Diploma Project

AIDA: An AI Doctor Assistant for Clinical Documentation and Analysis

Victor Economides

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2026

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

AIDA: An AI Doctor Assistant for Clinical Documentation and Analysis

Victor Economides

Advisor:

Dr Panayiotis Kolios

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2026

Declaration

I declare that this dissertation and any accompanying software are my own original work, conducted in full compliance with the University of Cyprus Code of Conduct for Research (https://www.ucy.ac.cy/legislation/kodikas-deontologias-kai-politikes/), and with full accountability on my part for the accuracy, integrity, and validity of all content.

Acknowledgements

I would like to thank my supervisor, Dr Panayiotis Kolios, for his guidance, constructive feedback and support throughout the development of this project and for presenting me with the opportunity to work on a problem as impactful and interesting as AI-driven clinical documentation.

ABSTRACT

The documentation burden placed on physicians represents a growing crisis in modern healthcare, with studies showing that clinicians spend over two hours per eight-hour shift on documentation alone. Compounding this, patient data across healthcare systems is frequently structured differently, making reliable exchange between institutions and vendors difficult. This dissertation presents AIDA (AI-powered Doctor Assistant), a system that addresses both challenges simultaneously through an end-to-end, locally hosted AI pipeline.

AIDA captures physician-patient consultations via a web interface, transcribes them using OpenAI's Whisper, extracts structured clinical facts using MedGemma, a medically fine-tuned large language model and persists the resulting data as HL7 FHIR resources to a HAPI FHIR server. The system additionally constructs a patient-level knowledge graph and exposes a GraphRAG analytics engine that enables physicians to query their patient population using natural language.

Evaluation of the system has demonstrated that the pipeline correctly transcribes clinical dialogue and generates valid FHIR resources, with strong fact extraction in structured consultations but reduced accuracy in informal, conversational speech where limited FHIR resource coverage led to clinical information being lost. The GraphRAG engine performs well on queries with well-defined anchors such as history retrieval, allergy search and population statistics.

The system's privacy-preserving design is a key architectural contribution, as all AI inference runs on controlled infrastructure to ensure patient data never leaves trusted environments. At the same time, limitations such as LLM hallucination risk and limited FHIR resource coverage highlight directions for future work.

Keywords: clinical documentation, large language models, FHIR, GraphRAG, electronic health records, healthcare informatics

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AIDA	AI-powered Doctor Assistant
AI	Artificial Intelligence
ASR	Automatic Speech Recognition
EHR	Electronic Health Record
FHIR	Fast Healthcare Interoperability Resources
GDPR	General Data Protection Regulation
HL7	Health Level Seven International
JSON	JavaScript Object Notation
LLM	Large Language Model
NLP	Natural Language Processing
RAG	Retrieval-Augmented Generation

1 Introduction

The delivery of high-quality healthcare depends not only on clinical expertise but also on the effective management of patient information. Today's medical practices generate large quantities of data with every patient encounter, including but not limited to, symptoms, diagnoses, prescribed medications, allergies and follow-up instructions all of which must be accurately recorded, stored, and made accessible to support future clinical decisions. In practice, this task burdens a physician with a significant proportion of their working day, often at the direct expense of time spent with patients [1]. Studies have found that physicians spend a mean of 5.8 hours per 8 patient-care scheduled hours actively working in the electronic health record (EHR). The most time spent is in documentation tasks with 2.3 hours per 8 patient-care scheduled hours. This disparity contributes to the clinician's burnout, reduced care quality and systemic inefficiency across healthcare organisations [2].

The emergence of Artificial Intelligence (AI) technologies and in particular recent advances in Large Language Models (LLMs), automatic speech recognition (ASR) and structured data extraction, presents a compelling opportunity to address this problem directly. AI systems are now capable of transcribing spoken language, understanding the clinical context of conversations, and transforming unstructured natural language into structured, semantically full medical data. These capabilities, if deployed responsibly within healthcare settings, could dramatically reduce the documentation burden on clinicians while simultaneously improving the completeness and accuracy of patient records [1, 3].

Interoperability is another central challenge in modern healthcare informatics. Patient data in various systems is frequently structured differently, making distribution of data across them difficult and even incompatible at times. The Fast Healthcare Interoperability Resources (FHIR) standard, developed and maintained by Health Level Seven International (HL7), has emerged as the leading framework for representing and exchanging electronic health data, solving this issue. FHIR defines a rich set of resource types, including Patient, Encounter, Observation, Condition, MedicationStatement, and AllergyIntolerance, as well as others, that provide a standardised vocabulary for clinical information, enabling systems developed by different vendors and institutions to exchange data reliably [4].

This dissertation presents the design and implementation of AIDA, an AI-powered Doctor Assistant system that addresses both the documentation burden and the interoperability challenge simultaneously. The system listens to physician-patient sessions, transcribes the dialogue using OpenAI's Whisper automatic speech recognition model, extracts clinically relevant facts from the transcript using a Large Language Model, specifically MedGemma developed by Google

DeepMind, converts those facts into valid FHIR resources and persists them to a HAPI FHIR server, an open-source, standards-compliant FHIR repository. In addition, the system exposes a Graph Retrieval-Augmented Generation (GraphRAG) analytics interface that allows physicians to query their patient population using natural language, enabling pattern recognition and population-level clinical analysis.

1.1 Aims and Objectives

The aim of this research is to demonstrate that an automated, AI-driven pipeline can bridge the gap between unstructured clinical conversations and structured, interoperable health records, while also enabling intelligent analysis of the resulting data. This aim is pursued through the following concrete objectives:

- To design and implement a pipeline that captures physician-patient dialogue, transcribes it automatically using ASR, and extracts structured clinical facts using a Large Language Model.
- To develop a robust mapping layer that converts extracted clinical facts into valid FHIR resources (Patient, Encounter, Observation, Condition, MedicationStatement, AllergyIntolerance) and submits them to a HAPI FHIR server via transaction bundles.
- To construct a Knowledge Graph over the stored FHIR data that preserves the relational structure of patient records, linking encounters to observations, conditions, medications, and allergies and supports graph traversal for context retrieval.
- To implement a GraphRAG analytics engine that enables physicians to query the patient knowledge graph using natural language and receive clinically meaningful, AI-generated responses grounded in the retrieved graph context.
- To integrate the entire system into a cohesive, deployable web application built on the Django framework, demonstrating practical feasibility for real-world adoption.

1.2 Contributions

This dissertation makes several contributions to the fields of healthcare informatics, applied AI, and clinical natural language processing (NLP):

FHIR generation pipeline. While prior work has addressed individual components of clinical NLP and usage of LLMs regarding this topic, this work integrates all stages into a single, deployable pipeline running within a standard web application framework. The pipeline proceeds

from raw audio capture through Whisper ASR, LLM-based fact extraction, FHIR bundle construction, and HAPI server submission, with each stage designed to be modular and replaceable.

GraphRAG for clinical analytics. This work introduces a graph-structured approach to healthcare data analysis in which FHIR resources are represented as typed nodes with explicit relational edges, and context for LLM prompts is assembled through deterministic graph traversal rather than approximate vector similarity. This preserves the relational semantics of medical data such as encounter-observation linkages and patient-condition associations in a way that vector retrieval cannot.

Privacy-preserving, locally hosted architecture. AIDA is designed to operate entirely on local infrastructure, using open-source components (HAPI FHIR, Whisper, MedGemma via llama.cpp) without transmitting patient data to third-party cloud services. This architectural choice directly addresses one of the key practical barriers to AI adoption in healthcare: the conflict between cloud-based AI services and patient data privacy regulations such as the General Data Protection Regulation (GDPR).

Open-source, replicable implementation. The complete implementation is publicly available, providing a reference architecture and codebase that practitioners and researchers can build upon, adapt, or evaluate independently.

1.3 Structure of the Thesis

The remainder of this thesis is organised as follows:

Chapter 2 reviews the prior work relevant to AIDA, covering AI-driven clinical documentation systems, the use of LLMs for clinical NLP, existing approaches to FHIR resource generation, and knowledge graph-based retrieval in healthcare.

Chapter 3 describes the system architecture of AIDA in detail, presenting the end-to-end pipeline from speech recognition and clinical fact extraction through to FHIR persistence, knowledge graph construction, and the GraphRAG analytics engine.

Chapter 4 evaluates the system through a series of scenario-based user stories representing realistic clinical consultations, assesses the GraphRAG component, and discusses the benefits and limitations of the current implementation.

Finally, Chapter 5 concludes the thesis with a summary of contributions and directions for future work.

1.4 Summary

This chapter has introduced the context and motivation for this research, establishing the clinical and technical problem of physician documentation burden and the interoperability challenges of modern healthcare. It has presented the system developed in this dissertation: an AI-powered Doctor Assistant that combines automatic speech recognition, large language model-based clinical fact extraction, FHIR-standard data persistence and GraphRAG-powered analytics and it articulated the aims, objectives and contributions that frame the work.

The documentation burden on healthcare professionals is a well-recognised and growing problem, and the technologies to address that have matured substantially in recent years. What has been lacking is an integrated implementation that demonstrates these technologies working together in a realistic workflow, while also ensuring interoperability by following standards. This dissertation presents such an implementation.

2 Related Work

This chapter examines the prior work that frames the design and implementation of AIDA. The system integrates several distinct technologies such as speech recognition, large language model-based clinical fact extraction, FHIR-standard data persistence, knowledge graph construction and retrieval-augmented generation, each of which has its own research lineage. The following sections review the most relevant work in each of these areas and identify the gaps that AIDA addresses.

2.1 AI-Driven Clinical Documentation

The burden of clinical documentation has been a well-recognised and growing problem in healthcare. Physicians spend a disproportionate fraction of their working hours in the EHR, a pattern that has been directly associated with burnout, reduced quality of care, and systemic inefficiency [1, 2]. This observation has motivated a wave of AI documentation systems, tools that listen to physician-patient conversations, transcribe them automatically, and generate structured clinical notes without manual data entry by the physician.

Several commercial platforms have emerged in this space, including Nuance DAX, Nabla, and Abridge. These tools have been clinically evaluated and consistently demonstrate reductions in documentation time, after-hours EHR use, and physician burnout [5, 6]. A randomised controlled trial comparing two of these scribe tools, DAX v2.0 and Nabla v1.5, found statistically significant reductions in note-taking time and improvements in physician well-being [5]. Moreover, a large-scale deployment at The Permanente Medical Group with more than 2.5 million patient encounters found that 84% of physicians reported a positive impact on visit interactions and 82% reported improved work satisfaction. Additionally, a majority of patients stated that the AI scribe tool had a positive impact on the quality of their visit [7].

Despite their effectiveness, these commercial systems share a critical architectural characteristic: patient audio is transmitted to vendor-managed cloud infrastructure where it is processed by proprietary AI models outside the healthcare provider's control. For example, Nuance DAX uploads patient audio to Microsoft Azure, where speech recognition and summarisation are performed before the resulting note is returned to the clinician [8]. AIDA addresses this directly by running all AI inference , transcription and fact extraction, using locally hosted open-weight models, ensuring that patient data is never transmitted to an external AI service regardless of the deployment scenario. In a self-hosted configuration this eliminates third-party data exposure entirely, in a managed hosting configuration the hosting provider processes data under a

standard Data Processing Agreement, but no data is forwarded beyond that single controlled boundary.

It is also worth noting that the existing documentation platforms are primarily designed to generate free-text clinical notes for physician review and entry into the EHR. They do not produce structured, standards-compliant health records, nor do they persist data in an interoperable format that can be consumed by other systems. AIDA differs fundamentally in this respect: rather than generating a narrative note, it extracts structured clinical facts and persists them directly as FHIR resources to a standards-compliant server. This means the output of AIDA is immediately interoperable with any FHIR-compatible system, without requiring a physician to manually transcribe AI-generated text into an EHR. Furthermore, by storing data in a structured knowledge graph, AIDA exposes a natural language analytics interface that enables population-level clinical queries across the accumulated patient record, a capability that is entirely absent from existing platforms.

