DISEASE DIAGNOSIS USING RAG LLM WITH SMART PROMPT ENGINEERING

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Qadeerullah Syed

May 2025

The Designated Project Committee Approves the Project Titled

DISEASE DIAGNOSIS USING RAG LLM WITH SMART PROMPT ENGINEERING

by

Qadeerullah Syed

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2025

Dr. Ching-Seh Wu Department of Computer Science

Dr. Fabio Di Troia Department of Computer Science

Dr. Genya Ishigaki Department of Computer Science

ABSTRACT

DISEASE DIAGNOSIS USING RAG LLM WITH SMART PROMPT ENGINEERING

by Qadeerullah Syed

Although recent trends indicate that LLMs outperform traditional methods in solving complex problems with enhanced reasoning, there has been barely any progress in replicating the quality of diagnoses like those of actual human doctors. The identification of an accurate diagnosis with thorough reasoning is still a significant challenge, even with advanced AI models. The process of performing accurate diagnosis remains challenging due to a lack of transparency in state-of-the-art models existing today, a lack of explanation in the diagnosis process, an emphasis on results rather than reasoning, and a lack of foundational knowledge in models, along with limited exploration of diseases. To solve these problems, we propose a RAG model with smart prompt engineering to develop a sound medical diagnostic agent. The rapidly evolving techniques in LLMs, particularly RAG models, have shown promising results in processing and interpreting complex data. With the use of smart prompt engineering techniques and the use of a RAG framework with strong databases, we have achieved desirable results and enhanced the diagnostic reasoning performance. Deepseek-R1-Distill-Qwen-7B, Mixtral-8x7B, and MedAlpaca were trained as RAG models with PubMed articles and PMC patient data by varying the number of documents. We observe an average increase of 15% in accuracy scores when we introduce relevant documents in the RAG framework. Additionally, prompt engineering guides the formulation of differential diagnosis and a chain-of-thought inference. Overall, the models show an improvement in predictions with an increase in the number of documents in context under controlled measures.

Index Terms - Large Language Models, Retrieval Augmented Generation, Knowledge Base, Differential Diagnosis, Chain-of-Thought

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to my professor and advisor, Dr. Ching-seh Wu, for his constant support and guidance throughout my Master's program. I found myself turning to his guidance often throughout this entire research. He has provided me with not only the concepts implemented in this research but also useful tools to help expedite my work. It was a pleasure working under his guidance, and this experience would help me advance in my career.

Additionally, I would like to extend my gratitude to Dr. Fabio Di Troia and Dr. Genya Ishigaki for their support throughout academia. I am also grateful for their time and effort in evaluating my research work.

TABLE OF CONTENTS

CHAPTER

LIST OF FIGURES

Chapter 1

INTRODUCTION

In recent years, the advent of Large Language Models has marked a significant shift in the field of artificial intelligence, leading to enhancements in natural language understanding and generation. Trained on vast amounts of data, LLMs have found a wide range of utility in applications like creative writing, customer service, research assistance, software development, and more. Given textual data, these models can understand the constructs of a language, additionally capturing the concepts beneath them. Their capabilities have astounded researchers, and there has been an increase in demand towards the mainstream deployment of these models in various sectors. Although these are great conversational agents, they still lack a niche in domain-specific knowledge, where they fail to emulate the skills of human experts in a particular field. Owing to their ease of use and capability to power other applications, there has been a growing demand for models with specialized knowledge. Healthcare is one of the domains trying to establish expert-level LLMs to emulate the reasoning capabilities of doctors. By simulating expert-level reasoning, LLMs can assist practitioners in differential diagnosis, medical education, clinical decision support, and patient communication.

While LLMs in healthcare provide substantial potential benefits, there are several challenges hindering their deployment in real-world medical settings. Unlike other domains, healthcare demands the highest standards of accuracy and precision. Minor inaccuracies and misunderstandings can lead to severe consequences for patient safety, making the deployment of LLMs a highly sensitive endeavor. Achieving production-level LLMs in healthcare requires overcoming several key obstacles like Reliability and Precision, Trustworthiness, Data Privacy,

and Security. Additionally, the quality of interaction with a patient plays a key role in customer satisfaction in a domain like healthcare.

This report streamlines the challenges faced by contemporary healthcare AI models into four sections. Firstly, it demands a close coupling to the foundational knowledge and expects the model to adhere closely to the core fundamentals of medicine. Secondly, it outlines the lack of explainability in recent models and demands a more critical analysis and explanation of patient queries. Moreover, contemporary models lack the provision of differential diagnosis that helps in narrowing down the potential diagnosis. Lastly, an open-source framework is lacking in commercial models, which raises potential privacy concerns. To tackle these challenges in the production of LLMs in healthcare, we implement a multi-faceted approach designed to enhance reliability, explainability, and transparency. Additionally, we try to establish a framework outlined by contemporary research and stick to standards set for successful implementation. The framework leverages a combination of techniques, including the Retrieval-Augmented Generation (RAG) framework, Chain of Thought (CoT) explanation, Differential Diagnosis formulation, and open-source architecture to create a robust and trustworthy system.

The RAG framework introduces a knowledge base to enhance the reasoning capabilities of LLM for the given task at hand. It is a hybrid model that utilizes generative AI in conjunction with information retrieval to ensure that the generated responses align with core foundational knowledge. This serves as an essential part of LLMs in healthcare, mainly because the models need to keep up with the current trends and up-to-date information. Moreover, it is crucial to ensure that the LLM adheres to the relevant information and does not hallucinate for complex queries.

The Chain of Thought (CoT) framework is essential to process the reasoning capabilities of diagnostic models. It enhances the reliability of LLM to provide an accurate diagnosis. Instead of providing answers without justification, CoT explanations allow the system to break down complex queries into smaller steps, emulating the reasoning framework used by medical professionals. Additionally, contemporary research indicates this is a crucial step in gaining the model's trust. This reasoning builds transparency and allows users to understand the conclusions made.

Differential Diagnosis establishes a robust framework, forcing the model to consider diagnoses from a larger set of potential explanations. Different diagnostic possibilities are weighed against each other to lower the risk of premature conclusions and improve the quality of recommendations.

Having an open-source framework is extremely crucial in a sensitive domain like medical diagnosis. Open-source models provide complete transparency in architecture, workflow, training, and decision-making. This allows the model to adhere to ethical standards and ensures that clinical guidelines are followed. Moreover, it encourages contributions from the community and drives the development of the model, allowing doctors and developers to customize and fine-tune models on various medical sub-domains without a breach in user privacy.

This research gives a unique insight into the performance of a medical diagnosis model by varying the number of documents in an RAG framework. We observe that the performance of the model increases under the RAG framework, but the performance is not linearly dependent on the number of documents input to the model. Different models have different tolerance numbers to reach peak performance under varying numbers of documents. This report is organized to present

the literature review first, covering the most relevant research in the field of healthcare assistants using AI. It defines the problems discussed above and presents the potential solutions to tackle them. The literature review highlights the key findings from the research and gives a comprehensive detail on the proposed model to tackle these problems. This is followed by an implementation section that outlines all the tools and resources required to implement the proposed model. This section is followed by details on the dataset used in our model and provides insights into the nature of our dataset. This is followed by a methodology section, which is an in-depth explanation of the steps involved in creating the proposed model. This section gives an explanation justifying the methods used and processes involved in our implementation. Lastly, we present our findings on the performance of our proposed model, followed by a conclusion and references.

Chapter 2

LITERATURE REVIEW

This literature study explores various implementations of medical diagnostics using AI. It investigates different Machine Learning approaches used to predict disease diagnosis of a patient. As we move further into our research, we narrow down our approach towards LLMs and their implementations. The objective of this literature study is to address the following questions: *What are the different ways AI is used to achieve digital diagnostics? What are the benefits of using an AI model in healthcare diagnostics? What are the problems faced by current implementations of this application? What are the different techniques used to potentially come up with a working model? What are the ethical concerns surrounding this application? What are the evaluation criteria to judge the performance or results of resultant diagnosis?* Conference proceedings and published papers were included in the articles that are chosen for this literature review.

The literature review is divided into the following sections: Section II discusses the problems that are faced by the AI diagnostic models today. We will be going through an in-depth exploration of existing and potential problems faced by these models. In Section III we will be looking into solutions suggested by the research papers. We will examine how the existing research tries to solve the problems discussed in the previous section. In section IV we will be discussing some Machine Learning techniques used to solve the problem and what role LLM plays to come up with a robust solution. In section V we will be exploring select methodologies in detail, these include the methodologies considered to be implemented in our proposed model. In section VI different datasets are discussed, and the best data is selected for the implementation. Section VII gives details on proposed methodology and implementation to build a better model. The literature

review comes to an end by describing the evaluation metrics for the proposed model. The literature survey is organized as shown in Fig 1.

