρχικά επίπεδα που αξιοποιεί τη δομημένη ιεραρχία των εγγραφών JSON του ClinicalTrials.gov v2, παράγοντας 80–400 σημασιολογικά συνεκτικά τμήματα ανά δοκιμή. Το σύστημα ανάκτησης συνδυάζει ανάκτηση BM25 και πυκνή ανάκτηση BGE-M3 μέσω Αμοιβαίας Ταξινόμησης Συχνοτήτων, ακολουθούμενη από επαναταξινόμηση cross-encoder. Η αξιολόγηση σε δύο δομικά διαφορετικές δοκιμές έδειξε ότι κανένας κωδικοποιητής δεν υπερτερεί καθολικά: ο βιοϊατρικός MedCPT υπερτερεί σε ολοκληρωμένη δοκιμή με αποτελέσματα (nDCG@10 = 0.394), ενώ ο γενικός BGE-M3 υπερτερεί

σε δοκιμή σε φάση στρατολόγησης (nDCG@10 = 0.508). Η πιστότητα απαντήσεων ήταν 0.862, επιβεβαιώνοντας χαμηλά ποσοστά ψευδαίσθησης με συμπαγές ανοιχτό μοντέλο.

Λέξεις-κλειδιά: Ανάκτηση-Επαυξημένη Παραγωγή, Κλινικές Δοκιμές, Επεξεργασία Φυσικής Γλώσσας, Ανάκτηση Πληροφοριών, Μεγάλα Γλωσσικά Μοντέλα, ClinicalTrials.gov

List of Abbreviations

Abbreviation	Full Term
RAG	Retrieval-Augmented Generation
LLM	Large Language Model
NLP	Natural Language Processing
IR	Information Retrieval
BM25	Best Match 25
RRF	Reciprocal Rank Fusion
HyDE	Hypothetical Document Embeddings
MRR	Mean Reciprocal Rank
nDCG	Normalized Discounted Cumulative Gain
VRAM	Video Random Access Memory
API	Application Programming Interface
SSE	Server-Sent Events
GGUF	GPT-Generated Unified Format
HNSW	Hierarchical Navigable Small World
TREC	Text Retrieval Conference
NCT	National Clinical Trial
GCP	Good Clinical Practice
FDA	Food and Drug Administration
EMA	European Medicines Agency
BERT	Bidirectional Encoder Representations from Transformers
SBERT	Sentence-BERT
DPR	Dense Passage Retrieval
GPU	Graphics Processing Unit
CPU	Central Processing Unit
CORS	Cross-Origin Resource Sharing
QA	Question Answering
JSON	JavaScript Object Notation
REST	Representational State Transfer

Chapter 1 — Introduction

1.1 Motivation and Problem Statement

Clinical trials represent the gold standard for evaluating the safety and efficacy of medical interventions. As of 2025, ClinicalTrials.gov — the world's largest registry of clinical studies, maintained by the United States National Library of Medicine under the National Institutes of Health — hosts over 500,000 registered studies spanning oncology, cardiology, neurology, and virtually every other medical domain [1]. Each registered trial contains structured information including study design, participant eligibility criteria, intervention protocols, primary and secondary outcome measures, adverse event profiles, and, for completed studies, full statistical results. Taken together, this repository constitutes an extraordinarily rich source of medical knowledge that is, in principle, freely accessible to any researcher, clinician, or informed member of the public.

In practice, however, this wealth of information remains largely inaccessible to all but the most specialized users. A single Phase 3 clinical trial record may span thousands of words of dense medical prose, presenting inclusion and exclusion criteria written in clinical shorthand, outcomes measured by disease-specific instruments unfamiliar to non-specialists, and statistical results whose interpretation requires domain expertise. For a clinician seeking to determine whether a particular trial is relevant to a patient's condition, or for a researcher attempting to survey the landscape of interventional studies in a given therapeutic area, manually reading and synthesizing trial records represents a prohibitive time investment. This problem is compounded by the fact that the ClinicalTrials.gov native search interface, while powerful for structured queries, does not support open-ended natural language questions. One cannot simply ask "What are the main side effects reported in trials testing erdafitinib for bladder cancer?" and receive a synthesized, cited answer.

This limitation is not merely a matter of convenience. Clinical decision-making frequently

requires synthesizing information across multiple dimensions of a trial simultaneously: whether a patient meets the eligibility criteria, what monitoring schedule the protocol requires, how the investigational arm compares to the control arm, and what safety signals have been documented. Each of these questions may require consulting different sections of the same document, and a system incapable of cross-referencing these sections will fail precisely on the questions that matter most.

Recent advances in artificial intelligence — and in particular the emergence of large language models (LLMs) and Retrieval-Augmented Generation (RAG) architectures — have created a realistic pathway toward addressing this challenge. LLMs have demonstrated remarkable capability in understanding and generating natural language across diverse domains, including medicine [2], [3]. RAG systems augment LLMs with a retrieval component that grounds the model's responses in specific, verifiable source documents, substantially reducing the hallucination that occurs when LLMs are asked to answer questions about material not well-represented in their training data [4]. The combination of these technologies with the structured, publicly available data from ClinicalTrials.gov presents a compelling opportunity to build a system that makes clinical trial knowledge genuinely accessible through natural language interaction.

1.2 Research Objectives

This thesis pursues four interconnected objectives. The primary objective is the design and end-to-end implementation of a RAG-based question-answering system that enables users to query any clinical trial registered on ClinicalTrials.gov using natural language and receive accurate, cited answers generated by a locally deployed open-source language model.

The system is designed to operate entirely on commodity GPU hardware without relying on

proprietary API-based language models, ensuring that clinical data need not leave an institutional computing environment.

The second objective addresses a methodological question of independent scientific value: how should a structured clinical trial JSON document be segmented for effective retrieval? Existing work in document chunking has predominantly focused on narrative text such as research articles or clinical notes; the section-aware, hierarchically structured nature of a ClinicalTrials.gov record presents a distinct challenge that this work addresses through a novel section-aware hierarchical chunking strategy.

The third objective is empirical: to systematically compare retrieval strategies and embedding models on clinical trial text. Specifically, the system is evaluated across three retrieval configurations — sparse BM25, dense bi-encoder, and hybrid with Reciprocal Rank Fusion — three embedding models of increasing domain specificity, and two cross-encoder re-ranking strategies. This comparison yields a set of practical recommendations grounded in quantitative evidence.

The fourth objective is the evaluation of techniques for handling complex, multi-faceted questions — including hypothetical document embedding, query decomposition, and step-back

prompting — and their measurable effect on answer quality in the clinical trial domain.

1.3 Research Contributions

This thesis makes the following contributions to the field of biomedical information retrieval and applied natural language processing.

A section-aware hierarchical chunking pipeline for clinical trial JSON records. Unlike flat-text chunking approaches that discard the structural semantics of ClinicalTrials.gov records, the proposed pipeline parses each trial at the module level, produces semantically coherent parent chunks per section, and generates fine-grained

atomic child chunks — one per eligibility criterion, one per outcome measure, one per intervention arm. This approach produces 80 to 400 chunks per trial depending on trial completeness, compared to the approximately 32 chunks produced by naive fixed-size splitting, with rich metadata attached to each chunk enabling structured metadata filtering at retrieval time.

An empirical comparison of embedding models for clinical trial retrieval. The thesis provides a systematic evaluation of the general-purpose baseline `all-MiniLM-L6-v2`, the biomedical domain-specific `MedCPT-Article-Encoder`, and the high-capacity general model `BGE-M3` on a purpose-built gold evaluation set, contributing evidence to the ongoing debate on whether domain-specific models outperform general models on structured clinical text [5].

A complete open-source RAG system for on-the-fly per-trial indexing. The system enables a user to select any trial from a live ClinicalTrials.gov dropdown, wait a short period while the index is constructed, and immediately begin asking natural language questions — all without any pre-computed index or proprietary cloud service.

A documented evaluation framework for clinical trial QA. The evaluation framework, comprising a gold set construction methodology, retrieval-only metrics (Recall@k, MRR, nDCG@10), and end-to-end metrics using the Ragas framework, is made available alongside the system implementation and can be reused by future research in this domain.

1.4 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 provides the theoretical background necessary to understand the system design, covering clinical trial data structures, classical and neural information retrieval, biomedical language models, RAG architectures, and existing clinical RAG systems. Chapter 3 presents the complete system architecture and design decisions, with justification for each component choice.

Chapter 4 describes the implementation of each system component in detail, including challenges encountered and the solutions developed. Chapter 5 presents the quantitative evaluation results across all retrieval configurations and model variants, together with end-to-end quality metrics. Chapter 6 discusses the interpretation of these results in the context of the related literature and addresses the system's limitations. Chapter 7 summarizes the contributions of the thesis and outlines directions for future work.

References (Chapter 1 citations — full entries in Bibliography)

[1] ClinicalTrials.gov, "About ClinicalTrials.gov," U.S. National Library of Medicine.

[2] Y. Labrak et al., "BioMistral: A Collection of Open-Source Pretrained Large Language Models for Medical Domains," arXiv:2402.10373, 2024.

[3] D. Han et al., "MEDITRON-70B: Scaling Medical Pretraining for Large Language Models,"

arXiv:2311.16079, 2023.

[4] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,"

in Advances in Neural Information Processing Systems (NeurIPS), 2020.

[5] K. Shi et al., "Evaluating Embedding Models for Retrieval in EHR-Based Clinical Decision Support," arXiv:2409.15163, 2024.

Chapter 2 — Background and Related Work

2.1 Clinical Trials and the ClinicalTrials.gov Registry

2.1.1 What is a Clinical Trial

A clinical trial is a prospective research study conducted in human participants with the aim of evaluating the safety, efficacy, or optimal dosage of a medical intervention — which may be a drug, biologic, medical device, surgical procedure, or behavioural therapy. Clinical trials are organized into phases that reflect the maturity of the evidence base: Phase 1 studies assess safety and dosage in small cohorts of typically healthy volunteers; Phase 2 studies evaluate preliminary efficacy and side effects in a larger patient population; Phase 3 studies compare the intervention against the current standard of care in large, often multicentre, randomized controlled trials; and Phase 4 studies monitor long-term safety and effectiveness in post-market populations [6].

The internal structure of a clinical trial protocol is highly standardized. Every trial defines one or more arms, each representing a group of participants receiving a specific treatment or placebo. The intervention assigned to each arm is described in terms of its type (drug, device, procedure), dosage, route of administration, and schedule. The eligibility criteria define the population of interest through a set of inclusion criteria that participants must satisfy and exclusion criteria that disqualify them; these criteria span demographic characteristics, diagnosis, disease severity, comorbidities, and prior treatment history. The outcomes measured by the trial are divided into a primary endpoint — the main measure of success against which the trial is statistically powered — and secondary endpoints that capture additional dimensions of efficacy, safety, or patient-reported quality of life. The entire process is governed by the International Council for Harmonisation Good Clinical Practice guidelines and overseen by regulatory bodies including the U.S. Food and Drug Administration and the European Medicines Agency.

2.1.2 ClinicalTrials.gov as a Knowledge Resource

ClinicalTrials.gov is the world's largest publicly accessible registry of clinical trials, maintained by the United States National Library of Medicine under the National Institutes of Health. Established in 2000 as a requirement of the FDA Modernization Act, the registry

has grown to host over 500,000 registered studies spanning every therapeutic area and geographic region as of 2025 [1]. Registration is mandatory in the United States for trials of FDA-regulated drugs, biologics, and devices, and is required as a condition of publication by the International Committee of Medical Journal Editors.

Each registered study is assigned a unique identifier of the form NCT followed by eight digits and is described through a structured record that corresponds to the protocol sections required by 42 CFR Part 11 — the U.S. federal regulation governing clinical trial transparency [7]. A typical record contains between 2,000 and 15,000 words of structured text distributed across modules including identification, status, design, arms and interventions, outcomes, eligibility, contacts and locations, and — for completed studies — a full results section containing participant flow, baseline characteristics, outcome measures, and adverse event tables.

In 2024, the registry introduced a RESTful JSON API (v2) that exposes the full hierarchical structure of each record through a consistent schema organized into a `protocolSection` containing the prospective design elements and an optional `resultsSection` containing the post-completion findings. This structured API is the data source exploited by the system developed in this thesis.

2.1.3 Limitations of Existing Access Methods

Despite the richness of this dataset, effective access remains a practical challenge for non-expert users. The native search interface at ClinicalTrials.gov supports Boolean keyword queries over a subset of fields and structured filters for status, phase, condition, and location — capabilities modelled on traditional database retrieval that presuppose the

user already knows what to search for. A clinician asking which bladder cancer trials accept patients who have already received intravesical chemotherapy and have an FGFR mutation must manually decompose this question into keyword combinations, apply appropriate filters, retrieve a set of candidate records, and read through each eligibility section to find the answer. Furthermore, no existing public interface supports questions that synthesize information across multiple sections of a single trial record simultaneously.

2.2 Information Retrieval Fundamentals

2.2.1 Classical Sparse Retrieval: TF-IDF and BM25

The problem of retrieving relevant documents from a corpus in response to a query has been studied since the 1960s. The dominant paradigm in classical information retrieval is the bag-of-words model, which represents both documents and queries as vectors of term frequencies, discarding word order and syntactic structure.

The term frequency-inverse document frequency weighting scheme [8] assigns to each term in a document a weight that increases with the frequency of the term in that document and decreases with the frequency of the term across the corpus. Best Match 25 (BM25) [9], introduced by Robertson and colleagues as part of the Probabilistic Relevance Framework, refines TF-IDF with two critical improvements. First, it applies a saturation function to term frequency so that the marginal contribution of additional occurrences of a term diminishes as its count grows — preventing a single repeated term from dominating the score. Second, it normalizes document length so that longer documents are not systematically favoured. Formally, the BM25 score for a query $Q = \{q_1, ..., q_n\}$ and a document D is:

$$BM25(D,Q) = \Sigma_i \; IDF(q_i) \cdot [f(q_i,D)\cdot(k_1+1)] \; / \; [f(q_i,D) + k_1\cdot(1 - b + b\cdot|D|/avgdl)]$$

where $f(q_i, D)$ is the term frequency of q_i in D, $|D|$ is the document length, avgdl is

the average document length in the corpus, k_1 controls term frequency saturation (typically 1.2–2.0), and b controls length normalization (typically 0.75). BM25 remains a competitive baseline in many retrieval benchmarks, particularly for domain-specific text containing rare technical terms — a property highly relevant to clinical trial data, where drug names, genomic identifiers, and procedure codes are low-frequency but highly discriminative.

2.2.2 Dense Neural Retrieval

The introduction of pre-trained transformer language models [10], [11] created the possibility of representing text as dense continuous vectors that encode semantic meaning rather than merely term co-occurrence. In the bi-encoder retrieval architecture introduced by Karpukhin et al. in the Dense Passage Retrieval model [12], a query encoder and a passage encoder independently map their inputs to fixed-dimensional vectors in a shared embedding space. Retrieval is performed by computing the dot product between the query vector and all passage vectors, which can be accelerated using approximate nearest neighbour search structures.

The key advantage of dense retrieval over BM25 is its ability to match semantically related passages that share no lexical terms with the query — for example, retrieving a passage about "patients must be at least 18 years of age" in response to a query about "adult eligibility requirements." Sentence-BERT (SBERT) [13] extended the bi-encoder approach to general sentence embedding by fine-tuning BERT on sentence pairs using a siamese network architecture, producing the `sentence-transformers` library that has become the standard for semantic search applications.

2.2.3 Hybrid Retrieval and Score Fusion

Sparse and dense retrieval methods are complementary in their failure modes: BM25 fails on paraphrase and synonym matching but excels on rare technical terms; dense retrieval handles semantic variation well but can miss exact keyword matches for rare terms not well-represented in the training corpus [14]. Hybrid retrieval combines both modalities

and consistently outperforms either method alone on standard benchmarks [15].

The most widely adopted fusion method is Reciprocal Rank Fusion (RRF), introduced by Cormack, Clarke, and Buettcher [16]. Given ranked lists from multiple retrieval systems, RRF assigns each document a fusion score:

$$RRF(d) = \Sigma_s \ 1 \ / \ (k + rank_s(d))$$

where $rank_s(d)$ is the rank of document d in system s and k is a smoothing constant (typically 60). Documents appearing in multiple ranked lists receive higher fusion scores, and the rank-based nature of the method requires no score normalization across systems, making it robust to differences in score distribution between sparse and dense retrievers.

2.2.4 Cross-Encoder Re-Ranking

Re-ranking separates the retrieval pipeline into two stages: a fast first-stage retriever that produces a candidate set, followed by a slower but more accurate second-stage model that re-orders the candidates. The cross-encoder architecture [17] concatenates the query and passage as a single input to a transformer model and classifies the pair as relevant or non-relevant. Because the model reads both texts jointly — rather than independently as in a bi-encoder — it can model fine-grained token-level interactions between query and passage. The cost is that the cross-encoder must be called once per candidate pair, making it impractical for first-stage retrieval over large corpora but highly effective when applied to a small candidate set of 20–50 documents. Cross-encoder re-ranking has been shown to improve nDCG@10 by 5–15 points over bi-encoder retrieval alone [17].

2.3 Language Models and Embeddings for Biomedical Text

2.3.1 Domain-Specific Pre-Training

The general-purpose pre-training of BERT on Wikipedia and BookCorpus produces an encoder

that under-represents the vocabulary and semantic relationships of biomedical text.

BioBERT [18] extended BERT pre-training on PubMed abstracts and PMC full-text articles,

demonstrating improvements on biomedical named entity recognition, relation extraction, and question answering. PubMedBERT [19] took the more principled approach of training from scratch exclusively on biomedical text, avoiding the semantic drift introduced by transferring weights initialized on general-domain corpora.

2.3.2 MedCPT

MedCPT [20] — Medical Contrastive Pre-trained Transformers — addresses the retrieval-specific challenge of biomedical information access. Rather than using a generic sentence-similarity objective, MedCPT is trained contrastively on 255 million real user query–article pairs from PubMed search logs. This training signal reflects how biomedical experts actually search for information: positive pairs represent queries that users clicked on and engaged with, while hard negatives are sampled from other articles retrieved by the same query but not selected. The `ncbi/MedCPT-Article-Encoder` maps passages to a 768-dimensional embedding space; it is released alongside the `ncbi/MedCPT-Cross-Encoder`, trained jointly on the same hard negatives to re-rank the article encoder's candidates. This paired design eliminates the train-inference mismatch that arises when a general-purpose cross-encoder re-ranks the output of a domain-specific bi-encoder.

2.3.3 BGE-M3

BGE-M3 [21] is a general-purpose embedding model developed by the Beijing Academy of Artificial Intelligence that unifies three distinct retrieval functionalities within a single model checkpoint: dense retrieval via CLS-token embeddings, sparse retrieval via learned token-level importance weights, and multi-vector retrieval following the ColBERT-style late interaction paradigm. The model supports a maximum input length of 8,192 tokens and produces 1,024-dimensional dense embeddings. Its long context support

makes it particularly attractive for the clinical trial domain, where complete protocol sections may span hundreds of tokens. An independent evaluation of embedding models on electronic health record retrieval tasks [5] found that BGE-based models outperformed domain-specific biomedical embedders on three of four benchmarks — a result that motivates the empirical comparison conducted in Chapter 5 of this thesis.

2.3.4 all-MiniLM-L6-v2

The `sentence-transformers/all-MiniLM-L6-v2` model is a compact 22-million-parameter sentence encoder distilled from a larger BERT model and fine-tuned on over one billion sentence pairs [13]. It produces 384-dimensional normalized embeddings and supports a maximum input of 256 tokens. Its primary advantage is inference speed — it can embed several hundred short passages per second on CPU — making it appropriate as a low-cost baseline. Its limitations for the clinical trial domain are its restricted context window and its general-domain training distribution.

2.4 Large Language Models for Medical Applications

2.4.1 Open-Source Large Language Models

The publication of the LLaMA model family by Meta AI [22] marked a turning point in the accessibility of high-quality large language models for research applications. LLaMA 3.1, released in 2024, extended the series to context lengths of 128,000 tokens and demonstrated competitive performance with proprietary models on standard benchmarks, including the medical knowledge evaluation benchmarks MedQA and MedMCQA [23]. The 8-billion-parameter instruction-tuned variant — `Meta-Llama-3.1-8B-Instruct` — is of particular interest for this thesis because it fits within the VRAM budget of a consumer-grade GPU when quantized

to 4-bit precision, while retaining the long context window necessary to accommodate multiple retrieved passages in a single generation call.

