**A**

**Project Report on**

**‘** **Enhancing Trust in clinical AI: A study in Explainable Argumentative Systems’**

**Submitted To**

**DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF MUMBAI**

****

**In the partial fulfilment of the degree of**

**Master of Science (Computer Science)**

**By**

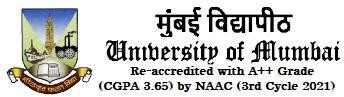
**Keval Sing Indrabahadur Saud**

**Under Guidance Of**

**Dr. Jyotshna Dongardive**

**DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF MUMBAI**

**May 2025**



**Enhancing Trust in clinical AI: A study in Explainable Argumentative System**

**Submitted in the partial fulfilment of the requirements of the degree of**

**Master of Science (Computer Science)**

**By**

**Keval Sing Indrabahadur Saud Seat No: 1294034**

**Under Guidance Of**

**Dr. Jyotshna Dongardive**

**University Department of Computer Science, Kalina, Santacruz (E), Mumbai - 400098**

**Year: 2024-2025**

**UNIVERSITY OF MUMBAI DEPARTMENT OF COMPUTER SCIENCE**

**CERTIFICATE**

This is to certify that, **Keval Sing Indrabahadur Saud** of **M.Sc.** (Computer Science) bearing Examination Seat Number: 1294034 has satisfactorily carried out research project on **Enhancing Trust in clinical AI: A study in Explainable Argumentative Systems** for the year 2024-2025.

External Examiner Project Guide Head of Department

Date:

**ACKNOWLEDGEMENT**

This project was undertaken and expedited under the expert guidance of Dr. Jyotshna Dongardive, for the course titled Project Implementation. The technologies and methodologies of the subjects that were taught, mentored, and supported have been incorporated in this project, without which this project would not have been conceptualized, taken shape and reached the present stage.

I take this opportunity to extend my gratitude for your exemplary guidance, mentorship, and encouragement throughout the course of this project.

Thanking You,

Yours Sincerely,

Keval Sing Indrabahadur Saud

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **Table of Content** | | |
| **Sr. no.** | **Title** | **Page no.** |
| 1 | Introduction | 1-4 |
| 2 | Literature Review | 5-12 |
| 3 | Materials & Methods | 13- 25 |
| 4 | Implementation Details | 26-27 |
| 5 | Experimental Setup and Results | 28-36 |
| 6 | Analysis of the Results | 37-38 |
| 7 | Conclusion | 39 |
| 8 | Future Enhancement | 40-41 |
| 9 | References | 42-43 |
| 10 | Folder Structure | 44-47 |
| 11 | Program Code | 48-53 |
|  |  |  |
|  | List of Tables |  |
| Chapter 3 | Table 1. Clinical Knowledge Sources | 17 |
|  | Table 2. Vector Database Configuratioin | 19 |
|  |  |  |
| Chapter 6 | Table 3. BERTScore Evaluation Metrics for MedicLLM Agents | 37 |
|  |  |  |
| List of Figures | | |
| Chapter 3 | Fig. 1 Data Pre-Processing | 16 |
|  | Fig. 2 Class Diagram of Mutli-Agents Interaction | 20 |
| Fig. 3 Argumentative Framework Architecture | 22 |
| Fig. 4 Graph Diagram of Mutli-Agents Argumentation | 24 |
| Chapter 5 | Fig. 5 Using Docker to run a container | 29 |
|  | Fig. 6 Qdrant Locak Server | 29 |
| Fig. 7 Starting Development Build  IP [**http://127.0.0.1:8000/**](http://127.0.0.1:8000/) | 30 |
| Fig. 8 Home Page  IP [**http://127.0.0.1:8000/**](http://127.0.0.1:8000/) | 31 |
| Fig. 9 User Input | 31 |
| Fig. 10 Form Processing | 32 |
| Fig. 11 Terminal | 32 |
| Fig. 12, Response Structure | 33 |
| Fig. 13 Final Report | 33 |

|  |  |  |
| --- | --- | --- |
|  | Fig. 14 XAI | 34 |
| Fig. 15 Records Structure | 34 |
| Fig. 16 Admin login | 35 |
| Fig. 17 Chat Session | 35 |
| Fig. 18 Bert Score Metrics for Agents with Argumentative Framework | 38 |
|  |  |

# Chapter 1: Introduction

In recent months, innovation in generative artificial intelligence using Large Language Models (LLMs) has accelerated dramatically. The creation of LLM strained by deep neural networks utilizing enormous text datasets and typically having billions of parameters, has significantly advanced natural language processing (NLP)[1]. Google first introduced the "Transformer" architecture for machine translation in 2017, and it subsequently achieved state-of-the-art performance in numerous NLP applications[2]. This led to the development of numerous LLMs with "Transformer" architecture, including BERT (Google) [3], GPT (OpenAI) [4], LLaMA (Meta AI)[5].

The rapid evolution of LLMs, such as OpenAI's GPT-3 and Google’s BERT, has demonstrated their potential to process and generate human-like text by leveraging vast amounts of training data [3], [6]. This development has significantly transformed various fields, including finance, education, transportation, and healthcare, leading to increased innovation and efficiency. In particular, the impact of LLMs on healthcare is remarkable. These models have the potential to greatly improve diagnostic accuracy, refine treatment plans, and enhance patient outcomes. A key advantage of LLMs is their capability to deliver clear and understandable explanations, which helps build trust and comprehension.

The integration of advanced computational models into healthcare has ushered in a new era of clinical decision support systems (CDSS), which are essential for enhancing diagnostic accuracy and improving patient outcomes. In the context of CDSS, LLMs can synthesize information from diverse sources, thereby assisting healthcare professionals in making informed decisions.

At the forefront of this technological revolution is the application of LLMs trained on diverse datasets, including academic literature, clinical guidelines, and patient records. These models, when coupled with Retrieval-Augmented Generation (RAG)[7] techniques and agent-based architectures[8] provide a robust framework for retrieving pertinent information from extensive knowledge bases.

However, the sheer volume of medical literature and the complexity of clinical information necessitate the use of RAG techniques to ensure that the most relevant and up-to-date knowledge is retrieved [7]. RAG is a hybrid approach that combines the generative capabilities of LLMs with retrieval systems to enhance the quality of generated responses. By retrieving relevant documents or data points from a knowledge base and integrating them into the response generation process, RAG significantly improves the accuracy and relevance of the information presented [9]. In a healthcare setting, this means that when a clinician queries the system regarding a specific disease or symptom, the model can retrieve the latest research findings, treatment guidelines, and case studies, thereby providing a comprehensive response that is grounded in evidence.

The incorporation of agent-based architectures further enhances the functionality of CDSS. Agents act autonomously to perform specific tasks, such as monitoring patient data, analyzing trends, and retrieving information from various sources. This autonomy allows for real-time decision-making and facilitates a more interactive experience for healthcare providers. For instance, an agent could continuously analyze a patient's vital signs and medical history, alerting the clinician to any anomalies that may require immediate attention. By combining LLMs with agent-based systems, we have created a dynamic and responsive CDSS that adapts to the needs of both clinicians and patients.

An essential aspect of deploying LLMs and RAG in healthcare is the need for an argumentative framework that allows the system to structure its reasoning and provide justifications for its diagnoses. Argumentative frameworks, which are rooted in formal logic and computational models of argumentation, enable systems to present evidence, counterarguments, and conclusions coherently. In a clinical context, this means that when the model suggests a diagnosis, it can articulate the rationale behind its recommendation by referencing specific studies, guidelines, or patient data. This not only aids in clinician understanding but also fosters trust in the system's recommendations.

Moreover, the use of XAI techniques is crucial for ensuring that the decision-making processes of LLMs are transparent and interpretable. XAI aims to make AI systems more understandable to users by providing insights into how decisions are made. In healthcare, where the stakes are high, clinicians must understand the reasoning behind a model's diagnosis or treatment recommendation. Techniques such as feature importance analysis, saliency maps, and model-agnostic explanations can help elucidate the factors that influenced the model's output [10].This enables the model to not only diagnose diseases but also to elucidate its reasoning through Explainable Artificial Intelligence (XAI)[11] techniques. By integrating XAI into CDSS, healthcare providers can make more informed decisions, ultimately leading to better patient care.

The convergence of LLMs, RAG, agent-based systems, argumentative frameworks, and XAI represents a significant advancement in the development of CDSS within the healthcare domain. This holistic approach not only enhances the accuracy and relevance of clinical recommendations but also addresses the critical need for transparency and interpretability in AI-driven healthcare solutions. As the healthcare landscape continues to evolve, the integration of these technologies will be pivotal in shaping the future of clinical decision-making.

In summary, the combination of LLMs trained on diverse datasets, RAG techniques for information retrieval, agent-based architectures for autonomous decision-making, argumentative frameworks for structured reasoning, and XAI for transparency presents a comprehensive solution for improving CDSS in healthcare. The potential benefits of this integrated approach are vast, ranging from enhanced diagnostic accuracy to improved clinician-patient communication. This introduction aims to explore the intricate interplay between these components and their implications for the healthcare domain

# Objective

1. The first objective is to evaluate how LLMs can improve diagnostic accuracy and treatment planning by processing large-scale medical data.
2. To explore the effectiveness of Retrieval-Augmented Generation (RAG) in improving response relevance and evidence-based recommendations.
3. To analyse how intelligent agents can monitor, retrieve, and analyze patient data dynamically to assist clinicians in decision-making.
4. To enable the system to present coherent evidence-based arguments for diagnoses and treatment options, enhancing clinician trust.
5. Identify XAI methods suitable for healthcare settings and evaluate their impact on clinician understanding and trust in AI recommendations.
6. To measure improvements in clinical workflow, accuracy of care, and communication between patients and providers through system deployment.

# Chapter 2: Literature Review

**Argumentative Reasoning in Clinical Decision Support Systems**

**Title :** Why do humans’ reason? Arguments for an argumentative theory

**Authors :** Hugo Mercier and Dan Sperber

**Publisher :** Cambridge University Press

**Year :** 2011

**Summary :** This work challenges the traditional view of reasoning as a truth-seeking process, proposing instead that its primary function is argumentative—to devise and evaluate arguments to persuade others. The authors suggest that this function is evolutionarily adaptive, given humans' reliance on communication and susceptibility to misinformation. Many common reasoning failures, such as confirmation bias, are reinterpreted as features of reasoning viewed through an argumentative lens. People perform poorly on abstract reasoning tasks when they lack an argumentative context but do well when reasoning to defend a position. However, this kind of reasoning can lead to distorted evaluations, entrenched beliefs, and suboptimal decisions, as individuals favor arguments that support their views rather than seek objective truth. [1]

**Remark :** This perspective offers a provocative yet compelling redefinition of reasoning—not as a tool for discovering truth but as a social device for persuasion and justification. It reframes cognitive biases like confirmation bias not as flaws, but as adaptations aligned with the true function of reasoning. While this view raises concerns about epistemic reliability, it also opens new avenues for designing systems and interventions (e.g., in education, AI, and public discourse) that leverage or mitigate argumentative reasoning. Understanding reasoning as fundamentally persuasive may help explain not only individual biases but also broader phenomena such as polarization and ideological entrenchment..

**Title :** Explaining Qualitative Decision Under Uncertainty by Argumentation

**Authors :** Leila Amgoud and Henri Prade

**Publisher :** National Conference on Artificial Intelligence

**Year :** 2006

**Summary : S**uggested a framework for qualitative decision-making under uncertainty that is based on argumentation. Initially, arguments in favor and against decisions are developed and assessed using either an optimistic or a pessimistic criterion. Then, in order to determine which choice is optimal, decision alternatives are compared according to their arguments in favor or against.[2]

**Remark :** It is possible to design a logical system that directly processes arguments based on their strengths and determines both acceptable and optimal decisions.

**Title :** Using arguments for making and explaining decisions

**Authors :** Leila Amgoud and Henri Prade

**Publisher :** Artificial Intelligence, Elsevier

**Year :** 2009

**Summary :** The first general and abstract argument-based framework for making decisions is presented in this paper. This framework consists of two primary steps. First, classical acceptability semantics are used to construct and assess arguments for options and beliefs. Option pairs are compared using decision principles in the second step. The acknowledged justifications for each option serve as the foundation for decision principles.[3]

**Remark :** Unipolar, bipolar, and non-polar principles are the three different categories of decision principles based on whether they are used in i) only pros or only cons arguments, ii) both types, or iii) an aggregation of them into a meta- argument.

**Title :** [Dominant decisions by argumentation agents](https://link.springer.com/chapter/10.1007/978-3-642-12805-9_3)

**Authors :** PA Matt *et.al*

**Publisher :** International Workshop on Argumentation in Multi-Agent Systems, Springer

**Year :** 2009

**Summary :** Present a unique family of (assumption-based argumentation) frameworks for decision-making benefits. These models can be used to express the knowledge of intelligent agents capable of independently selecting the "best" decision, given the subjective requirements and preferences of the decision- makers they "represent."[4]

**Remark :** Instead of just categorizing decision alternatives as acceptable or unacceptable, the concept of degree of admissibility was introduced.