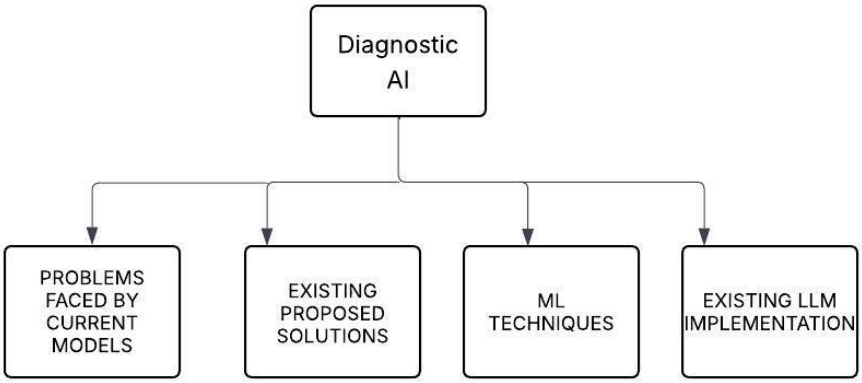

Fig 1. Organization of the literature survey

2.1 PROBLEMS IN CURRENT MODELS

Initial implementations of the application of AI in disease diagnosis focused just on predicting the correct output given a set of symptoms as input. [1] H. Wang et al. discussed the potential downsides of just focusing on mapping symptoms to diseases. Disease diagnosis is a more complex task than just a correlation between a set of symptoms and diseases. [2] M. Rosendal et al. state that the same symptoms can result in different diseases for different people. Medical consultation is carried out keeping in mind various factors like patient history, demographics, age, gender, and more. [1] H. Wang et al. add that contemporary models cannot emulate the general consultation scenario where a general practitioner is consulted first followed by the recommendation of a specialist. Instead, a single model is used for all types of diagnosis tasks.

Moreover, there is an emphasis on emulating the decision-making strategy of doctors in these models. The process an AI model takes to come up with a resultant diagnosis is a point of concern

for healthcare practitioners. [3] [4] [5] The bulk of the research suggests that for healthcare practitioners to gain confidence in AI diagnostic models, it has to provide detailed reasoning on the process of resultant diagnosis. [3] A.F. Tchango et al. describe that if the AI diagnostic models can provide the same reasoning as doctors in their disease diagnosis process, this would help them gain trust in the performance of the model. [3] A.F. Tchango et al. also state that the AI model should be more focused on the reasoning process to achieve correct diagnosis rather than just focusing on the result. [5] K. Singhal et al. states that we should focus on emulating the model to human experts in a particular domain to the highest level possible. Additionally, the model should be equipped to solve expert-level queries. [4] M. Abbasian et al. points out the significance of multi-step processing in diagnostic AI. This research suggests that intermediate results are equally as important as the final diagnosis, and the path taken to solve a problem holds crucial significance. [6][7] C. Wu et al. and Z. Chen et al. discuss the concept of inference while coming up with a diagnosis and call for a structured approach towards performing diagnosis. Therefore, we can conclusively infer that providing intermediate processing steps while diagnosis will help gain the trust of practitioners and users likely.

Personalized diagnosis is a central part of such organizations, and the contemporary model today fails to account for it. As discussed before, diagnosis differs with various characteristics for the same set of symptoms. To tackle this problem, [8] [9] [4] discuss the possibility of a personalized healthcare system. [8] B. Yang et al. suggests that details regarding the patients should be constantly tracked and the possibility of disease diagnosis should not be dependent on subjective symptoms, i.e. the model should constantly monitor the health of an individual and keep updating the status to potentially identify the risks. [9] M. Jin et al. demands that a personalized health management system should be designed that can be created using user health reports. This

accounts for a more in-depth analysis of patient health. [4] M. Abbasian et al. explains that conversations of models with patients should not be generic and should be more personalized, making patients feel more comfortable.

Another concept of knowledge base or domain knowledge is introduced by Z. Chen et al. and U. Naseem et al. [7] [10], which puts an emphasis on foundational knowledge. Traditional models lack domain-specific knowledge, and they tend to deviate from core concepts of diagnosis [10]. It is important for the model to put an extra emphasis on remembering the foundational knowledge. This knowledge shall remain constant and always in consideration [7]. Therefore, core foundational knowledge is key for the model to draw information from to keep the diagnosis process guided.

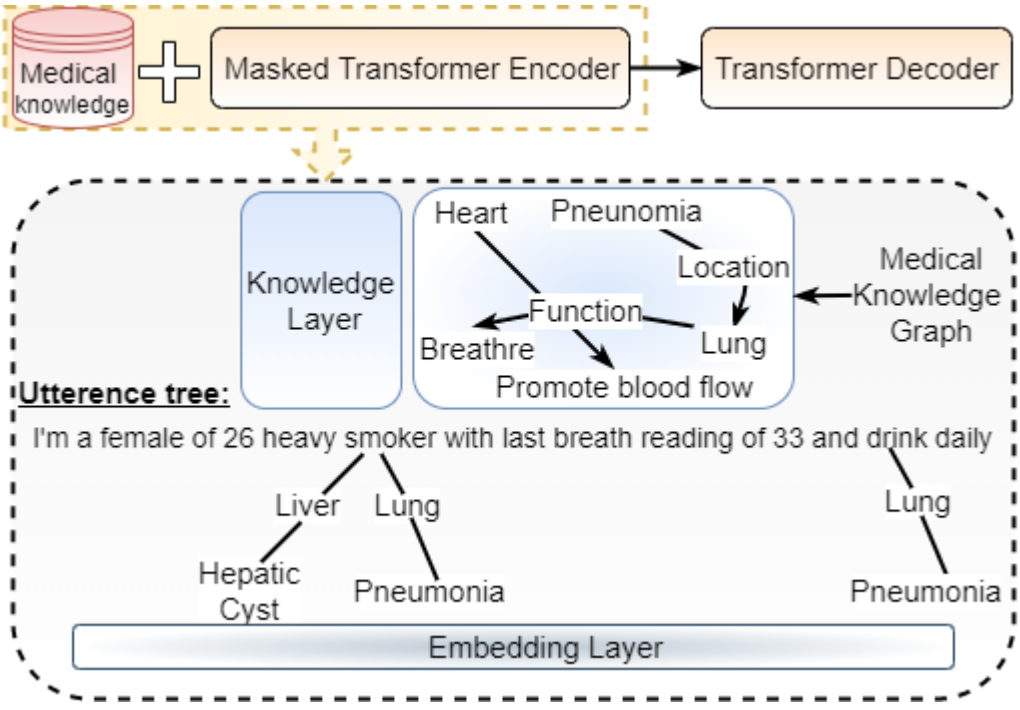

Fig. 2 Demonstration of Knowledge-base presented by U. Naseem et al. [10]

State-of-the-art models like GPT-4, GPT-3.5, Gemini, and others are closed-source models, and there has been a constant concern about implementing closed-source models for a sensitive domain like healthcare. [11] [12] T. Han et al. and A. Toma et al. raises the concern of potential risks of training closed-source models with sensitive information from healthcare. There has been an increase in demand for transparency of data and models used for healthcare applications. [11] [12] raise concerns over privacy risks and demand an open-ended view for the implementation of healthcare diagnosis.

Moreover, several other problems related to healthcare diagnosis are discussed in the domain, a few of which are listed below. [13] T. Tao et al. elaborates on the need for healthcare models to have a human touch. According to the situation during medical consultations, the model should be able to express empathy with the patient [13]. The consultant model should also be able to adapt to different people and different scenarios [13]. Therefore, we need models that can understand the situation and respond to it intelligently. The models should be able to show care and compassion towards the patient. Additionally, diagnostic models should be able to account for rare and complex diseases [14]. Failing to account for some of the complex diseases can lead to severe harm for the patients. Moreover, medical data is vast, diverse, and frequently unstructured [15]. The language models designed for diagnostic tasks should be equipped well to handle this complexity and come up with accurate diagnoses [15]. Therefore, we need to have a mechanism in place to handle this complexity. Confidence scores/levels in the output of disease diagnosis is a key feature in interpreting results. [16] [17] G.K. Gupta et al. and U. Dhakal et al. state that confidence scores are a key factor to determine the quality of the result output by the model. LLMs are designed in a way that they produce results even when the model is not sure about the result. To tackle this problem, [16] introduces the concept of confidence scores, which can help understand the extent

of truth in the answer. This is a key characteristic to arrive at a correct diagnosis. Traditional models also do not account for multimodal input and just rely on text-based input. [18] D.P. Panagoulias et al. demand the introduction of multimodal evaluation, stating that not all diagnoses can be carried out with just text-based inputs. Including multimodal formats of input is key to creating a robust model.