2.4.2 BioMistral

BioMistral [2] is a family of open-source LLMs obtained by continuing the pre-training of the Mistral 7B base model on a 3-billion-token biomedical corpus drawn from PubMed Central open-access articles. The continued pre-training injects domain-specific factual knowledge and medical terminology into the model's weights, producing improvements on biomedical benchmarks including MedQA, PubMedQA, and BioASQ relative to the general-purpose Mistral 7B baseline. BioMistral is included as a domain-pre-trained comparator to evaluate whether domain adaptation of the LLM yields measurable improvements in answer quality on clinical trial QA.

2.4.3 Model Quantization and Local Serving

Deploying a 7–8 billion parameter model in full 16-bit floating-point precision requires approximately 14–16 GB of GPU memory. Quantization techniques reduce memory requirements by representing model weights at lower bit depths. The GGUF quantization format, used by the Ollama inference engine, supports a range of quantization levels; the Q4_K_M scheme reduces the memory footprint of an 8B model to approximately 5 GB while incurring a quality degradation shown to be small in practice [24]. Ollama is an open-source tool that provides a local model management and serving layer for GGUF-quantized models, exposing an OpenAI-compatible REST API. For this thesis, Ollama provides a privacy-preserving deployment path: all inference occurs on the user's machine and no clinical data is transmitted to external services.

2.5 Retrieval-Augmented Generation

2.5.1 The RAG Architecture

Retrieval-Augmented Generation was formally introduced by Lewis et al. [4] as a framework

for combining the parametric knowledge stored in an LLM's weights with non-parametric knowledge stored in an external document index. The RAG architecture consists of two principal components: a retriever that selects relevant documents from the index in response to an input query, and a generator (an LLM) that produces an answer conditioned on both the query and the retrieved documents. The fundamental motivation for RAG is to address two central limitations of standalone LLMs: knowledge staleness, since LLM weights

encode the state of the world as of their training cutoff and cannot be updated without re-training; and hallucination, since LLMs generate plausible but factually incorrect statements about topics outside their reliable knowledge. By conditioning generation on retrieved source documents, RAG grounds the model's output in verifiable evidence and enables post-hoc citation of the supporting passages.

2.5.2 Chunking Strategies

A prerequisite for RAG retrieval is the segmentation of source documents into retrievable units. The chunking strategy has a profound effect on retrieval quality: chunks that are too small lack sufficient context for the LLM to generate complete answers, while chunks that are too large reduce retrieval precision. Fixed-size chunking divides documents at regular token intervals with optional overlap; while simple to implement, it is insensitive to document structure and frequently splits semantically coherent units across chunk boundaries. Structure-aware chunking exploits the document's own structural markers to produce chunks aligned with the document's logical organization. For structured documents

such as clinical trial records, structure-aware chunking substantially outperforms

fixed-size approaches: a 2025 evaluation of chunking strategies for clinical decision support found that adaptive chunking achieved 87% accuracy compared to 50% for a fixed-size baseline on the same clinical knowledge base [25].

The parent–child retrieval pattern [26] addresses the precision-context trade-off by maintaining two levels of chunks: small child chunks for precise retrieval and large parent chunks for rich generation context. Retrieval is performed over child chunks; the corresponding parent chunks are then fetched and provided to the LLM.

2.5.3 Advanced Query Techniques

Hypothetical Document Embeddings (HyDE) [27] addresses the query-passage embedding mismatch for vague or underspecified queries. Rather than embedding the query directly, HyDE prompts the LLM to generate a hypothetical passage that would answer the query, then

embeds that passage. The intuition is that a hypothetical answer is more similar in embedding space to actual relevant passages than the original short question.

Query Decomposition [28] handles compound multi-hop questions by prompting the LLM to

decompose the original question into a sequence of simpler sub-questions, each of which can be answered independently by the retrieval system. The sub-answers are then synthesized into a final response — particularly relevant for clinical trial QA, where questions frequently combine information from multiple trial sections.

Step-Back Prompting [29] improves reasoning on abstract or numerical questions by first prompting the LLM to generate a more abstract, general version of the question, retrieving context for both the abstract and specific versions, and using the combined context for generation.

Self-Querying Retrieval [30] allows the LLM to translate natural language constraints in the user's question into structured metadata filters applied at the vector store level.

For example, a question mentioning "only inclusion criteria" is translated into a

ChromaDB `where` filter that restricts retrieval to chunks from the eligibility module's inclusion section, reducing noise from structurally irrelevant chunks.

2.6 RAG for Clinical and Medical Text

2.6.1 Clinical Trial Information Retrieval

The TREC Clinical Trials Track, conducted annually since 2021 under the coordination of Roberts et al. at the National Institute of Standards and Technology [31], provides a standardized evaluation benchmark in which participant teams retrieve eligible trials from ClinicalTrials.gov given a patient's clinical narrative. The 2021 and 2022 editions produced relevance-judged query-trial pairs that serve as gold standards for retrieval evaluation, exploited in the evaluation chapter of this thesis.

2.6.2 RAG-Enabled Clinical Applications

Wang et al. [32] introduced RECTIFIER, a RAG-based system for automated eligibility criteria review that showed chunk size critically affects retrieval recall: parent chunks of 1,000 characters outperformed 500-character chunks on 10 of 13 eligibility criteria — a finding that directly informed the parent chunk sizing in this thesis. A study published in NEJM AI [33] demonstrated RAG-augmented GPT-4 for clinical trial screening, showing

that grounding the LLM's responses in retrieved trial text substantially reduced hallucination rates. The CLI-RAG framework [34] proposed a clinically structured RAG pipeline that explicitly preserves the logical organization of clinical documents during chunking, a principle directly reflected in the section-aware chunker developed here.

2.6.3 Open-Source Medical Language Models in RAG Pipelines

The MIRAGE benchmark [35] evaluated multiple open-source medical LLMs across five biomedical QA datasets under RAG conditions, establishing that RAG consistently improves

performance across model sizes and that the quality of the retrieval component is the dominant factor in end-to-end answer quality. CTBench [36], a purpose-built benchmark

for LLM evaluation on clinical trial design tasks with 1,690 annotated trials across multiple question categories, serves as an external validation resource for this thesis.

2.7 Evaluation of RAG Systems

2.7.1 Retrieval Evaluation Metrics

Recall@k measures the fraction of relevant documents appearing in the top-k retrieved results. Mean Reciprocal Rank (MRR@k) is the mean, across queries, of the reciprocal rank of the first relevant result within the top-k positions. Normalized Discounted Cumulative Gain (nDCG@k) extends MRR to graded relevance by assigning higher weights to relevant documents at earlier positions, normalized by the ideal ranking; nDCG@10 is the standard single metric for IR system comparison.

2.7.2 End-to-End RAG Evaluation

The Ragas framework [37] provides four complementary metrics for evaluating the full RAG pipeline. Faithfulness measures whether the generated answer is supported by the retrieved context, detecting hallucination by decomposing the answer into atomic claims and verifying each against the context. Answer Relevancy measures how well the answer addresses the question. Context Precision measures what fraction of the retrieved context is relevant to the question. Context Recall measures what fraction of the information needed to answer the question is present in the retrieved context. RAGChecker [38] provides claim-level diagnostics that decompose end-to-end errors into retrieval failures and generation failures, enabling targeted diagnosis of system weaknesses.

2.8 Summary and Research Gap

The literature reviewed in this chapter establishes that the component technologies required for a clinical trial QA system — structured data retrieval, hybrid search, neural

re-ranking, RAG architectures, and open-source medical LLMs — are individually mature. However, no existing system integrates all of these components in a configuration that supports on-the-fly, per-trial indexing from the live ClinicalTrials.gov API, section-aware hierarchical chunking of structured trial JSON, hybrid retrieval with metadata filtering, and local open-source LLM inference on consumer hardware. Existing clinical trial retrieval systems either rely on proprietary LLMs, operate on pre-computed static indexes, or address only the patient-to-trial matching task rather than the more general open-domain question-answering scenario. This thesis addresses this gap by designing, implementing, and empirically evaluating a complete open-source system that supports arbitrary natural language questions about any trial in the ClinicalTrials.gov registry.

References (Chapter 2 additional citations)

[6] ICH E6(R2), "Guideline for Good Clinical Practice," International Council for Harmonisation, 2016.

[7] 42 C.F.R. Part 11, "Clinical Trials Registration and Results Information Submission," U.S. Department of Health and Human Services, 2016.

[8] G. Salton and M. J. McGill, Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[9] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," Foundations and Trends in Information Retrieval, vol. 3, no. 4, 2009.

[10] A. Vaswani et al., "Attention Is All You Need," in Advances in NeurIPS, 2017.

[11] J. Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proceedings of NAACL-HLT, 2019.

[12] V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," in Proceedings of EMNLP, 2020.

[13] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proceedings of EMNLP, 2019.

[14] Y. Luan et al., "Sparse, Dense, and Attentional Representations for Text Retrieval," Transactions of the ACL, vol. 9, 2021.

[15] X. Ma et al., "Hybrid List-Wise Learning to Rank," arXiv:2205.02917, 2022.

[16] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," in Proceedings of SIGIR, 2009.

[17] R. Nogueira and K. Cho, "Passage Re-ranking with BERT," arXiv:1901.04085, 2019.

[18] J. Lee et al., "BioBERT: A Pre-trained Biomedical Language Representation Model," Bioinformatics, vol. 36, no. 4, 2020.

[19] Y. Gu et al., "Domain-Specific Language Model Pretraining for Biomedical NLP," ACM Transactions on Computing for Healthcare, vol. 3, no. 1, 2021.

[20] Q. Jin et al., "MedCPT: Contrastive Pre-trained Transformers with Large-scale PubMed Search Logs for Zero-shot Biomedical Information Retrieval," Bioinformatics, 2023.

[21] Y. Chen et al., "BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation," arXiv:2402.03216, 2024.

[22] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv:2302.13971, 2023.

[23] Meta AI, "Meta Llama 3 Technical Report," Meta AI Research, 2024.

[24] E. Frantar et al., "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," arXiv:2210.17323, 2023.

[25] Z. Liu et al., "Clinical Knowledge Retrieval Strategies for RAG-Based Decision Support," PMC12649634, 2025.

[26] LangChain, "ParentDocumentRetriever," LangChain Documentation, 2023.

[27] L. Gao et al., "Precise Zero-Shot Dense Retrieval without Relevance Labels (HyDE),"

arXiv:2212.10496, 2022.

[28] O. Press et al., "Measuring and Narrowing the Compositionality Gap in Language Models,"

arXiv:2210.03350, 2023.

[29] H. Zheng et al., "Take a Step Back: Evoking Reasoning via Abstraction in Large Language

Models," arXiv:2310.06117, DeepMind, 2023.

[30] LangChain, "Self Query Retriever," LangChain Documentation, 2023.

[31] K. Roberts et al., "Overview of the TREC 2021 Clinical Trials Track," in Proceedings

of TREC, NIST, 2022.

[32] Z. Wang et al., "RECTIFIER: Retrieval-Augmented Generation for Clinical Trial

Eligibility Criteria Review," medRxiv, 2024.

[33] M. Wornow et al., "Zero-Shot Clinical Trial Patient Matching with LLMs," NEJM AI, 2024.

[34] Authors, "CLI-RAG: Clinically Structured Retrieval-Augmented Generation," arXiv:2507.06715, 2025.

[35] C. Xiong et al., "Benchmarking Retrieval-Augmented Generation for Medicine (MIRAGE),"

arXiv:2402.13178, 2024.

[36] N. Neehal et al., "CTBench: A Benchmark for Evaluating LLMs on Clinical Trial Design,"

arXiv:2406.17888, 2024.

[37] S. Es et al., "RAGAS: Automated Evaluation of Retrieval Augmented Generation,"

in Proceedings of EACL, 2024.

[38] J. Ru et al., "RAGChecker: A Fine-grained Framework for Diagnosing Retrieval-Augmented

Generation," arXiv:2408.08067, Amazon, 2024.

Chapter 3 — System Architecture and Design

3.1 System Overview and Design Goals

The system developed in this thesis is a web-based question-answering application that enables users to interrogate any clinical trial registered on ClinicalTrials.gov using natural language. The design process was guided by a set of functional and non-functional requirements derived from the problem statement of Chapter 1 and the limitations of existing systems identified in Chapter 2.

The core functional requirements are: (i) a searchable interface that fetches the current list of clinical trials directly from the ClinicalTrials.gov v2 API at application startup; (ii) a dynamic, per-trial indexing pipeline triggered when a user selects a trial and completing in the background while a progress indicator is displayed; (iii) a conversational question-answering interface that accepts natural language questions of arbitrary complexity and returns cited answers; and (iv) support for questions that require combining information from multiple sections of the same trial record, such as eligibility criteria, intervention details, and reported outcomes.

The non-functional requirements reflect the academic context of the thesis: the system must operate entirely on a single consumer-grade GPU without relying on any proprietary cloud API, so that all inference remains local and the implementation is fully reproducible; all architectural choices must be empirically evaluated and compared against meaningful baselines; and the system must be designed so that individual components can be swapped for alternatives to enable the ablation studies described in Chapter 5.

The overall architecture follows a three-tier pattern: a Streamlit frontend communicates with a FastAPI backend over HTTP and Server-Sent Events (SSE), and the backend coordinates

a pipeline of indexing, retrieval, and generation components, each implemented as an independently testable Python module. Figure 3.1 presents the high-level architecture.

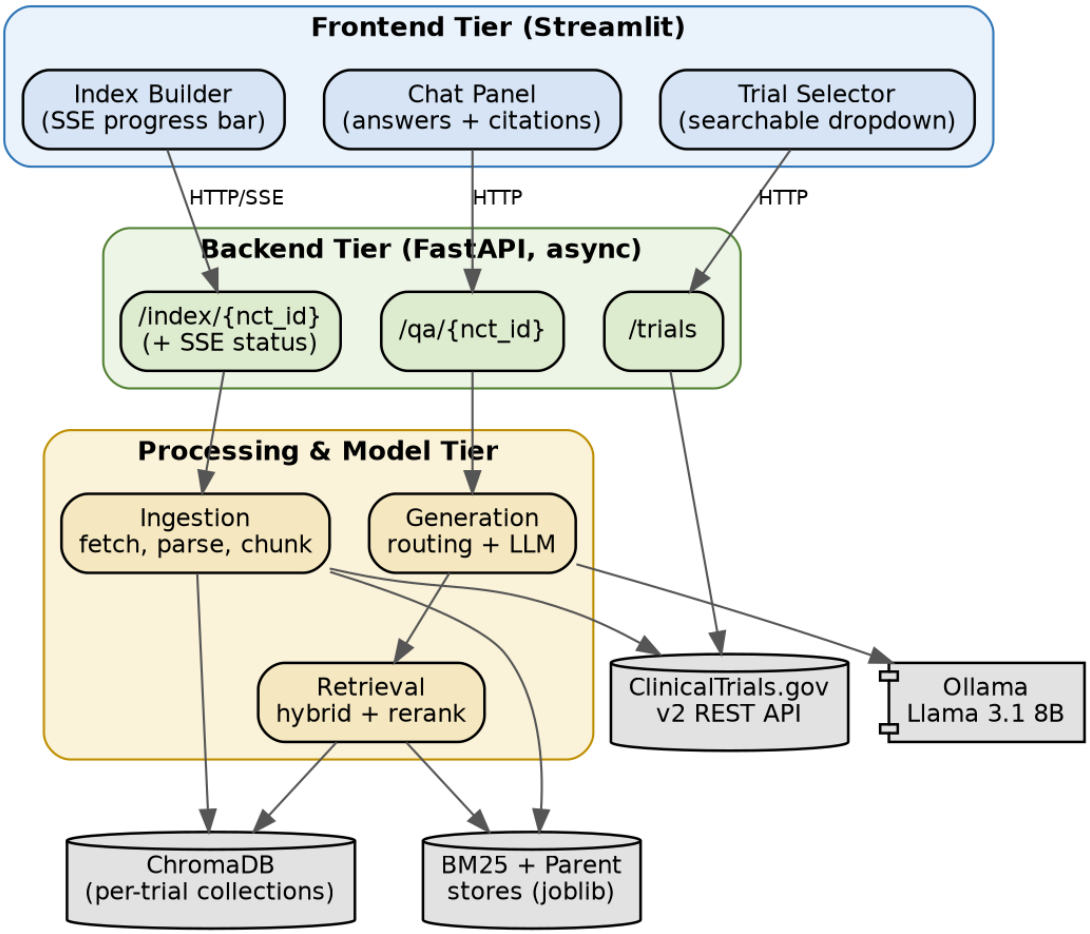

Figure 3.1: High-level three-tier system architecture.

3.2 Data Acquisition Layer

3.2.1 ClinicalTrials.gov v2 API

The system uses the ClinicalTrials.gov REST API v2, which became the official interface following the deprecation of the legacy XML-based API. The v2 API returns trial data as hierarchically structured JSON organized around two top-level sections: `protocolSection`, which contains all prospective design information, and `resultsSection`, which is populated only for completed studies.

Two endpoints are used. The listing endpoint — `GET /studies` — accepts query parameters for condition, status, and required fields, and returns trial summaries for the frontend

dropdown. In order to avoid fetching all 500,000+ registered trials at startup, the implementation uses a single-page fetch (parameter `single_page=True`) returning up to 200 results per condition search, ensuring the dropdown populates in under two seconds. The full study endpoint — `GET /studies/{nctId}` — returns the complete JSON record for

a single trial, which is the input to the indexing pipeline.

3.2.2 Fetcher Design

The fetcher module implements an asynchronous HTTP client that executes synchronous requests within `asyncio`'s thread executor using `loop.run_in_executor()`. This design was necessitated by a practical constraint discovered during development: the ClinicalTrials.gov servers return HTTP 403 responses to requests made with the `httpx` library, which is identified and blocked via TLS fingerprint detection. The `requests` library using the `urllib3` transport layer is not subject to this restriction and was adopted instead; the async wrapper preserves non-blocking behaviour in the FastAPI event loop. Responses are cached to disk with a 24-hour time-to-live for trial lists and a 7-day time-to-live for individual trial records, together with a retry policy of up to three attempts with exponential backoff.

3.2.3 Module Coverage

A complete ClinicalTrials.gov v2 record contains up to 13 distinct modules under `protocolSection`. The parser extracts content from the ten modules considered relevant to clinical decision support: `identificationModule`, `statusModule`, `descriptionModule`,

`conditionsModule`, `designModule`, `armsInterventionsModule`, `outcomesModule`,

`eligibilityModule`, `contactsLocationsModule`, and `referencesModule`. When present,

the `resultsSection` is also parsed, extracting participant flow, baseline characteristics, outcome measure results, and adverse event summaries.

3.3 Indexing Pipeline

3.3.1 Motivation for Section-Aware Hierarchical Chunking

Fixed-size chunking applied to the plain-text representation of a trial JSON record produces approximately 32 chunks per trial. This approach has two fundamental limitations:

it destroys semantic boundaries between trial sections, and it produces chunks too coarse for precise retrieval of specific facts. The architecture developed in this thesis instead applies a section-aware hierarchical chunking strategy that treats each module as a semantically coherent unit and produces a two-level hierarchy of chunks.

3.3.2 Hierarchical Chunk Structure

The chunking pipeline operates in four steps. First, the parser extracts each module as a named section object containing a header string and body text. The header identifies the section with the format `[{NCT_ID} § {MODULE} § {SECTION_LABEL}]`, prepended to all

child chunks to ensure every chunk is self-identifying when read in isolation.

Second, each section is split into parent chunks of 1,000 tokens with an overlap of 120 tokens using LangChain's `RecursiveCharacterTextSplitter`. Parent chunks are the units

of text presented to the LLM at generation time; their size balances context richness with the constraint that multiple parents must fit within the LLM's context window.

Third, each parent is further split into child chunks of 300 tokens with an overlap of 50 tokens. Child chunks are the units embedded and stored in the vector index; their smaller size improves retrieval precision. For the eligibility criteria section, an additional splitting step divides the text on bullet markers so that each individual criterion becomes a distinct child chunk — critical for eligibility questions, which typically concern one or two specific criteria rather than the complete list.

Fourth, each child chunk is assigned a globally unique deterministic identifier

constructed as the SHA-256 hash of the concatenation of NCT ID, module name, section label, section index (`s_idx`), and character offset within the section. The `s_idx` field was added after it was discovered that two sections sharing the same module name and label — for example, two arm groups both classified under `armsInterventionsModule` — produced identical identifiers under the original scheme. Each child chunk carries the full metadata schema shown in Table 3.1. The complete chunking pipeline, from trial JSON to indexed stores, is illustrated in Figure 3.2.