2.2 LLMs for Clinical Natural Language Processing

The use of NLP for extracting structured clinical information from text has a substantial history. However, the emergence of large generative language models has significantly changed the landscape of clinical NLP. Google DeepMind, with their medical LLM Med-PaLM 2, has demonstrated that medically fine-tuned LLMs could approach physician-level performance on clinical question-answering benchmarks such as MedQA and MultiMedQA. It has been shown that Med-PaLM 2 achieved 86.5% accuracy on the MedQA dataset, with physicians preferring its responses on eight out of nine clinical utility axes in human evaluation [9]. More recently, MedGemma, built on the Gemma 3 architecture and trained on a wide range of clinical datasets including MedQA and PubMedQA, as well as various clinical imaging reports, has demonstrated a strong capability in medical reasoning while at the same time being available as an open-weight model for local deployment [10], compared to Med-PaLM 2 which is a closed model. These facts have motivated the use of MedGemma in AIDA over general-purpose models of comparable size, as medical pre-training provides a more reliable foundation for interpreting the clinical language of physician-patient consultations.

2.3 LLM-Based FHIR Resource Generation

The problem of converting unstructured clinical text into structured, standards-compliant FHIR resources has received increasing research attention.

Recent work has explored the use of LLMs to perform this transformation. FHIR-GPT evalu-

ated GPT-4, Falcon, and Llama-2 on the task of generating FHIR MedicationStatement resources from discharge summaries, finding that the FHIR-GPT system (implemented with GPT-4) substantially outperformed both the smaller open-weight models and existing NLP pipelines on this task [11].

A parallel line of work, Infherno [12], proposed an agentic framework for FHIR resource synthesis from free-form clinical notes, LLM agents, code execution, and clinical terminology tools to comply with the FHIR document schema more reliably than single-prompt approaches. Infherno and AIDA were developed concurrently and independently from each other over the same period, with both systems converging on the same core premise, that LLMs can drive the transformation of unstructured clinical language into structured FHIR. This independent convergence provides a mutual validation of the approach. The two systems differ primarily in their extraction architecture: Infherno follows a multi-agent design in which specialised sub-agents handle distinct resource families, whereas AIDA uses a single structured prompt followed by a deterministic mapping layer, trading some flexibility for predictability and simplicity. As discussed in Section 4.5, Infherno's agentic extraction architecture represents a natural direction for extending AIDA in future work and the two systems could in principle be integrated. Infherno's extraction layer could replace AIDA's while leaving the downstream pipeline (FHIR persistence, knowledge graph construction, and GraphRAG analytics) unchanged.

AIDA holds a distinct position relative to both systems. It uses a locally hosted open-weight model, preserving patient data privacy throughout. FHIR-GPT on the other hand relies on cloud APIs and Infherno has used both commercial and open-weight models.

2.4 Knowledge Graphs and Graph-Based Retrieval for Healthcare

The key advantage of graph-structured representations over flat data is their ability to preserve relational semantics such as the fact that a condition belongs to a specific patient, an observation was recorded during a specific encounter or that a medication is associated with a specific allergy, enabling multi-hop inference.

Retrieval-Augmented Generation (RAG) has emerged as the primary approach for grounding LLM responses, reducing hallucination and enabling the model to reason over data it was not trained on. Traditional RAG systems retrieve context by computing the vector embedding of a query and then identifying the most semantically similar documents in a vector database. This is very effective for unstructured text retrieval, but it has a limitation when applied to structured clinical data: it treats each record as an independent document making it difficult to capture the relational structure that connects them.

Graph-based retrieval directly addresses this limitation. KARE, is a framework that works by generating a medical knowledge base from biomedical databases, clinical literature and LLM-generated insights and integrating knowledge graph community-level retrieval with LLM reasoning for healthcare prediction tasks. Through this framework, it has been demonstrated that graph-structured retrieval consistently outperforms standard RAG on clinical tasks [13]. AIDA extends these insights to a different problem. Rather than retrieving data from pre-existing biomedical knowledge bases, the GraphRAG analytics engine operates over a patient-level knowledge graph constructed dynamically from the FHIR data persisted during consultations. This allows for graph traversal over FHIR resource nodes, preserving the encounter-observation-condition-medication relationships that define the structure of a patient record. This approach foregoes the flexibility of vector similarity in favour of precise, semantics-preserving retrieval, a deliberate trade-off suited to the structured nature of FHIR data.

2.5 Summary

This chapter reviews the prior work relevant to AIDA across four interconnected areas:

AI Clinical Documentation: Commercial AI scribe tools have been proven effective at reducing documentation burden and physician burnout, but they rely on cloud infrastructure and produce free-text notes rather than structured, interoperable records. AIDA differentiates itself by running all inference locally and outputting standards-compliant FHIR resources directly.

LLMs for Clinical NLP: Medically fine-tuned models like Med-PaLM 2 and MedGemma demonstrate near-physician performance on clinical benchmarks. AIDA adopts MedGemma over general-purpose models of similar size precisely because medical pre-training offers a more reliable foundation for interpreting clinical language.

LLM-Based FHIR Generation: Both FHIR-GPT and Infherno have explored the use of LLMs to convert unstructured clinical text into FHIR resources. AIDA independently converged on the same premise but distinguishes itself through its local open-weight model, a single structured prompt with a deterministic mapping layer, and a downstream pipeline (FHIR persistence, knowledge graph, GraphRAG) that neither prior system includes.

Knowledge Graphs and Graph-Based Retrieval: Standard vector RAG struggles with relational clinical data. Graph-based retrieval, as demonstrated by frameworks like KARE, better preserves the semantic relationships in structured records. AIDA applies this insight by building a patient-level knowledge graph dynamically from FHIR data, enabling multi-hop clinical queries that flat retrieval approaches cannot support.

3 System Architecture

This chapter provides a comprehensive description of the design and implementation of AIDA. The system is composed of a set of cooperating subsystems: a Django-based web application, an automatic speech recognition module, a large language model pipeline responsible for clinical fact extraction, a deterministic mapping layer for FHIR resource generation, a HAPI FHIR server acting as the standards-compliant data storage backend, a knowledge graph constructed over the persisted FHIR data and a GraphRAG analytics engine that enables natural-language querying over the knowledge graph. Each subsystem is described in detail in the following sections.

The overall design philosophy consists of modularity and replaceability, meaning that every major component interacts with its neighbours and that individual modules can be upgraded or swapped without requiring changes to the rest of the system. This is particularly important in a domain as rapidly evolving as clinical AI or even AI in general, since the optimal choice of ASR model or LLM is likely to change within the lifespan of the implementation.

3.1 Pipeline Architecture

At the highest level, the system operates as a sequential processing pipeline with clear stages that transform, alter and enrich input data and then pass the result downstream, to the next stage. Figure 3.1 illustrates this end-to-end flow, while Figure 3.2 provides a detailed sequence diagram showing the message-level interactions between system components.

The pipeline is composed of six main stages. **In the first stage**, raw audio from a physician-patient consultation is captured via the browser and transmitted to the Django back end as a binary audio file. **In the second stage**, the audio is passed to the Whisper ASR module, which produces a plain-text transcript of the dialogue. **In the third stage**, the transcript is used as input to MedGemma, a medically fine-tuned LLM, together with a structured prompt that instructs the model to identify and extract clinically relevant facts. The model returns a JSON object containing clinical entities such as symptoms, conditions, medications and allergies as well as information about the patient. **In the fourth stage**, the extracted entities are converted into FHIR resources via a deterministic mapping layer and then are assembled into a transaction bundle, which is submitted to the HAPI FHIR server. **In the fifth stage**, a knowledge graph builder queries the FHIR server and builds an in-memory graph of the information. **In the sixth and final stage**, the GraphRAG engine accepts a physician's natural-language query, retrieves relevant subgraphs through deterministic graph traversal, assembles a context-rich prompt, and invokes the LLM to generate a response.

Figure 3.1: The AIDA end-to-end six-stage pipeline architecture.

This linear structure was chosen deliberately over an agentic approach. This is because the stages of the pipeline are well defined and sequential by nature, making the operational overhead of the alternative difficult to justify. Furthermore, due to this linearity, failures or issues at any point are immediately visible and can be reported back to the user with ease.

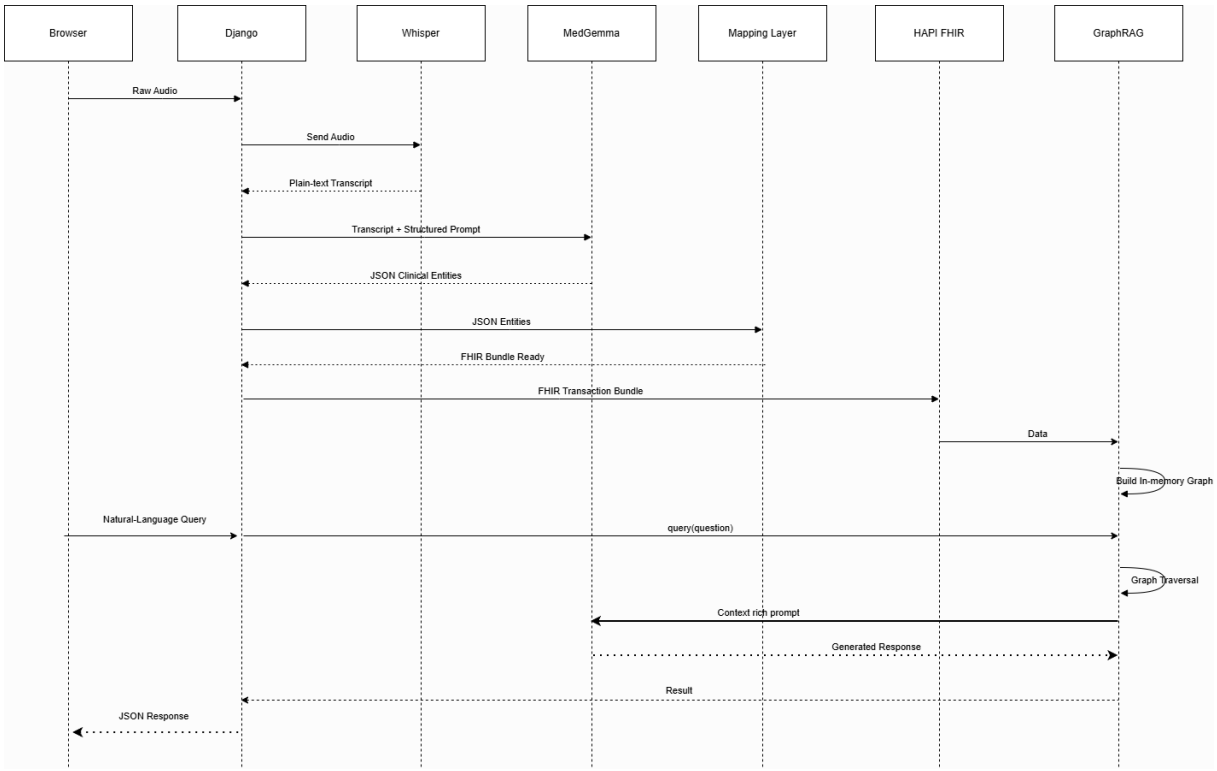


Figure 3.2: Sequence diagram showing the message-level interactions between AIDA system components.

3.2 Django Web Application Framework

Django was selected as the application framework for this project based on several considerations. As a batteries-included framework written in Python, Django provides out-of-the-box functionality such as URL routing, request handling, template rendering, session management, and a built-in administrative interface, reducing the amount of boilerplate code the developer must write and maintain. It makes it simple to deploy full-stack applications, with seamless communication of the frontend and the backend all in a single system. In addition, the Python

ecosystem provides first-class support to all the AI libraries required by the project, including Whisper and llama-cpp, making it possible to call those libraries directly from Django functions instead of resorting to communication with other services.

3.2.1 Views and Request Handling

In Django, views are Python entities that receive a web request and return a web response. There are two types: function-based and class-based views.

Throughout this project only function-based views are used in preference to class-based views. Function-based views were chosen because each endpoint in this system performs a distinct operation with minimal shared logic between endpoints (each view is a step of the pipeline) making the additional abstraction of class-based views an unnecessary complication.

Each view function is responsible for input validation, computation, error handling, and response serialisation. Separating the view layer from the service layer in this way ensures that the business logic of each pipeline stage can work and be tested independently.

3.2.2 Front-End

The front-end interface is lightweight, consisting of HTML templates served by Django, together with CSS styling and browser-native JavaScript for audio recording via the MediaRecorder API. No front-end framework was introduced to minimise operational complexity. The MediaRecorder API captures audio directly in the browser, encodes and posts it to the Django upload endpoint as a binary blob. This approach avoids the need for a native desktop application or a dedicated recording daemon.