One specific drawback of implementing a diagnostic AI that goes beyond the technical issues is the impact on healthcare practitioners due to the implementation of such a model. [19] A. Choudhury and Z. Chaudhry outlines the issue of the potential deskilling of doctors due to their over-reliance on diagnostic LLM models. It mentions that if the doctors get used to the consultation of diagnosis from LLMs, over time, we can observe the description of doctors who would not be able to differentiate between a correct diagnosis and an incorrect diagnosis from AI. Another implication of using AI models is the self-referential loop in training data [19]; this means that the AI model is trained on the data it produces itself and hence might be subject to bias and incorrect diagnosis.

In the next section, we will discuss the methods proposed in recent research to solve these problems.

2.2 PROPOSED SOLUTIONS

This section is dedicated to discussing the solutions to the above problems through recently proposed research work. [1] H. Wang et al. proposed a solution to emulate the real-world consultation process. It describes a framework known as Agent-Driven Multi-Specialist Consultation Framework (ASMC). In this framework, multiple agents are trained in multiple domains, and these models work in collaboration over the possibility of multiple diseases. Each

agent generates a probability distribution over the possibility of diseases. Finally, the disease with the highest probability is selected as the final diagnosis. This implementation achieves good results as it consists of dedicated models for each subfield. Another similar implementation is discussed in [20] where the diagnostic model works by collaboration of multiple LLMs over the possibility of diagnosis and comes to a consensus on the resultant diagnosis.

Another key feature implemented by many papers includes generating follow-up questions based on the user's input symptoms. This is essential in the process of emulating doctors, as a general consultation process includes a doctor asking follow-up questions related to the symptoms given by the user. Asking for implicit symptoms given explicit symptoms is key [1]. [9] M. Jin et al. generate follow-up questions based on the knowledge base created from health reports. Sample queries are generated based on health reports. [3] A. F. Tchango et al. asks for questions that could potentially eliminate diseases from differential diagnosis. This has been established as an important segment in the development of an accurate diagnosis because patients are usually unaware of some implicit symptoms. Only through the emphasis of doctors can they provide the information, and information is key to recognizing a diagnosis. Figure 3 gives a detailed

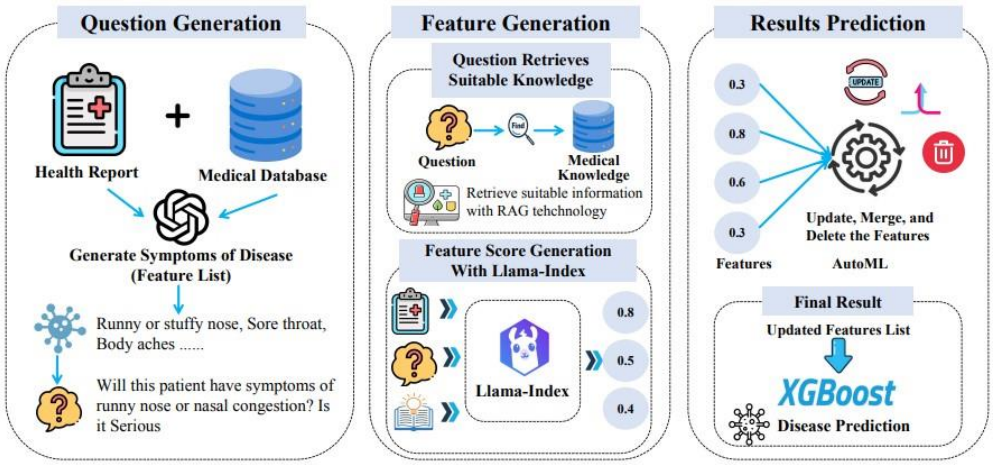

Fig. 3 Architecture of Health-LLM with RAG

explanation on RAG model implemented by M. Jin et al. [9] with health reports and medical data input to the LLM model. It uses the Llama indexing to retrieve documents and XGBoost to predict output.

Knowledge bases have been extensively used by researchers to create robust models because it is essential that the data is always aligned with the foundational knowledge. [8] B. Yang et al. use knowledge bases to extract data from external sources and others to maintain sensor data. [7] Z. Chen et al. consists of foundational knowledge in the form of replay data and makes sure that the model always draws from foundational knowledge and does not forget it. [4] M. Abbasian et al. include an API for knowledge bases. [10] U. Naseem et al. Introduces a unique concept of knowledge graphs to incorporate domain-specific knowledge. Domain knowledge is structured in the form of graphs, and it forms triples for input utterances. It consists of a knowledge layer that injects relationships from knowledge graphs into doctor-patient dialogue. This helps in enhancing the context of the model.

Multimodal input has been a topic of discussion lately. It accounts for the fact that doctors must visually see some things for accurate diagnosis. To take this into consideration, [18] D.P. Panagoulias uses the GPT-4 vision preview model for image input. The fact that we have extremely powerful image processing algorithms comes into play, too. [21] V. Pimprale et al. use image processing as a tool to detect visual markers that are not visible to the human eye. This feature highly endorses the use of AI as it provides early detection of critical diseases. [22] A. Serag et al. implements image segmentation using ResNets and U-Nets. These are state-of-the-art models capable of prolific image processing power. Additionally, [22] uses GANs to generate synthetic data that can be used for the Data Augmentation process. This technique helps to identify stained slides and images.

Multiple researchers have accounted for the importance of prompt engineering in the formulation of a correct diagnosis. [6] C. Wu et al. implement prompt engineering and fine-tuning using the QLoRa framework to enhance the performance of the model. [9] M. Jin et al. perform semi-automated feature engineering to extract meaningful features from data and fine-tune the predictions based on patient reports. [14] T. Abdullahi et al. introduce a unique Major Voting strategy. This strategy involves a prompt ensemble technique that selects the most common answer and comes to a consensus based on the output of various ensemble models. It uses prompt engineering to enhance results.

Data Augmentation has shown promising results in the performance of models. It is a strategy to augment the model with synthetic data close to the representation of actual data to improve predictions. Healthcare data is limited, and to enhance the performance of models across various domains, it needs data augmentation. [6] showed significant improvement in model performance after the data augmentation process.

Google introduced a unique technique to enable diagnostic models to adapt to various scenarios. Google used a self-play mechanism in their diagnostic model AIME, which consists of an inner self-play loop and an outer self-play loop [13]. The inner self-play loop is used to provide in-context critical feedback to enhance its performance in simulated dialogue with the patient [13]. The outer self-play loop fine-tunes the iterations of refined conversations. This new version is then again set for the inner loop. In addition to the self-play mechanism, it consists of a chain of reasoning. This model emphasizes the importance of the quality of conversation and shows empathy with patients during conversations. This model proved to be superior on 28 out of 32 evaluation axes when compared to physical care practitioners.

History taking and personalization are also key features of achieving quality diagnosis. Researchers have found that personalization is very important in the context of medical diagnosis. [15] Z. AlNazi and W. Peng account for maintaining patient history to come up with an accurate diagnosis. This introduces a patient-centered approach rather than a data-centered approach. [13] T. Tao et al. Places an extra emphasis on history taking and identifies it as a key approach to pave the way for accurate diagnosis. It claims that 60-80% of accurate diagnoses are achieved through history taking alone. [4] M. Abbasian et al. introduce a model that fosters a personalized conversation. It also consists of an API for healthcare data and AI analysis. [9] M. Jin et al. also implement a personalized healthcare management system that is created from the health reports of patients.

There have been serious concerns over the implementation of closed-source models for medical diagnosis. As described in the section above, state-of-the-art models like GPT-4 and Gemini are closed source, and this raises many concerns over privacy and transparency. [11] T. Han et al. introduce an open-source diagnosis model that is trained and fine-tuned over 160,000 entries. Additionally, it is fine-tuned using LoRa, which reduces its computational and memory requirements. [7] Z. Chen et al. also provide an open-source model. Additionally, it is trained under techniques like Chain of Thought (CoT) and self-consistency. Moreover, this model achieved a performance on MedQA data that is on par with human passing scores. Similarly, [12] A. Toma et al. provide an open-source model with a unique Dialogue-Based Knowledge Encoding. It converts dense medical texts into multi-turn conversations. It consists of a teacher model that transforms texts into dialogues by using prompts. Another model, named as student model, is fine-tuned on these dialogues. For simplicity, it limits the token size to 4096. The model is fine-tuned using the QLoRa framework.

Lastly, most of the approaches involve using OpenAI's ChatGPT model. [18] D.P. Panagoulias uses the GPT-4 vision preview model for multimodal diagnosis. [16] G.K. Gupta et al. compare famous models like GPT-4, GPT-3.5, and Gemini. The study reveals that GPT-4 outperforms all the other models. [23] OpenAI et al. give a technical report on the performance of GPT-4 on various applications. The results indicated that GPT-4 performed with a 75% accuracy on overall medical domain tasks. [24] A. Rios-Hoyo et al. test the performance of GPT-4 across various specialties in the medical domain. A study focused on feedback mechanisms revealed that the GPT-4 model performed better without feedback mechanisms and gave higher confidence levels without feedback.