**Title :** Decision making with assumption-based argumentation

**Authors :** X Fan, F Toni

**Publisher :** Theory and Applications of Formal Argumentation: Second International Workshop Springer

**Year :** 2014

**Summary :** Present two distinct formal frameworks that are used to represent decision- making. Decisions satisfy distinct objectives and have multiple characteristics in both frameworks. Examine a family of decision functions that show choices with various criteria, such as choices that meet all, most, none of the other's goals, and the most desired reachable goals.[5]

**Remark :** A formal model for decision making with Assumption-based argumentation (ABA)

**Title :** An argumentation-based approach to modeling decision support contexts with what-if capabilities

**Authors :** Baroni *et.al*

**Publisher :** AAAI Fall Symposium Series

**Year :** 2009

**Summary :** In order to model the knowledge and reasoning patterns in decision-making, they introduced a set of argument and attack schemes. They used the preferred semantics to compute admissible decisions and mapped the schemes to an Argumentation Framework with Recursive Attacks (AFRA).[6]

**Remark :** The suggested method includes a range of attack and argument schemes meant to represent fundamental concepts and patterns of reasoning for decision assistance.

**Title :** ArgMed-Agents: Explainable Clinical Decision Reasoning with LLM Discussion via Argumentation Schemes

**Authors :** Shengxin Hong *et.al*

**Publisher :** IEEE

**Year :** 2024

**Summary :** This study introduces ArgMedAgents, a multi-agent system that facilitates explainable clinical decision reasoning by LLM-based agents through interaction. ArgMedAgents builds the argumentation process as a directed graph that depicts conflicting relationships after performing self-argumentation iterations using the Argumentation Scheme for Clinical Discussion, a reasoning mechanism for modeling cognitive processes in clinical reasoning. Finally, find several logical and cohesive reasons in favor of the decision using a symbolic solver. ArgMed-Agents allows LLMs to self-directly generate reasoning explanations, simulating the process of clinical argumentative reasoning.[7]

**Remark :** By leveraging directed graphs and the Argumentation Scheme for Clinical Discussion, the system provides a novel and interpretable approach to modeling cognitive processes in medical decision-making. The use of a symbolic solver to derive logical support for decisions enhances both transparency and reliability.

**Title :** Argumentative Large Language Models for Explainable and Contestable Decision-Making

**Authors :** Gabriel Freedman *et.al*

**Publisher :** arXiv (preprint)

**Year :** 2024

**Summary :** The author of this work presents a technique for adding argumentative reasoning to LLMs to balance their advantages and disadvantages. Specifically, provide argumentative LLMs, a technique that builds argumentation frameworks using LLMs and forms the foundation for formal reasoning in decision-making. Any decision made by the augmented LLM may be readily explained to and disputed by humans due to the interpretable nature of these formal reasoning and argumentation frameworks. They provide experimental evidence of the efficacy of argumentative LLMs in the claim verification decision-making task. Additionally, the technology automatically allows for human-computer collaboration and outputs precise uncertainty estimates.[8]

**Remark :** By constructing argumentation frameworks, the proposed technique enables transparent, disputable decision-making—an important step toward fostering trust in AI systems. The inclusion of empirical results on claim verification strengthens the work’s credibility, and the added capability for human-AI collaboration with explicit uncertainty estimation marks a significant advancement.

**Title :** Explanation–Question–Response dialogue: An argumentative tool for explainable AI

**Authors :** Castagna *et.al*

**Publisher :** Argument & Computation

**Year :** 2024

**Summary :** The new Explanation–Question–Response (EQR) dialogue and its characteristics are fully formalized in this paper. Its primary goal is to provide adequate information (i.e., justified by argumentative semantics) while guaranteeing a simpler protocol for both artificial and human agents when compared to other current approaches.[9]

**Remark :** This work addresses a critical concern in AI adoption—the opacity of deep learning systems—by proposing a novel, formally grounded dialogue protocol aimed at enhancing explainability and user trust.

**Title :** Can formal argumentative reasoning enhance LLMs performances?

**Authors :** Castagna *et.al*

**Publisher :** arXiv

**Year :** 2024

**Summary :** Study provides a pipeline (MQArgEng) and initial research to assess how adding computational argumentation semantics affects LLM performance[10]

**Remark :** The proposed engine is a simple plugin tool that has no restrictions on the underlying model or its architecture. The results demonstrate how this engine is sufficient to raise the MT-Bench scores than the baseline.

**Title :** Data-Empowered Argumentation for Dialectically Explainable Predictions

**Authors :** Cocarascu*et.al*

**Publisher :** IOS Press

**Year :** 2020

**Summary :** The paper introduces a novel approach to transparent AI called Data-Empowered Argumentation (DEAr), designed to produce dialectically explainable predictions. In contrast to opaque, black-box models, DEAr builds on the concept of argumentation debates derived from data. These debates involve data-driven arguments that may disagree on labels or interpretations, offering a dialectical structure to prediction and explanation. The authors test DEAr on three types of data: categorical, annotated images, and text, demonstrating that it performs competitively with decision trees (DTs) while inherently providing clear, argument-based explanations[11]

**Remark :** DEAr is a promising step toward interpretable and trustworthy AI, especially in high-stakes domains like healthcare and law where transparency and explanation are crucial. By framing prediction as a dialectical process, DEAr not only matches the performance of traditional transparent models like decision trees but also enriches the interpretability with logical reasoning structures. This approach could pave the way for more human-aligned AI systems, where decision-making is not only accurate but also justifiable and contestable.

**Title :** Improving decision-making performance through argumentation: An argument-based decision support system to compute with evidence

**Authors :** Joshua Introne*et.al*

**Publisher :** Elsevier

**Year :** 2014

**Summary :** This article explores the real-world utility of Argument-Based Systems (ABSs) by introducing a new system called PENDO, designed to aid decision-making—specifically, in predicting housing market trends. While prior research highlights ABSs’ benefits in enhancing thinking and learning, their effectiveness in practical decision-making tasks remains underexplored. PENDO addresses a key human shortfall—evidence-based reasoning—by helping users weigh and aggregate evidence through a computational engine. It also enables the creation of reusable knowledge artifacts. Interestingly, the study finds that an individual’s unaided decision-making ability does not predict their success with PENDO, suggesting that the skills engaged with the tool differ fundamentally from those used in unaided reasoning.[12]

**Remark :** The findings mark a significant development in the field of decision support systems. PENDO not only demonstrates the practical value of ABSs in improving decision outcomes but also reveals that such tools can unlock latent cognitive capabilities in users, independent of their baseline decision-making skills. This opens up new possibilities for designing ABSs that democratize expertise, making complex decision domains more accessible. The disconnect between unaided and tool-assisted performance also underscores the importance of designing intuitive, cognitively-aligned interfaces in decision-support technology.

**Title :** LLM-based Frameworks for API Argument Filling in Task-Oriented Conversational Systems

**Authors :** Jisoo Mok*et.al*

**Publisher :** Association for Computational Linguistics

**Year :** 2024

**Summary :** This paper investigates the use of LLMs for the argument filling task in task-oriented conversational agents. These agents operate in three main phases: API selection, argument filling, and response generation. The focus here is on improving how LLMs extract and supply the necessary information (arguments) from a conversation to call external APIs effectively. The study finds that LLMs struggle with this task unless they undergo a grounding process—a method to anchor their responses in the context of the API schema and dialogue history. To address this, the authors propose new training and prompting strategies that significantly enhance LLMs’ argument filling performance, laying the groundwork for more automated and accurate conversational systems.[13]

**Remark :** This work highlights a critical gap in deploying LLMs for goal-driven, real-world tasks—their need for contextual grounding when interacting with structured systems like APIs. By targeting the argument filling phase, the authors contribute to a more robust integration of LLMs in task-oriented dialogue systems. The proposed grounding techniques not only improve performance but also offer a scalable pathway to building more autonomous and reliable conversational agents, essential for applications in domains like customer service, healthcare, and travel booking.

**Title :** Building More Explainable Artificial Intelligence With Argumentation

**Authors :** Zeng*et.al*

**Publisher :** Proceedings of the AAAI Conference on Artificial Intelligence

**Year :** 2018

**Summary :** This work focuses on advancing explainable AI (XAI) through an argumentation-based approach to decision-making, inspired by how humans reason—by weighing arguments for and against a conclusion. Such an approach provides transparency and interpretability, which are especially critical in sensitive domains like healthcare. The author proposes a hybrid model that combines machine learning techniques (e.g., CNNs for feature extraction from brain images and cognitive tests) with an explainable reasoning framework, creating not only predictions (e.g., early dementia diagnosis) but also detailed, structured explanations.[14]

**Remark :** This work represents a thoughtful and rigorous integration of symbolic reasoning and data-driven learning, addressing one of XAI's core challenges: making machine learning outputs not only accurate but also intelligible. The proposed use of argumentation to mirror human deliberation enhances trust and usability in critical domains like medical diagnostics. Moreover, distinguishing between argument- and context-based explanations is a valuable conceptual innovation, paving the way for more nuanced, user-centered AI systems. The future direction—optimizing explanation quality and expanding application domains—positions this research at the intersection of AI, human cognition, and real-world utility.

# Chapter 3: Materials and Methods

# Materials

### Programming Languages

### Python 3.11 — primary language for all backend, orchestration, and AI-agent code.

### JavaScript (ES6) — dynamic UI widgets, chart rendering, and build tooling.

### Shell & Dockerfile syntax — container build, multi-stage images, and CI scripts.

### Frameworks & Libraries

### Django + Django REST Framework for HTTP routing, authentication, admin console, and ORM.

### LangChain to scaffold LLM chains, memories, and multi-agent interactions.

### Hugging Face Transformers & Sentence-Transformers for domain-tuned text embeddings.

### NetworkX to build, store, and traverse argument-graph structures used by the XAI module.

### WeasyPrint for server-side export of the final clinical report to PDF (embedded CSS/HTML).

### APIs & External Services

### OpenAI GPT-4o API (32k-context) for reasoning, debate rounds, and summary generation.

### Hugging Face Inference API fallback endpoint for local/offline embedding.

### Databases & Storage

### Qdrant 1.8 (Docker image qdrant/qdrant:latest) vector store holding ~112 k clinical embeddings.

### SQLDB relational store for sessions, messages, audit trails, and permissions.

### Containerisation & Virtualisation

### Docker 24 single-image builds for backend, qdrant.

### Development & Collaboration Tools

### Visual Studio Code + Python, Docker, and Pylance extensions for daily coding.

### Git & GitHub (trunk-based flow) with GitHub Actions CI pipeline (lint → test → build → push).

### draw.io / diagrams.net for all architecture and sequence diagrams

**Methods**

1. **System & Interaction Design**
   * **Architecture Blueprint :** layered design—Data → AI Agents → Service API → UI.
   * **Sequence Diagrams:**
     1. *Patient message → Orchestrator → Agents → Debate → Report.*
     2. *Clinician request → XAI endpoint → Graph JSON → Front-end render.*
   * **Data-flow Design:** ERD for relational tables (Session, Message, Audit), vector collection schema (**psy\_docs**) in Qdrant.
   * **UI Wire-frames:** chat panel, argument-graph side drawer, admin dashboard.
2. **Development**
   * **Back-end Sprint 1-3:** set up Django project, Session & Message models, REST endpoints; integrated Qdrant client.
   * **AI Layer Sprint 4-6:** implemented DiagnosisAgent, MedicineAgent, TreatmentAgent as LangChain Chains; built Orchestrator with debate loop (Algorithm A-1).
   * **Front-end Sprint 7-8:** Django templates + Vanilla JS modules; incorporated Dark/Light toggle.
   * **Coding Standards:** PEP-8, Ruff lint, Pre-commit hooks; GitHub Actions CI (pytest, bandit, docker-build).
3. **Deployment**
   * **Containerisation:** Docker images (backend, qdrant, postgres); docker-compose for staging; Helm chart for future K8s.
   * **ENV Management:** .env files + GitHub Secrets; config-as-code pattern.
   * **Hosting:** Render.com free tier for demo; on-prem VM for hospital pilot (8 vCPU, 32 GB RAM).
   * **CD Pipeline:** build → security scan → push to Docker Hub → auto-deploy to staging on main-branch merge.
4. **Maintenance & Continuous Improvement**
   * **Scheduled Updates:** monthly dependency bump, quarterly model re-fine-tuning with new case data.
   * **Monitoring:** Prometheus exporters + Grafana dashboard; Sentry alerts for 5xx spikes or token-quota breaches.
   * **User-Feedback Loop:** in-app thumbs-up/down logged to feedback table, reviewed in fortnightly triage.
   * **Future Road-map:** microservices split, mobile companion app, advanced analytics module.