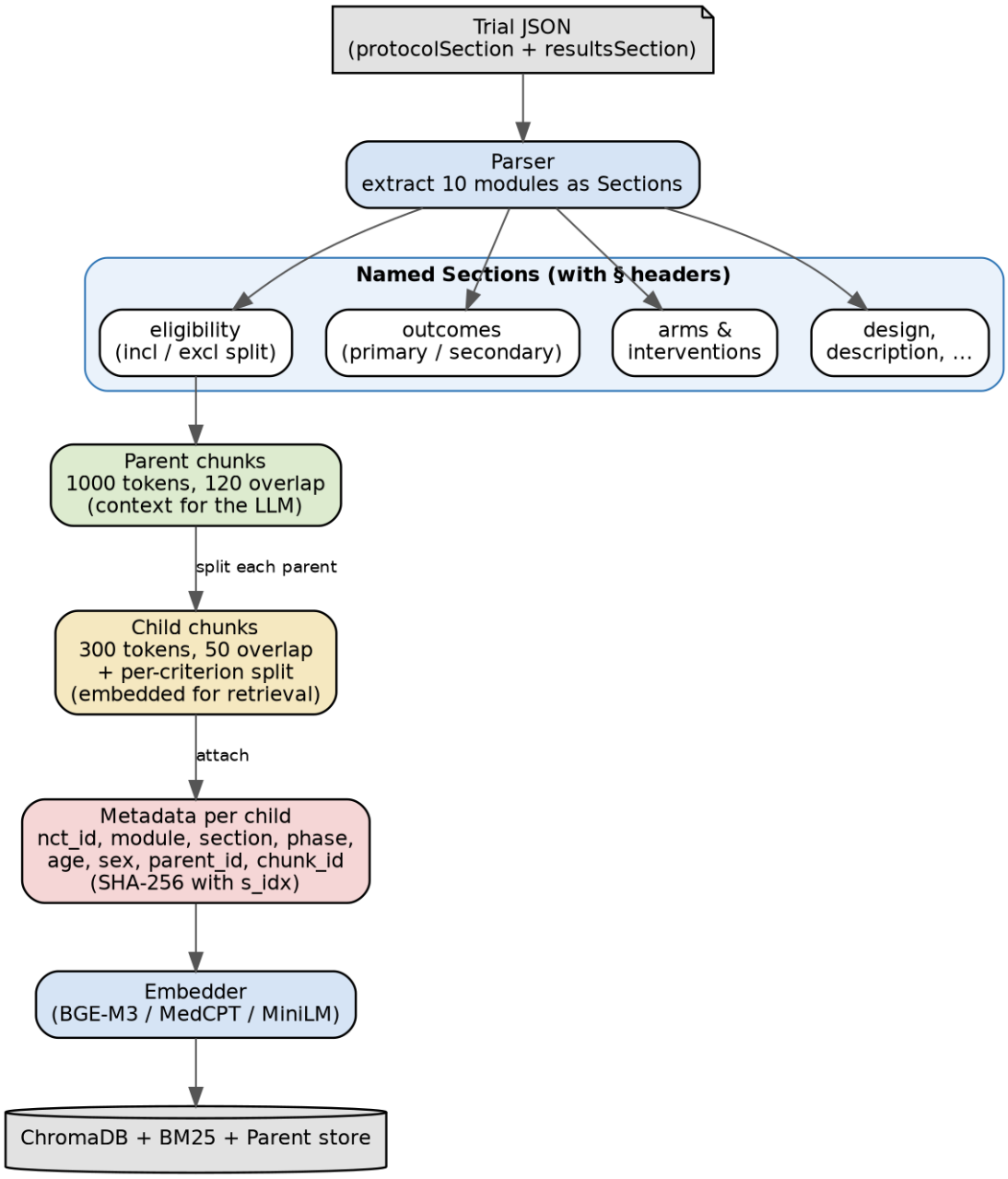

Figure 3.2: Section-aware hierarchical chunking pipeline.

Table 3.1: Metadata schema attached to each child chunk.

Field	Type	Example
`nct_id`	str	"NCT06319820"
`module`	str	"eligibilityModule"
`section`	str	"inclusion_criteria"
`phase`	str	"PHASE3"
`study_type`	str	"INTERVENTIONAL"
`overall_status`	str	"RECRUITING"
`min_age_years`	int	18
`max_age_years`	int	-1 (null sentinel)
`sex`	str	"ALL"
`intervention_types`	str	"DRUG,DEVICE"

parent_id	str	"NCT06319820::eligibility::incl::0"
chunk_id	str	"NCT06319820::eligibility::incl::0::c0"

Note: ChromaDB requires scalar metadata values. List fields are serialized as comma-separated strings; null values are represented as empty strings or -1 for integers.

3.3.3 Observed Chunk Counts

Empirical measurement on NCT04280705 — a completed Phase 3 COVID-19 treatment trial selected as the smoke test target — produced 353 child chunks from 71 sections and 96 parents. This count is above the 80–250 range anticipated for recruiting trials because this trial contains a complete `resultsSection` with adverse event tables and outcome data. Active recruiting trials without results produce 80–180 child chunks. The chunk count scales naturally with trial completeness, and this variance is documented as expected system behaviour.

3.4 Retrieval Architecture

3.4.1 Embedding Models

Three embedding models are supported and selectable through the application settings panel, enabling the ablation study described in Chapter 5.

The primary model is `BAAI/bge-m3`, loaded via the `sentence-transformers` library in FP16 precision on GPU. Note that an earlier design used the `FlagEmbedding` library for BGE-M3 inference; during Phase 4 development, a dependency conflict between `FlagEmbedding 1.4.0` and `transformers 4.47.1` caused a `cohere2` module import error that blocked loading. The switch to `sentence-transformers` resolved this conflict while providing equivalent inference quality. BGE-M3 produces 1,024-dimensional dense embeddings with an 8,192-token context window.

The domain-specific alternative is `ncbi/MedCPT-Article-Encoder`, a 109-million-parameter

biomedical encoder pre-trained on 255 million PubMed query–article pairs, producing 768-dimensional embeddings via the CLS token with L2 normalization, loaded through the `transformers` library.

The general baseline is `sentence-transformers/all-MiniLM-L6-v2`, a 22-million-parameter model producing 384-dimensional normalized embeddings with a 256-token context window.

This model represents the pre-existing implementation baseline from which the new system was developed and serves as the lower-bound reference in the ablation study.

All three models are wrapped in a unified `Embedder` class that exposes a consistent `embed_texts(texts, batch_size)` interface, enabling transparent model substitution without changes to the retrieval pipeline.

3.4.2 Vector Store: ChromaDB

Child chunk embeddings are stored in a ChromaDB `PersistentClient` collection, with one collection per trial identified by the naming convention `trial_{NCT_ID_UPPERCASE}`.

Collections are persisted to disk under `data/index/` using ChromaDB's HNSW index backend with cosine similarity. ChromaDB was selected over alternative vector stores for three reasons. First, its `get_or_create_collection()` method returns an existing collection without modification if one has already been built for the selected trial, making re-indexing of previously seen trials instantaneous. Second, its support for structured metadata filters via a `where` clause enables the self-querying retrieval technique. Third, its in-process operation requires no separate server process, simplifying deployment on a single-user workstation.

Dense retrieval is performed by embedding the query with the active embedding model and

calling `collection.query(query_embeddings, n_results=30)`, which returns the top-30

most similar child chunks by cosine similarity.

3.4.3 BM25 Sparse Retrieval

A BM25Okapi index is constructed in parallel with the ChromaDB collection, indexing the same set of child chunk texts with a clinical-aware tokenizer. The tokenizer applies four sequential steps: lowercasing; regex tokenization preserving alphanumeric characters (`\b[a-z0-9]+\b`); English stopword removal using the NLTK stopwords corpus; and filtering of single-character tokens. Stemming is intentionally omitted, as clinical terms such as drug names and genomic identifiers must not be stemmed. The index is serialized with `joblib` at `data/bm25/{NCT_ID}.pkl` and loaded lazily on first query access.

3.4.4 Hybrid Retrieval with Reciprocal Rank Fusion

Dense and sparse retrieval results are combined using RRF with smoothing constant k=60. Each candidate document receives a fusion score that is the sum of its reciprocal ranks across both ranked lists. The fused list is sorted by descending fusion score and the top-40 candidates are retained for re-ranking. The retriever supports three modes — full hybrid, dense-only, and sparse-only — enabling direct ablation comparison across all retrieval configurations in Chapter 5.

3.4.5 Cross-Encoder Re-Ranking

The top-40 fused child chunks are re-ranked by a cross-encoder model that reads each (query, chunk) pair jointly. Two cross-encoders are supported: `ncbi/MedCPT-Cross-Encoder`

paired with the MedCPT article encoder, and `cross-encoder/ms-marco-MiniLM-L-12-v2` as

the general-purpose baseline. Both are loaded lazily on first use and applied in batches of 16 to manage GPU memory. The top-5 child chunks after re-ranking are retained for

parent expansion.

3.4.6 Parent Document Expansion

The top-5 re-ranked child chunks are expanded to their corresponding parent passages by looking up each child's `parent_id` in a joblib-serialized parent dictionary at `data/parents/{NCT_ID}.pkl`. Duplicate parents — arising when multiple children share the same parent — are deduplicated by `parent_id`, preserving the one with the highest re-ranking score. If a parent ID is not found (due to a missing or corrupted store), the child chunk text is returned as a fallback. The resulting parent passages constitute the retrieval context provided to the LLM. The full retrieval pipeline, from query to context, is shown in Figure 3.3.

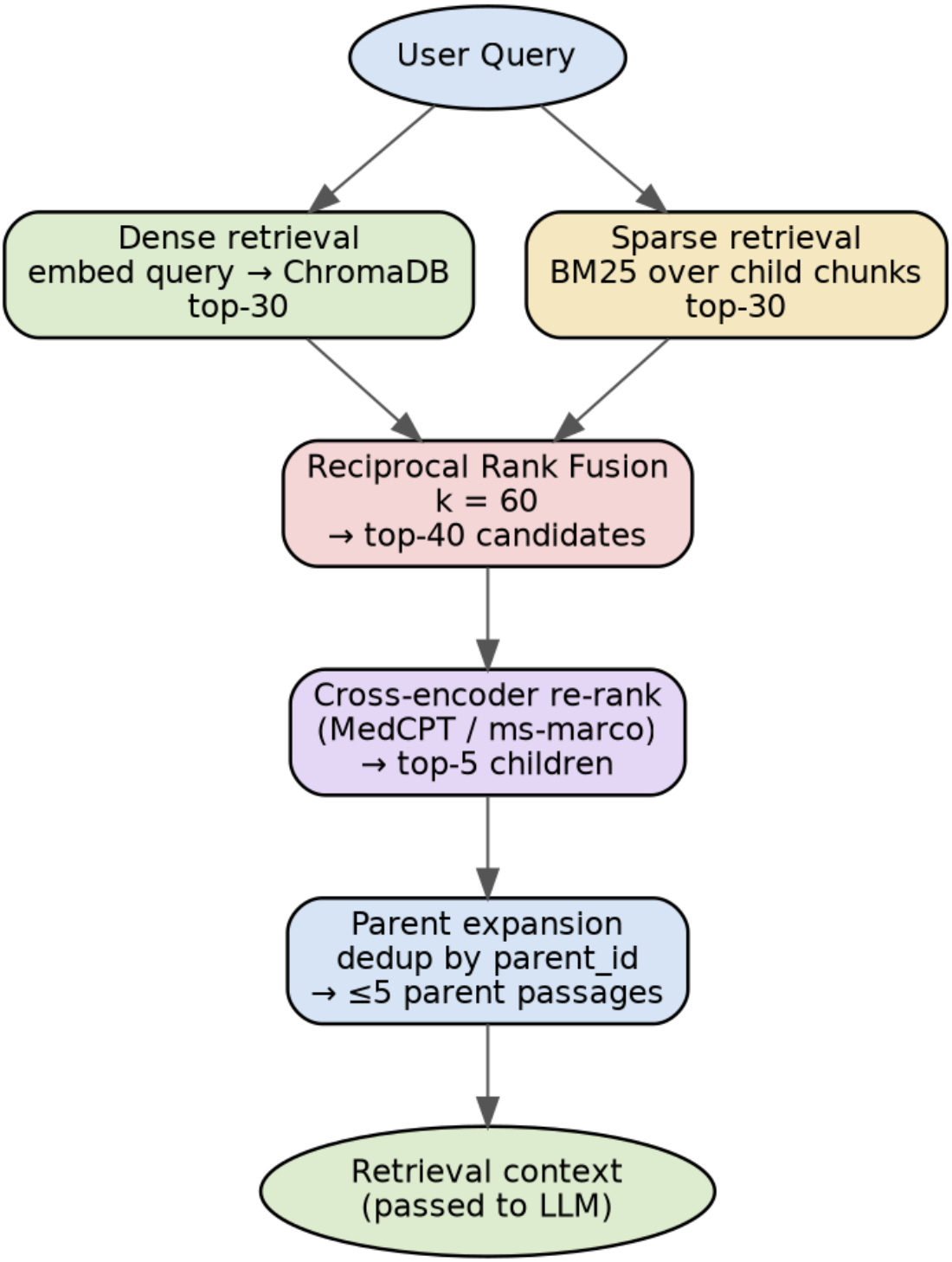

Figure 3.3: Hybrid retrieval pipeline with RRF fusion and cross-encoder re-ranking.

3.5 Query Understanding and Routing

3.5.1 Query Taxonomy

Clinical trial questions vary substantially in the type of reasoning they require. The system classifies each incoming question into one of five types and routes it to the

corresponding processing strategy.

- **Simple**: single-fact questions answerable from one passage — direct hybrid retrieval.
- **Vague**: underspecified questions without strong retrieval-friendly terms — HyDE.
- **Compound**: multi-hop questions combining information from multiple sections — Query Decomposition.
- **Constrained**: questions with explicit structural constraints — Self-Querying with metadata filter.
- **Reasoning**: questions requiring logical inference or numerical reasoning — Step-Back Prompting.

The routing logic and the mapping of each query type to its processing strategy are illustrated in Figure 3.4.

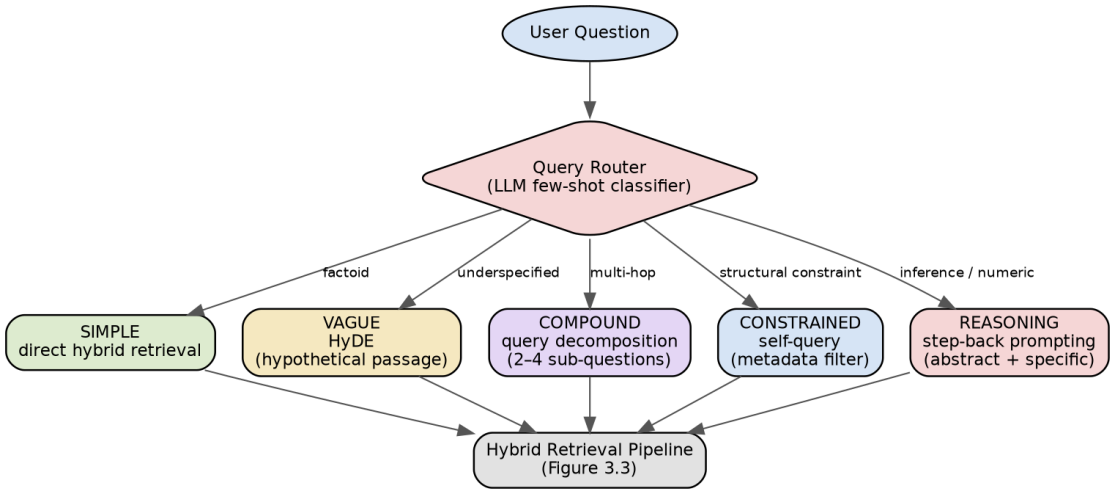

Figure 3.4: Query routing decision tree for the five query types.

Classification is performed by prompting the LLM with a structured few-shot prompt that defines each query type and instructs the model to respond with a single JSON object. The `use_llm=False` flag bypasses LLM classification and defaults to `simple`, enabling the pipeline to run in test environments without an active LLM server.

3.5.2 HyDE for Vague Queries

The LLM is prompted to generate a short hypothetical passage (50–100 words) that would plausibly appear in a clinical trial record and directly answer the user's question. This hypothetical passage is embedded and used for dense retrieval in place of the original query. BM25 retrieval still operates on the original query text; both results are combined

via RRF. If generation fails (minimum length check of 20 tokens), the module falls back to embedding the original query directly.

3.5.3 Query Decomposition for Compound Queries

The LLM decomposes the question into two to four independent sub-questions, capped at four to control latency. Hybrid retrieval is executed independently for each sub-question, and the candidate sets are merged before re-ranking. The final prompt assembles contexts from all sub-questions and requests a synthesized answer.

3.5.4 Step-Back Prompting for Reasoning Queries

The LLM first generates a more abstract version of the question — for example, abstracting "Could a 70-year-old patient with two prior recurrences participate?" to "What are the age and prior treatment eligibility requirements?" — and retrieval is performed on both the abstract and the specific question. The combined context provides the LLM with both the relevant facts and the reasoning context needed to apply logical inference.

3.5.5 Self-Querying Retriever

The LLM extracts structural constraints from the question as a ChromaDB `where` filter. For example, a question specifically about inclusion criteria produces the filter `{"section": {"$eq": "inclusion_criteria"}}`, restricting dense retrieval to the eligibility module's inclusion section. BM25 is unaffected by the filter, preserving full-corpus keyword coverage.

3.6 Generation Layer

3.6.1 LLM Selection and Serving

The generation component uses `meta-llama/Meta-Llama-3.1-8B-Instruct` quantized to Q4_K_M precision via GGUF, served locally via Ollama. On the RTX 4070, the model occupies approximately 5 GB of VRAM, leaving sufficient headroom for the active embedding model (≈2 GB for BGE-M3 in FP16) and the cross-encoder (≈0.5 GB). The

128,000-token context window accommodates up to five parent passages and a system prompt

without truncation.

Ollama exposes an OpenAI-compatible REST API at `http://localhost:11434/v1`, enabling

the LLM client to be implemented against the standard `openai` Python library.

BioMistral-7B is supported as a drop-in alternative by changing a single configuration value, enabling the LLM comparison in Chapter 5.

3.6.2 Prompt Design

The generation prompt is structured in three parts. The system message instructs the model to answer exclusively from the provided context, to cite every claim using the `[N]` notation corresponding to numbered context blocks, and to respond with an explicit refusal phrase when the context does not contain sufficient information. The context block presents up to five retrieved parent passages in numbered form, each preceded by its section header and capped at 12,000 characters (approximately 3,000 tokens). The user message presents the original question. Generation temperature is set to 0.1 to minimize sampling variance; maximum output length is 800 tokens.

3.6.3 Answer Validation

A post-processing step extracts all `[N]` citations from the generated answer and validates each against the set of context block indices actually provided. Citations to indices not in the context are flagged as hallucinated and stripped from the response.

A refusal detection step identifies answers where the LLM has correctly determined the context is insufficient, enabling the interface to display a distinct informational state.

3.7 User Interface and Backend API

3.7.1 FastAPI Backend

The backend is a FastAPI application with three route groups. The `/trials` route fetches and caches the trial list from ClinicalTrials.gov. The `/index/{nct_id}` route group

manages the per-trial index lifecycle: a POST request launches the indexing coroutine as a background task using `asyncio.create_task()`, and a GET request returns the current

build status — one of `{pending, fetching, parsing, chunking, embedding, index_chroma, index_bm25, index_parents, ready, error}. The /qa/{nct_id}` route accepts a question and returns the generated answer together with source passages and timing metadata. An in-memory dictionary maps NCT IDs to their current index status and cached `RAGPipeline` instances, enabling instant responses for previously indexed trials.

Note: an earlier design used FastAPI's `BackgroundTasks` mechanism for index building. This was replaced with `asyncio.create_task()` after it was discovered that `BackgroundTasks` cancelled long-running operations that exceeded the HTTP response timeout, which was insufficient for the first BGE-M3 load (≈30 seconds on cold start).