2.3 ML TECHNIQUES

2.3.1 Retrieval Augmented Generation (RAG) Models:

The RAG framework is presented in [25] and consists of a retrieval mechanism and pre-trained sequence-to-sequence (seq2seq) models. RAG improves performance on knowledge-intensive activities like fact verification, abstractive QA, and open-domain question answering by dynamically retrieving pertinent material from a dense vector index (such as Wikipedia). It provides two methods: RAG-Token, which dynamically retrieves documents for each token during creation, and RAG-Sequence, which conditions on a fixed set of retrieved documents. By gaining access to current external knowledge, enhancing factual correctness, and providing transparency through the provenance of retrieved content, RAG models perform better than conventional seq2seq baselines. It is especially useful for jobs needing real-time knowledge updates, such as medical decision support systems or diagnostic AI, because of its dynamic, retrieval-enhanced nature.

2.3.2 Chain of Thought (CoT):

By decomposing complicated problems into intermediate reasoning steps, known as "chains of thought," the paper [26] explores how large language models (LLMs) perform better. By breaking down inquiries into digestible, interpretable pieces, this method allows models to handle tasks such as arithmetic problems, symbolic reasoning, and commonsense logic. The strategy produces notable gains over conventional techniques by utilizing thoughtfully crafted prompts that direct the model to reason through difficulties. Applying the method to a 540-billion-parameter LLM produced state-of-the-art results on benchmarks like GSM8K for math word problems, demonstrating that the advantages increase with larger models.

Applications in healthcare and diagnostic AI benefit greatly from chain-of-thought prompting. Greater interpretability and dependability are ensured by clearly stating the reasoning behind decisions, which is crucial in high-stakes industries like healthcare. Clinical decision support systems can be enhanced by this technique, which will allow them to handle difficult cases more accurately and transparently justify their conclusions. The study emphasizes how reasoning-augmented AI might enhance technical and applied domains' ability to solve problems and make decisions.

2.3.3 Deep Reinforcement Learning (DRL):

The main DRL algorithms are the subject of [27] by V. Francois-Lavet et al. Action-value pairs are learned via value-based techniques such as Q-Learning and Deep Q-Networks (DQN), which use neural networks for high-dimensional problems and stabilizing strategies such as experience replay. Actor-Critic integrates policy optimization with value evaluation for enhanced stability, whereas policy-based techniques, such as policy gradients, optimize policies directly. Model-based techniques simulate results and improve sampling efficiency by learning the dynamics of the

environment. These methods address issues such as generalization and exploration-exploitation, and they serve as the foundation for sophisticated DRL systems.

2.4 EXISTING MODELS

2.4.1 AMIE

AMIE, a conversational AI model based on LLMs designed for diagnostic dialogue. It uses self-play mechanisms to improve diagnostic accuracy and reasoning through iterative interactions. History taking is at its center and the model accounts for empathy in conversations [13].

2.4.2 MedAlpaca

Fine-tuned open-source LLMs on medical datasets, achieves competitive diagnostic performance across USMLE questions. Consists of a dataset called Medical Meadow. It consists of 2 categories: a collection of NLP tasks and a collection of medical data from the internet [11].

2.4.3 Clinical Camel

Fine-tuned LLaMA-2-based LLMs using Dialogue-Based Knowledge Encoding (DBKE) to improve diagnostic reasoning and patient communication. A very implementation model that consists of a teacher and student model [12].

2.4.4 MEDITRON

Open-source model fine-tuned on Llama-2 architecture with 7B and 70B parameters. Pre trained in medical applications. Consists of medical replay data which consists of foundational medical knowledge. Uses Nvidia's Megatron-LM for large scale training. Achieves results on par with human passing scores on MedQA [7].

Chapter 3

IMPLEMENTATION

Almost all the computational processing of this research was performed on a High-Performance Computer (HPC). This section gives an overview of the tools required for the development of this project. Additionally, we discuss the libraries used to train the models and the development platform utilized to achieve this. This section outlines all the tools required for preprocessing of data, chunking, indexing, prompt engineering, and development of retriever and generator models.

3.1 IMPLEMENTATION PLATFORM

3.1.1 High-Performance Computer (HPC)

High-Performance Computer was employed for heavy weight tasks to speed up the processing time of the implementation. Except for preprocessing, all the steps involved in the project were carried out on HPC. This was essential because the models used for indexing and generating were too complex to be carried out on a generic machine. The indexing model MedCPT consists of 109M parameters, and all the three generator models consist of 7B parameters each. The configuration of this machine is

as follows:

• Operating System: Cent OS

• Accessed via: SSH provided connected to SJSU VPN

• CPU Memory: 100 GB

• Total GPU Nodes: 4

• GPUs Available:

– 2 x Tesla (Kepler) K40 GPU

– 2 x Xeon CPU E5-2680 v4 @ 2.40GHz(broadwell) (14 core/CPU)

• Python version: 3.7

• CUDA Version: 11.3

3.2 MACHINE LEARNING LIBRARIES

3.2.1 HuggingFace (Transformers)

HuggingFace provides access to implement and fine-tune open-source transformer models. It provides pre-trained models to perform Natural Language Processing (NLP) tasks. The diagnostic models used in this project have been obtained through hugging face. *AutoModelForCausalLM* and *AutoTokenizer* are utilized to set up models for the RAG framework

3.2.2 Transformers

Transformers is a library used by HuggingFace. It provides tools for training, loading and fine-tuning transformer models. It has been used to load specialized LLMs like Deepseek, Mixtral, and MedAlpaca. Additionally, it helps in tokenizing input queries and formatting the output. It is useful in managing model generation configurations.

3.2.3 Torch (PyTorch)

PyTorch is an open-source machine learning framework used for deep learning applications. Torch sensors are used to load and run deep learning models on CPU or GPU. It handles model generation and facilitates fine-tuning and evaluation of models.

3.3 DATA COLLECTION AND PREPROCESSING LIBRARIES

3.3.1 HuggingFace

In addition to transformer models, HuggingFace is also used to load datasets. It has been used to load and run the model on MedQA dataset for testing.

3.3.2 FAISS (Facebook AI Similarity Search)

FAISS is a library used for clustering dense vectors and performing efficient similarity searches. It is optimized to enhance the speed and scalability of indexing and search of embeddings. Chunks from the dataset are converted into vectors as embeddings using the FAISS library. Retriever searches for FAISS indices for similarity search.

3.4 VISUALIZATION

3.4.1 Matplotlib

It is a library used to visualize the data by creating graphs from the output of the model. The performance metrics of different models are converted into visual charts by using this library.

3.4.2 TQDM

Python library used to monitor the progress of model training visually. It is responsible for creating progress bars for loops and processes. It helps in benchmarking and provides a real-time training progress.

3.5 UTILITY TOOLS

3.5.1 Pandas

Pandas is a powerful Python library for data manipulation and analysis, providing data structures like DataFrame for structured data operations.

3.5.2 NumPy

NumPy is a core Python library for numerical computing, providing support for arrays, matrices, and high-level mathematical functions.

3.5.3 Scikit-learn

Scikit-Learn provides various utilities for machine learning evaluation and performance metrics.

Chapter 4

DATASET

A quality dataset is essential to essential to build a robust retrieval-augmented generation framework. Especially in the domain of medical diagnosis, the dataset must cover a wide range of cases while maintaining uniformity and consistency. For this biomedical question-answering application, two datasets were selected after careful consideration. For the task at hand, the PMC patients dataset and PubMed abstract articles were used. The PMC Patients dataset offers richly detailed clinical summaries for synthetic patient profiles. It consists of factors like age, gender, medical history, and clinical summary, and other related articles provided by PMID. The dataset consists of 167K articles of patient summaries extracted from PubMed Central (PMC) case reports. It simulates real-world patient scenarios and provides access to a wide range of content.

The second dataset used is the PubMed Article Summarization Dataset. It consists of scientific article abstracts, offering access to the latest peer-reviewed biomedical research. Additionally, this data set can be extended based on further research findings. While the earlier dataset provided access to real-world scenarios that can be paralleled with the experience of doctors, this dataset provides access to core foundational knowledge based on research. The combination of both datasets ensures that the system is equipped to handle a wide range of questions, from highly specific, patient-related queries to broader inquiries about medical conditions, treatments, and drugs. This dataset consists of 119K articles and their corresponding abstracts. The structure of each document had to be normalized to ensure consistency in schema, with standard keys such as "title," "abstract," and "PMID." Due to computational complexity, only 60% of the entire data was used.