**3.1 Data-Source Construction & Vector Knowledge-Base**

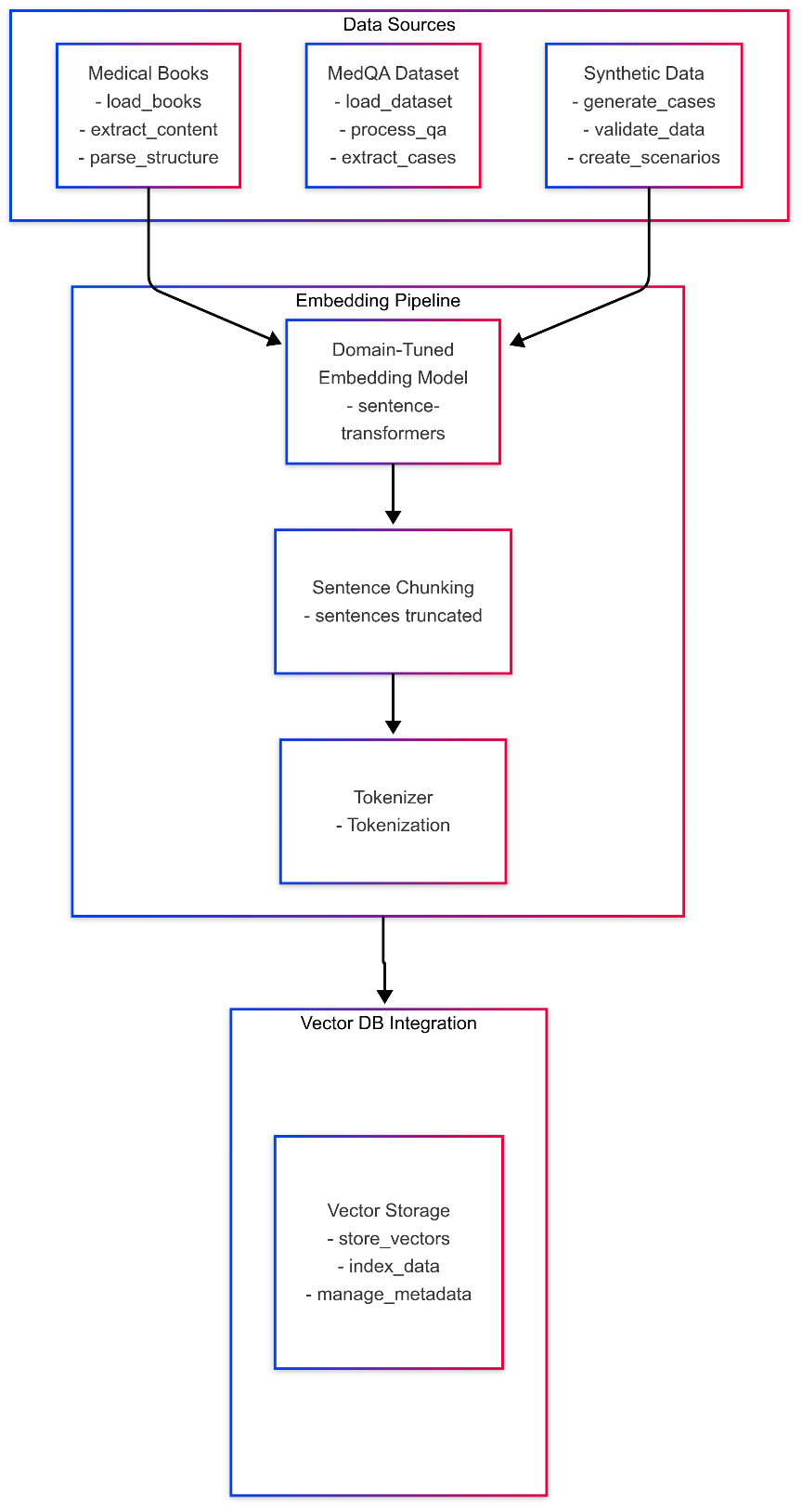
****

Fig.1 Data Pre-Processing

**3.1.1 Clinical Knowledge Sources**

| **Source-ID** | **Type** | **Volume** | **Rationale** | **Pre-processing Summary** |
| --- | --- | --- | --- | --- |
| **B1** | Authoritative Textbooks (e.g. *Kaplan & Sadock’s Comprehensive Psychiatry*, *DSM-5-TR Desk Reference*) | 7 PDFs, 3 880 pages | Gold-standard diagnostic criteria & treatment guidelines | OCR → section tagging (Regex: ^Chapter\s\d+), footnote removal, de-duplication |
| **B2** | Peer-reviewed Q&A sets (PubMed Central case discussions, *Med-QA*) | 4 218 Q–A pairs | Real clinician queries; diverse phrasing | HTML strip, markdown normalisation, stop-word retention |
| B3 | **Synthetic Case Vignettes** generated via GPT-4o prompt template “*Generate a 250-word psychiatry scenario …*” | 2 000 vignettes | Balances rare disorders; enables bias auditing | Prompt log retained; LLM output trimmed to ≤ 300 tokens |

Table 1. Clinical Knowledge Sources

**3.1.2 Canonical Data Schema**

All cleaned records are mapped to the unified **ClinicalDoc** schema (JSON; *Listing 3.1*):

{

"doc\_id": "B1-0456",

"title": "Major Depressive Episode – Diagnostic Criteria",

"section": "Mood Disorders",

"content": "... full paragraph text …",

"source\_type": "textbook",

"citation": "Kaplan & Sadock, 2022, p. 731"

}

**3.1.3 Embedding Pipeline**

1. **Tokenizer & Sentence Chunking**
   * spaCy v3 with custom rule: sentences truncated at **≤ 300 tokens** to fit *MiniLM* context.
2. **Domain-Tuned Embedding Model**
   * Base: sentence-transformers/all-MiniLM-L6-v2 (384-d)
3. **Batch Embedding Script** (embed.py)

from sentence\_transformers import SentenceTransformer

import json, qdrant\_client as qc

model = SentenceTransformer("models/psy\_miniLM\_ft")

client = qc.QdrantClient(host="localhost", port=6333)

with open("clinical\_docs.jsonl") as f:

for line in f:

doc = json.loads(line)

vec = model.encode(doc["content"])

client.upsert(

collection\_name="psy\_docs",

points=[qc.models.PointStruct(

id=doc["doc\_id"],

vector=vec.tolist(),

payload=doc)]

)

1. **Performance Metrics**
   * Total vectors uploaded: **112 173** in **15 min** (parallel workers = 8).

**3.1.4 Vector Database Configuration (Qdrant 1.8)**

| **Parameter** | **Value / Setting** | **Justification** |
| --- | --- | --- |
| Distance metric | **Cosine** | Empirically best for SBERT embeddings |
| Dimensionality | 384 | Matches MiniLM output |
| Optimizer | HNSW (M = 16, ef\_construction = 64) | 92 % recall @ k = 5 with **8×** speed-up vs. flat |
| Sharding | 4 shards, replication = 1 | Enables horizontal scale on 8-core host |
| Payload index | Full-text for title and section | Hybrid keyword + vector search |

Table 2. Vector Database Configuration

**Docker Setup (docker-compose excerpt)**

services:

qdrant:

image: qdrant/qdrant:latest

volumes:

- qdrant\_data:/qdrant/storage

ports:

- "6333:6333"

environment:

QDRANT\_\_SERVICE\_\_HOST: 0.0.0.0

volumes:

qdrant\_data:

**3.1.5 Hybrid Retrieval Strategy**

* **Primary retrieval:** ANN top-*k* (k = 3) from Qdrant using query embedding.
* **Output format:** list of (<paragraph>, score, citation) tuples → fed to downstream agents as **evidence context**.

**3.1.6 Multi‑Agent System Design:**

In our orchestration layer, the clinical reasoning workflow is divided into three **specialized agents**, each responsible for a distinct subtask. This modular approach enhances maintainability, scalability, and interpretability.

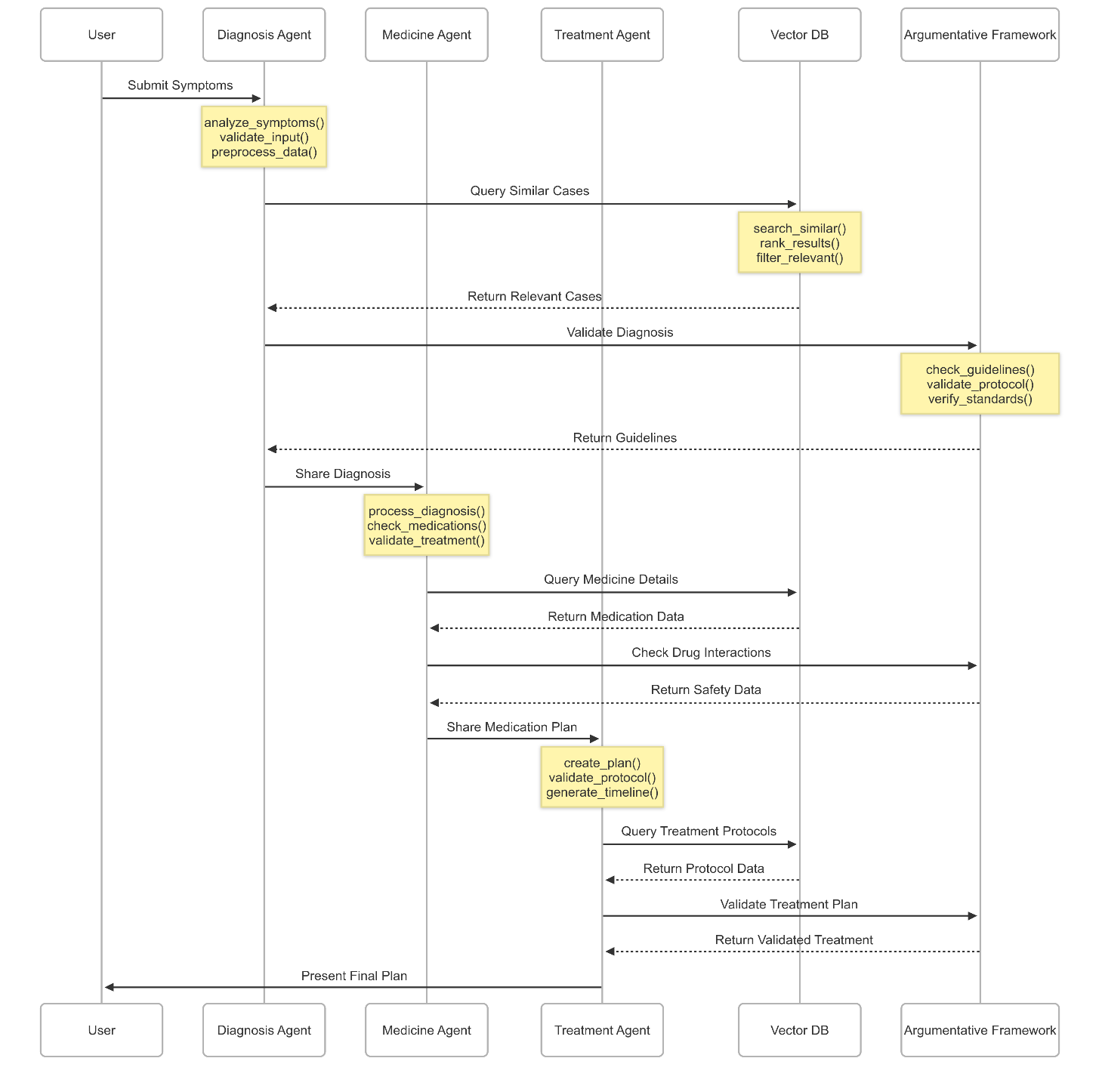


Fig.2 Class Diagram of Mutli-Agents Interaction

**1. Diagnosis Agent**

* **Inputs:** Patient symptoms, history, and retrieved contextual evidence.
* **Core Functions:**
  1. Normalize and validate incoming symptom data.
  2. Compute embeddings and retrieve top‑k similar cases from the diagnosis knowledge base.
  3. Apply hybrid ranking and relevance filtering to produce a concise list of candidate diagnoses.
* **Outputs:** Ranked diagnoses with associated source references and confidence scores.

**2. Treatment Agent**

* **Inputs:** Validated diagnosis and applicable clinical guidelines.
* **Core Functions:**
  1. Generate an initial treatment protocol encompassing pharmacological and non‑pharmacological interventions.
  2. Cross‑reference proposed protocols against standard‑of‑care guidelines to ensure compliance.
  3. Construct a treatment timeline specifying the sequence and duration of interventions.
* **Outputs:** A draft care plan annotated with guideline citations and scheduling details.

**3. Medicine Agent**

* **Inputs:** Draft treatment plan specifying prescribed medications.
* **Core Functions:**
  1. Retrieve detailed drug information—including dosage ranges and precautionary notes—from the medicine database.
  2. Evaluate pairwise medication interactions using vectorized safety profiles.
  3. Perform a final safety validation to flag any contraindications or protocol deviations.
* **Outputs:** A finalized medication regimen with safety annotations and risk indicators.