3.3 Whisper Automatic Speech Recognition Module

The ASR component is responsible for converting the raw audio recording of a consultation between physician and patient into a plain text transcript. OpenAI's Whisper was selected for this role based on its strong performance across various speech recognition benchmarks, its robustness to background noise and varying audio conditions and its ability to generalise well without fine-tuning on a specific task while also supporting local inference. Whisper was trained on approximately 680,000 hours of multilingual and multitask audio data, enabling it to handle conversational speech in diverse environments effectively. Furthermore, it has been shown that Whisper maintains strong performance under noisy conditions compared to other models and exhibits robust zero-shot generalisation across datasets [14]. These characteristics make it suitable for a clinical setting, where speech is unstructured and is recorded mostly in non-ideal

acoustic conditions. The interface responsible for this step is illustrated in Figure 3.3

3.3.1 Model Selection

While the properties described above make Whisper a strong candidate for this task, deploying the full large-v2 or medium.en checkpoints, at 1550 million and 769 million parameters respectively, on a CPU-only machine, proved impractical for real-time or near-real-time transcription. For this reason, distil-whisper/distil-medium.en was selected instead, a knowledge-distilled variant of whisper-medium.en introduced by Gandhi et al. [15] that was designed precisely to balance accuracy and inference speed in resource-constrained environments.

Distil-Whisper is obtained by applying pseudo-labelling-based knowledge distillation to the original Whisper model, resulting in a student model that is 5.8 times faster and contains about 51% fewer parameters. Specifically, the distilled model is initialised by copying the entire encoder from the medium.en teacher and freezing it during training, while the decoder is reduced to only two layers: the first and last decoder layers of the teacher, and discarding all other intermediate layers. As shown in Table 3 of the paper, distil-medium.en has 394 million parameters, compared to the 769 million of its medium.en teacher, making it substantially more tractable for CPU inference.

Despite this compression, accuracy is well preserved. On short-form audio distil-medium.en performs worse than medium.en by 1.6% WER, but on long-form audio, the more relevant setting for full consultation transcription, the gap narrows to just 0.1% WER, which is a negligible difference given the significant gain in inference speed [15].

Additionally, this model is distributed under the MIT licence and is available via the Hugging Face Hub, which enables local inference without reliance on any external API making it consistent with the privacy requirements of a clinical setting.

3.3.2 Transcript Handling

The Whisper ASR model returns the transcript as a plain text string, which is directly forwarded to the downstream LLM without additional post-processing. Although clinical speech may include filler words, informal phrasing, or abbreviations, no explicit normalisation or cleaning is applied at this stage. In a production-grade clinical system, additional preprocessing, such as abbreviation expansion or noise filtering, would likely be beneficial.

However, for the purposes of this proof-of-concept, this simplified approach is sufficient. Modern LLMs have shown a strong ability to interpret noisy and unstructured input when guided by an appropriately designed prompt. Empirically, the system was able to extract the required clinical information with acceptable reliability, making further preprocessing unnecessary within

the scope of this project.

3.4 LLM-Driven Clinical Fact Extraction

After a transcript has been produced, it must be parsed to extract clinical facts that will then, in later steps, be transformed into FHIR resources. Clinical conversations are unstructured and filled with implicit information that requires medical domain knowledge to interpret accurately. If we were to tackle this issue with a rule-based approach, it would require an exhaustive vocabulary of medical terms and it would not generalise properly to real consultations. In contrast, LLMs, with the use of an appropriate prompt, can leverage their understanding of language and medical context to interpret information, resolve ambiguities and extract the relevant clinical facts without relying on predefined rules. This flexibility allows us to generate the data we need even from diverse and unstructured clinical conversations. The results interface produced by this step is illustrated in Figure 3.4

3.4.1 Model Selection: MedGemma

MedGemma, developed by Google Research and Google DeepMind, is a collection of medically fine-tuned vision-language foundation models built on the Gemma 3 architecture. It was selected as the LLM for the extraction of clinical facts from the transcript for two primary reasons. First, MedGemma has been trained on a wide range of biomedical and clinical datasets, including medical question-answering corpora such as MedQA and PubMedQA, as well as clinical imaging reports, resulting in a significantly stronger understanding of medical terminology, diagnoses, and clinical relationships compared to general-purpose models of comparable size. Second, MedGemma is released as an open-weight model, meaning its weights can be downloaded and run locally without requiring a subscription or API key, which directly satisfies the privacy constraints of this project [10]. Third, the model can be used efficiently using llama.cpp, a high-performance engine that supports quantised model weights and runs on CPU hardware, making it accessible without specialised GPU infrastructure [16].

The llama-cpp library allows the model to be loaded and queried directly from within the Django service layer without a separate server. The model is loaded once at application start-up and held in memory for the lifetime of the process, avoiding the latency of repeated loading between requests.

3.4.2 Prompt Design

To generate accurate, high-quality results from an LLM, it is critical that we have good prompt design. The extraction prompt used in this system instructs the model to read the provided transcript and return a JSON object containing the following top-level keys: patient (demographic information), encounter (chief complaint), symptoms (a list of symptoms mentioned by the patient or the doctor, either present or not with metadata such as duration and severity), conditions (a list of diagnosed or reported conditions), medications (a list of prescribed or mentioned medications with dose and frequency where available) and allergies (a list of reported allergies with the nature of the reaction where mentioned).

The prompt follows a zero-shot strategy in which a detailed set of instructions and an explicit JSON schema are provided without any example of input-output. This allows the model to fully structure its response based on the schema and constraints. Having these strict rules helps the model generate exactly the response that we need for the next step of the pipeline, which is the deterministic mapping of the clinical facts to FHIR resources.

The prompt explicitly instructs the model to return only the JSON object and no surrounding text and to use null for any field for which information is not available in the transcript. It also instructs the LLM to not infer diagnoses and add medical knowledge, so it does not fabricate values that are plausible but not accurate to the consultation. This instruction is intended to mitigate the hallucination risk that is a known limitation of LLMs. By instructing the model to explicitly note absent information rather than filling gaps, the downstream FHIR mapping layer can distinguish between a field that was not mentioned and one for which the model has low confidence. (see Appendix I.1)

3.5 FHIR Resource Generation and Bundle Construction

The validated JSON produced by the LLM in the fact extraction stage is consumed by the FHIR mapping layer, which is responsible for constructing valid FHIR resources and assembling them into a transaction bundle for atomic submission to the HAPI FHIR server.

3.5.1 FHIR Resource Types

The system generates six different FHIR resource types, each corresponding to a category of clinical information that may be present in a consultation transcript. While FHIR provides a wide range of resource types, the selected six: Patient, Encounter, Observation, Condition, MedicationStatement, and AllergyIntolerance were chosen because they capture the essential information that a physician would extract during a consultation. This selection is sufficient for

the purposes of this proof-of-concept.

The Patient resource captures demographic information including the patient's full name and gender. In addition, the national ID of the patient is assigned to their record, allowing the system to detect and update existing records rather than creating duplicates on repeated encounters.

The Encounter resource represents the consultation event itself and records the chief complaint of the patient. It is linked to the patient via a Patient reference and serves as the focal point to which all clinical findings from the consultation are attached.

Observation resources represent each extracted symptom from the consultation. They include a textual description of the symptom and are linked to both the Patient and Encounter resources via references. The presence or absence of a symptom is also represented using either a boolean value (for explicitly absent symptoms) or a string value describing its presence. When available, additional details such as duration and severity are included within this value.

Condition resources are generated for each identified condition mentioned in the consultation. Each Condition includes a textual description of the condition and is linked to the Patient resource via a reference. In the current version of the system, all conditions are assigned a clinical status of "active," as no distinction is made between active and historical conditions.

MedicationStatement resources record medications that the patient is taking or has been prescribed during the encounter. Each resource includes the medication name and dosage quantity and dosage frequency if provided. These values are stored in a string field of the resource.

AllergyIntolerance resources record allergies reported by the patient or clinician. Each resource includes the substance name as well as a description of the patient's reaction to it, if mentioned in the transcript.

3.5.2 Transaction Bundle Construction

All resource types generated for a consultation are assembled into a single, transaction type FHIR Bundle. A transaction bundle allows multiple create operations to be submitted to the FHIR server as an atomic unit. Either all operations succeed or none are committed, preventing partial writes that would leave the server in an inconsistent state. Each entry in the bundle carries a request field specifying the HTTP method (POST) and the target URL.

The bundle is serialised to JSON and submitted to the HAPI FHIR server via an HTTP POST to the server's base URL. The python requests library is used for this HTTP interaction.

3.6 HAPI FHIR Server

HAPI FHIR is an open-source Java-based implementation of the HL7 FHIR specification [17]. It provides a fully standards-compliant FHIR RESTful API, including support for all standard CRUD operations, transaction bundles, search parameters and more [18]. The HAPI FHIR server was selected as the persistence backend for this project based on its conformance completeness, its active maintenance, its zero-cost licensing, and its suitability for local deployment [17].

HAPI FHIR is run as a Docker container on the same host machine as the Django application. The server is configured to use an embedded H2 database for resource storage (default database) [18], which is appropriate for a single-practitioner deployment or a research prototype but would need to be replaced with a production-grade database such as PostgreSQL for a multi-user clinical deployment [19]. The server is accessible only on localhost and is not exposed to the network.

3.6.1 Search and Retrieval

Beyond the submission of transaction bundles, the system also interacts with the HAPI FHIR server during knowledge graph construction. In this case, the system makes FHIR search requests to the server's RESTful API. For example, to retrieve all Observations associated with a given patient, the system sends an HTTP GET to the Observation endpoint with the patient search parameter set to the patient's server-assigned ID. The search results are returned as a FHIR Bundle, which is parsed by the application to extract the information needed.

All FHIR interactions are encapsulated within a dedicated FHIRClient class. This class provides an abstraction over the raw HTTP requests, handling URL construction, header setting (including the Accept: application/fhir+json header required by the FHIR specification) and response parsing. This encapsulation ensures that the rest of the application is separated from the details of the FHIR REST protocol.

3.7 Knowledge Graph Construction

The knowledge graph is the foundation of the GraphRAG analytics engine. It represents the FHIR data stored on the HAPI FHIR server as a graph with nodes corresponding to FHIR resources and edges corresponding to the references between them. Having the FHIR data structured as an explicit graph, enables the system to efficiently retrieve context through graph traversal, which preserves the semantics of clinical relationships in a way that vector-based retrieval cannot.

3.7.1 Graph Schema

The graph's structure is built based on the FHIR resources used, with six distinct node types: Patient, Encounter, Observation, Condition, Medication, and Allergy. Each of these nodes is implemented as a Python dataclass that stores a set of properties extracted from the corresponding FHIR resource:

- A **PatientNode** stores the patient's FHIR identifier, full name, gender, and national identifier
- An **EncounterNode** stores the encounter identifier, the patient identifier it refers to, the reason for the encounter, and its status.
- An **ObservationNode** stores the observation identifier, the patient and encounter identifiers it refers to, the observation code, and its recorded value.
- A **ConditionNode** stores the condition identifier, the patient identifier it refers to, the condition code, and its status.
- A **MedicationNode** stores the medication identifier, the patient identifier it refers to, the medication name, and its status.
- An **AllergyNode** stores the allergy identifier, the patient identifier it refers to, the substance, and the associated reaction description.

Every node type additionally provides a summary() method that returns a concise human-readable string describing the node, which is used when assembling context strings for LLM prompts.

The graph's edges were modeled using a directed multigraph (NetworkX MultiDiGraph) from the Python NetworkX library. Each edge carries a "rel" attribute that identifies the relationship type, with six relationship being used: has_encounter, connecting a Patient node to an Encounter node, has_observation, connecting a Patient node to an Observation node, recorded_in, connecting an Encounter node to an Observation node, has_condition, connecting a Patient node to a Condition node, has_medication, connecting a Patient node to a Medication node and has_allergy, connecting a Patient node to an Allergy node. In addition to the graph structure, the knowledge graph maintains a reverse feature index (_feature_index) that maps lowercase feature codes such as, condition names, observation codes, medication names, and allergy substances, to a set of patient identifiers associated with each feature. This index supports efficient population-level queries, such as finding all patients who share a given condition, without requiring a full graph traversal for every lookup.

3.7.2 Graph Construction Process

The graph is constructed by querying the HAPI FHIR server for resources and creating an in-memory graph implemented by the KnowledgeGraph class, which uses a NetworkX internally.