4.1 PubMed Abstract Dataset:

The PubMed abstract dataset consists of 2 columns and 119,832 rows as datapoints. Each datapoint in the first column consists of a comprehensive, detailed article that consists of useful medical information retrieved from relevant research. PubMed is a data repository widely used for medical applications throughout the world. Lately, this dataset has been widely used for Deep Learning NLP tasks. The second column consists of summaries of the articles corresponding to it in the same column. The summarized version of these articles is used as documents for the knowledge base in our RAG framework. This is particularly done to maintain simplicity and retain relevant information instead of burdening the model with long texts.

4.2 PMC Patient Dataset:

The PMC patient dataset consists of 167K articles extracted from PubMed Central. It is one of the components of the PubMed dataset collection. This dataset has been annotated by over 3.1M relevant articles and 293K similar patients. The dataset was extracted in the form of JSON files, where each JSON object corresponds to a patient case.

Figure 4 gives statistical insights into the dataset. The figure is divided into 4 graphs, where graphs A and B give insights into the PubMed articles dataset, whereas graphs C and D give insights into the PMC patient articles dataset. We observe the document length and sentence length of text and their distribution in each dataset. We observe that PubMed articles have much shorter abstract versions of articles when compared to PMC patient data. A similar pattern is observed in the length of the sentences for both datasets.

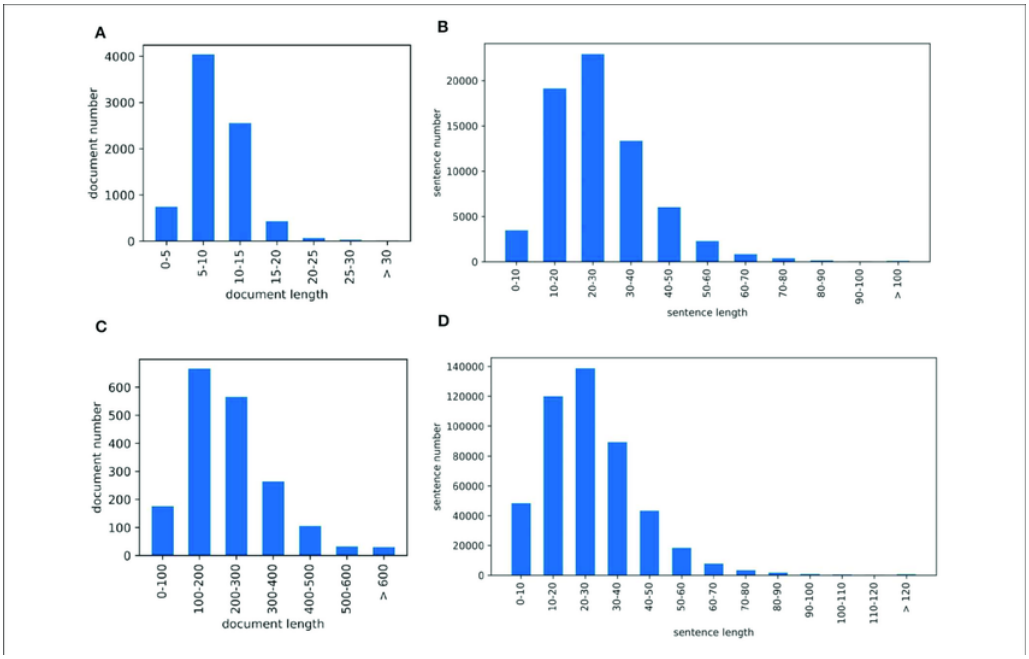

Fig. 4 Dataset Overview

Chapter 5

METHODOLOGY

The flow of data in our proposed model works as follows. Firstly, we take the PMC patients dataset and PubMed articles and convert them into JSON objects, each object is an article from the dataset. This is followed by the chunking process, where each datapoint is split into documents of size 500 words in length with a 20-word overlap in each. This is followed by indexing, where we convert documents into vectors and embed them in a vector space according to similarity. Input query from the user is used to retrieve relevant documents, which are then structured together using prompt engineering to get an output from the LLM model. Figure 5 gives a visual depiction of the flow of data and processes involved in our framework.

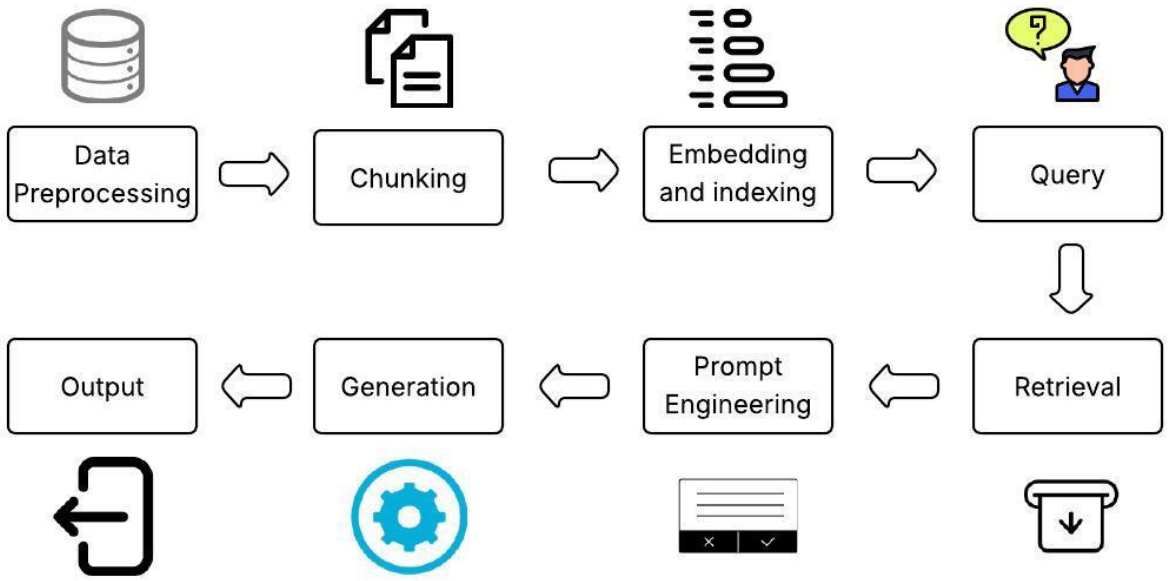

Fig. 5 Process Flow

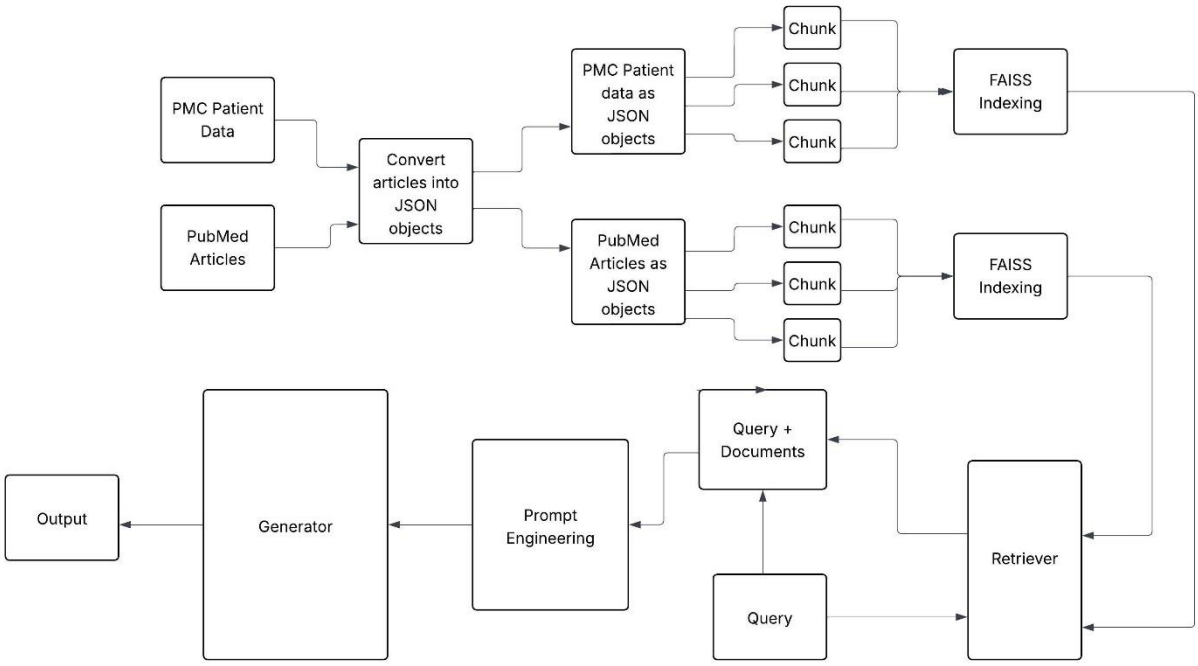

Fig. 6 Model Architecture

Figure 6 represents the architecture of our framework in detail. It represents the direction of flow of data and an in-depth explanation of how the data is processed, transformed, and executed. The following sections provide an in-depth description of this architecture.