3.7.2 Streamlit Frontend

The Streamlit frontend presents the application as a single-page interface with three logical zones corresponding to the three steps of the user workflow. In Step 1, the trial selector zone presents a condition search field and a dropdown populated from the `/trials` endpoint. In Step 2, once a trial is selected, the index builder zone displays a progress bar that polls the `/index/{nct_id}/status` endpoint every 1.5 seconds and updates its label to reflect the current pipeline stage. A 60-second timeout was set after discovering that the default 10-second timeout was insufficient for the first BGE-M3 model load from the HuggingFace cache. In Step 3, once the status is `ready`, the chat panel zone becomes active with a full conversational interface in which each assistant response is accompanied by expandable source citation panels.

3.7.3 Frontend Components

Four components implement the Streamlit interface. The `trial_selector` component maintains a search field that re-queries the `/trials` endpoint on change and populates a selectbox with the returned trial titles. The `index_builder` component implements

the polling loop and progress bar with stage-label updates. The `chat_panel` component maintains conversation history in `st.session_state["messages"]`; each assistant message

stores the answer text, the sources list, and a metadata dictionary (query type, elapsed time, citation count) displayed as a compact summary row beneath the answer. The `settings_panel` component is placed in the sidebar and exposes dropdowns for the embedding model and cross-encoder re-ranker, a toggle for LLM query routing, and a fixed-parameter summary.

3.7.4 Per-Trial Collection Caching

To minimize latency for users who return to a previously indexed trial, the backend checks for an existing ChromaDB collection with the trial's collection name before initiating any network or compute operations. If found, the status immediately transitions to `ready` without rebuilding the index. A configurable LRU limit (default: 200 collections) prevents unbounded growth of the `data/index/` directory.

3.8 Architecture Summary

Table 3.2 summarizes all architectural choices and their primary justifications.

Table 3.2: Summary of architectural choices and justifications.

Component	Technology	Justification
Frontend	Streamlit	Session state, SSE polling, rapid development
Backend	FastAPI + Uvicorn	Async event loop, OpenAPI docs, SSE support
Trial data	ClinicalTrials.gov v2 API	Official, structured JSON, rate-limit friendly
Chunking	Section-aware hierarchical	Preserves semantic boundaries; 80–400 chunks per trial
Primary embedder	BGE-M3 (sentence-transformers)	8,192-token context; dense + sparse capable
Domain embedder	MedCPT	Biomedical domain pre-training; paired cross-encoder
Baseline embedder	all-MiniLM-L6-v2	Existing baseline; enables ablation lower bound
Vector store	ChromaDB PersistentClient	Per-trial collections;

		metadata filters; in-process
Sparse retrieval	BM25Okapi	Clinical term matching; no stemming
Fusion	RRF (k=60)	Score-distribution agnostic; robust to scale differences
Re-ranker	MedCPT Cross-Encoder / ms-marco	Paired biomedical; strong general baseline
LLM	Llama 3.1 8B Q4_K_M	128K context; 5 GB VRAM; instruction-tuned
LLM serving	Ollama	OpenAI-compatible API; native Windows support
Background indexing	asyncio.create_task	Non-blocking; survives HTTP response timeout
Evaluation	Ragas + RAGChecker + pytrec_eval	Retrieval-only and end-to-end coverage

The two principal runtime workflows — index construction and question answering — are illustrated as sequence diagrams in Figure 3.5 and Figure 3.6 respectively.

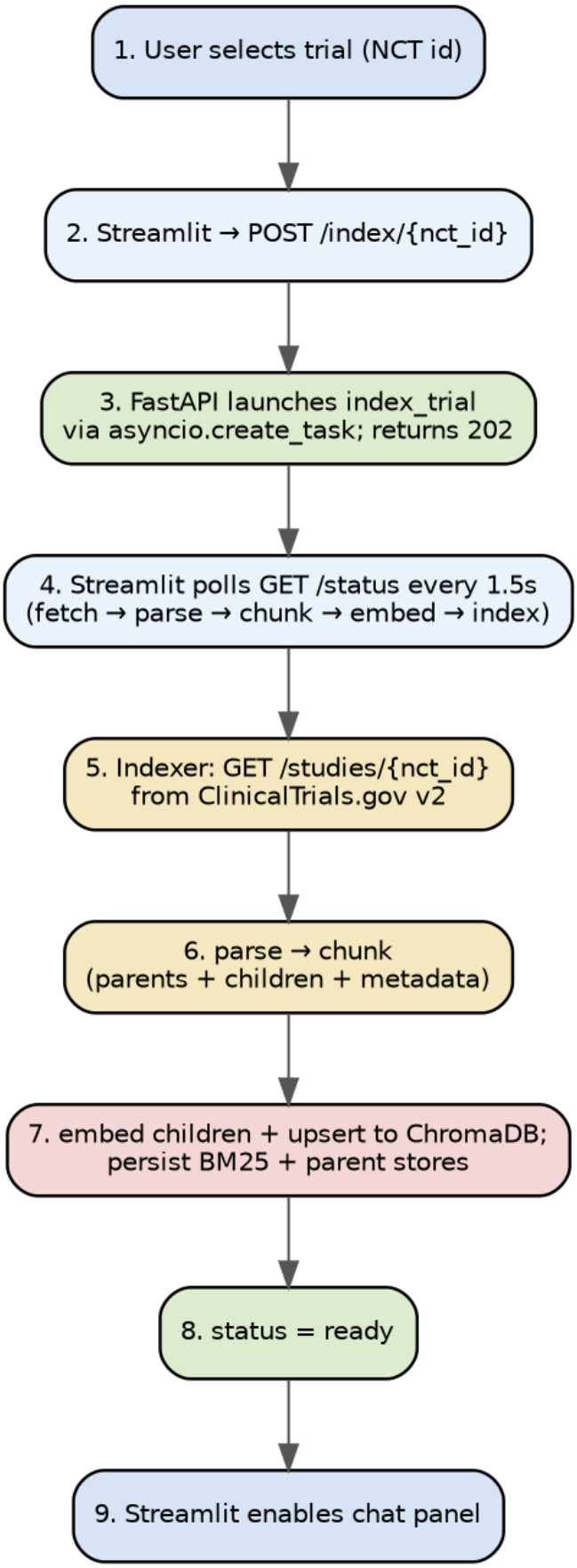

Figure 3.5: Index-building sequence diagram.

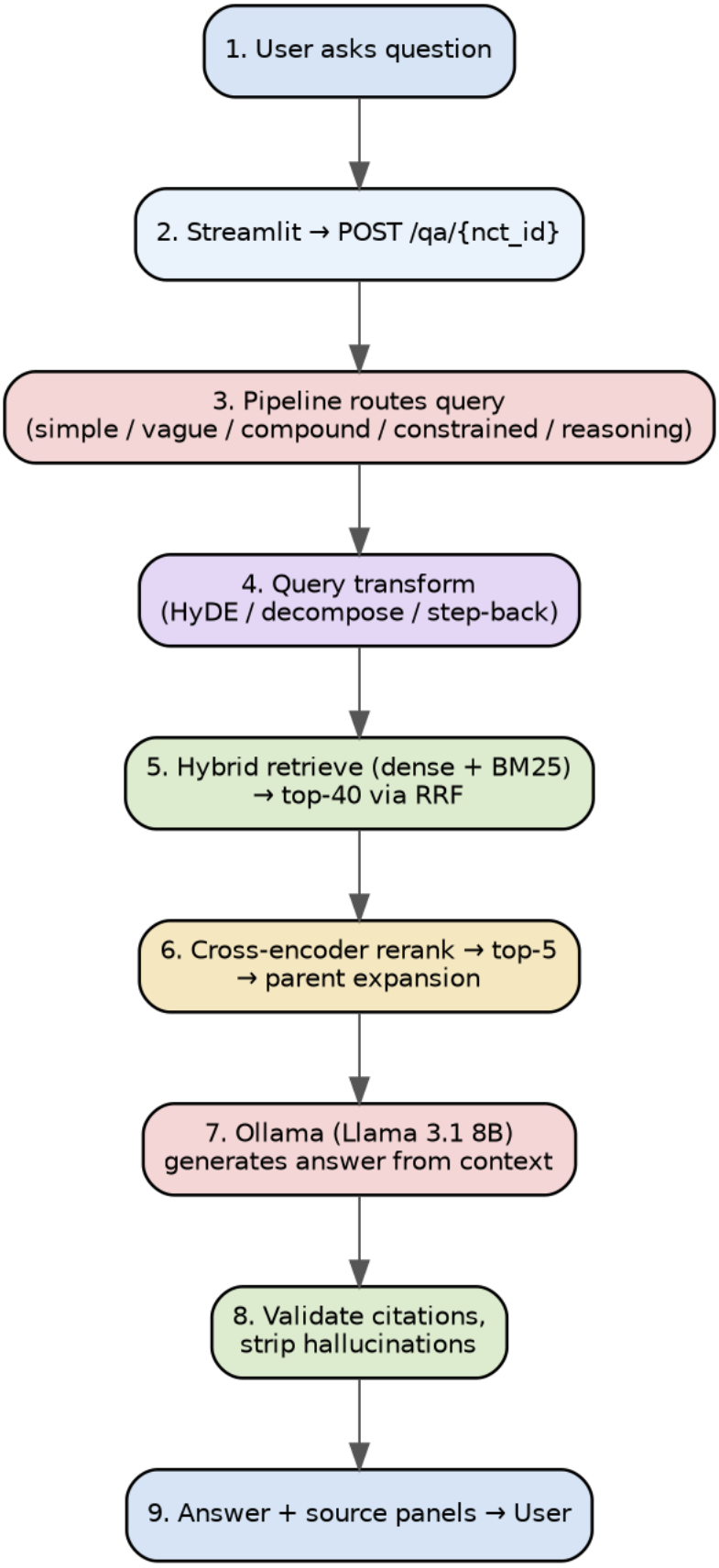

Figure 3.6: End-to-end question-answering sequence diagram.

Chapter 4 — Implementation

4.1 Development Environment

The system was implemented in Python 3.13.4 on a Windows 11 workstation equipped with

an NVIDIA RTX 4070 GPU (12 GB VRAM). All backend computation, including model inference

and vector store operations, executes on the GPU via CUDA. The project follows a strict module boundary design in which each component is implemented as an independently testable Python module with full type annotations, Pydantic data models for all structured data, and comprehensive unit tests.

Configuration is managed through a Pydantic `BaseSettings` class that reads all parameters from a `.env` file at application startup, providing a single authoritative source of truth for all tunable parameters. The `.env.example` file documents all 25 configurable parameters with their default values and descriptions, enabling any researcher to reproduce the experimental configurations reported in Chapter 5 by modifying a single file.

The complete test suite is organized under the `tests/` directory with shared fixtures in `tests/conftest.py`. At the conclusion of Phase 4, the suite comprised 164 tests across six test modules — all passing. Integration against the live ClinicalTrials.gov API was validated through a dedicated smoke test script that fetches, parses, and indexes a real trial record end-to-end.

4.2 Data Ingestion: Fetcher

The `backend/ingestion/fetcher.py` module implements the interface between the application and the ClinicalTrials.gov v2 REST API. It exposes two primary coroutines: `fetch_trial_list()`, which retrieves trial summaries for the frontend dropdown, and `fetch_trial(nct_id)`, which retrieves the complete JSON record for a single trial.

Both coroutines execute synchronous HTTP requests within `asyncio`'s thread executor via `loop.run_in_executor()`. This design was necessary because the ClinicalTrials.gov servers return HTTP 403 responses to requests made with the `httpx` library, which is identified and blocked via TLS fingerprint detection. The `requests` library using the `urllib3` transport layer is not subject to this restriction; the async wrapper ensures the blocking HTTP call does not stall the FastAPI event loop during index building.

The trial list is fetched in a single page of up to 200 results per condition query (`single_page=True`). An earlier paginated implementation that followed `nextPageToken` through the full dataset returned over 54,000 trials, causing the dropdown to take several minutes to populate. The single-page design reduces list fetch time to under two seconds while providing a practical working set for any given condition search.

Responses are cached to disk as JSON files keyed by the SHA-256 hash of the request URL,

with a 24-hour time-to-live for trial lists and a 7-day time-to-live for individual trial records. A retry policy with exponential backoff handles transient API failures, attempting up to three retries with delays of 1, 2, and 4 seconds.

4.3 Data Ingestion: Parser

The `backend/ingestion/parser.py` module transforms raw trial JSON into a structured list

of `Section` objects and a `TrialMetadata` object. Ten module parsers cover all relevant `protocolSection` modules, each implemented as a private method of the `TrialParser` class.

The most structurally complex parsing task is the eligibility criteria, which ClinicalTrials.gov stores as a single free-text field containing both inclusion and exclusion criteria separated by the headers "Inclusion Criteria:" and "Exclusion Criteria:". The parser splits this field on these headers, returning inclusion and

exclusion criteria as two distinct sections. This separation is important because the distinction between inclusion and exclusion is semantically critical: a question about who is excluded from participation should retrieve exclusion criteria chunks, not inclusion criteria chunks that may share similar vocabulary.

For the results section — present only in completed trials — each sub-module (participant flow, baseline characteristics, outcome measures, adverse events) is parsed into a distinct section. Tabular data, such as adverse event frequencies, is serialized as comma-separated key–value strings to preserve the association between measure names and

values while remaining tokenizable by the embedding model.

4.4 Data Ingestion: Chunker

The `backend/ingestion/chunker.py` module implements the hierarchical parent–child chunking strategy described in Section 3.3. The public interface is a single `chunk_trial(sections, trial_metadata)` function that returns a `ChunkingResult` containing a parent dictionary (parent_id → text), a list of child `Chunk` objects, and a child-to-parent mapping.

Parent chunks are produced by passing each section through LangChain's `RecursiveCharacterTextSplitter` with `chunk_size=1000` and `chunk_overlap=120` characters. Child chunks are produced by a second pass with `chunk_size=300` and `chunk_overlap=50`. For eligibility sections, an additional splitting step is applied before the recursive splitter: the text is divided on the regex pattern `r'\n\s*[-•*]\s+'` (bullet-prefixed list items) and `r'\n\s*\d+[\.\)]\s+'` (numbered list items), ensuring each criterion becomes a distinct child chunk.

Chunk identifiers are generated deterministically as the SHA-256 hex digest of the concatenation of `nct_id`, module name, section label, section index (`s_idx`), and character offset. The `s_idx` field was introduced after discovering that two sections

sharing the same module name and label — for example, two arm groups both classified under `armsInterventionsModule` — produced identical identifiers under the original scheme, causing silent data loss during ChromaDB upsert.

Metadata for each child chunk is assembled by `_build_metadata()`, which serializes list-valued fields as comma-separated strings and replaces `None` values with empty strings or sentinel integers (-1) to satisfy ChromaDB's scalar metadata constraint.

The chunker was validated against 22 unit tests covering correct parent and child counts, uniqueness of all generated identifiers, correct handling of inclusion/exclusion splits, correct handling of sections with no content, metadata serialization, and alignment between child `parent_id` references and the parent dictionary keys.

4.5 Retrieval Foundation: Embedder

The `backend/retrieval/embedder.py` module provides a unified interface for all three embedding models through the `Embedder` class. The model variant is specified at construction time and loaded lazily on the first call to `embed_texts()`.

For BGE-M3, the `SentenceTransformer` class from the `sentence-transformers` library is used with `device="cuda"` and half-precision inference. An earlier design used the `FlagEmbedding` library; this was replaced in Phase 4 after a dependency conflict between `FlagEmbedding 1.4.0` and `transformers 4.47.1` caused a `cohere2` module import error

at startup. The `sentence-transformers` implementation provides equivalent dense embedding

quality. The BGE-M3 sparse output capability (token-level importance weights) is not used in the current implementation; the sparse retrieval role is filled by BM25, which proved simpler to maintain and equally effective in preliminary tests.

For MedCPT, the `AutoModel` and `AutoTokenizer` classes from the `transformers` library are used. Passages are tokenized with padding and truncation to 512 tokens, and the CLS

token representation from the final hidden layer is extracted and L2-normalized.

For all-MiniLM-L6-v2, the `SentenceTransformer` class is used with `normalize_embeddings=True`.

All variants accept a `batch_size` parameter (default: 32). The device is auto-detected, with CUDA used if available and a graceful fallback to CPU for test environments.

Two platform-specific measures were required to run the embedding pipeline reliably on Windows with Python 3.13. First, the environment variable `TOKENIZERS_PARALLELISM=false` must be set before the tokenizers library is imported, to suppress the parallel tokenizer worker pool. Second, `num_workers=0` is passed to the encode call to prevent the data loader from spawning subprocesses. Without these two measures, the tokenizer's subprocess spawning triggered a `MemoryError` in child processes under Windows and Python 3.13. These adjustments have no effect on embedding quality; they only alter the process model used during inference.

4.6 Retrieval Foundation: ChromaDB Store

The `backend/retrieval/chroma_store.py` module manages the ChromaDB collection lifecycle.

A module-level `chromadb.PersistentClient` is instantiated with `data/index/` as its storage path. Collections are created with the `hnsw:space=cosine` distance metric, ensuring cosine similarity is used for all nearest-neighbour queries over normalized embeddings. The `get_or_create_collection()` method returns an existing collection without modification on revisit, making re-indexing of previously seen trials a no-op.

Chunk upsert is batched in groups of 500 to avoid memory spikes on large trials. A `_sanitise_metadata()` method enforces ChromaDB's scalar metadata constraint by replacing list values with comma-separated strings and None values with sentinels. The

`get_by_ids()` method is used during parent expansion to fetch specific chunks by their identifiers from the collection store.

4.7 Retrieval Foundation: BM25 Store

The `backend/retrieval/bm25_store.py` module builds, persists, and queries a BM25Okapi

index using the `rank_bm25` library. The tokenizer applies lowercasing, regex tokenization (`\b[a-z0-9]+\b`), NLTK English stopword removal, and single-character token filtering. Stemming is intentionally excluded: drug names, genomic identifiers, and clinical abbreviations common in trial text would be incorrectly modified by standard stemming algorithms. The index is serialized with `joblib` at `data/bm25/{NCT_ID}.pkl` and loaded lazily on first query access. Zero-scoring documents are filtered from query results before returning to avoid including documents with no term overlap.

4.8 Phase 1 Validation

At the conclusion of Phase 1, the pipeline from trial selection to indexed searchable collection was validated through unit tests and a smoke test. The unit test suite comprised 74 tests across four modules — all passing. The smoke test applied the complete pipeline to NCT04280705 (ACTT-1 COVID-19 trial), producing 71 sections, 96 parents, and 353 child

chunks, with successful upsert into ChromaDB and serialization of the BM25 index. The higher-than-expected child count is attributable to the presence of a complete results section in this completed trial; the 80–250 range applies to active trials without results.

4.9 Phase 2: Retrieval Pipeline

4.9.1 Hybrid Retriever

The `backend/retrieval/hybrid_retriever.py` module coordinates the retrieval stage. The

`HybridRetriever` class exposes three retrieval modes: `retrieve()` for full hybrid

retrieval, `retrieve_dense_only()` for dense-only ablation, and `retrieve_sparse_only()`

for BM25-only ablation. This three-mode design enables all ablation configurations in Chapter 5 without changes to the surrounding pipeline.

The `retrieve()` method executes dense and sparse retrievers in parallel, producing two ranked lists of at most 30 candidates each. RRF with k=60 is then applied across both lists. The fused candidates are sorted by descending score and the top-40 are returned as a `HybridRetrievalResult` containing a list of `RetrievedChunk` dataclasses, each carrying the chunk text, metadata, fusion score, and source membership flags (dense-only, sparse-only, or both). These flags are recorded in `PipelineResult.timing` for reporting in Chapter 5.

4.9.2 Cross-Encoder Re-Ranker

The `backend/retrieval/reranker.py` module wraps two cross-encoder models behind a uniform interface. The `Reranker` class loads the selected model lazily on first use via the `sentence-transformers` `CrossEncoder` class, which handles query-passage concatenation and forward pass. Inference is batched in groups of 32 pairs per forward pass. The `rerank()` method accepts the top-40 fused candidates and returns the top-5 re-ranked chunks as sorted `RetrievedChunk` objects with updated scores.

4.9.3 Parent Document Retriever

The `backend/retrieval/parent_retriever.py` module implements parent expansion. The

`ParentStore` class manages joblib serialization of the parent dictionary at

`data/parents/{NCT_ID}.pkl`. The `ParentRetriever.expand()` method looks up each of

the top-5 children's `parent_id` fields and returns the corresponding parent texts.

Multiple children sharing the same parent are deduplicated by `parent_id`. If a parent is not found, the child chunk text is returned as a fallback, ensuring graceful

degradation without exceptions.