For the actual construction logic, we have a GraphBuilder class that is responsible for converting raw FHIR resources into KnowledgeGraph nodes. To do this, we also implemented a FHIRClient class that exposes wrapper functions for HTTP request to the HAPI FHIR server. The GraphBuiler has a reference to this FHIRClient and exposes two methods. The build_for_patient method accepts a single raw FHIR Patient resource and makes five queries, via the FHIRClient, to retrieve all Encounters, Observations, Conditions, MedicationStatements, and AllergyIntolerances associated with that patient. For each retrieved resource, it initializes the corresponding dataclass node and calls the appropriate add_* method on the KnowledgeGraph, which adds the node to the graph as well as the reverse feature index and creates an edge. Observations are additionally connected to their parent Encounter node via a recorded_in edge if an encounter reference is present. The build_full_graph method retrieves all Patient resources from the FHIR server and calls build_for_patient for each one skipping any patient whose data cannot be retrieved without aborting the remainder of the build. The complete process results in a directed multigraph that reflects the relational structure of the FHIR data.

The graph is rebuilt automatically before each GraphRAG query or by a dedicated endpoint, rather than being incrementally maintained. This strategy was selected for its simplicity. Since the system in not multi-tenant at this stage, a full rebuild from the FHIR server typically completes in under one second and guarantees that the graph is always consistent with the persisted data. An incremental update mechanism would introduce complexity without a meaningful performance benefit at this scale.

3.8 GraphRAG Analytics Engine

The GraphRAG analytics engine enables a physician to query their patient data using natural language and receive a clinically grounded, AI-generated response. The engine combines deterministic graph traversal with LLM-based language generation, following the principles of Retrieval-Augmented Generation (RAG) while replacing the usual vector-similarity retrieval approach with structured graph traversal. The analytics dashboard is illustrated in Figure 3.5

3.8.1 Motivation for Graph-Based Retrieval

Standard RAG systems retrieve context for an LLM prompt by computing the vector embedding of the query and making a similarity check to a vector store to identify the most semantically

similar documents. While this approach is effective for unstructured text retrieval, it has a significant limitation when it is applied to structured clinical data. It treats each fact as an independent record and cannot capture the structured relationships between them. For example, if we have a query asking whether a patient has a condition while also being currently prescribed a specific medicine, would require the system to identify the joint occurrence of these two facts within the same patient record, a relationship that vector similarity cannot express since it could retrieve documents about the condition and the medicine irrelevant to the one patient.

Graph traversal, on the other hand, directly exploits the relational structure of the FHIR data. The engine can answer a query like the one above by traversing the graph from each Patient node to its Condition nodes and Medication nodes, filtering on the relevant values, and returning only those patients for whom both conditions are satisfied. The retrieved subgraph context is then injected into a prompt that provides the LLM the appropriate facts it needs to formulate a response.

3.8.2 Query Processing Pipeline

The GraphRAG query pipeline has three phases. In the first phase, the physician asks the system a question in natural language, which is then injected to a dedicated planning prompt (PLANNER_PROMPT) and passed to the LLM. This special prompt instructs the model to return a structured JSON object representing the graph traversal plan. The plan contains four fields: traversal_goal, which is one of five values (find_similar, get_history, compare, find_by_symptom, or population_stats), this identifies what kind of graph traversal we will have to perform, anchor_type, which identifies the type of the base entity referenced in the query (patient_id, national_id, symptom, condition, medication, allergy, or population), anchor_value, which holds the specific value or term referencing the anchor_type and extra_patient_ids, a list of additional patient identifiers present for comparison queries (see Appendix I.2). If the LLM fails to produce a valid JSON, the system falls back to a safe default plan of using population_stats.

In the second phase, the GraphRAG Engine uses the _build_context method to perform the appropriate traversal strategy based on the traversal_goal. For a get_history goal it retrieves the complete clinical record of a single patient by traversing all outgoing edges from that patient's node. A find_similar goal executes a two-step traversal: the first step collects all features reachable from the anchor patient and the second uses the reverse feature index to identify other patients sharing those features. It then ranks them by the number of shared features and returns a subgraph containing only the matching nodes from each similar patient rather than their full records. A compare goal retrieves the full context for two or more patients and appends a shared-feature summary derived from a similarity traversal. A find_by_symptom goal uses the reverse feature index to find all patients associated with the queried feature code (symptom) and

returns a context for the LLM containing only the matching condition or observation nodes for each patient. A population_stats goal reads the feature index and produces a ranked summary of the most prevalent features in the population. In every case, only the subgraph which is directly relevant to the query is included in the context, preventing the LLM from being overwhelmed with irrelevant patient data.

In the third phase, a message is created containing the retrieved subgraph context, followed by the physician's original question and a request to analyse and asses the data. This message is passed to the LLM alongside a system prompt (SYSTEM_PROMPT) that instructs the model to act as a clinical decision support assistant, to identify clinically significant patterns, to be concise and specific, to cite the provided data in its response, and to clarify that it is providing decision support rather than a diagnosis. The system prompt also instructs the model to state clearly when no patients match the query, preventing the generation of fabricated patient data (see Appendix I.3).

3.8.3 Response Generation

The assembled prompt is submitted to the locally hosted MedGemma model via the llama–cpp–python interface. The model's output is returned to the Django view layer as part of a dictionary that includes not only its response, but also the context_used string (the exact subgraph text that was passed to the model). Returning the context alongside the answer allows the interface to display the retrieved subgraph data transparently alongside the AI-generated response, enabling the physician to verify that the model's assertions are grounded with the actual records that were retrieved.

3.9 Privacy and Security

Privacy and security are prime concerns in this system, due to the sensitive nature of patient data and the strict regulations that govern its use.

The main privacy guarantee of AIDA is that all AI inference is performed on infrastructure under the provider's direct control and that patient data is never transmitted to an external AI API service. In a single-practitioner deployment, where the physician hosts AIDA on their own hardware, patient data does not leave their infrastructure at all. In a managed hosting scenario, where a third party operates the server on behalf of a healthcare provider, the hosting provider processes the data under GDPR, requiring a Data Processing Agreement. However, even in this case, there is a critical distinction: patient data is processed exclusively within that single controlled boundary and is never further forwarded to an external AI service.

This is achieved by the architectural decision to run the two sensitive AI components, the Whisper transcription module and the MedGemma model, on the same server as the application itself. No matter if the server is a laptop during development or a virtual machine in a private cloud, the audio recordings and clinical transcripts it produces never leave the infrastructure that is under the control of the provider. This directly contrasts the alternative approach, in which audio or text is sent to a third-party ASR or LLM API for processing, transferring clinical data to an external party and raising immediate concerns under regulations such as GDPR.

The Django web application, the HAPI FHIR server, and the AI models all run on the same server, while physicians access the system through a web browser on their own workstations. The browser and server must communicate over a network connection, which introduces some security considerations. To prevent the interception of patient audio or the transcript data in transit, the connection must be encrypted using Transport Layer Security (TLS). The internal communication between the Django application and the HAPI FHIR server remains confined to the server itself and does not traverse an external network since both are running on the same host.

Authentication and access control is another matter that has not been implemented in this proof-of-concept, due to the single-user prototype nature of the implementation but has an important role in the matter of security, especially in a multi-user clinical environment. Django provides a built-in authentication framework that supports session-based login and can be extended to role-based access control which should be used to enable restrictions such as limiting a physician's access to records for their own patients.

The HAPI FHIR server itself exposes a RESTful API that must be protected from unauthorised access. The recommended approach is to expose the FHIR server only to a network that is not reachable from outside the server, so that all access is controlled by the Django application, which enforces authentication. The FHIR server should not be accessible from the browser. This ensures that the access control logic is centralised in Django and that the FHIR API is not exposed without authentication.

In general, the privacy architecture of the system is based on a single, principal decision: all AI inference is performed on infrastructure controlled by the provider, so that patient data is never handed to a third party.

The security components mentioned are essential for a production release, but they do not require changes to the core pipeline architecture described in the preceding sections. This separation makes it easier to evolve the system and transition it into a production-ready state.

3.10 User Interfaces

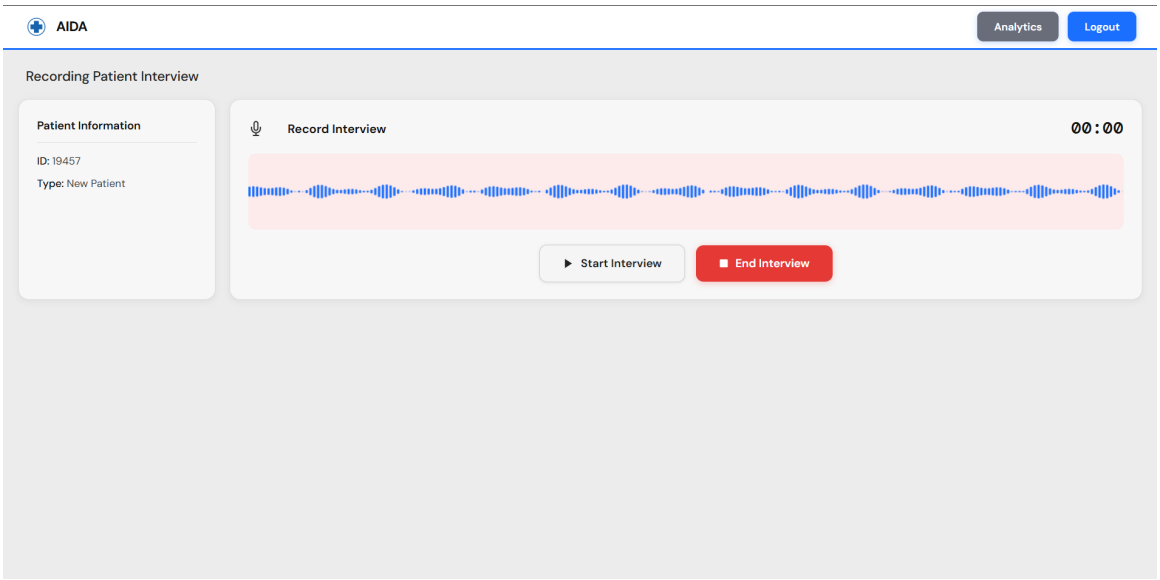


Figure 3.3: The AIDA recording interface, displaying the live audio waveform and interview controls used during physician-patient consultations.

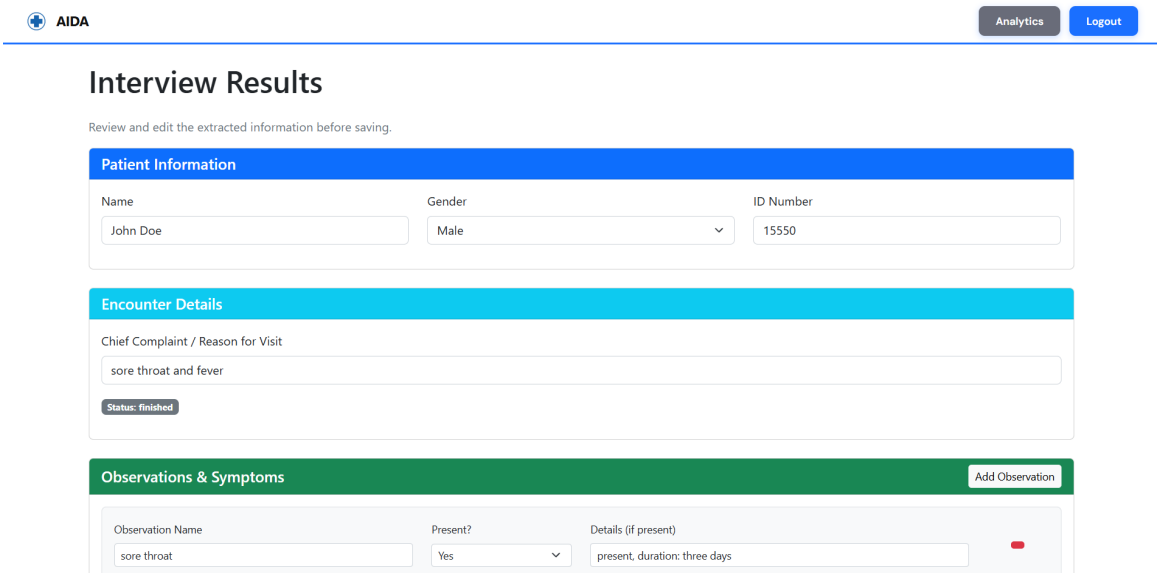


Figure 3.4: The Interview Results page, showing the structured medical record automatically extracted from the consultation transcript by the LLM processing pipeline.