5.1 DATA PREPROCESSING

For efficient indexing and reliable retrieval, preprocessing of the dataset plays a crucial role in getting the expected results. To ensure that each dataset aligned with standards of completeness and correctness, the pandas library was used to load and inspect the data. For the PMC Patients dataset, documents with missing values in crucial fields, such as title and abstract, were removed. This is a crucial step that ensures the embedding and retrieval stages operate on meaningful data. Fields such as age are standardized to ensure type consistency, and categorical fields like gender are cleaned to eliminate spelling variations.

In the case of the PubMed dataset, which was originally formatted with field names like "article"

and "summary," renaming operations were performed to match the schema of the PMC dataset. Documents with malformed or empty abstract fields were excluded to maintain the quality of the corpus. The cleaned datasets were then exported as JSON files—PMC_Patients_Clean.json. and pubmed_articles_Clean.json. This format was chosen for its compatibility with the FAISS indexing process, which required access to documents in structured, iterable formats. Figure 7 represents the preprocessing code, where we open each dataset file in CSV format and convert each datapoint into JSON format. We load the raw dataset using pandas and read them from a specific CSV path. This data essentially consists of either research articles or clinical summaries. Pandas library is again used to standardize column names in both datasets for a consistent and uniform retrieval. This is followed by extracting numerics like “year” and “age” using the pandas library and ignoring the NaN values. The outliers are removed for missing “title” and “abstract” columns to make sure outliers and inconsistent data is removed. We use the Pandas library and its functions to convert data from text to JSON format. Figure 8 shows the preprocessing flow, where we run separate scripts for each dataset and use the pandas library to convert them into JSON objects.

```python
print("Loading PMC Patients dataset from:", PMC_RAW_PATH)
df = pd.read_csv(PMC_RAW_PATH)

# Inspect raw data
print("\nRaw PMC Data Preview:")
print(df.head())
print("\nData types:")
print(df.dtypes)

# Rename columns as needed.
# Here, we rename the 'patient' column to 'abstract' so that we have a consistent field for text.
df.rename(columns={
    "patient": "abstract"
}, inplace=True)

# Optionally, you can rename other columns if necessary. For example, if you want to rename
# 'Patient_Title' to 'title', do that here. (Based on your preview, the 'title' column is already present.)

# Convert 'year' and 'age' columns to numeric if they exist.
for col in ["year", "age"]:
    if col in df.columns:
        df[col] = pd.to_numeric(df[col], errors="coerce")

# Drop rows missing essential fields (title and abstract)
df = df.dropna(subset=["title", "abstract"]).reset_index(drop=True)

print("\nCleaned PMC Data Preview:")
print(df.head())
print("\nData types after cleaning:")
print(df.dtypes)

# Save the cleaned dataset as JSON for later use (e.g., indexing)
df.to_json(PMC_CLEAN_PATH, orient="records", indent=4)
print(f"\n✅ Cleaned PMC dataset saved to: {PMC_CLEAN_PATH}")
```

Fig. 7 Preprocessing Code

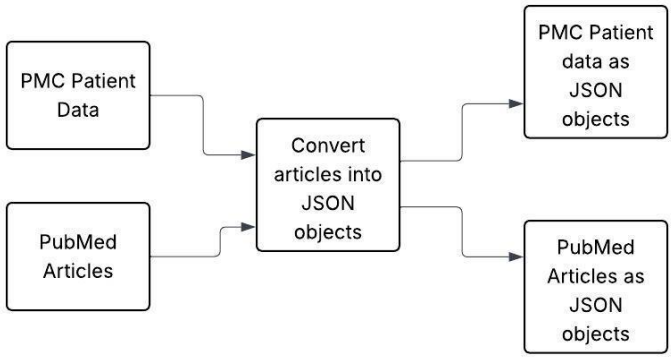

Fig. 8 Preprocessing Flow

5.2 CHUNKING AND EMBEDDING

Chunking is a necessary step in the RAG pipeline, especially when dealing with long texts that exceed the input length of transformer models. Moreover, the model starts to hallucinate when it is bombarded with lengthy articles. To avoid this, chunking was implemented using a sliding window approach. Each document was broken down into segments of 500 tokens, with an overlap of 50 tokens between adjacent chunks. This overlap ensured that key terms and phrases near the boundaries of one chunk were still captured in the next, preserving semantic continuity and minimizing information loss.

Tokenization was initially carried out using the tokenizer associated with the embedding model. In our case, we initially used the tokenizer from the NeuML/pubmedbert-base-embeddings model. However, this model specializes in extracting sentences from the set of corpus. Therefore, we had to shift towards a more sophisticated model that is streamlined to capture the entire concept from the article. For this reason MedCPT tokenizer was used. Once tokenized, the input was truncated to the required length and converted into tensor format. A mean pooling operation was applied across the token dimension to collapse the output into a fixed-size vector. Each of these vectors was then stored in the FAISS index. To improve throughput, embedding was performed in batches. This batch processing not only reduced computational overhead but also took advantage of GPU acceleration, thereby speeding up the embedding pipeline. Figure 9 represents the embedding code, where we use the ncbi/MedCPT-Article_Encoder. After loading the HuggingFace model, AutoTokenizer and AutoModel are used from the transformers library to load the pretrained tokenizer and model weights. The embed_text_batch() function generated an embedding using batch processing. It splits the text into batches and tokenizes it with padding and truncation. The tokenized input is passed to the model to extract the last hidden state for a single embedding per

input text. The chunk_text() function splits text into overlapping chunks where the input text is tokenized and sliced iteratively into overlapping spans. The reversed tokenizing process is implemented using the convert_tokens_to_string() function from the tokenizer. All the chunks are then returned to ensure the entire data is retained. Figure 10 shows that each dataset is processed into chunks separately. The JSON objects go through the embedding and chunking process before going through indexing.

```python
# Load ncbi model and tokenizer
MODEL_NAME = "ncbi/MedCPT-Article-Encoder"
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
model = AutoModel.from_pretrained(MODEL_NAME).eval()

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

def embed_text_batch(texts, batch_size=16):
    """Embed a list of texts using batching."""
    all_embeddings = []

    for i in range(0, len(texts), batch_size):
        batch_texts = texts[i:i + batch_size]
        inputs = tokenizer(batch_texts, return_tensors="pt", truncation=True,
                           max_length=512, padding=True).to(device)
        with torch.no_grad():
            outputs = model(**inputs)
            embeddings = outputs.last_hidden_state.mean(dim=1).cpu().numpy()
            all_embeddings.extend(embeddings)

    return all_embeddings


def chunk_text(text, chunk_size=500, overlap=50):
    """Split text into overlapping chunks of tokens."""
    tokens = tokenizer.tokenize(text)
    chunks = []
    start = 0
    while start < len(tokens):
        end = min(start + chunk_size, len(tokens))
        chunk = tokenizer.convert_tokens_to_string(tokens[start:end])
        chunks.append(chunk)
        start += chunk_size - overlap
    return chunks
```

Fig. 9 Chunking Code

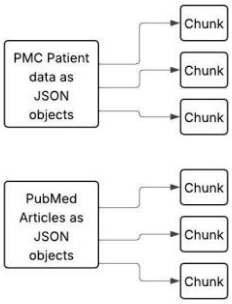

Fig. 10 Chunking Flow

5.2.1 Indexing with FAISS

Indexing constitutes the backbone of the retrieval system. In this project, the FAISS (Facebook AI Similarity Search) library was selected as the indexing engine due to its robustness and scalability. FAISS is a high-performance library developed by Facebook AI that is optimized for searching large-scale collections of dense vectors [28]. It supports both exact and approximate nearest neighbor searches, offering multiple indexing strategies. For our application, IndexFlatL2 was used initially to perform a brute-force search with Euclidean (L2) distance, ensuring high accuracy with slower retrieval time. The flat index stores indices in a sequential format, and a brute force search is performed for retrieval [28].