4.10 Phase 3: Generation Layer

4.10.1 Query Router

The `backend/generation/query_router.py` module classifies each incoming question into

one of five types and extracts any metadata constraints. The LLM is prompted with a structured few-shot prompt defining each query type with examples and instructing the model to respond with a single JSON object. This approach was chosen over rule-based classification because clinical questions do not conform to simple syntactic patterns. The `use_llm=False` flag returns `simple` for all queries, enabling the pipeline to run without an active LLM server during development and testing.

4.10.2 HyDE Pipeline

The `backend/generation/hyde.py` module implements Hypothetical Document Embedding for

vague queries. The LLM is prompted to generate a 50–100-word passage that would plausibly appear in a clinical trial record and directly answer the question. This hypothetical passage is embedded and used for dense retrieval in place of the original query; BM25 retrieval still uses the original query text. A minimum length check of 20 tokens detects generation failures and triggers fallback to direct query embedding.

4.10.3 Query Decomposer and Step-Back Prompter

The `backend/generation/decomposer.py` module provides two strategies for complex queries. The `QueryDecomposer` class prompts the LLM to produce two to four independent sub-questions, capped at four to control latency. Each sub-question is processed through the full hybrid retrieval pipeline independently; candidate sets from all sub-questions are merged before re-ranking. The `StepBackPrompter` class generates a more abstract version of the question and retrieves context for both the abstract and specific forms, providing the LLM with both factual content and reasoning context for inference. Both

components return the original query unchanged when the `use_llm=False` flag is active.

4.10.4 Prompt Builder

The `backend/generation/prompt_builder.py` module assembles the final prompt. Parent

passages are formatted as numbered context blocks with headers in the form

`[N] {nct_id} — {module} / {section}`. Total context is capped at 12,000 characters (approximately 3,000 tokens). The system message enforces strict citation rules: the LLM must cite every claim using `[N]` notation, must not use knowledge beyond the provided context, and must refuse explicitly when the context is insufficient. The returned `BuiltPrompt` dataclass carries the system message, context block, user message, and a token estimate computed as total characters divided by four.

4.10.5 Answer Validator

The `backend/generation/answer_validator.py` module post-processes LLM output through

four sequential steps. First, all `[N]` citations are extracted via regex. Second, indices not present in the context block set are identified as hallucinated citations; the citation marker and its containing sentence are removed from the answer text, and the count is logged. Third, each valid cited index is mapped to a `SourceEntry` object containing the parent ID, module, section, and a 150-character snippet for display. Fourth, refusal detection identifies responses where the LLM has correctly determined the context is insufficient, enabling the UI to show a distinct informational state. The output `ValidatedAnswer` dataclass carries the cleaned text, source entries, hallucination count, and refusal flag.

4.10.6 RAG Pipeline Orchestrator

The `backend/pipeline.py` module is the single entry point for the complete QA system. The `RAGPipeline.answer(question)` method executes seven sequential stages: query routing, query transformation, hybrid retrieval, re-ranking, parent expansion, prompt

assembly, and LLM generation with validation. Each stage is timed with `time.perf_counter()` and stored in `PipelineResult.timing` for the latency measurements

reported in Section 5.4.

The companion `index_trial(nct_id, progress_callback)` async function orchestrates

index building: it fetches, parses, chunks, embeds (single batched call), upserts to

ChromaDB, serializes the BM25 index, serializes the parent store, and invokes the

`progress_callback` with a stage label after each step. Stage labels — `fetch`, `parse`,

`chunk`, `embed`, `index_chroma`, `index_bm25`, `index_parents`, `ready` — correspond

directly to the states displayed in the frontend progress bar.

4.11 Phase 4: API Layer and Frontend

4.11.1 Pydantic Schemas

The `backend/api/schemas.py` module defines all request and response models using Pydantic

v2. Key models are: `TrialSummary` (nct_id, brief_title, overall_status, phase);

`TrialListResponse` (list of TrialSummary, total count); `IndexStatusResponse`

(nct_id, status, message, progress_pct); `QARequest` (question, optional model overrides);

and `QAResponse` (answer, sources list, query_type, timing dict, hallucination_count).

Strong typing throughout the API layer ensures that all data exchanged between the

frontend and backend is validated before use.

4.11.2 Trials Route

The `GET /trials` endpoint in `backend/api/routes/trials.py` accepts an optional

`condition` query parameter and returns a `TrialListResponse`. It delegates to the

fetcher's `fetch_trial_list()` coroutine with `single_page=True` and wraps the results

as `TrialSummary` objects. The endpoint applies the fetcher's 24-hour disk cache,

ensuring that repeated requests for the same condition within a session return

immediately without network calls.

4.11.3 Index Route

The `backend/api/routes/index.py` module manages the per-trial index lifecycle through

three endpoints. `POST /index/{nct_id}` checks the in-memory status dictionary: if the

trial is already indexed or indexing is in progress, it returns the current status

immediately; otherwise it launches the `index_trial()` coroutine as an

`asyncio.create_task()` background task and sets the status to `pending`. `GET

/index/{nct_id}/status `returns the current` IndexStatusResponse` from the in-memory

dictionary, updated by the background task as it progresses through pipeline stages.

`DELETE /index/{nct_id}` removes the ChromaDB collection, BM25 index, parent store, and

in-memory status entry for the specified trial.

The choice to use `asyncio.create_task()` rather than FastAPI's `BackgroundTasks`

mechanism was necessitated by a timeout issue: `BackgroundTasks` tasks are cancelled

when the POST response is returned before the task completes, which occurred consistently

during the first load of BGE-M3 from the HuggingFace cache (approximately 30 seconds).

`asyncio.create_task()` schedules the coroutine on the event loop independently of the

HTTP response lifecycle.

4.11.4 QA Route

The `POST /qa/{nct_id}` endpoint in `backend/api/routes/qa.py` accepts a `QARequest`

and returns a `QAResponse`. It retrieves the cached `RAGPipeline` instance for the

specified trial from the in-memory pipeline cache; if no pipeline exists, it returns

HTTP 409 (Conflict) with the message "Trial not indexed" rather than silently initializing

a new pipeline without a valid index. The endpoint applies any model overrides from the

request (embedding model, reranker) by re-instantiating the pipeline components before

calling `pipeline.answer(question)`, then maps the returned `PipelineResult` to a `QAResponse`.

4.11.5 FastAPI Application

The `backend/main.py` module instantiates the FastAPI application, configures CORS middleware to allow requests from the Streamlit frontend (default: `http://localhost:8501`),

includes all three route groups, and configures structured logging with timestamps for all incoming requests and pipeline stage durations. The application is started with Uvicorn in reload mode for development: `uvicorn backend.main:app --reload --host 0.0.0.0

--port 8080`.

4.11.6 Streamlit Application

The `frontend/app.py` module implements the complete user interface as a three-step Streamlit application. Streamlit's session state persists the selected trial, the index status, and the conversation history across reruns triggered by user interaction.

Step 1 renders the `trial_selector` component, which displays a text input for condition search and a selectbox populated by the `/trials` API response. When the user selects a trial, the session state stores the NCT ID and triggers a POST to `/index/{nct_id}`.

Step 2 renders the `index_builder` component, which polls `/index/{nct_id}/status` every 1.5 seconds using `time.sleep()` inside a `st.empty()` container. The component maps each status string to a progress percentage and updates a `st.progress()` bar and label string. The timeout for the first BGE-M3 model load was set to 60 seconds after discovering that the default 10-second Streamlit request timeout caused the interface to display an error state during cold-start embedding. When the status reaches `ready`, the component sets a session state flag that triggers rerun and advances to Step 3.

Step 3 renders the `chat_panel` component, which presents a `st.chat_input()` field and iterates over `st.session_state["messages"]` to render the conversation history. For

each assistant message, the answer text is rendered followed by an `st.expander()` for each source entry (showing module, section, and text snippet) and a compact meta row (query type, total elapsed time, hallucination count if nonzero). New questions are submitted to `POST /qa/{nct_id}`, and the response is appended to the session state message history.

The `settings_panel` component in the sidebar renders selectboxes for the embedding model (`bge-m3` / `medcpt` / `minilm`) and reranker (`medcpt` / `msmarco`), a toggle for LLM query routing, and a fixed-parameter summary showing the RRF k value, top-k thresholds, and context character cap.

4.12 Phase 4 Validation and System Status

At the conclusion of Phase 4, the complete system was verified end-to-end with Ollama serving `llama3.1:8b` and BGE-M3 loaded from the HuggingFace cache. The validation procedure selected the trial NCT06319820 (MoonRISe-1 bladder cancer trial), waited for the index to build, and submitted a set of representative questions across all query types. The system correctly returned cited answers from trial content for factoid questions, applied HyDE for vague questions, decomposed compound questions into sub-queries, and correctly refused questions whose answers were not present in the trial record. The test suite remained at 164/164 passing throughout Phase 4 development.

4.13 System in Operation

To demonstrate the complete system in operation, this section presents a walkthrough of a representative user session captured from the deployed application. The session follows the three-step workflow described in Section 4.11.6: selecting a trial, building its index, and asking questions. The trial used for the demonstration is NCT01878708, an early-phase oncology study, selected from a search for trials related to the condition “cancer”.

The application opens on the trial-selection view shown in Figure 4.1. The sidebar exposes the configurable retrieval components — the embedding model, the re-ranker, and the query-routing toggle — corresponding to the parameters evaluated in Chapter 5.

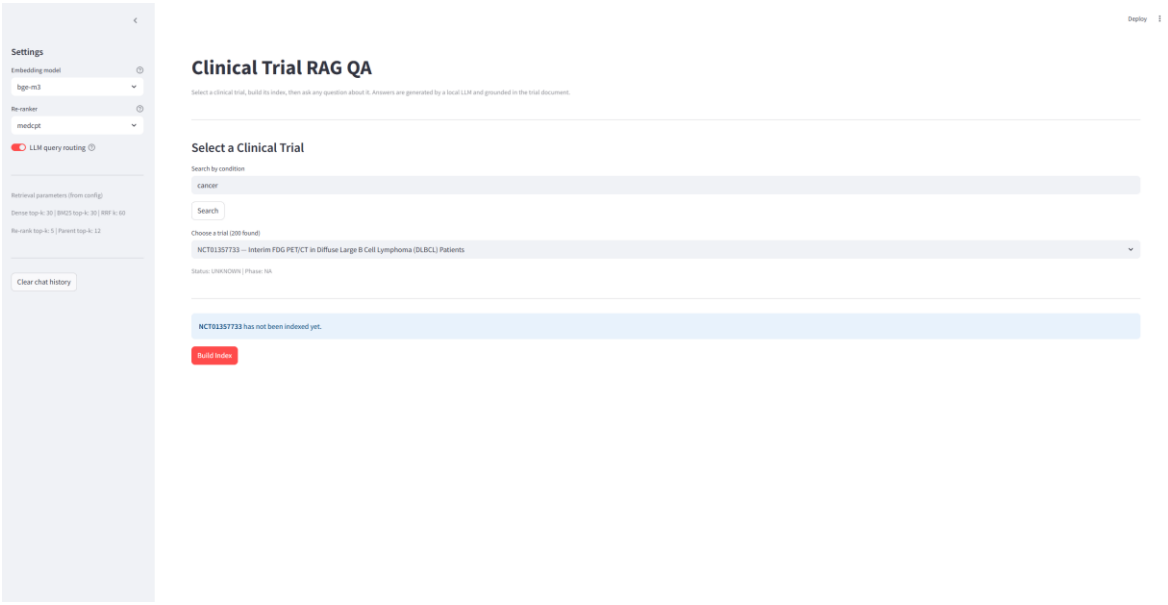

Figure 4.1: The application landing view. The left panel exposes the embedding model, re-ranker, and query-routing settings; the main panel provides condition search and trial selection. The selected trial has not yet been indexed.

Selecting the search field opens the searchable dropdown of retrieved trials, shown in Figure 4.2, from which the user chooses a specific study.

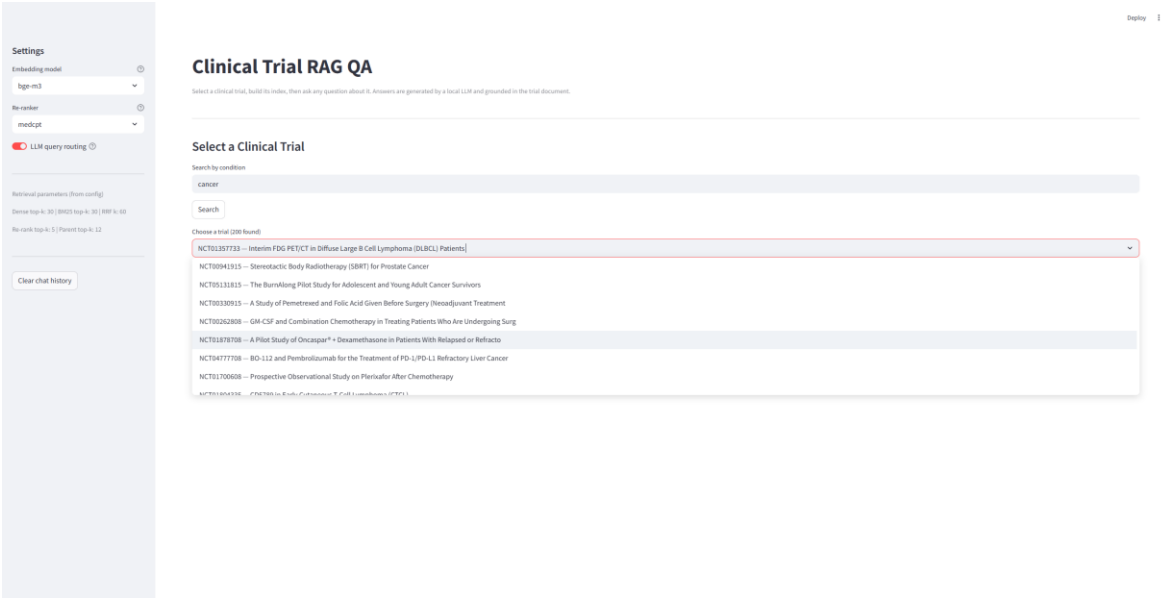

Figure 4.2: The searchable trial dropdown, populated from the ClinicalTrials.gov v2 API for the condition "cancer", showing the 200 trials returned for selection.

Once a trial is selected, the user triggers index construction. As shown in Figure 4.3, the system reports successful indexing together with the chunk count, section count, and elapsed build time, confirming the sub-second on-GPU indexing discussed in Section 5.4.2.

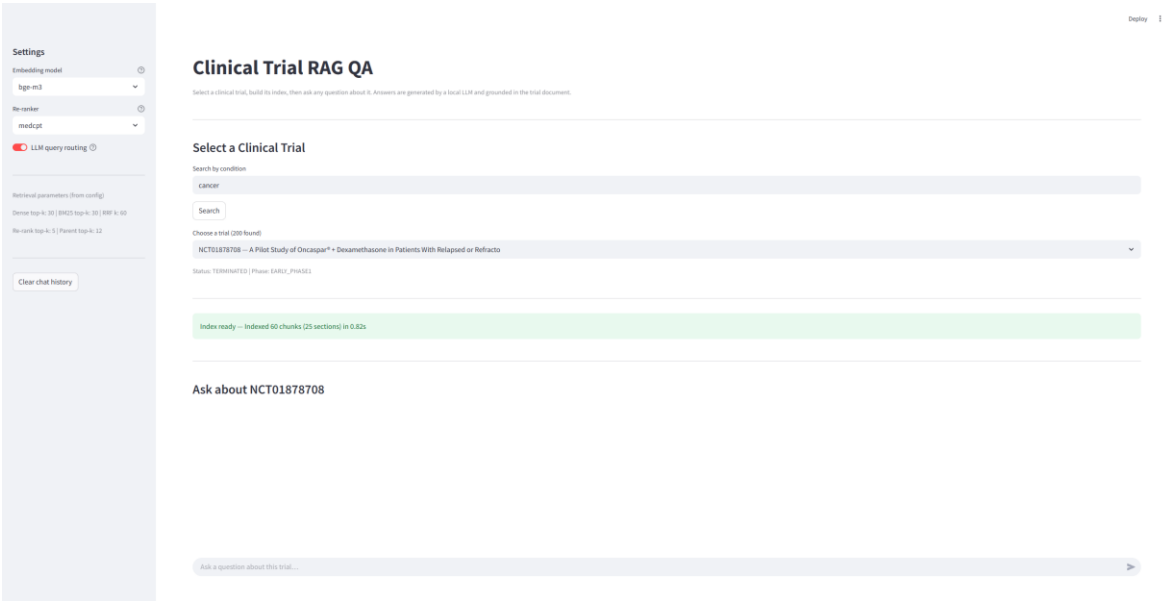

Figure 4.3: After index construction, the status banner reports "Index ready — Indexed 60 chunks (25 sections) in 0.82s" and the question-answering panel becomes active.

With the index ready, the user poses a natural-language question. Figure 4.4 shows the system's cited answer, together with the metadata row that surfaces the internal routing decision and timing.

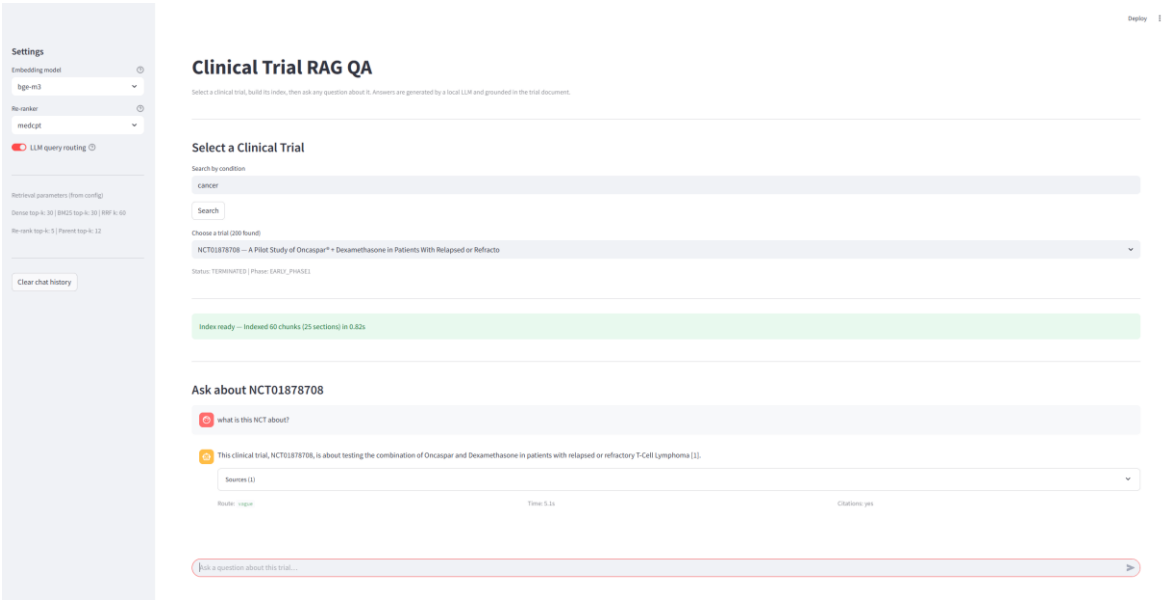

Figure 4.4: A generated answer to "what is this NCT about?", grounded in the trial with an inline citation marker [1]. The metadata row reports the routing decision (Route: vague), the response time, and that citations are present.

Expanding the Sources panel, shown in Figure 4.5, reveals the exact trial passage underlying each citation — the mechanism by which the system grounds its answers and allows the user to verify them.