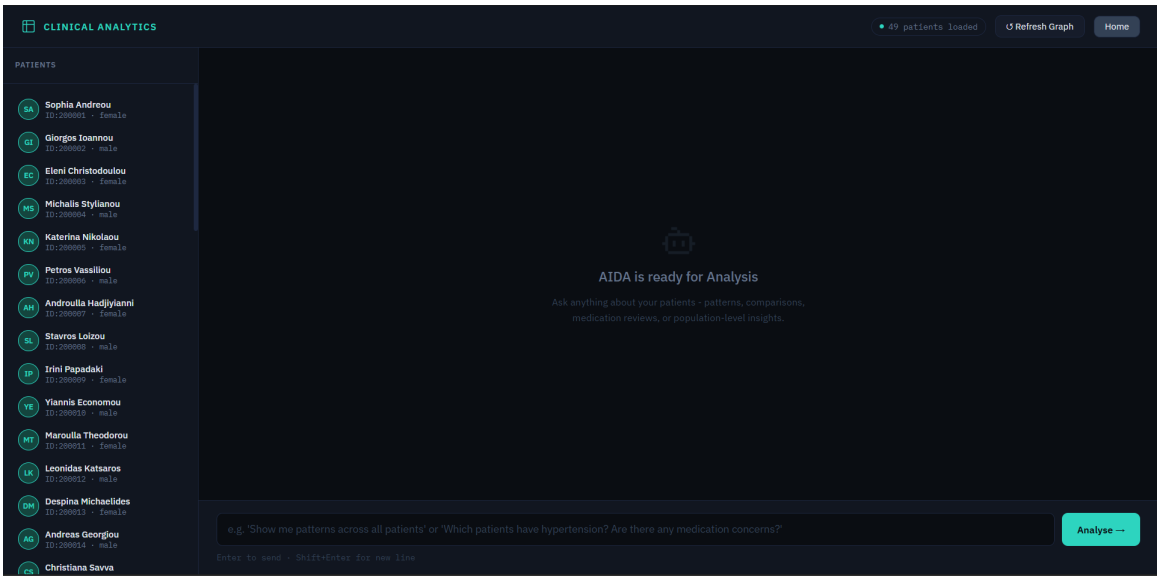


Figure 3.5: The Clinical Analytics dashboard, enabling physicians to query patient data using natural language through the GraphRAG Analytics Engine.

Additional system interfaces, including the homepage and patient management screen, are provided in Appendix II.

3.11 Summary

This section summarised the principal architectural decisions made for the design of the system and the reasoning behind them

We chose Django as the web framework because it uses Python, providing a simple entry point to the AI ecosystem and because it is batteries-included which reduces the amount of boilerplate code written. For the LLM, MedGemma was chosen because it has been medically fine-tuned, which improved fact extraction accuracy relative to a general-purpose model of comparable size and because it is available for local deployment due to its open-weights. As an ASR module, we used OpenAI's Whisper because of its strong performance in handling conversational speech in diverse environments, its robustness to background noise and because it also supports local inference. HAPI FHIR was chosen as the FHIR server because it is the most complete open-source FHIR implementation available and because it can be deployed locally without a cloud subscription. For the implementation of the knowledge graph, we used NetworkX because it provides an in-process graph data structure which alleviates the overhead of a dedicated graph database. Finally, the GraphRAG engine was designed for deterministic graph traversal instead of vector similarity retrieval, since the relational semantics of the FHIR data are better expressed as a graph rather than embeddings.

These decisions were based on the commitment to privacy, standards compliance, modularity, and the ability to deploy effectively on commodity hardware without requiring specialised infrastructure.

4 System Evaluation

This chapter evaluates AIDA against the aims and objectives described in Section 1.1. AIDA is a proof-of-concept system and so, the evaluation follows a qualitative methodology based on structured scenario testing. Due to hardware constraints (both the Whisper ASR module and MedGemma LLM are run entirely on a consumer laptop without a dedicated GPU), quantitative statistical benchmarking that requires repeated large-scale inference was impractical within the time scope of this project. Therefore, the evaluation focuses on demonstrating system correctness through physician-patient scenarios, analysing the observed behaviour of each pipeline stage, and discussing the limitations and benefits of the implemented approach.

4.1 Evalution Methodology

The evaluation methodology consists of three parts.

First, a set of physician-patient consultation scripts were designed to cover a range of clinical scenarios, from straightforward single-complaint encounters to more complex consultations involving various conditions, multiple medications, allergy disclosure and implicit information. These scripts were read aloud and recorded and then fed through the full AIDA pipeline via the web application: ASR transcription, LLM-based fact extraction, FHIR resource generation, and HAPI FHIR server persistence. The resulting FHIR resources were inspected for correctness and completeness against the ground-truth content of the scripts.

Second, the GraphRAG analytics engine was evaluated by making natural language queries over the patient data and assessing whether the responses are relevant and accurate.

Third, the chapter concludes with a discussion of the system's key benefits and its current limitations, framing the latter as directions for future work.

4.2 Scenario-Based User Stories

Three physician-patient consultation scenarios were designed and evaluated. Each scenario is presented with the consultation transcript, the ASR generated transcript, the structured JSON produced by the LLM extraction stage and an assessment of correctness.

4.2.1 Scenario 1: New Patient - Respiratory Complaint

The first scenario involves a presentation of an upper respiratory infection in an adult male. This scenario was designed to test the baseline capability of the pipeline to correctly identify patient demographics, a chief complaint, a set of symptoms with associated metadata, a medication, and a drug allergy. The patient is also a new client of the physician, so we are also testing whether a new patient record will be created (the ID of the patient is provided via the user interface before the recording page)

4.2.1.1 Consultation Transcript:

Doctor: *Good morning. What brings you in today?*
Patient: *Hi, I've had a sore throat and fever for the past three days. I also have a runny nose and a mild headache.*
Doctor: *Any cough or shortness of breath?*
Patient: *A little dry cough, but no shortness of breath.*
Doctor: *Any known allergies?*
Patient: *Yes, I'm allergic to penicillin. It gives me a rash.*
Doctor: *Alright. Based on what you are telling me, this looks like an upper respiratory tract infection. I will prescribe azithromycin 500mg once daily for five days. Take paracetamol for the fever. Get plenty of rest and fluids.*
Patient: *Okay, thank you.*
Doctor: *Patient name is John Doe, male, 34 years old*

4.2.1.2 Generated Transcript (Whisper):

Good morning. What brings you in today? Hi, I've had a sore throat and fever for the past three days. I also have a runny nose and a mild headache. Any cough or shortness of breath? A little dry cough, but no shortness of breath. Any known allergies? Yes, I'm allergic to penicillin. It gives me a ***rush****. All right, based on what you're telling me, this looks like an upper respiratory tract infection. I will prescribe* ***azithromisin*** *500 milligram once daily for five days. Take* ***parasitamol*** *for the fever, get plenty of rest and fluids. Okay, thank you. Patient name is John Doe, male 34 years old.*

We can see that the module produced some errors: rush instead of rash, azithromisin instead of azithromycin and parasitamol instead of paracetamol. Excluding those issues, the transcript is correct and complete.

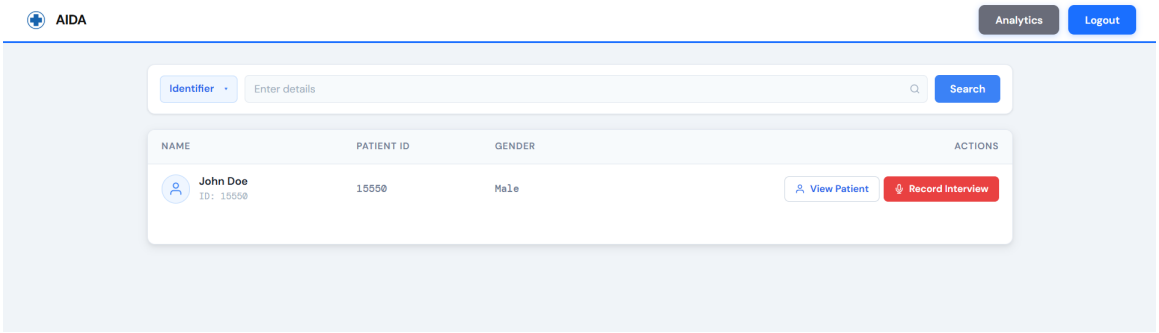
4.2.1.3 Extracted JSON output (MedGemma):


```
{
  "patient": {
    "name": "John Doe",
    "age": 34,
    "gender": "male"
  },
  "encounter": {
    "reason": "sore throat and fever"
  },
  "symptoms": [
    {
      "text": "sore throat",
      "present": true,
      "duration": "past three days",
      "severity": null
    },
    {
      "text": "fever",
      "present": true,
      "duration": "past three days",
      "severity": null
    },
    {
      "text": "runny nose",
      "present": true,
      "duration": null,
      "severity": null
    },
    {
      "text": "mild headache",
      "present": true,
      "duration": null,
      "severity": null
    },
    {
      "text": "dry cough",
      "present": true,
      "duration": null,
      "severity": null
    },
    {
      "text": "shortness of breath",
      "present": false,
      "duration": null,
      "severity": null
    }
  ],
  "conditions": [
    {
      "text": "upper respiratory tract infection"
    }
  ],
  "medications": [
    {
      "text": "azithromycin",
      "dosage": "500 milligram",
      "status": "active"
    },
    {
      "text": "paracetamol",
      "status": "active"
    }
  ],
  "allergies": [
    {
      "text": "penicillin",
      "reaction": "rush"
    }
  ]
}
```


The extraction was correct and complete. All six symptoms were identified, including the

negated symptom (shortness of breath) which was assigned a present value of false as required by the prompt. The model correctly identified the prescribed medication as azithromycin and paracetamol even though there was a typographical error in the generated transcript. The penicillin allergy was captured along with the reaction description, all though in this case it wasn't able to fix the error (rush instead of rash). The FHIR mapping layer generated one Patient resource, one Encounter resource, six Observation resources, one Condition resource, two MedicationStatement resources, and one AllergyIntolerance resource, all of which were successfully submitted to the HAPI FHIR server as a transaction bundle.

We can also see that the new patient was added to the list of existing patients:



4.2.1.4 Assessment

Pipeline Stage	Output	Correctness
ASR Transcription	Accurate transcript with a few typographical errors	Mostly correct
Fact Extraction	All symptoms, conditions, medications, and allergies captured correctly	Yes
Negation Handling	Correct identification of negated clinical findings (e.g. no shortness of breath)	Yes
Medication Correction	Medication names extracted and corrected	Yes
FHIR Generation	1 Patient, 1 Encounter, 6 Observations, 1 Condition, 2 Medications, 1 Allergy	Yes
HAPI Submission	Transaction bundle accepted, all resources persisted	Yes

Table 4.1: Scenario 1: Evaluation

4.2.2 Scenario 2: Paediatric Consultation with Multiple Allergies

This scenario involves a paediatric patient brought in by a parent, presenting with a skin rash and gastrointestinal symptoms. The consultation mentions two distinct allergies with different reaction profiles. This scenario tests whether AIDA can extract the multiple allergies as well as handle indirect speech, since the patient's history is provided by the accompanying parent rather than the patient directly.

4.2.2.1 Consultation Transcript:

Parent: *Hi doctor, this is my daughter Maria Agathokleous she is 8 years old*
Doctor: *Good morning. What seems to be the problem with the little one today?*
Parent: *She has had a red, itchy rash on her arms and stomach since yesterday afternoon. She also vomited twice last night and says her tummy hurts.*
Doctor: *Has she eaten anything new recently?*
Parent: *We had a birthday party and she ate some food with nuts. She is allergic to peanuts, gets hives and her throat swells up a little. She is also allergic to latex, it gives her a rash.*
Doctor: *Has she had any difficulty breathing?*
Parent: *No, nothing like that.*
Doctor: *Okay, this is consistent with a mild allergic reaction. I will prescribe cetirizine 5mg once daily for five days for the itching and loratadine if the cetirizine is not enough. Monitor her closely. If the throat swelling worsens, go to the emergency department immediately. For the record: patient is female*

4.2.2.2 Generated Transcript (Whisper):

Hi Doctor, this is my daughter Maria Agathocleus. She is 8 years old. Good morning. What seems to be the problem with the little one today? She has had a red ***itchie rush*** *on her arms and stomach since yesterday afternoon. She also vomited twice last night and says her tummy hurts. Has she eaten anything new recently? We had a birthday party and she ate some food with nuts. She is allergic to peanuts get hives and her throat swells up a little. She's also allergic to latex. It gives her a* ***rush****. Has she had any difficulty breathing? No, nothing like that. Okay, this is consistent with a mild allergic reaction. I will prescribe* ***ceteerizin*** *5 milligram once daily for 5 days for the itching and loratadine if the* ***ceteerizin*** *is not enough. Monitor her closely. If the throat swelling worsens, go to the emergency department immediately. For the record, patient is female.*

Once again, the transcript is mostly correct with minor typographical errors: rush instead of rash (like previously) but most importantly it got one of the medicine names wrong: ceteerizin instead of cetirizine. This is acceptable since the results are presented to the doctor for evaluation, but it would have been better if it extracted all medicine names correctly.