The process of indexing began with the embedding of each document. However, due to the token limitations of transformer models, long abstracts had to be chunked into smaller segments. Each segment was embedded independently, and the resulting vectors were stored in the FAISS index. For each indexed vector, a corresponding metadata entry was also created and stored in a separate JSON file. This metadata included details like the original document, the chunk index, and the chunk text, enabling the system to retrieve not just the vector but also the human-readable content and its context. This separation of index and metadata facilitated modular access during the retrieval phase. Figure 11 shows the code for FAISS indexing. The function calls for chunking and

embedding have been described in the previous section. After we get the chunked and embedded data, we format them in the form of vectors and create an index depending on the similarity of vectors with each other. Each vector represents a document in our framework. We use the IndexFlat2 function from the FAISS library to implement this. The create_index function is called to initiate the indexing process. After chunking and embedding the text, the numpy library is used to stack all the chunks as vectors using np.stack(). After indexing through IndexFlatL2 we add the vectors to faiss index using faiss.write_index().

```
def create_index(json_path, index_path, text_field="abstract", chunk_size=500, overlap=30):
    with open(json_path, "r", encoding="utf-8") as f:
        documents = json.load(f)

    # ✅ Randomly sample 60% of documents
    sample_size = int(0.6 * len(documents))
    documents = random.sample(documents, sample_size)

    vectors = []
    metadata = []

    for doc in documents:
        text = doc.get(text_field, "")
        if not text:
            continue

        chunks = chunk_text(text, chunk_size=chunk_size, overlap=overlap)
        if not chunks:
            continue

        # ✅ Batched embedding for speed
        chunk_embeddings = embed_text_batch(chunks, batch_size=16)
        for i, (chunk, embedding) in enumerate(zip(chunks, chunk_embeddings)):
            vectors.append(embedding)
            metadata.append({
                "original_doc": doc,
                "chunk_index": i,
                "chunk_text": chunk
            })

    vectors_np = np.stack(vectors)
    dim = vectors_np.shape[1]
    index = faiss.IndexFlatL2(dim)
    index.add(vectors_np)

    # Save FAISS index
    faiss.write_index(index, index_path)
    print(f"FAISS index saved to: {index_path}")
```

Fig. 11 Embedding Code

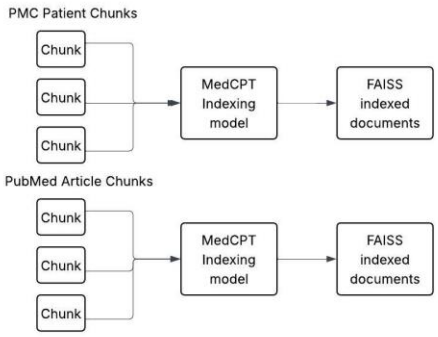

Fig. 12 Embedding and Indexing Flow

Figure 12 builds up from the previous flow shown in Figure 10, where the chunked data goes through the MedCPT embedding model, followed by using the FAISS library features to get a similarity-based vector space.

5.3 RETRIEVER ARCHITECTURE

The retriever component plays a vital role in ensuring that the generator receives only the most relevant information. The retriever works by first embedding the user's query using the same embedding model used for the documents. This ensures that both query and document vectors lie in the same semantic space. The embedded query is then passed to the FAISS index, which returns the top-k nearest neighbors based on L2 distance. These neighbors correspond to the document chunks that are most semantically similar to the query.

In our implementation, we maintained two separate FAISS indexes—one for PMC and one for PubMed. When a query was received, it was passed through both indexes independently. The top K results from each were returned and collated into a context block. This dual-source retrieval strategy was a deliberate choice to combine the strengths of both datasets: the specific, case-based reasoning found in PMC and the broader, research-oriented knowledge from PubMed. The retrieved chunks were sorted and concatenated to form the contextual input to the generator.

Figure 13 gives the code for our retriever model. We use the same embedding model used for indexing. the input query is first converted into an embedded vector, and a search is performed to get the most relevant k documents from each vector space. Embedding of the input query is done in the same way as explained in the previous section, the embed_text() function is called to implement this. After getting the query in the form of a Numpy embedding, the FAISS space is searched using the faiss.read_index() function with the specified path for our FAISS index. Along with the embedded query, the k parameter is passed to determine the number of documents to be retrieved. Finally these indexes are passed to generator after converting them back into strings.

```
# Load the MedCPT model using Transformers
model_name = "ncbi/MedCPT-Article-Encoder"
model = AutoModel.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Load FAISS indices
index_pmc = faiss.read_index(FAISS_INDEX_PMC_PATH)
index_pubmed = faiss.read_index(FAISS_INDEX_PUBMED_PATH)

# Load cleaned documents
with open(PMC_CLEAN_PATH, "r", encoding="utf-8") as f:
    pmc_documents = json.load(f)
with open(PUBMED_CLEAN_PATH, "r", encoding="utf-8") as f:
    pubmed_documents = json.load(f)

def encode_text(text):
    # Tokenize the input text
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True)

    # Extract embeddings using the model
    with torch.no_grad():
        embeddings = model(**inputs).last_hidden_state.mean(dim=1)  # Mean pooling

    # Convert to numpy array
    return embeddings.cpu().numpy()

def retrieve_documents(query, top_k=1):
    # Compute query embedding using the new encode_text function
    query_embedding = encode_text(query)

    # Retrieve from PMC index
    distances_pmc, indices_pmc = index_pmc.search(query_embedding, top_k)
    if len(indices_pmc) == 0 or len(indices_pmc[0]) == 0:
        print("No results found in PMC index.")
        retrieved_pmc = []
    else:
        retrieved_pmc = [pmc_documents[idx] for idx in indices_pmc[0] if idx < len(pmc_documents)]

    # Retrieve from PubMed index
    distances_pubmed, indices_pubmed = index_pubmed.search(query_embedding, top_k)
    if len(indices_pubmed) == 0 or len(indices_pubmed[0]) == 0:
        print("No results found in PubMed index.")
        retrieved_pubmed = []
    else:
        retrieved_pubmed = [pubmed_documents[idx] for idx in indices_pubmed[0] if idx < len(pubmed_documents)]

    return retrieved_pmc, retrieved_pubmed
```

Fig. 13 Retriever Code

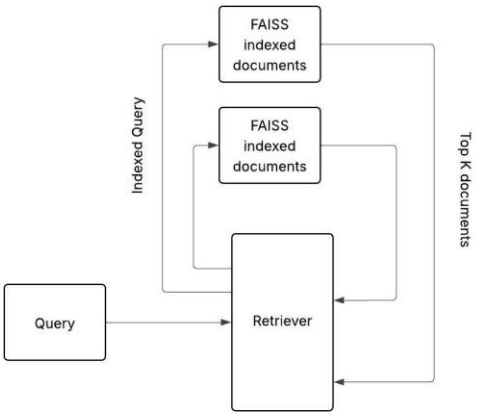

Fig. 14 Retriever Flow

Figure 14 shows the input and retrieval from the retrieved model. The input query is passed into the retriever along with parameter k, specifying the number of documents to be retrieved. The same query is searched in both the indexes to retrieve the most similar documents. The retrieved documents are then passed into the generator model.

5.4 GENERATOR MODELS

Initially, the generative component of the system used Meditron-7B, a biomedical fine-tuned transformer model available through Hugging Face. While Meditron was explicitly trained for the medical domain, it often failed to follow instruction-style prompts. It tended to generate long, verbose answers that did not always stay grounded in the provided context. Meditron model experienced hallucinations when subject to instruction-based prompts and thus often failed to provide answers.

Three models were used to test and verify the performance of the architecture:

- DeepSeek-R1-Distill-Qwen-7B
- Mixtral-8x7B-v0.1
- Medalpaca-7b

The retrieved chunks from the FAISS indexes were embedded into a prompt that was passed to the model prompt. The prompt was structured in three parts: a system message defining the assistant's role, a user query, and the context block with retrieved documents. The models returned a generated response, which was then displayed to the user. This setup ensured that all answers were grounded in the retrieved context and adhered closely to the user's question. Figure 15 shows the code for the generator. Firstly, we define the prompt here, which is discussed in the following section. Prompt is followed by setting the hyperparameters for the model and defining the behavior of our generator model. The model configuration is set to utilize a GPU when available and fall back to CPU in the absence of a GPU. This ensures efficient utilization of hardware resources. The tokenizer is used to convert the entire prompt into token IDs. The truncation is set to true to avoid overflow, and the maximum length and padding are set to 4096 in order to get consistent output for each query. The model.generate() function is used to set parameters for output. The temperature is set to a very low value of 0.2 to avoid randomness in answers. The output length is set to 500 to keep a concise answer, and do_sample is set to false to keep the output deterministic. The model is developed to generate consistent and concise answers. Finally, the tokenizer.decode() is used to convert the tokens back into human human-readable format. The correct diagnosis option is appended to "Final Answer:" and "No valid answer generated" is returned in case the model is confused.