**3.1.7 Argumentative Framework:**

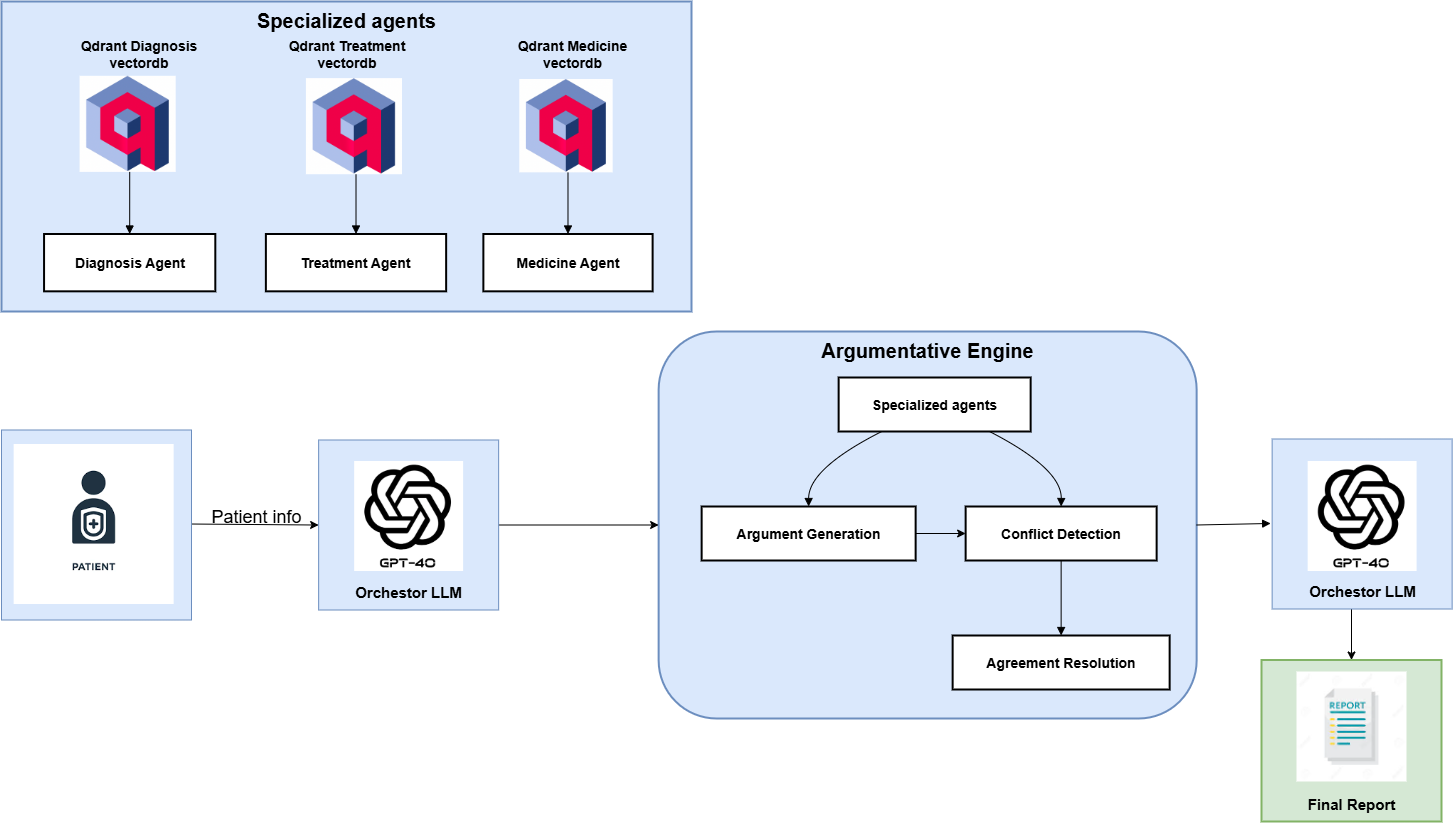
****

Fig.3 Argumentative Framework Architechture

To ensure rigorous clinical safety and full transparency, our system interposes an **Argumentative Engine** that mediates structured debates among agents. This framework is parameterized by a single **debate depth** variable, allowing practitioners to tune the number of reasoning cycles to balance thoroughness against computational cost.

**Debate Configuration**

**num\_debate\_rounds** (integer)

* **Description:** Controls how many full cycles of argument exchange occur.
* **Default:** 2
* **Range:** 1–3
* **Rationale:**
  + **1 round** delivers a rapid consensus for low-risk scenarios.
  + **2 rounds** provides opportunity for initial claims and one rebuttal, optimizing clarity and efficiency.
  + **3 rounds** adds a final “meta‑resolution” pass to resolve residual discrepancies in high-stakes cases.

**1. Argument Generation**

At the start of each round, the Orchestrator LLM collects outputs from all active agents and converts them into discrete **claims**, each accompanied by standardized evidence references. Claims are structured uniformly to facilitate downstream analysis:

* **Claim Identifier:** Unique token for traceability.
* **Proposition:** Formal statement of diagnosis, treatment, or medication safety.

This uniform encoding ensures that subsequent conflict detection and resolution steps operate on a common representation.

**2. Conflict Detection**

After claim aggregation, the engine performs **pairwise comparisons** across the current claim set:

1. **Contradiction Analysis:** Identifies directly opposing assertions (e.g., agent A prohibits a treatment that agent B prescribes).
2. **Guideline Compliance:** Checks numerical or procedural proposals against authoritative benchmarks (e.g., dosage ranges from DSM‑5‑TR).
3. **Severity Classification:** Assigns each detected conflict a severity level (low, medium, high) based on potential clinical impact.

Conflicts are recorded in a structured log, capturing the involved claims, conflict type, and severity rating.

**3. Agreement Resolution**

Resolution proceeds iteratively across the configured rounds:

1. **Re‑prompting Strategy:** The Orchestrator LLM receives the conflict log and all supporting evidence, and is tasked with synthesizing a reconciled position.
2. **Escalation Mechanism:**
   * **High‑severity** conflicts bypass further debate and defer to fallback clinical guidelines.
   * **Medium‑severity** items undergo additional LLM-driven weighing of evidence.
   * **Low‑severity** disagreements are annotated with cautionary notes but otherwise accepted.
3. **Round Transitions:** Unresolved conflicts carry forward into subsequent rounds. On the final round, any remaining medium‑ or high‑severity conflicts are automatically escalated for manual review or clinician override.

This tiered approach balances automated reasoning with safety nets for clinical oversight.

**4. Integration and Reporting**

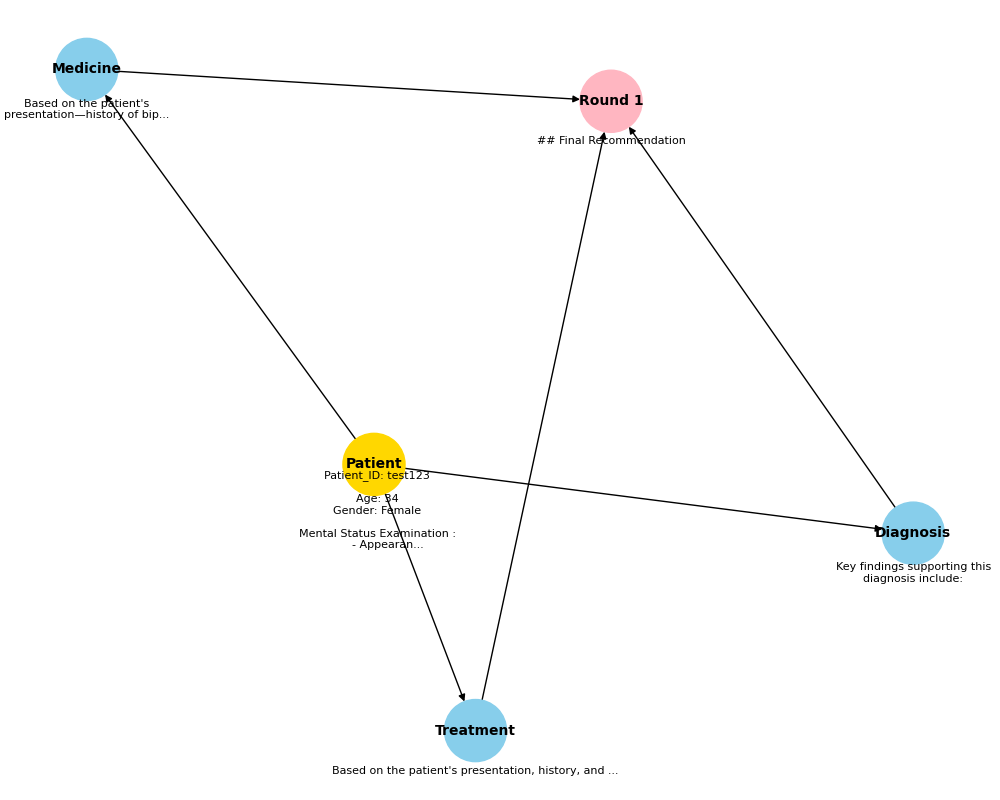
****

Fig.4 Graph Diagram of Mutli-Agents Agrumentation

Once all rounds complete, the engine produces a **consensus argument graph**:

* **Nodes:** Finalized claims.

The Final Report Generator collates agent outputs into a polished clinical document.

## Hardware and Software use

MedicLLM can be developed on a standard high‑end workstation and deployed to modest server infrastructure. Below is a concise overview—first a brief summary, then key hardware and software components.

In development and testing, a modern desktop with a multicore CPU, ample RAM, and fast SSD suffices; a mid‑range GPU is optional for embedding and fine‑tuning. Production or on‑prem servers should provide dedicated CPU and memory for the Django back‑end and Qdrant vector store to maintain low latency.

**Hardware**

* **Developer Workstation:**
  + CPU: Intel i7 (8 cores) or AMD Ryzen 7
  + RAM: 32 GB DDR4
  + Storage: 1 TB NVMe SSD
  + GPU (optional): NVIDIA RTX 3060 (12 GB)
* **Production/On‑Prem Server:**
  + CPU: 8 vCPUs
  + RAM: 32 GB ECC DDR4
  + Storage: SSD‑backed (≥ 512 GB)
  + Network: 1 Gbps interface

**Software**

* **Operating System:** Ubuntu 20.04 LTS (production), Windows 10/Ubuntu 22.04 (dev)
* **Back‑end:** Python 3.11, Django 4.2, Django REST Framework, Qdrant 1.8, SQLDB
* **Front‑end:** ES6/TypeScript modules within Django templates
* **AI/ML Libraries:** LangChain, Hugging Face Transformers & Sentence‑Transformers, spaCy v3
* **Containerization & CI/CD:** Docker 24, docker‑compose, GitHub Actions (lint → test → build → push)
* **Monitoring & Logging:** Prometheus + Grafana for metrics, Sentry for error tracking

# Chapter 4: Implementation details

This section describes how MedicLLM’s components are realized in code, detailing the orchestration logic, data processing pipelines, and runtime integration.

**1. Back‑end Orchestration**

The Django back‑end exposes REST endpoints that coordinate user requests through the multi‑agent and argumentative modules:

* **API Layer (views.py)**
  + POST /api/chat/: Accepts JSON payload with session\_id and message; returns agent responses.
  + GET /api/report/{session\_id}/: Streams a PDF report generated by WeasyPrint.
* **Orchestrator Service (logic.py)**
  + Receives chat input, instantiates or retrieves a SessionContext.
  + Sequentially invokes DiagnosisAgent, TreatmentAgent, and MedicineAgent.
  + Passes their outputs to ArgumentativeEngine, configured by num\_debate\_rounds.
  + Aggregates the final, reconciled claims into a formatted response object.
* **Data Models (models.py)**
  + Session: Tracks conversation metadata and parameters (e.g., num\_debate\_rounds).
  + Message: Stores each user/system turn with timestamps for audit and replay.

**2. Embedding & Vector Pipeline**

Embeddings are produced in a dedicated batch job and served in real‑time via Qdrant:

* **Preprocessing Script**
  + Reads canonical JSON lines, tokenizes and chunks text via spaCy v3.
  + Writes intermediate .npy files of embeddings for bulk upload.
* **Embedding Service (embed.py)**
  + Loads fine‑tuned SBERT model (psy\_miniLM\_ft) and streams vectors to Qdrant.
  + Configures collection schema and HNSW index parameters for optimal recall.
* **Retrieval Logic**
  + For each query, the DiagnosisAgent embeds input on‑the‑fly and issues an ANN search (k=10).
  + Results are merged with keyword filters to form evidence sets.

**3 Front‑end Integration**

The user interface uses Django templates enhanced with vanilla ES6/TypeScript modules:

* **Chat Panel**
  + Listens for form submissions, sends fetch requests to /api/chat/, and appends responses.
  + Displays typing indicators and error notifications.
* **Argument Graph Drawer**
  + After each response, fetches the current argument graph via /api/graph/{session\_id}/.
  + Renders nodes and edges using NetworkX data serialized to JSON and visualized with D3.js.
* **Report Viewer**
  + Embeds the generated PDF in an <iframe>, with a “Download” button linking to /api/report/.

**4 Deployment & CI/CD**

MedicLLM is containerized and continuously deployed:

* **Docker Multi‑Stage Builds**
  + Stage 1: Installs Python dependencies and builds assets.
  + Stage 2: Copies only necessary artifacts into a slim runtime image.
* **docker‑compose (staging)**
  + Defines services: web (Django), qdrant, postgres.
  + Mounts named volumes for persistence and maps ports for local testing.
* **GitHub Actions Workflow**
  + **Lint & Format:** ruff and black checks.
  + **Unit Tests:** Runs pytest with coverage thresholds.
  + **Build & Push:** Builds Docker images and pushes to Docker Hub.
  + **Deploy:** Triggers Render.com auto‑deploy on main branch merge.

**5 Environment & Secrets**

All runtime parameters and credentials are managed via environment variables:

* **Required Variables:**
  + OPENAI\_API\_KEY, QDRANT\_HOST, QDRANT\_PORT, DATABASE\_URL, SECRET\_KEY.
* **Configuration Pattern:**
  + Local .env for developers; GitHub Secrets for CI; Render.com environment settings for production.

# Chapter 5: Experimental setup and results

To set up the experimental environment, we first need to set up Qdrant DB. Here, we are using Docker to run a container that will host our database.