4.2.2.3 Extracted JSON output (MedGemma):


```
{
  "patient": {
    "name": "Maria Agathocleus",
    "age": 8,
    "gender": "female"
  },
  "encounter": {
    "reason": "allergic reaction"
  },
  "symptoms": [
    {
      "text": "red itchy rash",
      "present": true,
      "duration": "since yesterday afternoon",
      "severity": null
    },
    {
      "text": "vomited",
      "present": true,
      "duration": "last night",
      "severity": null
    },
    {
      "text": "tummy hurts",
      "present": true,
      "duration": null,
      "severity": null
    },
    {
      "text": "difficulty breathing",
      "present": false,
      "duration": null,
      "severity": null
    }
  ],
  "conditions": [
    {
      "text": "allergic reaction"
    }
  ],
  "medications": [
    {
      "text": "ceteerizin",
      "dosage": "5 milligram",
      "status": "active"
    },
    {
      "text": "loratadine",
      "dosage": null,
      "status": "active"
    }
  ],
  "allergies": [
    {
      "text": "peanuts",
      "reaction": "hives, throat swells up"
    },
    {
      "text": "latex",
      "reaction": "rush"
    }
  ]
}
```

All symptoms, medications and allergies were extracted correctly based on the generated transcript. An important detail is that the LLM actually corrected one of the typographical errors from the context: it extracted ”red itchy rash” correctly, even if the transcript contained ”red itchie rush”. Another notable observation is that loratadine was recorded with a status of active despite being mentioned as a contingency prescription. The prompt instructs the model to record explicitly stated medications, and as loratadine was mentioned as a prescription in the consultation it was included, which is technically consistent with the prompt even if clinically it should perhaps be recorded differently. Also worth mentioning, is the reason why the doctor provided the patient’s gender in the consultation. After some exhaustive testing and prompt tuning, we have realised that the LLM had difficulties identifying gender through pronouns only, especially when the patient’s history is provided by another person. So mentioning it directly ensures that the LLM provides the correct output.

4.2.2.4 Assessment

Pipeline Stage	Output	Correctness
ASR Transcription	Accurate transcript with a few typographical errors	Mostly correct
Fact Extraction	All symptoms, conditions, medications, and allergies captured correctly	Yes
Negation Handling	Correct identification of negated clinical findings	Yes
Medication Correction	Both medications extracted but only one corrected	Mostly correct
Allergy and Reaction extraction	Both allergies were extracted correctly with their reactions	Yes
FHIR Generation	1 Patient, 1 Encounter, 4 Observations, 1 Condition, 2 Medications, 2 Allergies	Yes
HAPI Submission	Transaction bundle accepted, all resources persisted	Yes

Table 4.2: Scenario 2: Evaluation

4.2.3 Scenario 3: Conversational Consulation with Implicit Clinical Information

This scenario is the most linguistically challenging of those tested. It is a highly informal, conversational consultation where almost all clinically relevant information is conveyed implicitly, without the physician-patient structure that the ASR and extraction stages are optimised for. Several difficulties arise.

4.2.3.1 Consultation Transcript:

Doctor: *So how are things? Still smoking?*
Patient: *Trying to cut down. I was on twenty a day, now maybe ten.*
Doctor: *Good. How's the knee?*
Patient: *The right one is still bad. It locks up sometimes when I climb stairs. The left one is fine now actually, been fine for months.*
Doctor: *Are you still taking the ibuprofen for it?*
Patient: *On and off. I don't like taking it every day. My wife makes me take omeprazole with it because of the stomach.*
Doctor: *Smart. Has the physio helped?*
Patient: *Went twice, then stopped going. It's too far.*
Doctor: *You really should continue. Any other problems?*
Patient: *My sleep is terrible. I wake up at three in the morning and can't get back to sleep. Been like this for maybe six weeks.*
Doctor: *Okay. Let's refer you to the sleep clinic.*

4.2.3.2 Generated Transcript (Whisper):

So how are things? Still smoking? Trying to cut down. I was on 20 a day, now maybe 10. Good, how's the knee? The right one is still bad. It locks up sometimes when I climb stairs. The left one is fine now actually, been fine for months. Are you still taking the ibuprofen for it? On and off, I don't like taking it every day. My wife makes me take omeprazole with it because of the stomach. Smart. Has the physio helped? Went twice, then stopped going. It's too far. You really should continue. Any other problems? My sleep is terrible. I wake up at 3 in the morning and can't get back to sleep. Been like this for maybe six weeks. Okay, let's refer you to the sleep clinic.

The transcript is fully correct and accurate, with all medication names captured correctly and no other typographical error.

4.2.3.3 Extracted JSON output (MedGemma):


```
{
  "patient": {
    "name": Null,
    "age": Null,
    "gender": Null
  },
  "encounter": {
    "reason": "General check-up"
  },
  "symptoms": [
    {
      "text": "knee pain",
      "present": true,
      "duration": null,
      "severity": null
    },
    {
      "text": "knee locking",
      "present": true,
      "duration": null,
      "severity": null
    },
    {
      "text": "terrible sleep",
      "present": true,
      "duration": "six weeks",
      "severity": null
    }
  ],
  "conditions": [
    {
      "text": "smoking"
    }
  ],
  "medications": [
    {
      "text": "ibuprofen",
      "dosage": null,
      "status": "active"
    },
    {
      "text": "omeprazole",
      "dosage": null,
      "status": "active"
    }
  ],
  "allergies": []
}
```


The extraction output has a mixture of correct, partially correct, and incorrect results that reveal the system's difficulties with informal, conversational clinical language.

The encounter reason was recorded as "General check-up", which is a reasonable inference but is not explicitly stated anywhere in the transcript. Due to the absence of a clear chief complaint, the model imposed this label, which is an example of inference beyond the transcript.

Smoking was classified as a condition and physiotherapy was not captured at all. The system has no implementation of FHIR resources for these scenarios. Proper matching would be the case where smoking is classified as a Social History Observation and physiotherapy as a Procedure Resource. While this is understandable given the schema's constraints and the current implementation of the system, it highlights the gap in resource coverage identified in Section 4.4.

The referral to the sleep clinic was also lost due to the resource coverage limitation.

The knee symptoms were also partially captured: knee pain and knee locking were both extracted, which is correct for the right knee. However, the resolved left knee was not mentioned at all and the left/right knee clarification was lost entirely. A physician reviewing these records would have no indication of which knee is affected or that the other knee had previously been involved and resolved.

The sleep symptom with the six weeks duration was correctly extracted.

Both medications were recorded as active with no dosage, which is partially correct. The intermittent nature of ibuprofen (”on and off”) and the informal, non-physician-prescribed nature of the omeprazole are both lost.

4.2.3.4 Assessment

Pipeline Stage	**Output**	**Correctness**
ASR Transcription	Fully Accurate	Yes
Encounter Reason	Inferred not stated	Partial
Social history (smoking)	Not extracted: no schema field available	No
Physiotherapy (discontinued)	Not captured: Procedure resource not implemented	No
Sleep clinic referral	Not captured: ServiceRequest resource not implemented	No
Resolved symptom	Not captured: only active symptom extracted	No
Medication Correctness	Both medications recorded as active. Lost intermittent status and non-prescribed context	Partial
Insomnia Symptom	Correctly extracted with six week duration	Yes
FHIR Generation	1 Encounter, 3 Observations, 1 Condition, 2 Medications	Yes
HAPI Submission	Transaction bundle accepted, all resources persisted	Yes

Table 4.3: Scenario 3: Evaluation

4.3 GraphRAG Evaluation

For the evaluation of the GraphRAG analytics engine we loaded data for 50 fictional patients, including the ones in the scenarios above, with different medical records and issued a number of natural language queries to examine the results.

4.3.1 Symptom Search

Query 1:
Physician Query: Which patients are allergic to penicillin and what reaction do they have?

The planner classified this as a *find_by_symptom* query with anchor_type of *allergy*. The LLM response correctly identified the two patients that match the query along with their reactions.

Query 2:
Physician Query: Which patients have low vitamin D and what are their levels?

The planner classified this as a *find_by_symptom* query with anchor_type *condition*. Within the patient population, one patient had an explicit diagnosis of vitamin D deficiency, while another exhibited reduced vitamin D levels without a formally coded diagnosis.

The LLM successfully identified both patients and correctly differentiated between a confirmed clinical condition and a laboratory finding suggestive of insufficiency:

- **Patient 1071 (Katerina Nikolaou, female)**
  - Observation: Vitamin D: 28 nmol/L
  - Condition: Vitamin D deficiency [active]
- **Patient 1548 (Kyriakos Philippou, male)**
  - Observation: Vitamin D: 52 nmol/L

(This level is considered insufficient, not deficient, according to standard cutoffs which often use <20 nmol/L as deficient and <30 nmol/L as insufficient).

4.3.2 Population Statistics

Query 1:
Physician Query: What are the most common conditions across all my patients?

The planner classified this as a *population_stats* query. The LLM correctly recognised *hypertension* as the main condition across the patient population. However, an important subtlety should be noted: the dataset also contains cases of *ocular hypertension* and *portal hypertension*. The LLM successfully differentiated between these clinically distinct conditions and identified

the more prominent one, but this does not guarantee that it distinguishes correctly in all cases, especially when the query is ambiguous or terminology overlaps semantically. This risk could be alleviated through the use of standardized clinical terminology systems such as SNOMED CT. The absence of terminology normalization is therefore a limitation of the current implementation and is discussed further in Section 4.5.

Query 2:
Physician Query: Which patients are at risk of cardiovascular disease?

The planner incorrectly classified this as a *find_similar* query. The LLM stated that it could not definitively assess the risk of cardiovascular disease (CVD) for any individual patient without more comprehensive information and then it presented various patients that it believes have some potential risk factors. The patients mentioned are the ones retrieved from the *find_similar* traversal so it is not complete since it did not check the whole population.

Even if the planner were correct, concepts like cardiovascular risk are derived rather than recorded, they must be inferred from a combination of raw values, clinical thresholds, and established risk criteria. The LLM receives raw observation values as plain text with no reference ranges, no flagging of abnormal results, and no temporal ordering to detect worsening trends. Any risk assessment it produces is therefore based on unstructured inference over incomplete context, with no guarantee of consistency or completeness across patients.

4.3.3 History Retrieval

Query 1:
Physician Query: What medication is patient Petros Vassiliou (ID: 200006) currently on?

The planner classified this as a *get_history* query. The LLM correctly retrieved the medications currently prescribed to the patient and additionally identified the associated clinical conditions to provide context for their use. Beyond simple retrieval, the model also generated treatment-related decision support information. The response appropriately emphasized that any treatment decisions or modifications should remain under the supervision of the responsible clinician.

This result demonstrates the system's ability to combine patient-specific medication history with contextual clinical reasoning while maintaining appropriate safeguards regarding medical decision-making.

4.3.4 Record Comparison

Query 1:
Physician Query: What do patients Voula Demetriou (ID: 200017) and Nikos Aristodemou (ID: 200034) have in common?

The planner classified this as a *compare* query with anchor_type: *patient_id*, anchor_value: *200017* and extra_patient_ids: *['200034']*. The retrieval engine successfully gathered the relevant records for both patients and the LLM correctly identified that both individuals had been diagnosed with different forms of cancer. However, the response also included an incorrect observation, stating that the two patients had the same weight despite this not being supported by the underlying data.