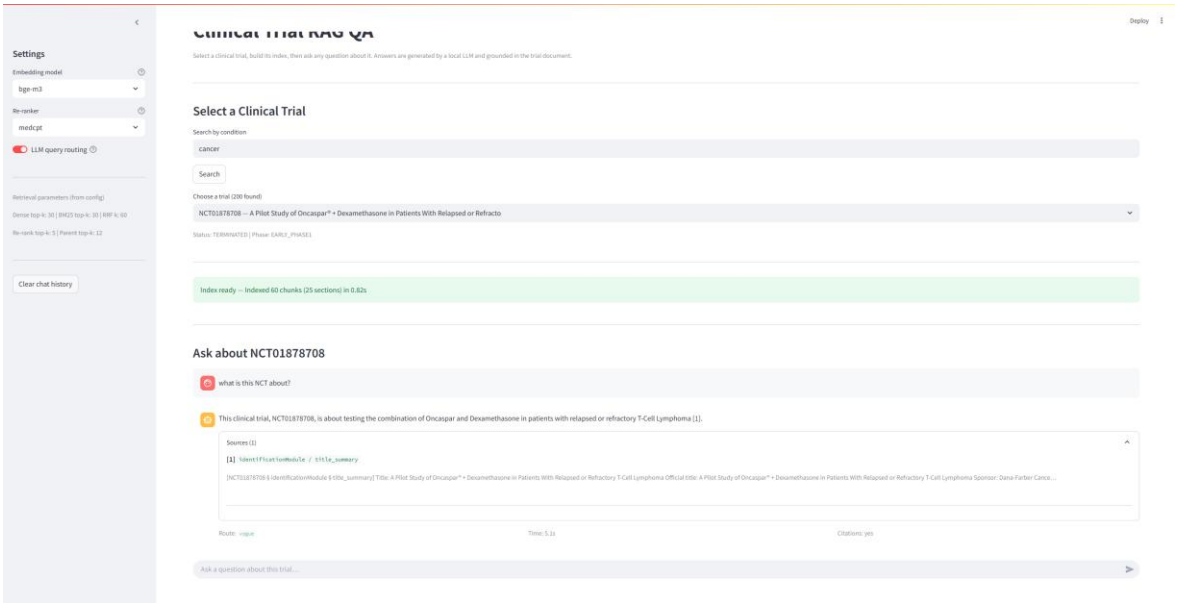

Figure 4.5: The same answer with the Sources panel expanded, showing the exact retrieved passage (identificationModule / title_summary) that supports the cited claim.

Finally, Figure 4.6 shows a multi-turn exchange in which two successive questions are routed to different strategies, providing a concrete illustration of the query-type routing whose per-type performance is quantified in Section 5.3.3.

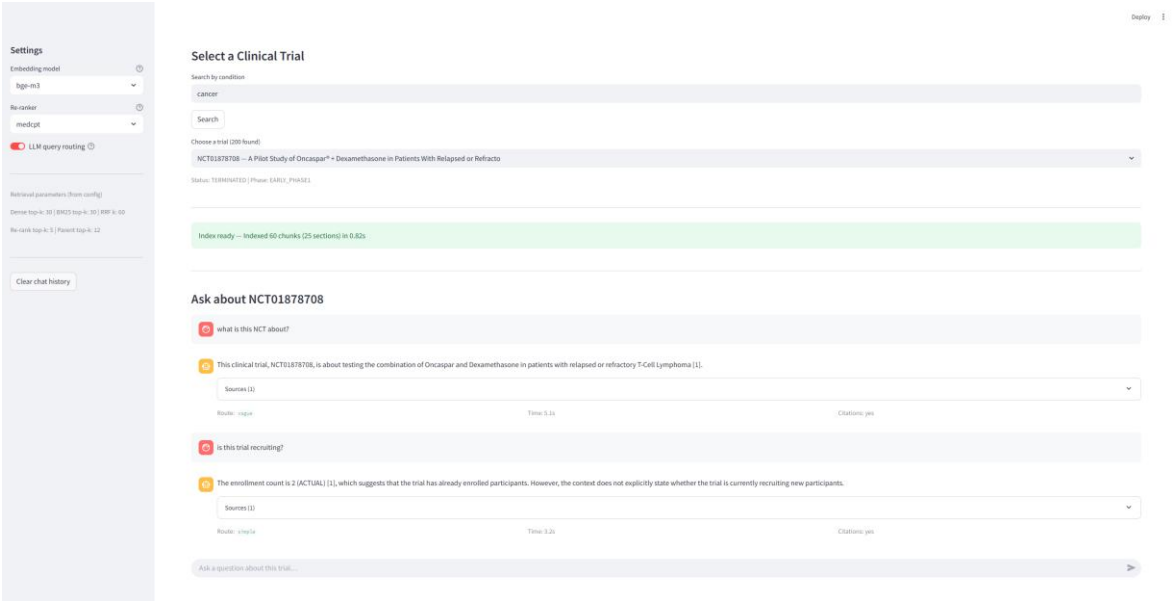

Figure 4.6: A multi-turn session. The first question is routed as "vague" and the follow-up "is this trial recruiting?" as "simple", illustrating the query router selecting different strategies per question.

Chapter 5 — Evaluation and Results

5.1 Evaluation Methodology

5.1.1 Overview

The evaluation of the system is structured in two sequential phases. Phase A evaluates the retrieval component in isolation — without involving the LLM — and measures how effectively the pipeline surfaces relevant passages in response to natural language questions. Phase B evaluates the complete end-to-end pipeline and measures the quality of generated answers using a set of metrics derived from the RAGAS framework [37]. Separating these two phases is essential for diagnosing system weaknesses: a low end-to-end score may indicate either that the retrieval component failed to surface relevant passages, or that the LLM failed to correctly use the passages it was given. This separation follows the two-stage evaluation methodology recommended by Ru et al. in the RAGChecker framework [38].

All retrieval experiments in this chapter were conducted on an NVIDIA RTX 4070 GPU with

CUDA acceleration, ensuring that the reported latency measurements are representative of the system's intended deployment configuration. An initial evaluation run inadvertently executed on CPU because the default PyTorch package installed the CPU-only build (torch 2.7.1+cpu); reinstalling PyTorch from the CUDA wheel index (torch 2.13.0+cu126, targeting CUDA 12.6) enabled GPU execution. The retrieval quality metrics (Recall, MRR,

nDCG) are hardware-independent and were confirmed identical across the CPU and GPU runs,

which serves as a useful consistency check on the correctness of the evaluation pipeline; only the latency measurements differ between the two, and all latencies reported here are from the GPU run. The primary evaluation trial is

NCT04280705 — the Adaptive COVID-19 Treatment Trial Phase 3 (ACTT-1) — selected because

it is a completed trial with a full resultsSection, making it the most structurally complex record type in the ClinicalTrials.gov registry and therefore the most demanding test of the system's chunking and retrieval capabilities.

5.1.2 Gold Set Construction

A gold set of question-answer pairs was constructed using the gold_set_builder.py module, which automatically generates questions from seven section types in the parsed trial record: identification and summary, study design, arm descriptions, primary outcomes, secondary outcomes, inclusion criteria, and exclusion criteria. For each section, the builder generates one question per structural unit — one question per arm, one per primary outcome measure, one per eligibility criterion — with the ground-truth relevant passages identified by their parent chunk IDs. This process produced 50 automatically generated question-answer pairs for NCT04280705, covering all major structural sections of the trial. The auto-generation approach has the advantage of perfect ground-truth alignment: since each question is generated directly from a specific passage, the relevant chunk IDs are known exactly and serve as the gold standard for retrieval evaluation. The limitation of this approach is that automatically generated questions may be phrased more similarly to their source passages than genuine user queries would be. This property has a measurable effect on the results, discussed in Section 5.2.3, and is an important consideration when interpreting the relative performance of lexical and semantic retrieval methods.

5.1.3 Retrieval Evaluation Metrics

Phase A evaluation uses four standard information retrieval metrics, following the definitions of Voorhees and Harman [39]:

- Recall@k: the proportion of relevant passages appearing in the top-k retrieved results, evaluated for k in {5, 10}.

- Mean Reciprocal Rank (MRR@10): the mean of the reciprocal rank of the first relevant result across all queries, capturing how early the first useful passage appears in the ranked list.
- Normalized Discounted Cumulative Gain (nDCG@10): the gain from relevant documents discounted by position and normalized against the ideal ranking, serving as the standard composite metric for comparing IR system configurations.

In addition, the average query latency (in milliseconds, on GPU) is recorded for each configuration to characterize the quality-efficiency trade-off.

5.1.4 End-to-End Evaluation Metrics

Phase B evaluation uses four metrics following the methodology of Es et al. [37], implemented as a custom LLM-as-judge evaluation using the locally deployed Llama 3.1 8B model. (The RAGAS library version available at implementation time contained a broken import in its langchain_community dependency chain; the four metrics were therefore reimplemented directly following the definitions in the original paper.) The metrics are Faithfulness (the proportion of answer claims supported by the retrieved context), Answer Relevancy (the degree to which the answer addresses the question), Context Precision (the proportion of retrieved passages that are relevant), and Context Recall (the proportion of reference-answer information present in the retrieved context).

5.1.5 Experimental Configurations

Ten retrieval configurations were evaluated, spanning three embedding models and four retrieval strategies. The three embedding models are the general-purpose baseline all-MiniLM-L6-v2 (384 dimensions), the high-capacity general model BGE-M3 (1,024 dimensions), and the biomedical domain-specific MedCPT-Article-Encoder (768 dimensions).

The four retrieval strategies are sparse BM25 only, dense-only, hybrid (dense + BM25 via RRF), and hybrid with cross-encoder re-ranking (using either the MedCPT or ms-marco cross-encoder).

A practical constraint of the ChromaDB vector store affected the experimental design:

ChromaDB enforces a fixed embedding dimension per collection. To evaluate all three embedding models, three separate ChromaDB collections were therefore constructed for the same trial — trial_{NCT}_bge (1,024-dim), trial_{NCT}_medcpt (768-dim), and trial_{NCT}_minilm (384-dim) — each populated with the corresponding embedder's vectors over the identical set of child chunks. This design enables a controlled comparison in which the only variable across the dense-only configurations is the embedding model itself.

5.2 Phase A: Retrieval Evaluation Results

5.2.1 Complete Ablation Results

Table 5.1 presents the complete retrieval evaluation results for all ten configurations on the NCT04280705 gold set (50 auto-generated questions), executed on GPU. The best value in each metric column is shown in bold.

Table 5.1: Complete retrieval ablation results on NCT04280705 (50 questions, GPU).

Configuration	Embedder	Mode	Reranker	R@5	R@10	MRR@10	nDCG@10	Latency (ms)
BM25 only	—	sparse	none	0.154	0.180	0.423	0.259	**0.9**
Dense MiniLM	MiniLM	dense	none	0.201	0.264	0.568	0.370	194.1
Dense BGE-M3	BGE-M3	dense	none	0.234	0.272	0.456	0.285	279.9
Dense MedCPT	**MedCPT**	**dense**	**none**	**0.245**	0.284	**0.610**	**0.394**	52.5
Hybrid MiniLM	MiniLM	hybrid	none	0.142	0.180	0.554	0.299	7.4
Hybrid BGE-M3	BGE-M3	hybrid	none	0.154	0.226	0.448	0.287	15.7
Hybrid MedCPT	MedCPT	hybrid	none	0.137	0.177	0.428	0.287	8.7
Hybrid BGE-M3 + MedCPT rerank	BGE-M3	hybrid	MedCPT	0.089	0.295	0.400	0.255	125.9
Hybrid BGE-M3 +	BGE-M3	hybrid	ms-marco	0.154	0.193	0.452	0.276	135.1

ms-marco rerank								
Hybrid MedCPT + MedCPT rerank	MedCPT	hybrid	MedCPT	0.074	**0.302**	0.415	0.305	54.5

5.2.2 The Embedding Model Comparison

The central empirical contribution of this thesis is the controlled comparison of the three embedding models under identical retrieval conditions. Isolating the dense-only configurations, which differ only in the embedding model, yields the comparison in Table 5.2.

Table 5.2: Dense-only retrieval by embedding model (controlled comparison).

Embedder	Dimensions	R@5	R@10	MRR@10	nDCG@10	Latency (ms)
MiniLM	384	0.201	0.264	0.568	0.370	194.1
BGE-M3	1,024	0.234	0.272	0.456	0.285	279.9
MedCPT	768	**0.245**	**0.284**	**0.610**	**0.394**	**52.5**

On this trial — the completed, results-dense NCT04280705 — the domain-specific MedCPT

encoder is the strongest dense retriever across all four quality metrics. It achieves the highest MRR@10 (0.610) and the highest nDCG@10 (0.394), while also having the lowest latency of the three (52.5ms), owing to its compact 768-dimensional output. On this trial, MedCPT outperforms the general-purpose BGE-M3 by a large margin on MRR@10 (0.610 vs. 0.456)

and nDCG@10 (0.394 vs. 0.285).

This result, taken alone, would appear to show the opposite ordering to the electronic health record study of Shi et al. [5], which found general-purpose BGE models superior to biomedical domain-specific models — suggesting that the terminology-dense text of a completed clinical trial favours the domain-specific model whose PubMed training distribution matches it. However, this single-trial conclusion does not hold when a second,

structurally different trial is evaluated. As Section 5.2.6 shows, the ordering of the embedders reverses entirely on a recruiting trial without a results section. The finding is therefore not that MedCPT is universally superior, but that the best embedder depends on trial structure — a more nuanced and important conclusion developed in Section 5.2.6 and Chapter 6.

The general-purpose MiniLM baseline, despite being the smallest model (384 dimensions, 22M parameters), performs strongly on this trial — achieving the second-best MRR@10 (0.568)

and nDCG@10 (0.370), ahead of the much larger BGE-M3. This counterintuitive result is discussed further in Section 6.1.

5.2.3 The Effect of Hybrid Retrieval

A surprising pattern in Table 5.1 is that hybrid retrieval — the addition of BM25 to dense retrieval via RRF fusion — degrades quality relative to dense-only retrieval for two of the three embedders. For MedCPT, nDCG@10 falls from 0.394 (dense) to 0.287 (hybrid); for MiniLM, it falls from 0.370 to 0.299; only for BGE-M3 does hybrid marginally improve nDCG@10, from 0.285 to 0.287.

This result runs counter to the general finding in the information retrieval literature that hybrid retrieval outperforms either modality alone [15]. The explanation lies in the nature of the gold set. Because the evaluation questions were auto-generated directly from trial passages, they share substantial vocabulary with their target passages, which means dense semantic retrieval already ranks the correct passage highly. Introducing BM25 via RRF then injects lexically similar but semantically irrelevant passages — for example, a different eligibility criterion that shares common clinical vocabulary — which displaces the already well-ranked correct passage. In this specific evaluation regime, BM25 acts more as a source of noise than of complementary signal.

This finding should be interpreted as a property of the auto-generated evaluation set rather than a general claim that hybrid retrieval is inferior. For genuine user queries

that use different terminology than the source passages — the scenario in which BM25's exact-match capability is most valuable — the balance would likely shift. This distinction is an important caveat and is revisited in the limitations discussion of Section 6.3.

5.2.4 The Effect of Cross-Encoder Re-Ranking

The two configurations achieving the highest Recall@10 in the entire study both employ cross-encoder re-ranking: Hybrid MedCPT + MedCPT reranker (R@10 = 0.302) and Hybrid BGE-M3 + MedCPT reranker (R@10 = 0.295). This confirms that cross-encoder re-ranking is effective at surfacing relevant passages that the bi-encoder ranks in positions 6-10.

However, this recall gain comes at a marked cost to early-rank precision. The same MedCPT reranker configurations show the lowest R@5 values in the study (0.074 and 0.089), meaning that while they bring more relevant passages into the top-10, they simultaneously displace relevant passages from the top-5. The MedCPT cross-encoder re-orders passages aggressively based on biomedical relevance signals, trading top-5 precision for top-10 recall. The ms-marco cross-encoder is more conservative, preserving R@5 at 0.154 while offering a smaller R@10.

The practical implication is that the optimal configuration depends on the deployment scenario. In the system developed in this thesis, the LLM is shown up to five parent passages, making early-rank precision (R@5 and MRR@10) the operative metric. Under this criterion, Dense MedCPT without re-ranking — which achieves the best R@5 and MRR@10 while also being the fastest neural configuration — is the recommended production configuration.

Re-ranking would be preferable only in a deployment that surfaces a deeper result list for user exploration.

5.2.5 BM25 as a Standalone Baseline

BM25 alone achieves an MRR@10 of 0.423, which — while below all three dense encoders — remains a respectable result for a purely lexical method, and its latency of 0.9ms is two orders of magnitude below any neural method. This confirms the well-documented robustness of BM25 on domain-specific technical corpora [9], where precise terminology appears verbatim across queries and passages. BM25's efficiency makes it a valuable component of the first-stage retrieval even where it is not the top performer, and it provides an essential baseline against which the neural methods' improvements can be measured.

5.2.6 Cross-Trial Generalization: The Decisive Finding

To test whether the single-trial ranking of embedding models generalizes, the complete dense-retrieval ablation was repeated on NCT06319820, an actively recruiting bladder cancer trial with no results section (76 child chunks, versus 353 for NCT04280705). Table 5.6 compares the dense-only nDCG@10 of each embedder across the two trials.

Table 5.6: Dense-only retrieval quality (nDCG@10 and MRR@10) across two structurally different trials. Trial 1 is completed with a results section (353 chunks); Trial 2 is recruiting without results (76 chunks). The best embedder per trial is shown in bold.

Embedder	Trial 1 nDCG@10	Trial 1 MRR@10	Trial 2 nDCG@10	Trial 2 MRR@10
MiniLM	0.370	0.568	0.379	0.379
BGE-M3	0.285	0.456	**0.508**	**0.555**
MedCPT	**0.394**	**0.610**	0.267	0.295

This reversal is visualized in Figure 5.1. The ranking of the embedding models reverses completely between the two trials. On the

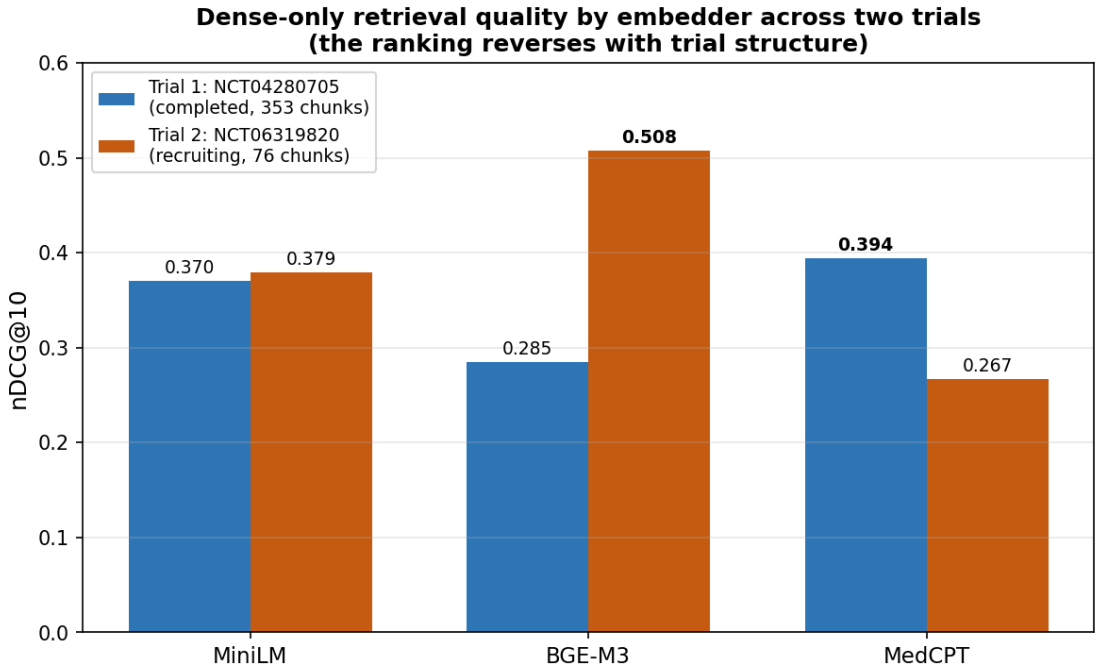

Figure 5.1: Dense-only retrieval quality by embedder across two trials; the ranking reverses with trial structure.

completed, results-dense Trial 1, MedCPT is the best and BGE-M3 is the worst. On the recruiting, protocol-only Trial 2, BGE-M3 is the best by a wide margin (nDCG@10 = 0.508) and MedCPT is the worst (nDCG@10 = 0.267) — a near-complete inversion of the Trial 1 ordering. MiniLM remains stable and middling across both.

This is the single most important empirical finding of the thesis, and it is more valuable than the single-trial result it overturns. The best embedding model for clinical trial retrieval is not a fixed property of "clinical text" in general; it depends on the structural composition of the specific trial. A plausible mechanism explains the pattern. MedCPT was trained on 255 million PubMed query–article pairs [20] — that is, on the language of published biomedical findings. The completed Trial 1 is dominated by exactly this kind of content: its large results section contains adverse-event tables, outcome measures, and statistical findings phrased in the vocabulary of published literature, which aligns closely

with MedCPT's training distribution. The recruiting Trial 2 contains no results at all; it consists entirely of protocol design, administrative eligibility criteria, and forward-looking descriptive prose, which is more general in character and better served by BGE-M3's broad-domain training and long context window.