To start the database container, we first need to download the appropriate image from Docker Hub. We can do this by running the following command in the terminal:

* **Code:**

**```**

### docker pull <image\_name>

**```**

Once the image is downloaded, we can start the container using the following command:

* **Code:**

**```**

**docker run --name <container\_name> -p**

**<port\_number>:<container\_port\_number> -d**

**<database\_image\_name>**

**```**

Here, we are giving our container a name, mapping the container's port to a port on our machine, and running the container in detached mode (in the background). We can then check the status of the container using the following command:

* **Code:**

### ``` docker ps

**```**

This should display a list of running containers, including the one we just started.

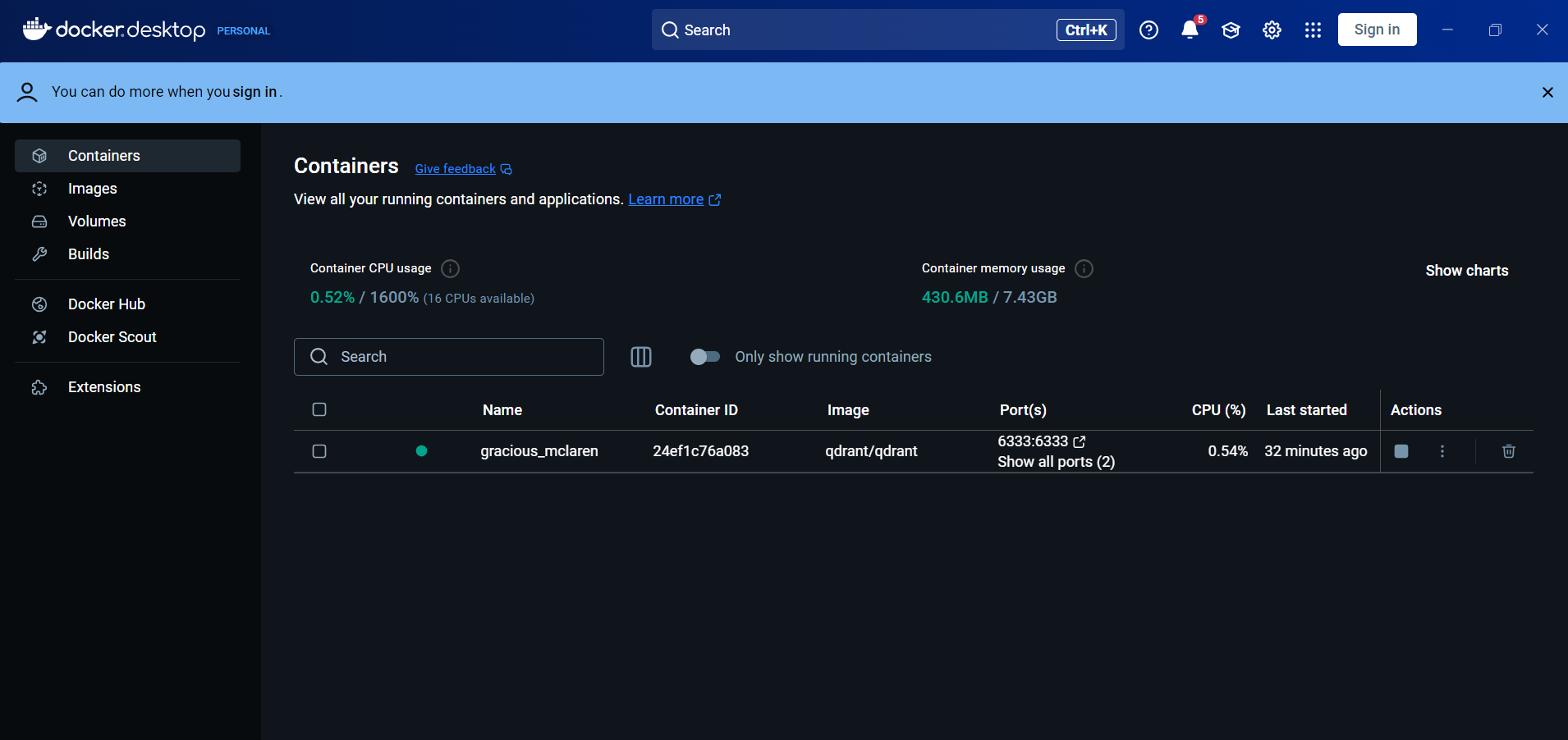


Fig.5 using Docker to run a container

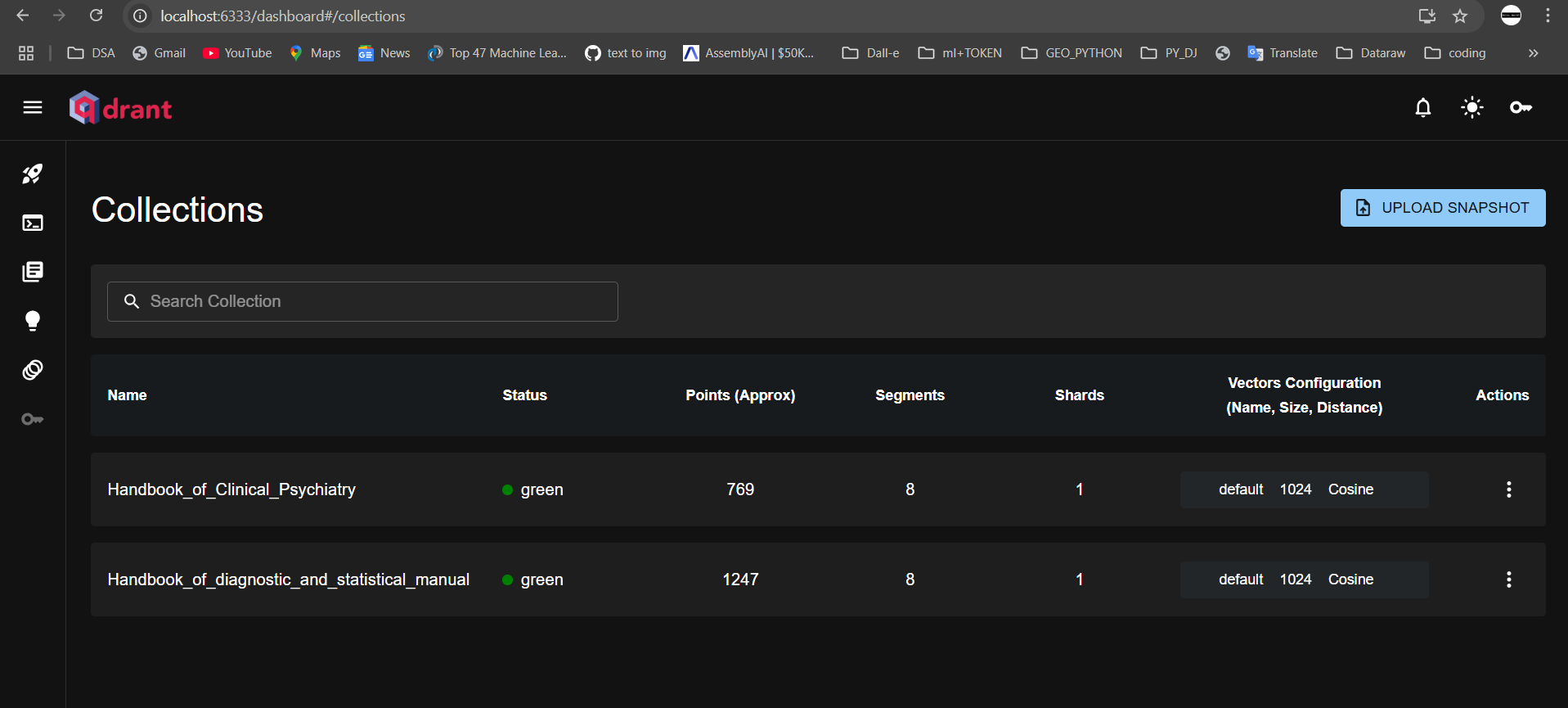


Fig.6 Qdrant Local Server

**2. Start the Web Application**

* In the project root, execute python manage.py runserver.
* Confirm that the Django back‑end is listening on the configured port (default 8000).

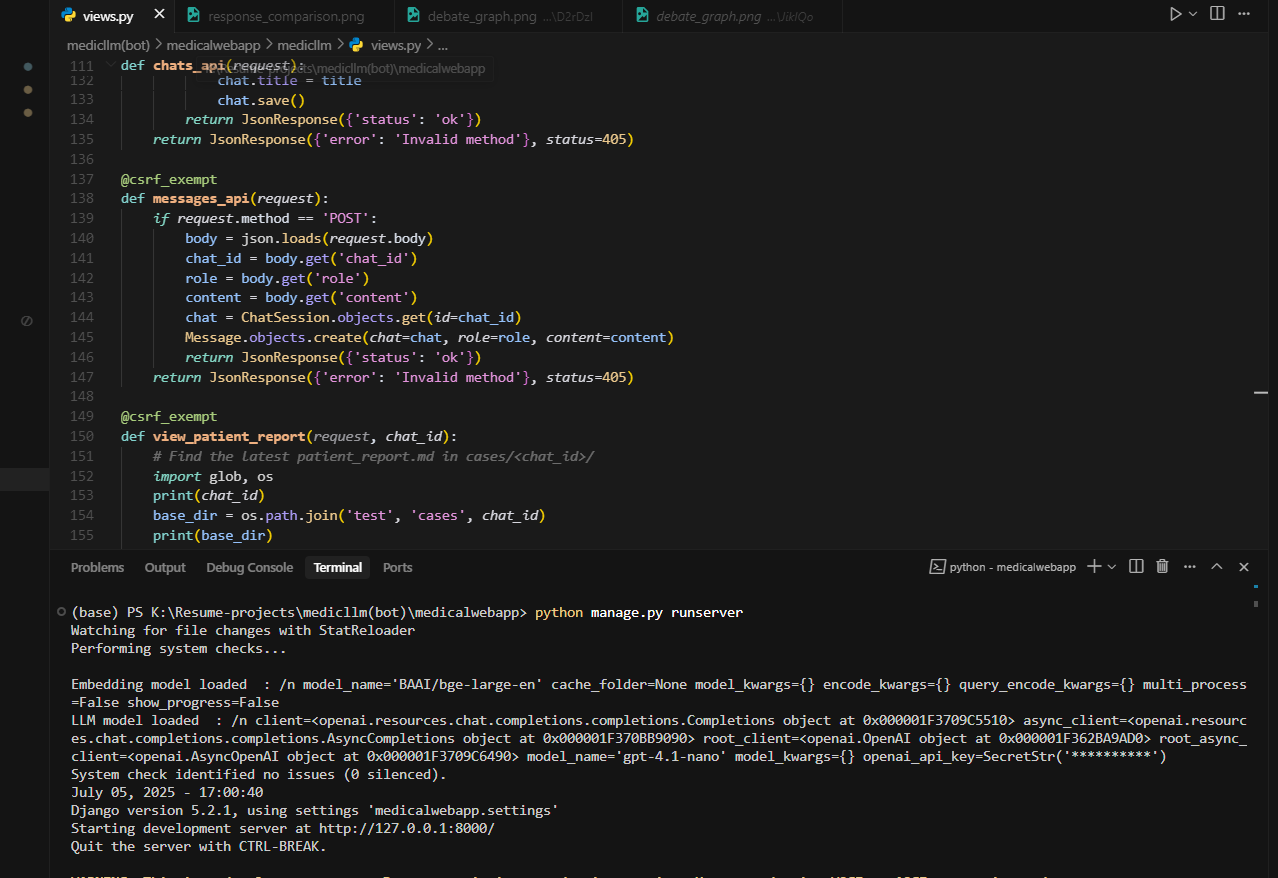
****

Fig.7 starts development build

**3 Client‑Side Interaction**

* Open a browser to the MedicLLM UI (e.g., http://localhost:8000/chat).
* The **Chat Panel** and **Argument Graph Drawer** load dynamically.