4.3.5 Conclusion

Overall, the GraphRAG analytics engine demonstrates good performance. It correctly handles the majority of cases but it shows particular strength in queries that have well-defined anchors, such as retrieving a specific patient's medication history or identifying patients with a recorded allergy. However, the evaluation also exposes clear limitations. Clinical concepts that are derived instead of explicitly stated such as cardiovascular risk cannot be reliably assessed from raw observation values alone. The absence of standardised clinical terminology introduces ambiguity in queries where conditions overlap semantically. Additionally, some hallucinations were observed, highlighting that LLM-generated responses require clinician oversight and should not be consumed without verification.

4.4 Benefits of AIDA

The evaluation scenarios demonstrate several benefits of the AIDA system that are worth examining.

Automation of Documentation

The main benefit of AIDA is that it converts the unstructured physician-patient dialogue into structured, FHIR records without any manual data entry by the physician. The time a physician would spend typing symptoms, conditions, and medications into an EHR is replaced by this pipeline, directly addressing the documentation burden identified in the literature review.

Natural Language Clinical Analytics

AIDA provides a capability that goes beyond what a standard EHR system offers: the ability for a physician to query their entire patient population using natural language and receive AI-generated responses grounded in structured patient data.

The graph-structured retrieval approach makes this particularly powerful for queries that span multiple resource types within a single patient record or across the population, preserving relational context such as which observations belong to which encounter, or which conditions co-occur with which medications. This relational retrieval is difficult to achieve with flat search

or vector similarity approaches alone.

When the retrieved context is sufficient, this analytics capability positions AIDA as a lightweight decision support tool intended to augment, not replace, physician judgment.

Standards Compliance and Interoperability

All data that AIDA generates, is persisted as HL7 FHIR resources to a standards-compliant HAPI FHIR. This means that the output is not specific to this system. Any other system that is compatible with FHIR, such as a hospital information system, could in principle retrieve and consume this data via the FHIR REST API. This interoperability property is a significant advantage over custom documentation formats.

Privacy-Preserving Local Architecture

A characteristic that distinguishes AIDA from cloud-based alternatives is that all AI inference is performed locally. Patient audio and transcripts are never transmitted to any third-party service. This is a critical property for healthcare deployments due to data protection regulations such as GDPR, where processing sensitive personal information via external APIs without explicit data processing agreements is legally problematic. The use of open-weight models (Whisper, MedGemma) and a locally hosted FHIR server means the entire pipeline operates on infrastructure entirely controlled by the provider.

Modular and Replaceable Architecture

The system was designed with modularity as a core principle. As the rapidly evolving sector of clinical AI produces better ASR models or more accurate medical LLMs, individual components can be upgraded without requiring changes to the rest of the pipeline. For example, replacing Whisper with a newer ASR model or substituting MedGemma with a stronger medically fine-tuned model requires only minor changes to the relevant layers.

4.5 Limitations of AIDA

Synthetic Evaluation Data

All evaluation scenarios used are manually written scripts that were read aloud in controlled conditions, rather than real physician-patient consultations. Real clinical conversations contain features that the synthetic scripts do not fully capture: overlapping speech, mid-sentence self-corrections, heavy use of medical abbreviations, background noise from clinical environments and non-native speaker disfluencies. The system's performance on real consultations may therefore differ from the results reported here.

LLM Hallucination and Schema Adherence

MedGemma, like all LLMs, is susceptible to hallucinations. While the extraction prompt explicitly instructs the model not to infer diagnoses or add medical knowledge that is not present in the transcript there is no formal guarantee that the model will always comply with these instructions. The model's clinical pre-training may lead it to complete patterns it recognises and therefore a production deployment would require a validation layer that checks extracted facts against the transcript before FHIR resources are generated.

Absence of Standardised Clinical Terminology Encoding

The FHIR resources generated by the system do not include standardised terminology encodings. Clinical concepts such as diagnoses, medications, conditions and allergies are stored as plain-text strings extracted directly from the transcript, without being mapped to codes from established vocabularies such as SNOMED CT, LOINC, or RxNorm. FHIR resources are designed to carry coded representations of clinical concepts within their coding fields precisely to enable interoperability across systems. Without these encodings, the generated resources cannot be meaningfully exchanged with external EHR systems, clinical decision engines, or population health analytics platforms that rely on standardised terminologies. Even the GraphRAG engine would perform better with standarised terminology since a condition or a medication would always be represented in the same way and not based on the wording of the consultation. A production deployment would require an automated coding step that maps extracted text to the appropriate standardized terminology codes before the FHIR resources are persisted.

Limited FHIR Resource Coverage

The system generates only six FHIR resource types: Patient, Encounter, Observation, Condition, MedicationStatement, and AllergyIntolerance. The FHIR standard provides a excessive list of resources that should be implemented for a full medical record system. The current selection is sufficient for demonstrating the concept but would need to be extended substantially to cover the full range of information that arises in real clinical consultations.

Scalability of the Single-Prompt Extraction Architecture

The current fact extraction stage relies on a single LLM prompt that extracts all relevant information from a transcript in one go. This produces a fixed JSON schema that is then deterministically mapped to the six FHIR resource types implemented. While this architecture is appropriate for the limited resource coverage of the current proof-of-concept, it does not scale gracefully to a more complete FHIR implementation.

As demonstrated in Scenario 4.2.3, the system drops clinical data simply because the extraction schema has no fields for them. Extending the schema to cover additional resource types such as, Procedure, ServiceRequest, CarePlan, FamilyMemberHistory and more, would rapidly increase the complexity and length of the extraction prompt and would make it increasingly difficult to

engineer properly and reliably. As the prompt and schema grow, the model's ability to follow all instructions degrades and the risk of hallucination increases.

A more scalable approach for a production system would be an agentic architecture. This would work in the following way: A coordinating agent first analyses the transcript and identifies which FHIR resource families are relevant to the consultation. Then, it dispatches specialised sub-agents, each responsible for extracting facts for a specific resource family, operating with a prompt tuned to that domain. For example, a dedicated medication agent could be responsible for medication names, dose histories, stopped medications, and conditional prescriptions. Similarly, a dedicated lab results agent could be specifically prompted to extract numerical values and units, a task the current single-prompt schema cannot accommodate at all.

This approach would also make the system more maintainable. Adding support for a new FHIR resource type simply means adding a new specialised agent rather than modifying the monolithic prompt which could in turn break behaviour across all other resource types and would need to be revalidated. The linear pipeline architecture chosen for AIDA was deliberately preferred over an agentic approach for its simplicity and predictability, as discussed in Section 3.1, but this advantage diminishes, especially in the fact extraction step of the pipeline, as the amount of required resource coverage grows. A hybrid design, where the outer pipeline remains linear and deterministic while the extraction stage is internally agentic, represents a natural improvement for future work.

Inference Speed

Running both Whisper and MedGemma on CPU hardware without GPU acceleration results in inference times that are substantially longer than they would be on a GPU-equipped server. Deploying the system on a machine with even a modest GPU would reduce inference time by an order of magnitude.

4.6 Summary

This chapter evaluated AIDA through a qualitative, scenario-based methodology. Three physician-patient consultation scenarios were processed through the full pipeline, demonstrating that the system correctly transcribes clinical dialogue, extracts structured facts including negated symptoms, revised prescriptions, and multiple allergies, generates valid FHIR resources, and persists them to the HAPI FHIR server. It also demonstrated the GraphRAG analytics engine's ability to perform patient history retrieval, population-level symptom queries, and condition summarisation, with responses grounded in the actual graph-retrieved data.

The key benefits of the system are its documentation automation, standards-based FHIR output, privacy-preserving local architecture, its modular architecture and clinically meaningful graph-

structured analytics. The principal limitations are the LLM hallucination risk, limited FHIR resource coverage, the non-scalable extraction stage and the CPU-constrained inference speed, which can be alleviated by running on infrastructure with a GPU. These limitations define a clear set of directions for future work that would be required to transition AIDA from a research prototype to a production-ready clinical tool.

5 Conclusion

5.1 Summary of Contributions and Findings

This dissertation presented AIDA, an AI-powered Doctor Assistant designed to reduce the burden of clinical documentation on physicians by automating the production of standards-compliant, interoperable health records while also providing a data analysis engine for decision support. The motivation for this system was driven by a well-documented and growing problem: physicians spend a disproportionate amount of their working hours on documentation which actively contributes to burnout, reduced care quality and systemic inefficiency across healthcare organisations. Commercial solutions that address the documentation burden exist, but they produce free-text clinical notes rather than structure and interoperable records while also transmitting sensitive patient data to third-party services.

AIDA addresses the clinical documentation burden, the interoperability gap, and concerns regarding third-party handling of sensitive patient data through a six-stage pipeline that operates entirely on controlled infrastructure. The pipeline begins with the audio capture of a physician-patient consultation, proceeds to audio transcription via Whisper and clinical fact extraction via MedGemma and concludes with the generation of valid FHIR resources that are persisted to a HAPI FHIR server. A knowledge graph is then constructed over the stored resources and is used by the GraphRAG analytics engine to enable natural-language querying over the patient population. Every AI inference step runs locally using open-weight models, ensuring that patient data never leaves the provider's infrastructure. This architectural choice directly addresses one of the principal barriers to AI adoption in clinical settings: the conflict between cloud-based AI services and patient data protection regulations such as GDPR.

The evaluation demonstrated that the system achieves its primary objectives across a range of clinical scenarios. In the first two scenarios, a straightforward respiratory consultation and a paediatric consultation involving multiple allergies and indirect speech, the pipeline correctly transcribed dialogue with minor typographical errors, extracted structured clinical facts including negated symptoms and multiple allergies, generated valid FHIR bundles, and submitted them successfully to the HAPI FHIR server. The third scenario, a highly informal and conversational consultation with implicit information, produced more mixed results that exposed the system's current boundaries. While the main symptom and both medications were identified, several clinically important details were lost: smoking was misclassified as a condition rather than a social history observation and physiotherapy as well as the referral to another clinic were dropped entirely due to the absence of corresponding FHIR resource types. These failures were

caused by the limited FHIR resource coverage and the constraints of the single-prompt extraction schema rather than to fundamental weaknesses in the approach, and they define a clear direction for future work. The GraphRAG engine demonstrated particular strength on queries with well-defined anchors, such as retrieving a specific patient's medication history, identifying patients with a recorded allergy, or summarising population-level condition prevalence. However, it struggled with queries requiring derived clinical reasoning, such as cardiovascular risk assessment, where the absence of reference ranges, flagging of abnormal results and temporal ordering limits the reliability of any inference. The absence of standardised clinical terminology could also introduce ambiguity in queries where conditions overlap semantically. Lastly, some isolated cases of hallucination were observed in the responses, reinforcing the need for clinician oversight and verification of LLM-generated analytics before use.

The evaluation identified several limitations. The system generates only six FHIR resource types, which is sufficient for this proof-of-concept but insufficient to capture the full range of clinically relevant information, as illustrated by the third scenario, where smoking history, physiotherapy, and a referral were all lost due to the absence of corresponding resource types. Clinical concepts are stored as plain-text strings without mapping to standardised terminologies such as SNOMED CT or LOINC, limiting interoperability with external systems and introducing ambiguity in the GraphRAG engine when conditions overlap semantically. Another fundamental concern is the LLM hallucination risk: the extraction prompt constrains the model's behaviour but provides no formal guarantee against fabricated output, so a production deployment would require a validation layer that verifies extracted facts against the source transcript. Finally, running both Whisper and MedGemma on CPU hardware results in inference times that would be substantially reduced by GPU-equipped infrastructure.

5.2 Directions for Future Work

The limitations identified above define a clear roadmap for transitioning AIDA from a research prototype to a production-ready clinical tool.

The most impactful architectural improvement would be the adoption of an agentic clinical fact extraction stage. The current single-prompt architecture does not scale well with the growth of FHIR resource coverage because by extending the prompt schema to cover additional resource types, the prompt's complexity would increase and a risk of degraded response accuracy would arise. An agentic approach on the other hand, in which a main coordinating agent identifies the relevant resource families for a given consultation and dispatches specialised sub-agents for each family, would make the extraction stage both more capable and more maintainable. This direction is particularly well-suited for integration with systems such as Infherno, whose multi-

agent extraction layer could in principle replace AIDA's single-prompt stage while leaving the downstream pipeline unchanged.

Standardised clinical terminology encoding represents another high-priority extension. Adding an automated coding step that maps extracted text to standard clinical healthcare terminology codes before FHIR resources are persisted, would improve both interoperability with external systems and the precision of the GraphRAG engine's retrieval.