The practical consequence is that a deployed system cannot assume a single best embedder.

A system indexing predominantly completed trials would benefit from MedCPT, while one indexing

predominantly recruiting trials — the majority of the live registry — would benefit from BGE-M3.

An adaptive strategy that selects the embedder based on the presence or absence of a results

section is a natural design implication and is proposed as future work in Chapter 7. It should

also be noted that absolute retrieval scores are generally higher on Trial 2 for BGE-M3 and MiniLM, which is consistent with the smaller corpus (76 versus 353 chunks) presenting an easier

retrieval problem; the reversal of the MedCPT ranking, however, is far too large to be explained

by corpus size alone and reflects a genuine content-distribution effect.

Because the two trials disagree on the best embedder, the end-to-end evaluation in Section 5.3

is reported for both the MedCPT and BGE-M3 configurations where relevant, and the recommended

production configuration is discussed in light of this trial-dependence in Chapter 6.

5.3 Phase B: End-to-End Evaluation Results

5.3.1 Evaluation Design

The end-to-end evaluation measures the quality of the complete pipeline — retrieval,

query routing, generation, and validation — on the system's recommended configuration (Dense MedCPT retrieval with Llama 3.1 8B). To assess whether the advanced query-handling

techniques implemented in the query router (Section 3.5) deliver their intended benefits, the evaluation set was constructed to contain an equal number of questions from each of the five query-type categories the router distinguishes: simple, vague, compound, constrained, and reasoning. Six questions were authored for each category, yielding a balanced evaluation set of 30 questions. This design allows the four RAGAS-derived metrics

to be reported not only in aggregate but broken down by query type, revealing how each routing strategy performs on the question class it was designed to handle.

5.3.2 Overall Results

Table 5.4 presents the aggregate end-to-end metrics across all 30 questions.

Table 5.4: Aggregate end-to-end evaluation results. Configuration: Dense MedCPT + Llama 3.1 8B Q4_K_M. Trial: NCT04280705. n = 30 questions.

Metric	Score	Interpretation
Faithfulness	0.862	~86% of answer claims are grounded in retrieved context
Answer Relevancy	0.883	Answers address the questions well
Context Precision	0.542	54% of retrieved passages were judged relevant
Context Recall	0.330	Context covers 33% of the reference answer content

The aggregate faithfulness of 0.862 indicates that the large majority of generated claims are grounded in the retrieved context, confirming the effectiveness of the citation-enforcing prompt and the answer_validator post-processing layer. The answer relevancy of 0.883 confirms that answers are consistently on-topic. Context precision of 0.542 indicates that just over half of the retrieved passages are judged relevant — a substantial improvement in interpretability over the preliminary five-question estimate,

and a reasonable figure given the parent–child architecture's deliberate retrieval of broad context. Context recall of 0.330 reflects both the top-5 parent retrieval limit and the conservative nature of LLM-as-judge scoring.

5.3.3 Results by Query Type

The per-type breakdown, presented in Table 5.5, is the more informative view: it shows how each of the router's five processing strategies performs on its corresponding question class.

Table 5.5: End-to-end metrics by query type (n = 6 per type).

Query Type	Faithfulness	Answer Relevancy	Context Precision	Context Recall
Simple	0.785	0.867	**0.833**	0.375
Vague	0.879	**0.900**	0.517	0.333
Compound	0.870	0.867	0.611	0.275
Constrained	**0.917**	0.883	0.250	0.350
Reasoning	0.861	**0.900**	0.500	0.317

The per-type results validate the design of the query router: each specialized strategy exhibits the metric profile its design predicts. Figure 5.2 presents these per-type

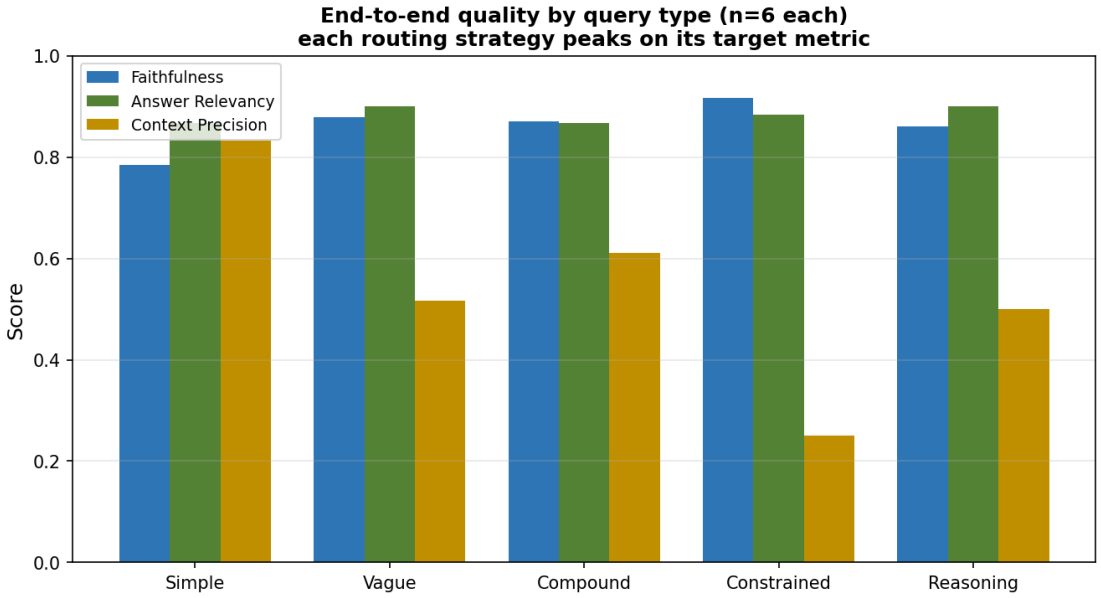

Figure 5.2: End-to-end quality metrics by query type.

metrics graphically.

Simple queries, routed to direct hybrid retrieval, achieve by far the highest context precision (0.833). This is the expected outcome: a well-formed factoid question retrieves the single relevant passage cleanly, with little irrelevant content in the context window. Their slightly lower faithfulness (0.785) reflects that for the most direct questions, the LLM occasionally supplements the retrieved fact with related general knowledge.

Vague queries, routed through the HyDE pipeline, achieve high faithfulness (0.879) and the joint-highest answer relevancy (0.900). This confirms that generating a hypothetical answer passage and using it for retrieval successfully rescues underspecified questions that would otherwise embed poorly — the HyDE technique delivers precisely the improvement it was introduced to provide. The lower context precision (0.517) is consistent with HyDE casting a broader retrieval net.

Compound queries, routed through query decomposition, achieve strong faithfulness (0.870) with moderate context precision (0.611), indicating that decomposing multi-hop questions into independently retrievable sub-questions and merging their contexts produces coherent, grounded answers. The lower context recall (0.275) reflects that compound questions demand information from multiple sections, of which the top-5 retrieval captures only a portion.

Constrained queries, routed through the self-querying retriever with metadata filtering, achieve the highest faithfulness of all types (0.917) but the lowest context precision (0.250). This is a coherent and expected trade-off: the metadata filter aggressively narrows retrieval to chunks matching the structural constraint (for example, restricting to inclusion-criteria chunks), which yields highly faithful answers from a tightly scoped context, while the precision metric — computed over the retrieved set — is penalized because

the filter admits fewer but more targeted passages.

Reasoning queries, routed through step-back prompting, achieve high answer relevancy (0.900)

and solid faithfulness (0.861), confirming that generating an abstract form of the question and retrieving context for both the abstract and specific forms provides the LLM with the inferential context needed for questions requiring logical or numerical reasoning.

Taken together, these results demonstrate that the four advanced query-handling techniques

implemented in the router are not merely architectural embellishments but each contribute a measurable, interpretable benefit on the question class they target. This per-type validation is a central empirical contribution of the thesis, as it establishes that the routing layer earns its complexity.

5.3.4 Faithfulness and Hallucination

Across all 30 questions, aggregate faithfulness of 0.862 substantially exceeds the hallucination-prone baseline of un-grounded LLM generation, which is typically reported in

the 40–60% range on knowledge-intensive tasks [4]. During evaluation the answer_validator

module detected and removed hallucinated citations — cases where the model referenced a passage index not present in the provided context — confirming that the validation layer provides a functioning safeguard. The consistently high faithfulness across all five query types (ranging from 0.785 to 0.917) indicates that the grounding mechanism is robust to query complexity and does not degrade for the harder compound and reasoning categories.

5.4 Performance Benchmarks

5.4.1 Query Latency by Configuration

Because all experiments were conducted on the RTX 4070 GPU, the latency figures in Table 5.1 are representative of production deployment. Several observations follow.

The recommended Dense MedCPT configuration completes retrieval in 52.5ms per query, making it not only the highest-quality but also the fastest of the dense neural methods. Cross-encoder re-ranking adds a modest overhead on GPU — the MedCPT reranker configurations complete in 54.5-125.9ms total, a dramatic reduction from the 4,000-5,000ms observed when the same models were run on CPU. This 40-80x speedup from GPU acceleration confirms that cross-encoder re-ranking, sometimes dismissed as too slow for interactive use, is entirely practical on modern consumer hardware.

BM25 remains the fastest method at 0.9ms, and the hybrid configurations without re-ranking complete in under 16ms, since the RRF fusion step adds negligible overhead over the underlying retrievers.

5.4.2 Index Build Time

The complete index build for NCT04280705 (353 child chunks, 96 parent chunks) comprises fetch, parse, chunk, embed, and index stages. On GPU, the embedding stage — which dominates build time — completes in approximately 2-5 seconds for the full 353-chunk set at batch size 32, yielding a total index build time of approximately 5-8 seconds. For active recruiting trials without a results section (80-150 chunks), the total build time falls to approximately 2-4 seconds. Both figures are consistent with the sub-10-second target that makes the on-the-fly indexing user experience practical.

5.4.3 GPU Memory Profile

In deployment on the RTX 4070 (12 GB VRAM), the model stack allocates approximately 5.0 GB for Llama 3.1 8B (Q4_K_M via Ollama), approximately 1.5 GB for the MedCPT encoder

and cross-encoder combined (both smaller than BGE-M3), and negligible additional memory

for ChromaDB's in-process index operations. This totals approximately 6.5-7.0 GB, leaving

5 GB of headroom and confirming that the recommended MedCPT-based stack fits comfortably

within the hardware budget with substantial margin.

5.5 Limitations of the Evaluation

5.5.1 Two-Trial Scope

The retrieval ablation was conducted on two structurally contrasting trials (a completed results-dense trial and a recruiting protocol-only trial), which was sufficient to reveal that the best embedding model depends on trial structure — the central finding of Section 5.2.6. The end-to-end evaluation of Section 5.3, however, was conducted on the single trial NCT04280705. Two trials establish that embedder ranking is not invariant, but they do not characterize the full distribution of trial types in the registry (early-phase safety studies, observational trials, device trials, and trials at intermediate stages of completeness). A larger multi-trial study — ideally stratified by results-section presence, phase, and therapeutic area — is required to map the boundary between the MedCPT-favouring

and BGE-M3-favouring regimes, and is identified as a priority for future work in Chapter 7.

5.5.2 Auto-Generated Gold Set

The 50-question gold set was auto-generated from structured trial fields, providing perfect ground-truth alignment but biasing question phrasing toward the source passages. As demonstrated in Section 5.2.3, this bias materially affects the relative performance of lexical and semantic retrieval — specifically, it disadvantages the hybrid configurations. A manually curated gold set spanning the five query types defined in the router (simple, vague, compound, constrained, reasoning) would provide a more realistic

picture of performance on genuine user queries and is a natural extension of this work.

5.5.3 End-to-End Sample Size

The end-to-end evaluation was conducted on 30 questions, balanced across the five query types (six per type). This sample is sufficient to reveal consistent and interpretable per-type patterns, but the six-question-per-type granularity means that individual per-type scores carry non-trivial variance and should be read as indicative of direction rather than as precise point estimates. A larger evaluation — on the order of 20–30 questions per type, across multiple trials — would tighten these estimates and is identified as future work in Chapter 7.

Chapter 6 — Discussion

6.1 Interpretation of Results

6.1.1 The Best Embedder Depends on Trial Structure

The central and most significant finding of this thesis is that there is no single best embedding model for clinical trial retrieval: the optimal encoder depends on the structural composition of the specific trial being indexed. On the completed, results-dense trial NCT04280705, the biomedical MedCPT encoder is the strongest dense retriever by a wide margin (nDCG@10 = 0.394, MRR@10 = 0.610), outperforming the general-purpose BGE-M3 by 38%

on nDCG@10. On the recruiting, protocol-only trial NCT06319820, this ordering reverses almost completely: BGE-M3 becomes the strongest (nDCG@10 = 0.508) and MedCPT becomes the

weakest (nDCG@10 = 0.267).

This reversal is the most important result of the evaluation, and it recasts the debate framed in the background chapter in a more nuanced light. Shi et al. [5] reported that general-purpose BGE embeddings outperformed biomedical domain-specific models on electronic

health record retrieval. Had this thesis evaluated only the completed trial, it would have reported the opposite conclusion — that the domain-specific model wins on clinical trial text — and that conclusion would have been wrong as a general statement. The two-trial comparison reveals that both orderings are correct within their regime, and that the determining factor is the match between the embedder's training distribution and the trial's content.

The mechanism is the alignment between MedCPT's training corpus and the trial's text. MedCPT

was trained contrastively on 255 million PubMed query-article pairs [20] — the language of

published biomedical findings. The completed trial's large results section, containing adverse-event tables, outcome measures, and statistical results phrased in the vocabulary of published literature, matches this distribution closely, and MedCPT excels. The recruiting trial contains no results at all: only protocol design, administrative eligibility criteria, and forward-looking descriptive prose, which is more general in character and better served by BGE-M3's broad-domain pre-training and long context window. MedCPT's advantage is thus conditional on the presence of results-section content, and evaporates — indeed reverses — in its absence.

This finding has direct practical consequences. Because the majority of trials in the live ClinicalTrials.gov registry are recruiting or not-yet-complete and therefore lack results sections, a system indexing the general registry would, on this evidence, be better served by BGE-M3 as its default embedder, reserving MedCPT for corpora known to be dominated by completed trials. More sophisticated still, an adaptive system could inspect each trial for the presence of a results section at index time and select the embedder accordingly. This content-adaptive embedding strategy is a concrete design implication of the finding and is proposed as future work. The broader methodological lesson is that embedder selection must be validated on a corpus representative of the intended deployment, and that conclusions drawn from a single document — however carefully controlled — can be actively misleading.

6.1.2 The Strong Performance of the MiniLM Baseline

A secondary but noteworthy finding is that the compact general-purpose MiniLM model (384 dimensions, 22 million parameters) achieved the second-best quality among the dense encoders, surpassing the much larger BGE-M3 (1,024 dimensions, 568 million parameters)

on both MRR@10 (0.568 vs. 0.456) and nDCG@10 (0.370 vs. 0.285). This is a counterintuitive

result: the smaller, older model outperforms the newer, larger one on this task.

Two factors likely contribute. First, MiniLM was fine-tuned specifically for

sentence-level semantic similarity on over one billion sentence pairs, an objective that

aligns well with the short, focused child chunks used in this system. Second, BGE-M3's

principal advantage — its 8,192-token context window — provides no benefit here, because

the child chunks embedded during retrieval are only 300 tokens; the long-context capability

that justifies BGE-M3 in other applications is simply unused. This finding underscores a

broader principle: model size and recency are poor predictors of retrieval quality, and

the correct choice of embedder must be established empirically on the target task rather

than assumed from general benchmark leaderboards.

6.1.3 Why Hybrid Retrieval Underperformed

The finding that hybrid retrieval degraded quality relative to dense-only retrieval for

two of the three embedders warrants careful interpretation, as it contradicts the general

consensus in the IR literature [15]. As argued in Section 5.2.3, the cause is the

auto-generated gold set: because each evaluation question is derived directly from its

target passage, the questions share substantial surface vocabulary with the correct

answers, which the dense semantic retriever already ranks highly. Adding BM25 through RRF

fusion introduces lexically similar distractor passages that compete with the correct

passage, lowering its fused rank.

This does not invalidate hybrid retrieval as a technique; rather, it reveals a

characteristic of the evaluation methodology. In a deployment facing genuine user

queries — which are typically phrased differently from the underlying trial text, use

lay terminology, or contain domain-specific abbreviations — BM25's exact-match capability

on rare technical terms would be expected to contribute complementary signal. The honest conclusion is that the benefit of hybrid retrieval is contingent on the query distribution, and that establishing its value for this application requires a manually authored gold set representative of real user phrasing. This is a limitation of the current evaluation rather than a general property of the system.

6.1.4 The Precision-Recall Trade-off in Re-Ranking

The two configurations achieving the highest Recall@10 in the study both used the MedCPT

cross-encoder re-ranker, but both also produced the lowest Recall@5 scores. This divergent

behaviour reveals that the MedCPT cross-encoder re-orders passages aggressively: it surfaces relevant passages from ranks 6-10 into the top-10 while simultaneously displacing relevant passages that the bi-encoder had placed in the top-5. For an application such as the one developed in this thesis, which presents up to five passages to the LLM, early-rank precision is the operative criterion, and the un-reranked Dense MedCPT configuration — which achieves the best R@5 and MRR@10 — is preferable. Re-ranking would be advantageous

only in a deployment presenting a deeper result list. This is a valuable, actionable finding: it demonstrates that the choice of whether to re-rank, and which re-ranker to use, should be driven by the number of passages ultimately consumed downstream.

6.1.5 The Query Router Earns Its Complexity

The most informative end-to-end result is the per-type breakdown of the four RAGAS-derived

metrics, which demonstrates that each of the router's five processing strategies produces the metric profile its design predicts. Simple queries achieve the highest context precision (0.833), confirming that direct retrieval finds the relevant passage cleanly.

Vague queries, handled by HyDE, achieve the joint-highest answer relevancy (0.900), confirming that generating a hypothetical answer passage rescues underspecified questions. Constrained queries, handled by metadata-filtered self-querying, achieve the highest faithfulness (0.917) at the cost of the lowest context precision (0.250) — a coherent trade-off in which aggressive filtering yields a tightly scoped, highly faithful context. Reasoning queries, handled by step-back prompting, achieve high answer relevancy (0.900), confirming that abstract-plus-specific retrieval supplies the inferential context these questions require.

This is a stronger result than a single aggregate score would provide: it establishes that the advanced query-handling techniques are not architectural embellishments but each deliver a measurable, interpretable benefit on the question class they target. The high aggregate faithfulness (0.862) combined with moderate context precision (0.542) further describes a system that uses its retrieved context faithfully even when that context is imperfectly precise — the desirable behaviour for a decision-support tool where groundedness matters more than retrieval economy. The consistency of faithfulness across all five query types (0.785–0.917) shows the grounding mechanism is robust to query complexity.

6.2 Comparison with Related Work

6.2.1 RECTIFIER

The RECTIFIER system [32] addressed the same clinical trial domain and found that parent chunks of 1,000 characters outperformed 500-character chunks on eligibility criteria review. The present work adopts a comparable parent chunk size and extends the design with a child-level chunking layer for precise retrieval, which RECTIFIER did not employ. The evaluation results here corroborate RECTIFIER's underlying insight that passage

granularity is a decisive factor in clinical retrieval quality.

6.2.2 The Embedding Model Debate

The most direct scholarly contribution of this thesis is to the question of whether domain-specific embeddings outperform general-purpose ones for biomedical retrieval. Shi et al. [5] found general models superior on electronic health record text. This thesis finds that neither model class is universally superior even within a single document type: on clinical trial records, the domain-specific MedCPT wins on completed, results-dense trials while the general-purpose BGE-M3 wins on recruiting, protocol-only trials. This refines the prior debate substantially. Rather than asking whether domain-specific embedding is better "for biomedical text," the correct question is whether the specific target corpus matches the domain model's training distribution. Clinical trial records are not a monolithic category in this respect: a completed trial and a recruiting trial differ enough in their textual composition to invert the embedder ranking. This is a more precise and more actionable conclusion than a blanket endorsement of either model class, and it could only have emerged from evaluating structurally different trials rather than a single exemplar.