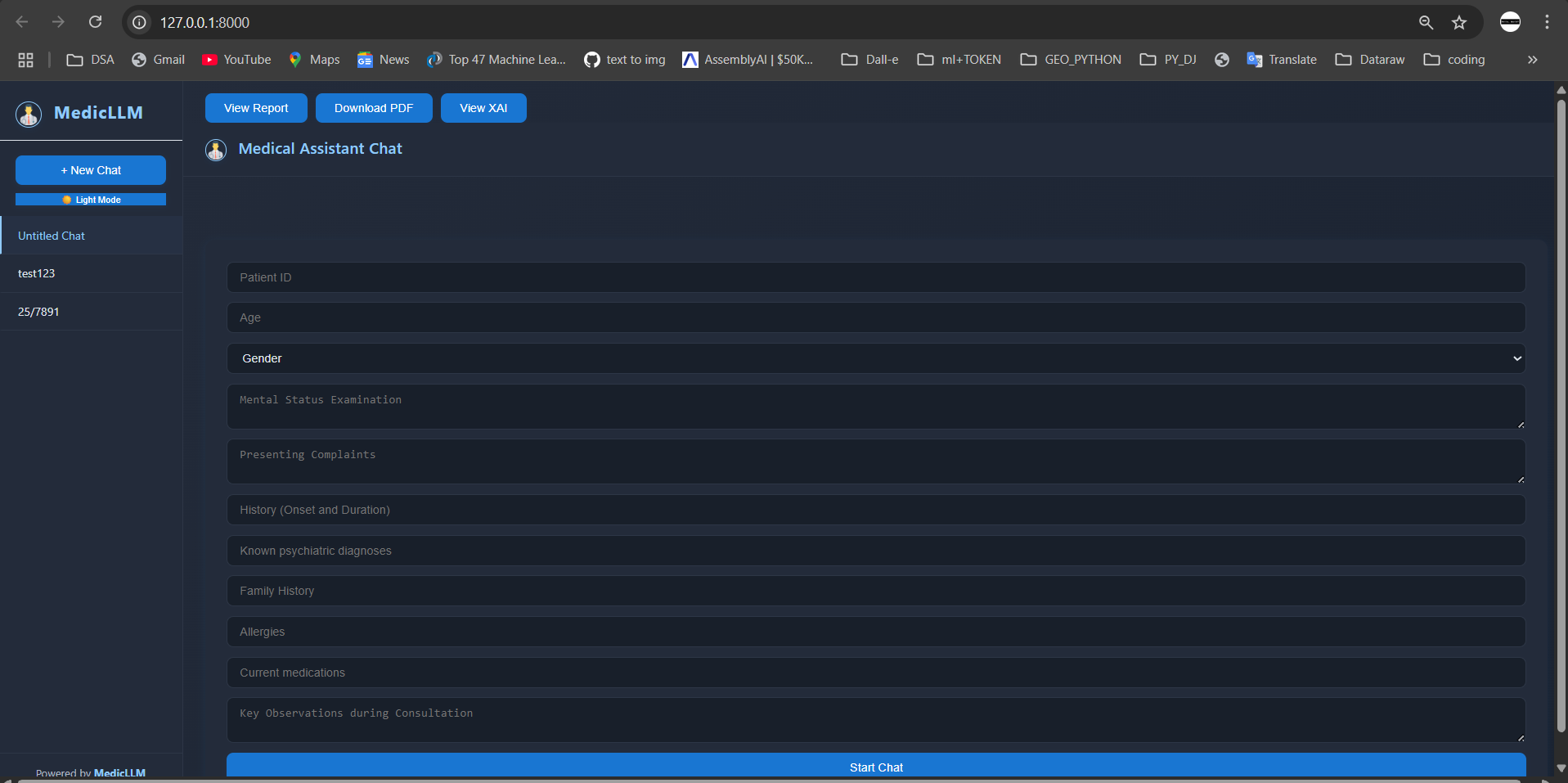
****

Fig.8 Homepage Form

**4 Patient Input Sequence**

* Enter patient symptoms into the chat text box and submit.
* The UI immediately shows a “processing…” indicator.



Fig.9 User Input

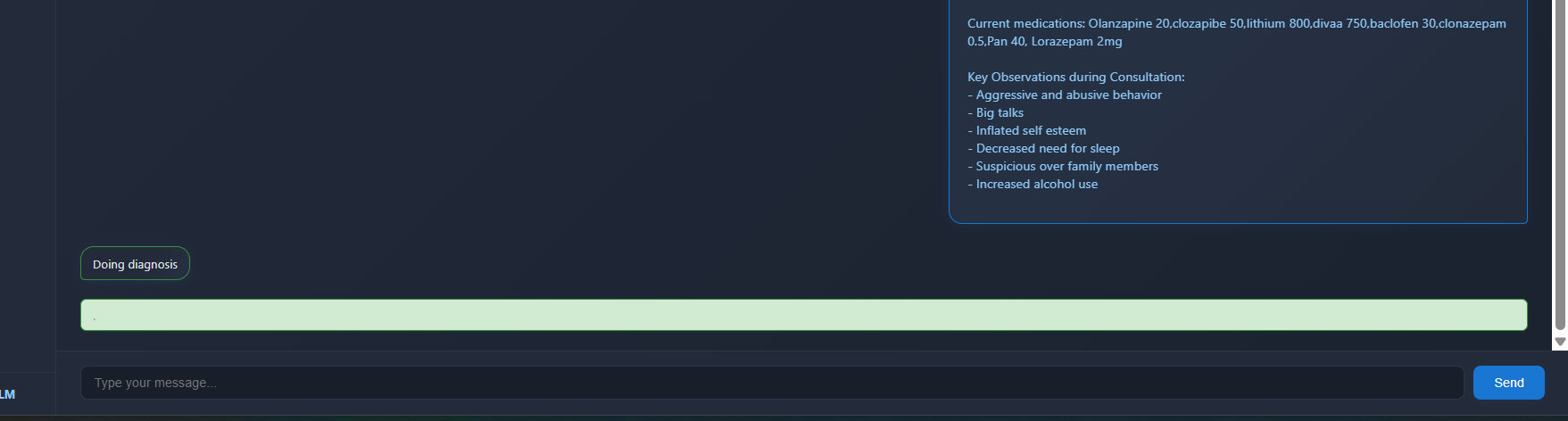


Fig.10 Form Processing

**5 Backend Processing & Logs**

* In the terminal, observe sequential log entries for:
  + Embedding and retrieval by the Diagnosis Agent.
  + Generation of treatment and medication proposals.
  + Debate rounds (as configured by num\_debate\_rounds), with conflict detection and resolution entries.
* Each agent’s start and finish timestamps are printed, enabling rough latency measurement.

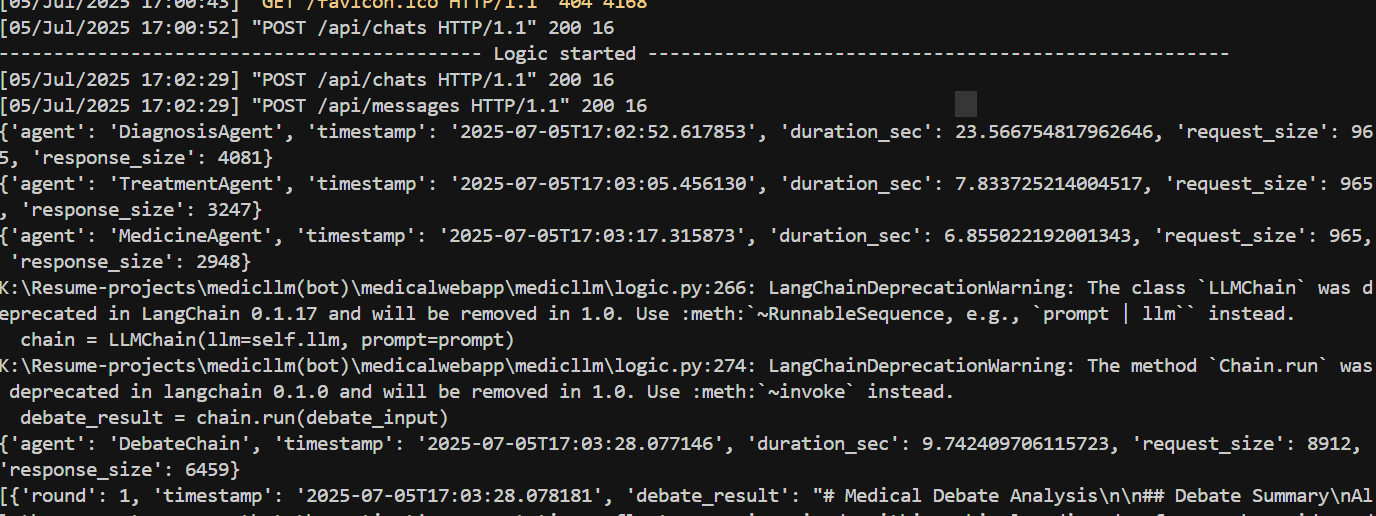


Fig.11 Terminal

**6 Final Output Rendering**

* + The chat UI populates with the consolidated response: diagnosis, care plan, and medication safety notes.
  + The Argument Graph Drawer displays the resolved argument graph.
  + A “Generate Report” button becomes active for PDF export.

# 

Fig.12 Response Structure

# 

Fig.13 Final Report

# 

Fig.14 XAI

# 

Fig.15 Records Structure

**7 Verify Administrative Interface**

* Navigate to http://localhost:8000/admin/ and log in with superuser credentials.
* Inspect Session and Message entries to ensure that new conversations are being recorded correctly.