Other important additions would be a hallucination mitigation layer, which cross-checks extracted facts against the source transcript before FHIR generation, as well as authentication and role-based access control.

Taken together, these extensions would build directly on the foundation that AIDA establishes. The core pipeline architecture, speech recognition, LLM-based extraction, FHIR persistence, knowledge graph construction, and graph-based analytics are robust, modular, and designed to accommodate such improvements without requiring great changes to the system as a whole.

This dissertation demonstrates that an integrated, privacy-preserving, standards-compliant AI documentation system is practically achievable on commodity hardware and provides an open-source reference implementation on which future work can build on.

BIBLIOGRAPHY

[1] S. Pavuluri, R. Sangal, J. Sather, and R. A. Taylor, "Balancing act: the complex role of artificial intelligence in addressing burnout and healthcare workforce dynamics," *BMJ Health Care Informatics*, vol. 31, no. 1, p. e101120, 2024. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC11344516/

[2] A. J. Holmgren, C. A. Sinsky, L. Rotenstein, and N. C. Apathy, "National comparison of ambulatory physician electronic health record use across specialties," *Journal of General Internal Medicine*, vol. 39, no. 14, pp. 2868–2870, 2024. [Online]. Available: https://doi.org/10.1007/s11606-024-08930-4

[3] European Commission: Directorate-General for Health and Food Safety, PwC EU Services EEIG, and Open Evidence, "Study on the deployment of ai in healthcare - final report," Publications Office of the European Union, Tech. Rep., 2025. [Online]. Available: https://data.europa.eu/doi/10.2875/2169577

[4] Health Level Seven International (HL7), "Fast healthcare interoperability resources (fhir)," 2026, accessed: 2026-04-09. [Online]. Available: https://hl7.org/fhir/

[5] M. Albrecht, D. Shanks, T. Shah, T. Hudson, J. Thompson, T. Filardi, K. Wright, G. A. Ator, and T. R. Smith, "Enhancing clinical documentation with ambient artificial intelligence: a quality improvement survey assessing clinician perspectives on work burden, burnout, and job satisfaction," *JAMIA Open*, vol. 8, no. 1, p. ooaf013, 2025.

[6] P. J. Lukac, W. Turner, S. Vangala, A. T. Chin, J. Khalili, Y.-C. T. Shih, C. Sarkisian, E. M. Cheng, and J. N. Mafi, "A randomized-clinical trial of two ambient artificial intelligence scribes: Measuring documentation efficiency and physician burnout," 2025, preprint. PMID: 40672471; PMCID: PMC12265753. [Online]. Available: https://www.medrxiv.org/content/10.1101/2025.07.10.25331333

[7] A. A. Tierney, G. Gayre, B. Hoberman, B. Mattern, M. Ballesca, S. B. Wilson Hannay, K. Castilla, C. S. Lau, P. Kipnis, V. Liu, and K. Lee, "Ambient artificial intelligence scribes: Learnings after 1 year and over 2.5 million uses," *NEJM Catalyst Innovations in Care Delivery*, vol. 6, no. 5, 2025.

[8] Microsoft, "Nuance's dragon ambient experience (dax) helps doctors document care faster so they can spend more time with patients delivering high-quality care with ai on azure," 2022, microsoft Customer Story, accessed: 2026-05-04. [Online]. Available: https://www.microsoft.com/en/customers/story/1415193159282858383-nuance-partner-professional-services-azure-machine-learning

[9] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, M. Amin, L. Hou, K. Clark, S. R. Pfohl, H. Cole-Lewis, D. Neal, Q. M. Rashid, M. Schaekermann, A. Wang, D. Dash, J. H. Chen, N. H. Shah, S. Lachgar, P. A. Mansfield, S. Prakash, B. Green, E. Dominowska, B. Agüera y Arcas, N. Tomašev, Y. Liu, R. Wong, C. Semturs, S. S. Mahdavi, J. K. Barral, D. R. Webster, G. S. Corrado, Y. Matias, S. Azizi, A. Karthikesalingam, and V. Natarajan, "Toward expert-level medical question answering with large language models," *Nature Medicine*, vol. 31, pp. 943–950, 2025. [Online]. Available: https://www.nature.com/articles/s41591-024-03423-7

[10] A. Sellergren, S. Kazemzadeh, T. Jaroensri *et al.*, "Medgemma technical report," 2025, accessed: 2026-04-11. [Online]. Available: https://arxiv.org/abs/2507.05201

[11] Y. Li, H. Wang, H. Z. Yerebakan, Y. Shinagawa, and Y. Luo, "FHIR-GPT enhances health interoperability with large language models," *NEJM AI*, vol. 1, no. 8, p. AIcs2300301, 2024. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC12312630/

[12] J. Frei, N. Feldhus, L. Raithel, R. Roller, A. Meyer, and F. Kramer, "Infherno: End-to-end agent-based FHIR resource synthesis from free-form clinical notes," 2025, accessed: 2026-05-05. [Online]. Available: https://arxiv.org/abs/2507.12261

[13] P. Jiang, C. Xiao, M. Jiang, P. Bhatia, T. Kass-Hout, J. Sun, and J. Han, "Reasoning-enhanced healthcare predictions with knowledge graph community retrieval," 2024, accessed: 2026-05-09. [Online]. Available: https://arxiv.org/abs/2410.04585

[14] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 28 492–28 518. [Online]. Available: https://proceedings.mlr.press/v202/radford23a.html

[15] S. Gandhi, P. von Platen, and A. M. Rush, "Distil-whisper: Robust knowledge distillation via large-scale pseudo labelling," 2023, accessed: 2026-05-04. [Online]. Available: https://arxiv.org/abs/2311.00430

[16] G. Gerganov, "llama.cpp: LLM inference in C/C++," 2023, accessed: 2026-04-13. [Online]. Available: https://github.com/ggml-org/llama.cpp

[17] HAPI FHIR Project, "HAPI FHIR: The Open Source FHIR API for Java," 2025, accessed: 2026-04-13. [Online]. Available: https://hapifhir.io/hapi-fhir/

[18] ——, "Get Started - HAPI FHIR Documentation," 2025, accessed: 2026-04-13. [Online]. Available: https://hapifhir.io/hapi-fhir/docs/server_jpa/get_started.html

[19] ——, "Database Support - HAPI FHIR Documentation," 2025, accessed: 2026-04-13. [Online]. Available: https://hapifhir.io/hapi-fhir/docs/server_jpa/database_support.html

APPENDICES

APPENDIX I

Prompt Templates

I.1 MEDGEMMA_PROMPT

```
Extract only explicitly stated facts from the text.
Do not infer diagnoses.
Do not add medical knowledge.
If doctor mentions medication but does not actually prescribe it to the patient, do not include it.
Medication example:
{{
     "text": amlodipine,
     "dosage": 50mg daily,
     "status": "active"
}}
Return ONLY valid JSON and nothing else. The Schema is provided below

Schema:
{{
  "patient": {{
    "name": string or null,
    "age": number or null,
    "gender": string or null
  }},
  "encounter": {{
    "reason": string or null
  }},
  "symptoms": [
    {{
      "text": string,
      "present": boolean,
      "duration": string or null,
      "severity": string or null
    }}
  ],
  "conditions": [
    {{
      "text": string (condition_name)
    }}
  ],
  "medications": [
    {{
      "text": string,
      "dosage": string or null,
      "status": "active" | "stopped" | null
    }}
  ],
  "allergies": [
    {{
      "text": string,
      "reaction": string or null
    }}
  ]
}}

----
If a symptom or condition is mentioned but negated, for example: "do you have a headache. No I don't"
     you must still include it like this:
"symptoms": {{"text": headache, "present": false}}
OR
"Any chest pain", "No" -> "symptoms": {{"text": chest pain, "present": false}}

Transcript:
{text}

Respond with the JSON object only.
```

I.2 PLANNER_PROMPT

```
You are a medical query parser. Extract a structured graph traversal plan from
    the doctor's question.

Return ONLY a valid JSON object.
DO NOT include explanations, markdown, or any text before or after the JSON.
DO NOT wrap the JSON in code fences

Traversal goal guide:
- find_similar: "patients like X", "similar symptoms to X", "who else has what X
    has". RULE: only ONE patient is named and you are searching the population for
    matches.
- find_by_symptom: "who has diabetes", "patients with fever", no specific patient
    anchor
- get_history: "history of patient X", "what does patient X have", "show me
    patient X"
- compare: "compare patient A and B", "differences between X and Y", "what do A
    and B have in common", "similarities between patient A and B". RULE: if two or
    more specific patients are named, ALWAYS use compare.
- population_stats: "common conditions", "most frequent", "across all patients"


Return ONLY a valid JSON object with exactly these fields:

If more than one patient IDs are present:
- Put the FIRST patient ID in "anchor_value"
- Put ALL remaining patient IDs in "extra_patient_ids"

Schema
{{
  "traversal_goal": one of ["find_similar", "get_history", "compare",
      "find_by_symptom", "population_stats"],
  "anchor_type": one of ["patient_id", "national_id", "symptom", "condition",
      "medication", "population", "allergy", null],
  "anchor_value": the specific value (patient ID, symptom name, etc.) or null if
      not applicable,
  "extra_patient_ids": list of additional patient IDs for comparison queries (may
      be empty),
  "filters": list of any extra constraints mentioned (e.g. ["active conditions
      only", "female patients"])
```

```
}}


(Respond with complete JSON. Do not stop early. The JSON format must match
    exactly)
```

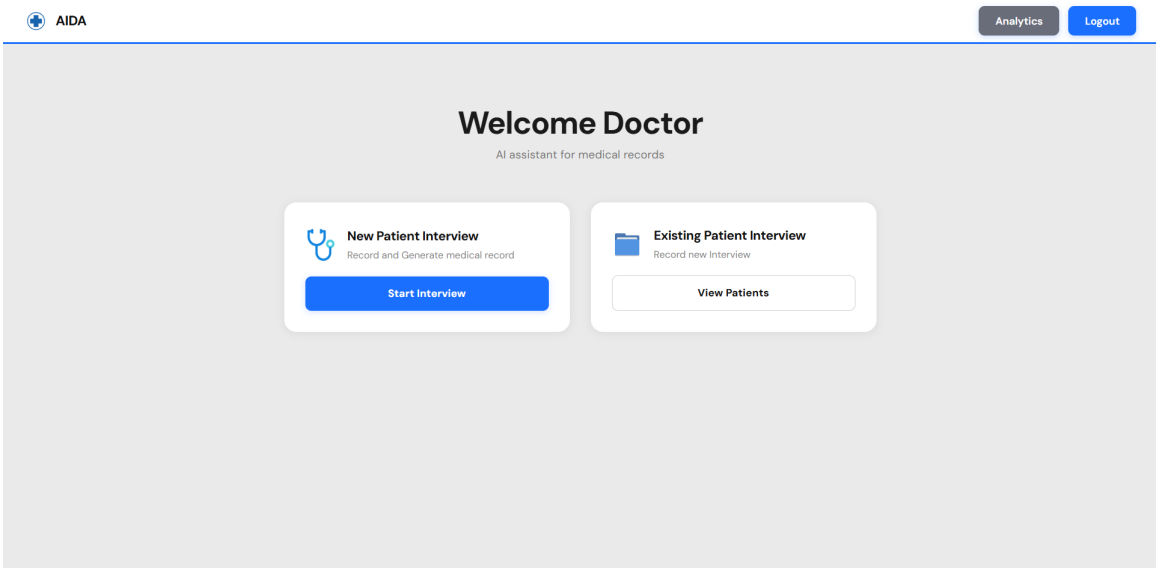
I.3 SYSTEM_PROMPT

```
You are a clinical decision support assistant helping doctors analyze patient
    data.
You have access to structured medical records including encounters, observations,
    conditions, medications, and allergies.

When analyzing patient data:
- Identify clinically significant patterns (symptom clusters, medication
    interactions, risk factors)
- Compare patients objectively when asked
- Highlight any red flags or notable correlations
- Be concise but thorough -doctors need actionable insights, not lengthy prose
- Always clarify that you are providing decision support, not a diagnosis
- Use medical terminology appropriately
- If no patients match the doctor's query, say so clearly

Format your response with clear sections when appropriate. Be specific and cite
    the data provided.
```

APPENDIX II

System User Interface

II.1 Home Page



II.2 Patient Management Screen