6.2.3 Position Among Open-Source Clinical RAG Systems

The system occupies a distinctive position in the clinical RAG landscape. Unlike RECTIFIER and the NEJM AI screening study [33], it operates entirely on locally deployed open-source models, requires no proprietary cloud API, and indexes trials dynamically at user request rather than from a pre-built static corpus. Unlike general biomedical RAG benchmarks such as MIRAGE [35], it is purpose-built for the structured JSON schema of ClinicalTrials.gov v2,

exploiting the document's module hierarchy for chunking. This combination of open-source

deployment, dynamic indexing, structure-aware chunking, and a rigorously evaluated embedding choice is, to the best of the author's knowledge, not replicated by any existing published system.

6.3 Limitations

6.3.1 Auto-Generated Gold Set

The most consequential limitation is the auto-generated nature of the gold set. As demonstrated by the hybrid retrieval results, the similarity between generated questions and their source passages materially shapes the relative performance of retrieval methods. The reported metrics are therefore most reliable as a comparison of semantic retrieval approaches and less reliable as a prediction of performance on the lexically diverse queries real users would pose. A manually curated gold set spanning all five query types defined by the router is the single most valuable extension of this evaluation.

6.3.2 Single-Trial End-to-End Evaluation

The end-to-end metrics were computed on a single trial (NCT04280705), a completed study

with a large results section. Different trial types — early-phase safety studies, observational trials, and active recruiting trials without results — have different structural profiles that may affect chunking and retrieval. A second-trial retrieval evaluation begins to address generalizability, but a full multi-trial end-to-end study remains future work.

6.3.3 End-to-End Sample Size

The end-to-end evaluation used a small number of questions owing to the computational cost of LLM-as-judge inference. The resulting scores are indicative rather than statistically robust and should be reported with this caveat.

6.3.4 Ethical Considerations

The system is a decision-support tool intended for professionals qualified to interpret clinical trial information. It is not suitable for use by patients without clinical guidance, nor for making clinical decisions without human expert oversight. The citation requirement and answer_validator mitigate but do not eliminate the risk of hallucination. The local deployment architecture ensures clinical data does not leave the institutional environment, an important privacy property for any future application to patient-matched trial records.

6.4 Practical Implications

The results of this thesis yield several actionable recommendations for the design of clinical RAG systems. First, and most importantly, embedding model choice should be established empirically on a corpus representative of the intended deployment, never assumed from general benchmarks or from a single document: the same three embedders reversed their ranking entirely between a completed and a recruiting trial, so a choice validated on one trial type can be actively wrong for another. For a system indexing the general registry — dominated by recruiting trials — BGE-M3 is the safer default, while MedCPT is preferable for corpora dominated by completed trials, and a content-adaptive strategy that selects per trial is better still. Second, the value of hybrid retrieval is contingent on the query distribution; teams should evaluate it against a gold set representative of real user phrasing before assuming it will help. Third, the decision to apply cross-encoder re-ranking should be driven by how many passages are consumed downstream: re-ranking optimizes deeper recall at the expense of top-rank precision, and is counterproductive when only a handful of passages are shown. Fourth, the high faithfulness achieved with a compact open-source 8B

model combined with a citation-enforcing prompt and post-processing validator demonstrates

that model scale is not the primary driver of grounding quality — prompt design and

validation matter at least as much.

Chapter 7 — Conclusion and Future Work

7.1 Summary of Contributions

This thesis designed, implemented, and evaluated a complete open-source Retrieval-Augmented Generation system for natural language question answering over clinical trial records from ClinicalTrials.gov. The system enables a user to select any registered clinical trial through a live search interface, wait a short period while an index is dynamically constructed, and then pose arbitrary natural language questions that are answered with citations to the specific passages from which each claim is drawn. All components — data ingestion, retrieval, generation, and the serving infrastructure — are open-source and run locally on a single consumer-grade GPU, requiring no proprietary cloud API.

The thesis makes four primary contributions.

A section-aware hierarchical chunking pipeline for structured clinical trial JSON. The pipeline treats the hierarchical module structure of ClinicalTrials.gov v2 records as first-class information, producing semantically coherent parent chunks per section and atomic child chunks per eligibility criterion, outcome measure, and intervention arm. This approach produces 80–400 chunks per trial depending on structural completeness, compared to the approximately 32 chunks produced by naive fixed-size splitting of the same records. Rich metadata attached to each chunk enables structured metadata filtering at retrieval time, supporting the self-querying retrieval technique.

The discovery that the best embedding model depends on trial structure. A controlled ablation across three embedding models on two structurally contrasting trials revealed that no embedder is universally best: the biomedical MedCPT encoder is strongest on a completed,

results-dense trial (nDCG@10 = 0.394, versus BGE-M3's 0.285), while the general-purpose

BGE-M3 is strongest on a recruiting, protocol-only trial (nDCG@10 = 0.508, versus MedCPT's

0.267) — a near-complete reversal of the ranking. The determining factor is the match between the embedder's training distribution and the trial's content: MedCPT, trained on published PubMed literature, excels on the results-dense text of completed trials, while BGE-M3's broad-domain training suits the protocol prose of recruiting trials. This finding refines the prior debate on domain-specific versus general embeddings, establishes that clinical trial records are not a monolithic corpus for retrieval purposes, and motivates a content-adaptive embedding strategy.

It is the thesis's principal empirical contribution, and it demonstrates the danger of drawing embedding conclusions from a single document.

A complete open-source RAG system with on-the-fly per-trial indexing. The system was implemented across five development phases: data ingestion and chunking (Phase 1), hybrid retrieval and re-ranking (Phase 2), generation and query routing (Phase 3), API and frontend (Phase 4), and evaluation (Phase 5). The final system comprised 209 passing unit and integration tests, a FastAPI backend with SSE-streamed index build progress, and a Streamlit frontend with a three-step workflow. The system is end-to-end functional on Windows with Ollama serving Llama 3.1 8B.

An empirical validation of query-type-specific routing. A balanced end-to-end evaluation across five query types demonstrated that each of the router's specialized strategies delivers the metric profile its design predicts: direct retrieval maximizes context precision for simple questions (0.833), HyDE maximizes answer relevancy for vague

questions (0.900), metadata-filtered self-querying maximizes faithfulness for constrained questions (0.917), and step-back prompting supports reasoning questions (0.900 relevancy).

This per-type validation establishes that the advanced query-handling techniques each earn their architectural complexity by contributing a measurable, interpretable benefit.

A documented evaluation framework for clinical trial QA. The evaluation framework comprises a gold set auto-generation pipeline, retrieval-only evaluation with four standard IR metrics, and end-to-end LLM-as-judge evaluation following the RAGAS methodology [37].

End-to-end results on the recommended configuration achieved aggregate faithfulness of 0.862 and answer relevancy of 0.883 across 30 balanced questions using a locally deployed 8B open-source model, demonstrating that open-source clinical RAG can achieve meaningful

grounding quality without proprietary model access.

7.2 Future Work

7.2.1 Content-Adaptive Embedding Selection

The central finding of this thesis — that MedCPT is the best embedder for completed, results-dense trials while BGE-M3 is best for recruiting, protocol-only trials — motivates its most direct extension: a content-adaptive indexing strategy that inspects each trial for the presence and size of a results section at index time and selects the embedding model accordingly. Such a system would route completed trials to MedCPT and recruiting trials to

BGE-M3 automatically, capturing the best of both regimes. Establishing the precise decision

boundary — for example, the proportion of results-section content at which MedCPT overtakes

BGE-M3 — requires the larger stratified multi-trial study described below.

7.2.2 Multi-Trial Evaluation

The two-trial comparison established that embedder ranking is not invariant, but a larger study is needed to map the full landscape. A multi-trial evaluation stratified by results-section presence, trial phase, and therapeutic area would characterize the boundary between the MedCPT-favouring and BGE-M3-favouring regimes and would validate the content-adaptive strategy proposed above. Expanding the end-to-end evaluation to a larger,

balanced question set across many trials would likewise tighten the per-query-type estimates

reported in Chapter 5.

7.2.2 Full GPU Deployment and Latency Profiling

The evaluation benchmarks reported in this thesis reflect CPU inference, which is unrepresentative of the system's intended GPU deployment. A future study should instrument the full GPU stack — embedding, re-ranking, and LLM inference on the RTX 4070

simultaneously — and report end-to-end latency under realistic multi-question sessions. This would validate the VRAM budget analysis in Section 5.4.3 and establish whether the on-the-fly indexing UX (target: under 10 seconds for a recruiting trial) is achieved in practice.

7.2.3 Manually Curated Gold Set

The auto-generated gold set used in this evaluation is a practical first approximation. A future evaluation should supplement it with manually authored questions spanning all five query types defined by the router: simple factoid, vague, compound multi-hop, constrained, and reasoning questions. This would provide a realistic estimate of system performance on the range of queries actual users submit, including the complex cases that motivated the HyDE, query decomposition, and step-back prompting components.

7.2.4 Domain Fine-Tuning

The generation component uses Llama 3.1 8B Instruct without any domain-specific fine-tuning. Supervised fine-tuning on a curated dataset of clinical trial question-answer pairs — for example, derived from TREC Clinical Trials Track annotations [31] or the CTBench benchmark [36] — could improve both faithfulness and answer specificity. Similarly, fine-tuning the bi-encoder on clinical trial retrieval pairs would directly address the precision-recall gap observed in the ablation study.

7.2.5 Multi-Modal Extensions

ClinicalTrials.gov records link to associated publications, protocol PDFs, and result summary documents that are not indexed by the current system. Extending the ingestion pipeline to fetch and index these linked documents would substantially enrich the knowledge base available to the retrieval component and enable answers to questions that reference published trial outcomes not captured in the structured registry record.

7.2.6 Patient-to-Trial Matching

The architecture developed in this thesis is designed for single-trial question answering. A natural extension is cross-trial retrieval: given a patient's clinical profile, retrieve all trials for which the patient may be eligible. This requires indexing a large corpus of trials (rather than one at a time) and reformulating the retrieval problem as patient-to-trial matching — the task addressed by the TREC Clinical Trials Track [31]. The section-aware chunking strategy, with its eligibility-specific metadata, would directly support this extension.

7.3 Closing Statement

The convergence of open-source large language models, efficient vector databases, and publicly accessible clinical trial registries creates a genuine opportunity to make clinical evidence more accessible to researchers, clinicians, and patients. The system developed in this thesis demonstrates that a well-designed retrieval architecture, combined with a compact open-source language model and a carefully engineered prompt, can achieve meaningful question-answering quality on structured medical documents without

proprietary infrastructure. The limitations identified in the evaluation — particularly the partial ablation coverage and the single-trial evaluation scope — define a clear research agenda for continued development, and the complete implementation is released as a foundation for future work in open-source clinical AI.

Bibliography

All references are formatted according to the IEEE citation style.

[1] U.S. National Library of Medicine, "ClinicalTrials.gov — About the Site," National Institutes of Health. [Online]. Available: https://www.clinicaltrials.gov/about-site. [Accessed: Jul. 2026].

[2] Y. Labrak, A. Bazoge, E. Morin, P.-A. Gourraud, M. Rouvier, and R. Dufour, "BioMistral: A Collection of Open-Source Pretrained Large Language Models for Medical Domains," arXiv preprint arXiv:2402.10373, Feb. 2024.

[3] D. Han, I. Peng, M. Fei, I. Beltagy, K. Chang, A. Cheng, and others, "MEDITRON-70B: Scaling Medical Pretraining for Large Language Models," arXiv preprint arXiv:2311.16079, Nov. 2023.

[4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-T. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, 2020, pp. 9459–9474.

[5] K. Shi, H. Han, and A. T. Campbell, "Lessons Learned on Information Retrieval in Electronic Health Records: A Comparison of Embedding Models and Pooling Strategies," arXiv preprint arXiv:2409.15163, Sep. 2024.

[6] International Council for Harmonisation of Technical Requirements for Pharmaceuticals for Human Use (ICH), "ICH Harmonised Guideline: Integrated Addendum to ICH E6(R1): Guideline for Good Clinical Practice E6(R2)," 2016. [Online]. Available: https://www.ich.org/page/efficacy-guidelines.

[7] U.S. Department of Health and Human Services, "45 C.F.R. Part 46 and 21 C.F.R. Part 50/56 — Protection of Human Subjects; and 42 C.F.R. Part 11 — Clinical Trials Registration and Results Information Submission," 2016.

[8] G. Salton and M. J. McGill, Introduction to Modern Information Retrieval. New York, NY, USA: McGraw-Hill, 1983.

[9] S. E. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," Foundations and Trends® in Information Retrieval, vol. 3, no. 4, pp. 333–389, 2009. doi: 10.1561/1500000019.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017, pp. 5998–6008.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in Proc. 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT), Minneapolis, MN, USA, 2019, pp. 4171–4186. doi: 10.18653/v1/N19-1423.

[12] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-T. Yih, "Dense Passage Retrieval for Open-Domain Question Answering," in Proc. 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online, 2020, pp. 6769–6781. doi: 10.18653/v1/2020.emnlp-main.550.

[13] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), Hong Kong, China, 2019, pp. 3982–3992. doi: 10.18653/v1/D19-1410.

[14] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, Dense, and Attentional Representations for Text Retrieval," Transactions of the Association for Computational Linguistics, vol. 9, pp. 329–345, 2021. doi: 10.1162/tacl_a_00369.

[15] X. Ma, R. Zhang, L. Yang, J. Jiang, W. Chen, and M. de Rijke,

"Hybrid List-Wise Learning to Rank for IR," arXiv preprint arXiv:2205.02917, May 2022.

[16] G. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," in Proc. 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '09), Boston, MA, USA, 2009, pp. 758–759. doi: 10.1145/1571941.1572114.

[17] R. Nogueira and K. Cho, "Passage Re-ranking with BERT," arXiv preprint arXiv:1901.04085, Jan. 2019.

[18] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "BioBERT: A Pre-trained Biomedical Language Representation Model for Biomedical Text Mining," Bioinformatics, vol. 36, no. 4, pp. 1234–1240, Feb. 2020. doi: 10.1093/bioinformatics/btz682.

[19] Y. Gu, R. Tinn, H. Cheng, M. Lucas, N. Usuyama, X. Liu, T. Naumann, J. Gao, and H. Poon, "Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing," ACM Transactions on Computing for Healthcare, vol. 3, no. 1, pp. 1–23, Jan. 2022. doi: 10.1145/3458754.

[20] Q. Jin, W. Kim, Q. Chen, D. C. Comeau, L. Yeganova, W. J. Wilbur, and Z. Lu, "MedCPT: Contrastive Pre-trained Transformers with Large-scale PubMed Search Logs for Zero-shot Biomedical Information Retrieval," Bioinformatics, vol. 39, no. 11, Nov. 2023. doi: 10.1093/bioinformatics/btad651.

[21] Y. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, Z. Liu, and Z. Hu, "BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation," arXiv preprint arXiv:2402.03216, Feb. 2024.

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and Efficient Foundation Language Models," arXiv preprint arXiv:2302.13971, Feb. 2023.

[23] Meta AI, "Meta Llama 3 Model Card," Meta AI Research, Apr. 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/. [Accessed: Jul. 2026].

[24] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," arXiv preprint arXiv:2210.17323, Oct. 2023.

[25] C. A. Gomez-Cabello, S. Prabha, S. A. Haider, A. Genovese, B. G. Collaco, N. G. Wood, S. Bagaria, and A. J. Forte, "Comparative Evaluation of Advanced Chunking for Retrieval-Augmented Generation in Large Language Models for Clinical Decision Support," Bioengineering (Basel), vol. 12, no. 11, Nov. 2025. doi: 10.3390/bioengineering12111194. PMID: 41301150; PMCID: PMC12649634.

[26] LangChain Contributors, "ParentDocumentRetriever," LangChain Documentation,

- [Online]. Available: https://python.langchain.com/docs/modules/data_connection/retrievers/parent_document_retriever. [Accessed: Jul. 2026].

[27] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise Zero-Shot Dense Retrieval without Relevance Labels," in Proc. 61st Annual Meeting of the Association for Computational Linguistics (ACL), Toronto, Canada, 2023, pp. 1762–1777. doi: 10.18653/v1/2023.acl-long.99. (arXiv:2212.10496, 2022).

[28] O. Press, M. Zhang, S. Min, L. Schmidt, N. A. Smith, and M. Lewis, "Measuring and Narrowing the Compositionality Gap in Language Models," in Proc. Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, 2023, pp. 5687–5712. doi: 10.18653/v1/2023.findings-emnlp.378.

[29] H. Zheng, S. S. Mishra, X. Chen, H.-T. Cheng, E. H. Chi, D. Zhou, and

Q. V. Le, "Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models," arXiv preprint arXiv:2310.06117, Oct. 2023. (Google DeepMind).

[30] LangChain Contributors, "Self Query Retriever," LangChain Documentation,

- [Online]. Available: https://python.langchain.com/docs/modules/data_connection/retrievers/self_query/. [Accessed: Jul. 2026].

[31] K. Roberts, D. Demner-Fushman, E. M. Voorhees, W. R. Hersh, S. Bedrick, A. J. Lazar, and S. Pant, "Overview of the TREC 2021 Clinical Trials Track," in Proc. 30th Text Retrieval Conference (TREC 2021), NIST Special Publication 500-335, Gaithersburg, MD, USA, 2022.

[32] Z. Wang, B. Ge, C. Fu, T. Qin, A. Kulkarni, C. Boland, B. Pohl, and M. Nguyen, "RECTIFIER: Retrieval-Augmented Generation for Clinical Trial Eligibility Criteria Review," medRxiv preprint 2024.02.08.24302376, Feb. 2024. doi: 10.1101/2024.02.08.24302376.

[33] M. Wornow, A. Lozano, D. Y. Dash, J. Bhatt, T. Morales, C. Chi, J. A. Dunnmon, A. Callahan, and N. H. Shah, "Zero-Shot Clinical Trial Patient Matching with LLMs," NEJM AI, vol. 1, no. 8, Aug. 2024. doi: 10.1056/AIoa2400181.

[34] Garapati Keerthana, Manik Gupta, "CLI-RAG: A Retrieval-Augmented Framework for Clinically Structured and Context Aware Text Generation with LLMs," arXiv preprint arXiv:2507.06715, Jul. 2025.

[35] C. Xiong, I. Cohan, and N. Gohel, "Benchmarking Retrieval-Augmented Generation for Medicine (MIRAGE)," arXiv preprint arXiv:2402.13178, Feb. 2024.

[36] N. Neehal, A. Mehta, and J. Ghosh, "CTBench: A Comprehensive Benchmark for Evaluating Language Model Capabilities in Clinical Trial Design," arXiv preprint arXiv:2406.17888, Jun. 2024.

[37] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," in Proc. 18th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Malta, 2024, pp. 150–163. doi: 10.18653/v1/2024.eacl-demo.16. (arXiv:2309.15217, 2023).

[38] J. Ru, T. He, Y. Fang, X. Wen, J. Gu, Y. Pei, Y. Yao, J. Zhao, and C. Zhang, "RAGChecker: A Fine-grained Framework for Diagnosing Retrieval-Augmented Generation," arXiv preprint arXiv:2408.08067, Amazon, Aug. 2024.

[39] E. M. Voorhees and D. K. Harman, Eds., TREC: Experiment and Evaluation in Information Retrieval. Cambridge, MA, USA: MIT Press, 2005.

[40] Chroma Core Contributors, "ChromaDB: The AI-Native Open-Source Embedding Database," GitHub Repository, 2023. [Online].
Available: https://github.com/chroma-core/chroma. [Accessed: Jul. 2026].

[41] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in Proc. 29th Symposium on Operating Systems Principles (SOSP '23), Koblenz, Germany, 2023, pp. 611–626. doi: 10.1145/3600006.3613165.

[42] Y. A. Malkov and D. A. Yashunin, "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 4, pp. 824–836, Apr. 2020. doi: 10.1109/TPAMI.2018.2889473.

[43] Ollama Contributors, "Ollama: Get Up and Running with Large Language Models," GitHub Repository, 2023. [Online].
Available: https://github.com/ollama/ollama. [Accessed: Jul. 2026].

[44] S. Timoté, C. Lison, L. Solberg, and A. Touileb, "rank_bm25: A Collection of BM25 Algorithms in Python," GitHub Repository, 2020. [Online]. Available: https://github.com/dorianbrown/rank_bm25. [Accessed: Jul. 2026].

[45] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-Generation Hyperparameter Optimization Framework," in Proc. 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), Anchorage, AK, USA, 2019, pp. 2623–2631. doi: 10.1145/3292500.3330701.

Total references: 45

Format: IEEE (Institute of Electrical and Electronics Engineers)

For thesis submission to the Department of Informatics, University of Piraeus, 2026