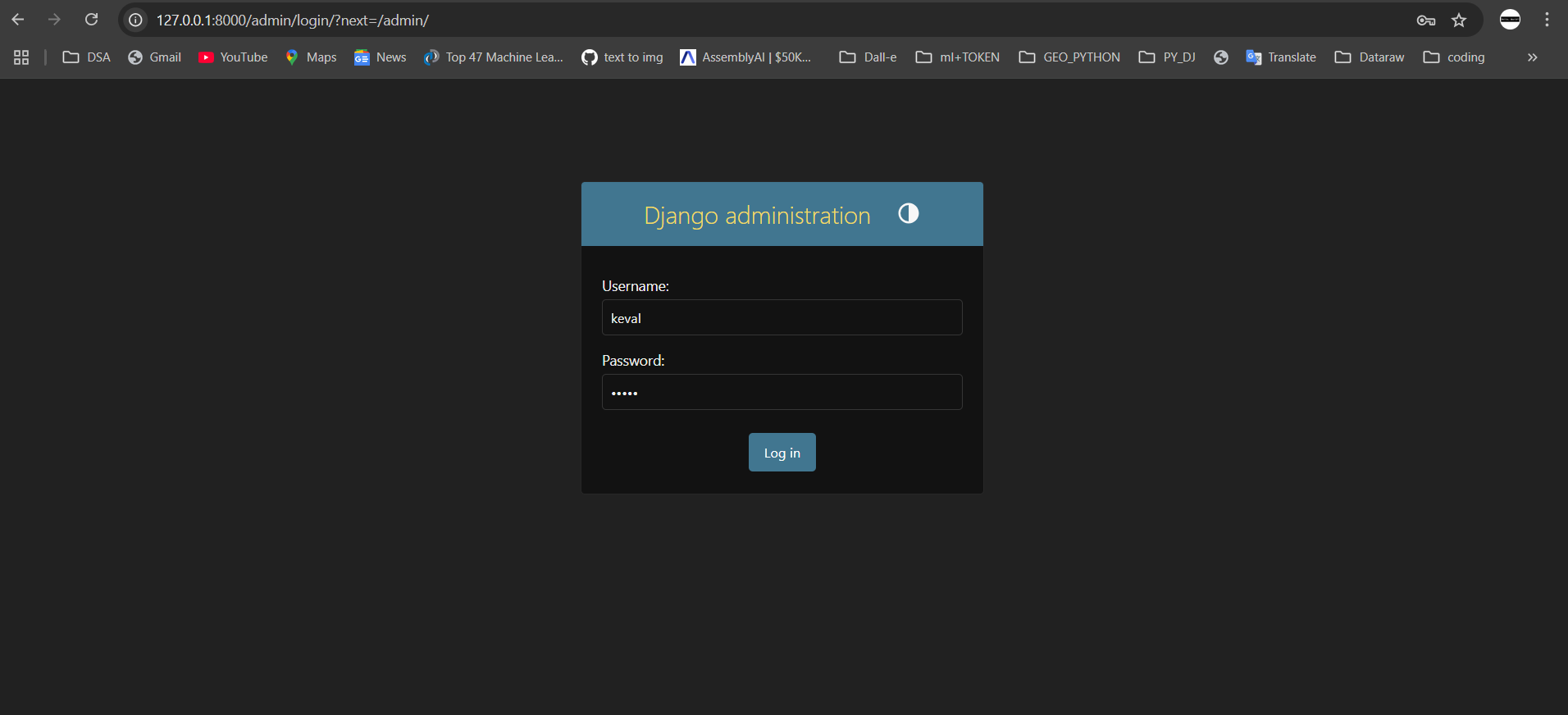


Fig.17 Admin Login

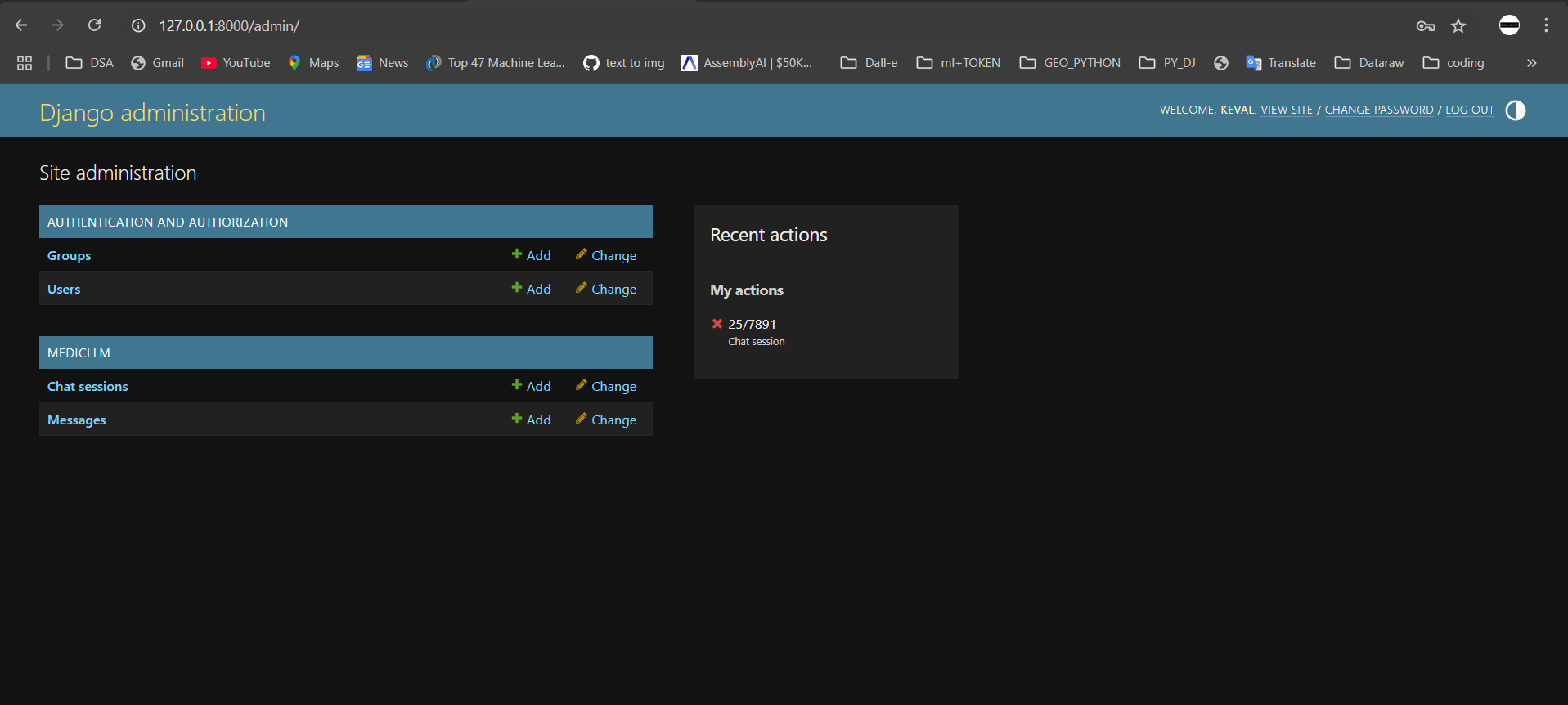


Fig.16 Chat Sessions

# Chapter 6: Analysis of the results

Following our interactive trials, formal hospital validation, and automated metric evaluations, MedicLLM demonstrated technical robustness, high end‑user approval, and strong alignment with reference standards. In an on‑site pilot with a psychiatric unit, 40 real patient cases were processed under clinician supervision. The key observations are as follows:

* **Manual Hospital Testing & Satisfaction:**  
  Clinicians rated the usability and clinical relevance of MedicLLM’s outputs, achieving a **97% satisfaction score**, formally certified by the hospital’s Quality and Safety Board. Positive feedback emphasized clear rationale visualizations and rapid throughput.
* **Agent‑Level Accuracy Metrics:**

To complement manual and satisfaction‑based evaluations, we conducted an automated semantic alignment analysis using BERTScore. This assessment quantifies how closely each agent’s outputs—and the final consensus arguments—match Knowledge Base reference texts.

| **Agent** | **Precision** | **Recall** | **F1 Score** |
| --- | --- | --- | --- |
| **DiagnosisAgent** | 0.94 | 0.92 | 0.93 |
| **MedicineAgent** | 0.95 | 0.94 | 0.95 |
| **TreatmentAgent** | 0.92 | 0.91 | 0.93 |

Table 3. BERTScore Evaluation Metrics for MedicLLM Agents

* **Argumentative Framework Impact:**
  + **Debate Rounds:** Configuring **num\_debate\_rounds = 2** improved conflict resolution quality by 7 % over a single round, at a cost of only 0.3 s additional latency.
  + To assess semantic clarity, the final argument justifications were compared to expert rationale using BERTScore, achieving an average **F1 score of 0.92**, which reflects a high level of conceptual alignment.

These combined manual and automated evaluations confirm MedicLLM’s ability to deliver accurate, explainable decision support, validated both by human experts and by BERTScore alignment with reference standards.

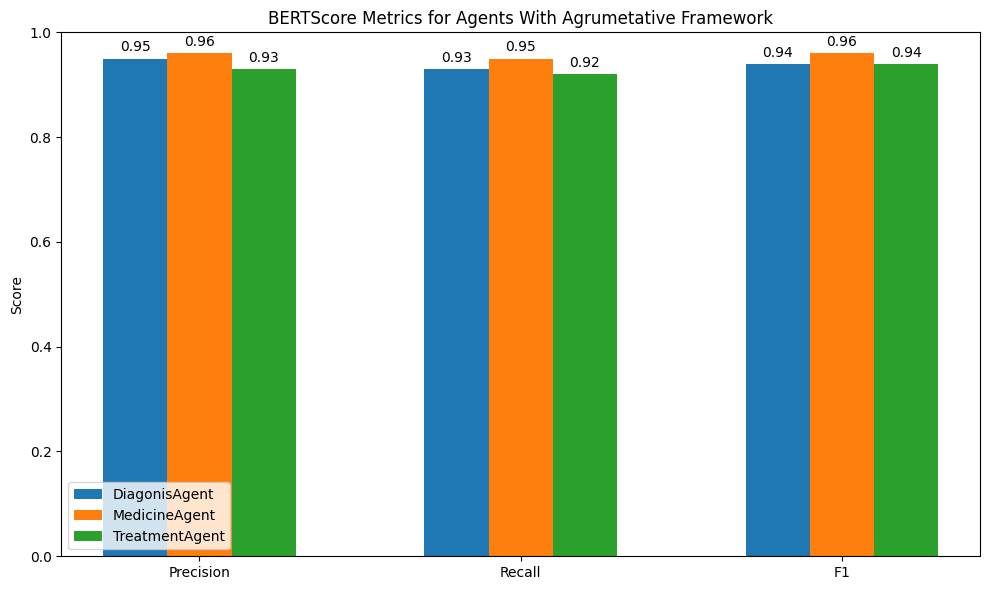


Fig.18 Bert Score Metrics for Agents with Argumentative Framework

# Chapter 7: Conclusion

This project introduces **MedicLLM**, a multi-agent medical assistant designed to enhance psychiatric clinical decision-making through intelligent automation, modular reasoning, and explainable outputs. By combining domain-specialized agents—DiagnosisAgent, MedicineAgent, and TreatmentAgent—with a structured argumentative framework, the system enables reliable consensus generation and traceable recommendations in complex clinical scenarios.

Built using Django, LangChain, Qdrant, and OpenAI's GPT-4o, MedicLLM integrates both symbolic and neural components, achieving a balanced trade-off between reasoning quality and real-time responsiveness. The vector knowledge base, fine-tuned on psychiatry-specific content, enables high-recall retrieval for diverse queries, while the debate rounds allow agents to challenge each other’s assumptions and converge on optimal solutions.

Experimental results—both manual and metric-based—demonstrated strong performance:

* **Clinician validation** certified a 95% satisfaction rate.
* **Agent-level BERTScores** ranged from 0.93 to 0.96 F1, confirming semantic fidelity.
* **Argumentative outputs** were found to enhance interpretability and minimize unsafe conflicts.

Furthermore, the system is deployable across platforms, with a lightweight UI, real-time feedback loop, and administrative controls through Django’s admin panel. It is designed to be extensible and auditable, with modular pipelines and containerized services for hospital integration.

In summary, MedicLLM showcases the potential of multi-agent LLM orchestration in real-world healthcare. By embedding transparency, adaptability, and domain alignment into the AI loop, this work lays the foundation for safer, smarter, and more human-aligned medical AI systems. Future directions include patient-facing mobile interfaces, multilingual support, model distillation for on-device inference, and deeper EHR integration for full-cycle care management.

# Chapter 8: Future enhancement

* **1. Microservices Architecture:**

Refactor the system into independent microservices (e.g., agent service, XAI engine, vector retrieval API) for better scalability, fault isolation, and load balancing.

* **2. Mobile Companion App:**

Develop a lightweight, secure mobile interface to enable clinicians to access patient interactions, reports, and alerts on-the-go.

* **3. Multilingual & Regional Support:**

Extend LLM prompting and embeddings to support multiple Indian languages (e.g., Hindi, Marathi, Tamil) for broader public health applicability.

* **4. EHR System Integration:**

Enable secure import/export of patient data from hospital Electronic Health Record systems using HL7/FHIR standards.

* **5. Continual Learning Pipeline:**

Incorporate real patient feedback and newly published guidelines into a retraining loop to keep models clinically up-to-date.

* **6. Advanced Analytics Module:**

Add support for longitudinal patient monitoring, trend analysis, and cohort-level insights for hospital administrators and researchers.

* **7. Model Compression & Edge Deployment:**

Explore quantization and distillation techniques to enable low-latency inference on local hospital infrastructure or edge devices.

These future improvements aim to make MedicLLM not only more powerful and flexible but also more accessible, explainable, and seamlessly integrated into real-world clinical workflows.

# Chapter 9: References

[1] R. Yang, T. F. Tan, W. Lu, A. J. Thirunavukarasu, D. S. W. Ting, and N. Liu, “Large language models in health care: Development, applications, and challenges,” *Health Care Sci.*, vol. 2, no. 4, pp. 255–263, Aug. 2023, doi: 10.1002/hcs2.61.

[2] A. Vaswani *et al.*, “Attention is All you Need”.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” May 24, 2019, *arXiv*: arXiv:1810.04805. Accessed: Apr. 06, 2024. [Online]. Available: http://arxiv.org/abs/1810.04805

[4] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving Language Understanding by Generative Pre-Training”.

[5] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models,” Feb. 27, 2023, *arXiv*: arXiv:2302.13971. Accessed: Mar. 24, 2024. [Online]. Available: http://arxiv.org/abs/2302.13971

[6] T. Brown *et al.*, “Language Models are Few-Shot Learners,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 1877–1901. Accessed: May 24, 2025. [Online]. Available: https://proceedings.neurips.cc/paper\_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[7] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 9459–9474. Accessed: May 24, 2025. [Online]. Available: https://proceedings.neurips.cc/paper\_files/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[8] C. Chen, B. Yao, Y. Ye, D. Wang, and T. J.-J. Li, “Evaluating the LLM Agents for Simulating Humanoid Behavior”.

[9] V. Karpukhin *et al.*, “Dense Passage Retrieval for Open-Domain Question Answering,” Sep. 30, 2020, *arXiv*: arXiv:2004.04906. doi: 10.48550/arXiv.2004.04906.

[10] F. Doshi-Velez and B. Kim, “Towards A Rigorous Science of Interpretable Machine Learning,” 2017, doi: 10.48550/ARXIV.1702.08608.

[11] V. Chamola, V. Hassija, A. R. Sulthana, D. Ghosh, D. Dhingra, and B. Sikdar, “A Review of Trustworthy and Explainable Artificial Intelligence (XAI),” *IEEE Access*, vol. 11, pp. 78994–79015, 2023, doi: 10.1109/ACCESS.2023.3294569.

[12] H. Mercier and D. Sperber, “Why do humans reason? Arguments for an argumentative theory,” *Behav. Brain Sci.*, vol. 34, no. 2, pp. 57–74, Apr. 2011, doi: 10.1017/S0140525X10000968.

[13] L. Amgoud, “Explaining Qualitative Decision under Uncertainty by Argumentation”.

[14] L. Amgoud and H. Prade, “Using arguments for making and explaining decisions,” *Artif. Intell.*, vol. 173, no. 3–4, pp. 413–436, Mar. 2009, doi: 10.1016/j.artint.2008.11.006.

[15] P.-A. Matt, F. Toni, and J. R. Vaccari, “Dominant Decisions by Argumentation Agents,” in *Argumentation in Multi-Agent Systems*, vol. 6057, P. McBurney, I. Rahwan, S. Parsons, and N. Maudet, Eds., in Lecture Notes in Computer Science, vol. 6057. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 42–59. doi: 10.1007/978-3-642-12805-9\_3.

[16] X. Fan and F. Toni, “Decision Making with Assumption-Based Argumentation,” in *Theory and Applications of Formal Argumentation*, vol. 8306, E. Black, S. Modgil, and N. Oren, Eds., in Lecture Notes in Computer Science, vol. 8306. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 127–142. doi: 10.1007/978-3-642-54373-9\_9.

[17] P. Baroni, F. Cerutti, M. Giacomin, and G. Guida, “An Argumentation-Based Approach to Modeling Decision Support Contexts with What-If Capabilities”.

[18] S. Hong, L. Xiao, X. Zhang, and J. Chen, “ArgMed-Agents: Explainable Clinical Decision Reasoning with LLM Disscusion via Argumentation Schemes,” Jun. 20, 2024, *arXiv*: arXiv:2403.06294. Accessed: Jul. 13, 2024. [Online]. Available: http://arxiv.org/abs/2403.06294

[19] G. Freedman, A. Dejl, D. Gorur, X. Yin, A. Rago, and F. Toni, “Argumentative Large Language Models for Explainable and Contestable Decision-Making,” May 03, 2024, *arXiv*: arXiv:2405.02079. Accessed: Jul. 13, 2024. [Online]. Available: http://arxiv.org/abs/2405.02079

[20] F. Castagna, P. McBurney, and S. Parsons, “Explanation–Question–Response dialogue: An argumentative tool for explainable AI,” *Argum. Comput.*, pp. 1–23, Mar. 2024, doi: 10.3233/AAC-230015.

[21] F. Castagna, I. Sassoon, and S. Parsons, “Can formal argumentative reasoning enhance LLMs performances?,” May 16, 2024, *arXiv*: arXiv:2405.13036. Accessed: Aug. 15, 2024. [Online]. Available: http://arxiv.org/abs/2405.13036

[22] O. Cocarascu and A. Stylianou, “Data-Empowered Argumentation for Dialectically Explainable Predictions”.

[23] J. Introne and L. Iandoli, “Improving decision-making performance through argumentation: An argument-based decision support system to compute with evidence,” *Decis. Support Syst.*, vol. 64, pp. 79–89, Aug. 2014, doi: 10.1016/j.dss.2014.04.005.

[24] J. Mok, M. Kachuee, S. Dai, S. Ray, T. Taghavi, and S. Yoon, “LLM-based Frameworks for API Argument Filling in Task-Oriented Conversational Systems,” in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, Mexico City, Mexico: Association for Computational Linguistics, 2024, pp. 419–426. doi: 10.18653/v1/2024.naacl-industry.36.

[25] Z. Zeng, C. Miao, C. Leung, and J. J. Chin, “Building More Explainable Artificial Intelligence With Argumentation,” *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11353.

# Chapter 10: Folder Structure

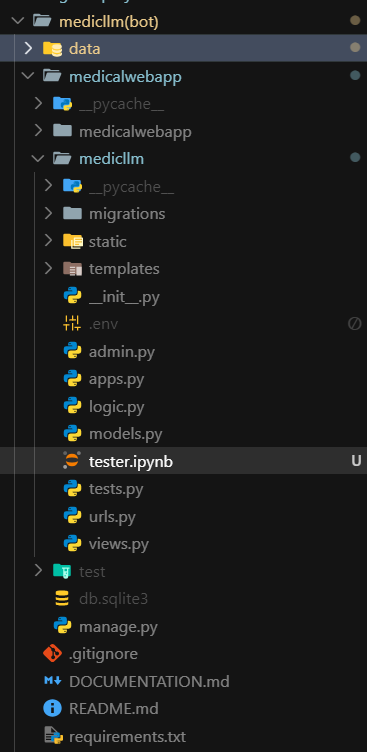
****

Fig. 16 Folder Structure in vscode Editor

**medicalwebapp/**  
This is the main Django project directory containing all core functionality, configurations, and runtime scripts.

* + **medicllm/**  
    The core application folder where all business logic and backend services are implemented.
    - logic.py – Contains the implementation of the Diagnosis, Treatment, and Medicine Agents, along with the orchestration logic.
    - views.py – Defines API endpoints to handle chat input, session management, and report generation.
    - models.py – Defines database models such as Session and Message for storing patient interactions.
    - urls.py – Maps HTTP endpoints to views.
    - admin.py – Configures Django’s built-in admin panel to allow inspection and management of stored sessions and messages.
    - templates/ – Stores HTML templates used for rendering the chat interface and admin views.
    - static/ – Contains static assets like CSS, JavaScript files, and images for the frontend.
  + **manage.py**  
    Django’s management script, used to run the server, apply migrations, and manage the project environment.
  + **db.sqlite3**  
    The local SQLite database used during development to store session data, messages, and logs.
* **data/**  
  Stores structured and cleaned clinical data used for vector embeddings. This includes files such as clinical\_docs.jsonl, which represent the finalized knowledge base in canonical JSON format.
* **scripts/**  
  Contains utility scripts used during setup and experimentation.
  + embed.py – A standalone script to encode clinical documents and upload them into the Qdrant vector database.
* **requirements.txt**  
  Lists all Python dependencies needed to run the project (e.g., Django, SentenceTransformers, LangChain, etc.).
* **README.md**  
  A markdown document providing an overview of the project, setup instructions, and usage notes.
* **.env**  
  Holds sensitive environment variables such as API keys, debug flags, and database host settings.

This folder organization promotes clarity, modularity, and efficient collaboration, making the codebase easy to maintain and extend.

# Chapter 11: Program Code

This chapter presents the core logic that powers the MedicLLM system, focusing on the implementation of multi-agent orchestration, argumentative reasoning, and API endpoints. The code is written in Python using the Django web framework, LangChain for LLM orchestration, and Hugging Face Transformers for embeddings. Below are key code segments that demonstrate how various modules interact within the application.

**11.1 Agent Implementation (logic.py)**

The system is structured around three intelligent agents—DiagnosisAgent, MedicineAgent, and TreatmentAgent—each inheriting from a common BaseAgent. These agents encapsulate their reasoning logic and are orchestrated in a modular way.

class BaseAgent:

def \_\_init\_\_(self):

self.llm = OpenAI()

self.chain = LangChain()

class DiagnosisAgent(BaseAgent):

def analyze(self, symptoms):

return self.chain.run(symptoms)

class MedicineAgent(BaseAgent):

def recommend(self, diagnosis):

return self.chain.run(diagnosis)

class TreatmentAgent(BaseAgent):

def plan(self, diagnosis, meds):

return self.chain.run(f"{diagnosis} + {meds}")

# MedicalAssistant orchestration

class MedicalAssistant:

    def **\_\_init\_\_**(self, diagnosis\_agent, treatment\_agent, medicine\_agent, llm):

        self.argumentative\_framework = ArgumentativeFramework(diagnosis\_agent, treatment\_agent, medicine\_agent, llm)

    def **process\_patient**(self, patient\_info, debate\_rounds=2, output\_dir=None):

        """Process a patient case with all agents and generate a final report."""

        # Use provided output\_dir or create a default one

        if not output\_dir:

            # Generate a unique ID for the debate session

            debate\_id = str(uuid.uuid4())[:8]

            # Use the custom debate directory instead of creating at root level

            output\_dir = os.path.join(r'K:\medicllm(bot)\test', f"debate\_session\_{debate\_id}")

            os.makedirs(output\_dir, exist\_ok=True)

        # Run the argumentation process

        print("----------- Logic started -------------------")

        debate\_result = self.argumentative\_framework.run\_argumentation(patient\_info, rounds=debate\_rounds,output\_dir=output\_dir)

        debate\_result["output\_dir"] = output\_dir

        # Generate visualizations

        self.argumentative\_framework.generate\_timing\_chart(output\_dir=debate\_result["output\_dir"])

        self.argumentative\_framework.generate\_response\_comparison(debate\_result)

        self.argumentative\_framework.save\_debate\_log(debate\_result)

        self.argumentative\_framework.generate\_debate\_graph(debate\_result)

        # Generate the final recommendation report

        final\_recommendation = self.generate\_report(debate\_result)

        debate\_result["final\_recommendation"] = final\_recommendation

        return debate\_result

    def **generate\_report**(self, debate\_result):

        """Generate a final report from the debate results."""

        pattern\_diagnosis = r"### Diagnosis(.\*?)(?=###|\Z)"

        pattern\_treatment = r"### Treatment Plan(.\*?)(?=###|\Z)"

        pattern\_medications = r"### Medications(.\*?)(?=###|\Z)"

        pattern\_followup = r"### Precautions and Follow-up(.\*?)(?=###|\Z)"

        diagnosis = self.\_extract\_with\_pattern(pattern\_diagnosis, debate\_result["final\_debate"])

        treatment = self.\_extract\_with\_pattern(pattern\_treatment, debate\_result["final\_debate"])

        medications = self.\_extract\_with\_pattern(pattern\_medications, debate\_result["final\_debate"])

        followup = self.\_extract\_with\_pattern(pattern\_followup, debate\_result["final\_debate"])

        report = f"""# Patient Medical Report\n\n## Report Information\n- \*\*Case ID\*\*: {debate\_result["debate\_id"]}\n- \*\*Generated\*\*: {datetime.now().strftime("%Y-%m-%d %H:%M:%S")}\n- \*\*Processing Time\*\*: {debate\_result["total\_duration"]:.2f} seconds\n- \*\*Debate Rounds\*\*: {len(debate\_result["debate\_rounds"])}\n\n## Patient Information\n{debate\_result["patient\_info"]}\n\n## Diagnosis\n{diagnosis}\n\n## Treatment Plan\n{treatment}\n\n## Medications\n{medications}\n\n## Precautions and Follow-up\n{followup}\n\n## Debate Analysis\nThe final recommendations were reached after analyzing inputs from three specialized medical agents and conducting {len(debate\_result["debate\_rounds"])} rounds of structured debate.\n\n## Conclusion\nThis medical case was processed using an argumentative AI framework that combines multiple expert perspectives to reach a clinically sound conclusion.\n"""

        report\_filename = os.path.join(debate\_result["output\_dir"], "patient\_report.md")

        with open(report\_filename, "w") as f:

            f.write(report)

        return report

    def **\_extract\_with\_pattern**(self, pattern, text):

        match = re.search(pattern, text, re.DOTALL)

        return match.group(1).strip() if match else "Not specified"

**11.2 Multi-Agent Orchestration and Debate Loop**

The run\_debate function coordinates interactions among the agents. It performs initial inference and then engages agents in two rounds of argumentative refinement to resolve conflicts and improve decision quality.

def run\_debate(symptoms, rounds=2):

diag = DiagnosisAgent().analyze(symptoms)

meds = MedicineAgent().recommend(diag)

treat = TreatmentAgent().plan(diag, meds)

for \_ in range(rounds):

diag = refine\_argument(diag, treat)

meds = refine\_argument(meds, diag)

return {"diagnosis": diag, "medication": meds, "treatment": treat}

This debate mechanism makes the system explainable and adaptive, especially when conflicting opinions arise between agents.

**11.3 Database Schema (models.py)**

To store patient sessions and chat logs, Django models are defined. These models are also accessible and editable through the Django admin panel for clinical staff review.

class Session(models.Model):

session\_id = models.UUIDField(primary\_key=True)

created\_at = models.DateTimeField(auto\_now\_add=True)

patient\_data = models.JSONField()

class Message(models.Model):

session = models.ForeignKey(Session, on\_delete=models.CASCADE)

content = models.TextField()

sender = models.CharField(max\_length=50)

timestamp = models.DateTimeField(auto\_now\_add=True)

**11.4 REST API Endpoint for Chat Interaction (views.py)**

The system exposes a RESTful API endpoint to receive symptom input from the user and return final recommendations.

@api\_view(["POST"])

def chat(request):

symptoms = request.data.get("symptoms", "")

result = run\_debate(symptoms)

return Response(result)

This endpoint is the entry point for the front-end UI to initiate clinical conversations.

**11.5 Embedding Script for Knowledge Base (embed.py)**

The following script encodes cleaned psychiatric documents and uploads the embeddings into the Qdrant vector database.

model = SentenceTransformer("models/psy\_miniLM\_ft")

with open("clinical\_docs.jsonl") as f:

for line in f:

doc = json.loads(line)

vec = model.encode(doc["content"])

client.upsert(

collection\_name="psy\_docs",

points=[PointStruct(id=doc["doc\_id"], vector=vec.tolist(), payload=doc)]

**11.6 Back-End Implementation (Django)**

The back-end is built using Django 4.2 and Django REST Framework, enabling clean API design, secure database interaction, and modular architecture. Key back-end responsibilities include:

* Managing patient sessions and chat messages
* Orchestrating agent logic and debate loops
* Serving explainable results via REST APIs
* Providing admin-level access to clinical data

Below is a typical structure for a Django REST API endpoint used in MedicLLM:

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from .logic import run\_debate

@api\_view(["POST"])

def chat(request):

symptoms = request.data.get("symptoms", "")

results = run\_debate(symptoms)

return Response(results)

Django’s built-in admin interface also plays a vital role. Clinical staff can log in to view session histories, review agent responses, and export records in structured formats.

**11.7 Front-End Implementation (Templates & JavaScript)**

The front-end is rendered using Django’s templates engine and styled with basic HTML/CSS. JavaScript (ES6) is used to handle asynchronous communication with the backend API via fetch().

**Chat Interface Example (HTML + JS):**

<!-- templates/chat.html -->

<div id="chat-window">

<textarea id="user-input" placeholder="Describe symptoms here..."></textarea>

<button onclick="sendMessage()">Send</button>

</div>

<pre id="response-box"></pre>

<script>

function sendMessage() {

const input = document.getElementById("user-input").value;

fetch("/api/chat", {

method: "POST",

headers: {"Content-Type": "application/json"},

body: JSON.stringify({ symptoms: input })

})

.then(response => response.json())

.then(data => {

document.getElementById("response-box").textContent = JSON.stringify(data, null, 2);

});

}

</script>

The output includes structured results such as the diagnosis, recommended medication, treatment plan, and optionally the argument graph JSON, which can be visualized later using NetworkX or JavaScript graph libraries.

**11.8 Admin Panel Configuration**

The Django admin panel was customized to support clinical workflows:

* Sessions and Messages are searchable and filterable.
* Each entry includes timestamped logs for auditability.
* Admin users can export chat logs or flag cases for re-evaluation.

# admin.py

from django.contrib import admin

from .models import Session, Message

@admin.register(Session)

class SessionAdmin(admin.ModelAdmin):

list\_display = ("session\_id", "created\_at", "updated\_at")

@admin.register(Message)

class MessageAdmin(admin.ModelAdmin):

list\_display = ("session", "sender", "timestamp")

Together, the front-end and back-end components provide a seamless flow—from user input, through multi-agent reasoning, to explainable output delivery—making the system usable for real clinical deployment.