Figure 16 demonstrates that the query and retrieved documents from the retriever are combined to form a prompt. Smart prompt is used to drive the output of the generator, and the output is obtained and used for evaluation.

```
prompt = prompt_template.format(query=query, context=context)

# Determine device: use GPU if available, else CPU.
device = "cuda" if torch.cuda.is_available() else "cpu"

inputs = tokenizer(
    prompt,
    return_tensors="pt",
    truncation=True,
    max_length=4096,
    padding="max_length"
)
input_ids = inputs.input_ids.to(device)
attention_mask = inputs.attention_mask.to(device)

# Generate output
with torch.no_grad():
    output = model.generate(
        input_ids=input_ids,
        attention_mask=attention_mask,
        max_new_tokens=500,
        do_sample=False,
        temperature=0.2,
        pad_token_id=tokenizer.eos_token_id
    )

# Decode the generated tokens
output_text = tokenizer.decode(output[0], skip_special_tokens=True)

# Extract the answer using LangChain's pattern recognition capabilities
if "Final Answer:" in output_text:
    answer = output_text.split("Final Answer:")[-1].strip()
else:
    answer = "No valid answer generated."

return answer
```

Fig. 15 Generator Code

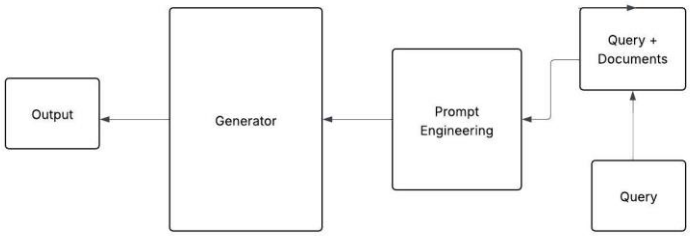

Fig. 16 Generator Flow

5.5 PIPELINE EXECUTION AND OPTIMIZATION

The entire system was orchestrated through a single Python script (test_query.py) which served as the user interface. Upon receiving a query, the script invoked the retriever to obtain relevant document chunks. These chunks were then assembled into a prompt and passed to the generator module. The generator returned the answer, which was printed to the console. This modular pipeline structure made it easy to debug and extend individual components.

Several optimizations were implemented to improve performance and robustness. Embedding was batched to minimize latency. Metadata files were created for every indexed document, enabling fast reverse lookups. Finally, random sampling was used during development to reduce runtime while preserving the statistical diversity of the dataset.

5.6 PROMPT ENGINEERING

5.6.1 Initial Prompt: Naive Implementation

The first approach directly concatenated the medical query and context without structural guidance ("Answer: {query} using {context}"). This failed because the model lacked clear directives to distinguish between its internal knowledge and the context provided. It freely blended outdated training data with context details, producing responses that mixed generic advice with inappropriate specifics. The absence of output constraints also allowed verbatim copying of context text, such as quoting demographic details from irrelevant studies, making responses clinically unreliable.

5.6.2 Second Phase: Basic Structural Separation

Introducing XML-like tags (<MEDICAL_QUERY>, <CONTEXT>) created visual separation but

insufficient operational guidance. While the model recognized distinct sections, it treated them as formatting rather than processing stages. The tags failed to establish analysis protocols, allowing the model to draw flawed connections between unrelated clinical scenarios. Responses often included phrases like "as shown above" referring to context content, violating patient-facing communication standards. The model failed to decouple the actual question from the context provided.

5.6.3 Third Iteration: Explicit tagged instructions

To allow the model to understand the core features of the prompt, an <IMPORTANT> tag was used to enforce the model on a specific set of instructions. Examples of these included asking the model not to repeat the context and question in the generated response. This was especially important as the model utilized its output token limit on regenerating the entire prompt with the context provided. However, this strategy was barely successful as the number of retrieval documents increased, and the models generated random responses again.

5.6.4 Structured Prompting with LangChain

Langchain is widely considered for its ability to seamlessly integrate and interact with data to be used in LLM applications [29].

The integration of LangChain facilitated structured prompting and more control over the input and output of the model. It helped enforce consistency and improve the performance of the model. PromptTemplate and retriever integration contributed significantly towards improving the overall architecture and output quality of the model.

5.6.4.1 Phased Processing Architecture

Langchain provides prompt template to construct specialized prompts [29]. Adaption of LanChain's prompt template was crucial in performing a phased processing architecture. This ensured the structured processing of the input and reduced model hallucinations. It helped enforce a separation between the query and the context, which turned out to be crucial in generating a consistent and correct response. The phased processing helped separate the processing of query and context along with a separate set for additional instructions.

5.6.4.2 Silent Context Integration

Silent context integration was a key advancement towards generating a correct response. Because the context might not always contain text related to query at hand, it is crucial that symptoms are evaluated independently, without citing sources from the context. The model is encouraged to utilize the relevant information and generate an appropriate response. This helped model from deviating from the task at hand.

5.6.4.3 Structured Output Protocol

LangChain's prompt setup allows for explicit instructions that enhance clarity and coherence of responses. By enforcing the format *"Final Answer: [Option Letter] - [Explanation]"*, the system achieves a standardized output structure that aligns well with clinical communication standards. The structured output protocol also enhances evaluation efficiency by simplifying the extraction of answers during benchmarking and assessment. Although this format generated the best results, when the size of the context increased the model occasionally deviated from the required output format.

5.6.4.4 Response Trigger Isolation

An important feature of LangChain is its ability to delineate between different stages of the prompt through triggers such as RESPONSE:. This helped create a clear separation between different sections of the prompt. By using a standardized trigger for response generation, LangChain eliminates confusion in the instructions provided.

5.7 CURRENT PROMPT IMPLEMENTATION WITH LANGCHAIN

This section provides the final prompt created using LangChain for our RAG model. PromptTemplate is a function that LangChain uses to structure the input given to our model. The input variables array defines the input given to the model apart from general instructions. The query object corresponds to the input query by the user, whereas the context object contains documents returned by the retriever for that specific query. The template feature is used to arrange instructions and custom input for the model.

The template starts with instructions, where we define the roles and responsibilities of the model. This is essentially useful for general-purpose LLMs, not specifically fine-tuned for medical tasks. This is followed by the query, context, and system prompts. The query is the input medical question that our model tries to solve. The context is the relevant scientific information and similar cases surrounding the input query. Lastly, the system prompts drive the direction of output for our LLM model, including additional features like Chain-of-Thought inference and differential diagnosis. Finally, we define the start token for our response for consistent output retrieval. Following is the format of our prompt in code format.

Final Prompt:

```
prompt_template = PromptTemplate(
    input_variables = ["query", "context"],
    template=
    """
You are a highly skilled medical diagnostic AI. Based on the provided context and your knowledge, respond to the query.

    Query: {query}

    Context:
    {context}

    Instructions:
    - Only provide the final answer in the format: "Final Answer: [Option Letter] - [Explanation]".
    - Provide the Differential Diagnosis
    - Do not repeat the query or context in the response.
    - Provide diagnosis and concise explanation

    RESPONSE:
    """
)
```

Chapter 6

EXPERIMENTS AND RESULTS

The experiments in this research are aimed at improving the performance of medical diagnostic models using the RAG framework. Tests were carried out by varying the number of relevant documents fed into the model for each LLM in consideration. Tests were carried out on the MedQA dataset to test the performance of the model on complex medical queries. MedQA is an open-domain question answering dataset from medical exams. Each point in the dataset consists of a complex medical query with 4 options, one of them being the correct answer. MedQA is the most widely used model to benchmark the performance of diagnostic AI models. Due to the computational complexity, only a fraction of this dataset was randomly sampled to test our models. Therefore, the results obtained in these set of experiments might not indicate the true metrics for the entire dataset. However, the key observation of these experiments is to monitor the gradual progression of the model in our RAG framework. This section discusses the experiments performed and results obtained by using pre-trained LLM models on our RAG framework. We also establish a distinction between a model's true performance against its performance in our framework. The aim is to compare these models and derive meaningful conclusions. For model evaluation HuggingFace dataset has been used due to its ease of use and customization. A dataset named *'bigbio/med_qa'* has been used for model evaluation. In the following discussions, K indicates the number of documents used in the RAG framework from each of the two datasets.

6.1 EVALUATION METRICS

This section outlines the metrics used to evaluate the performance of the models, including accuracy and F-1 score.

6.1.1 Accuracy

Accuracy is the most fundamental and widely used metric to test performance on classification tasks. It is more commonly used in machine learning and statistical models. It is the most straightforward and intuitive metric to measure the efficiency of a model. It is defined as the ratio of the number of correct predictions to the total number of predictions [30].

$$Accuracy = \frac{Number\ of\ correct\ precdictions}{Total\ Number\ of\ Predictions}$$

However, accuracy fails to truly reflect the performance of the model if the dataset is skewed or imbalanced. It fails to account for all types of errors. It is very beneficial only when we have balanced data with equal distribution among classes.

6.1.2 F-1 Score

The F-1 score is the harmonic mean of precision and recall. It is a measure to strike a balance between the two metrics. Precision is the ratio of true positives over all the values predicted as positive [31]. Whereas recall is the ratio indicating the measure of true predicted over all the actual positives in the dataset. It overcomes the limitations of accuracy and is a good indicator for imbalanced datasets.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F-1\ score = \frac{2\ x\ Precision\ x\ Recall}{Precision + Recall}$$

6.2 DeepSeek-R1-Distill-Qwen-7B

DeepSeek-R1-Distill-Qwen-7B belongs to a family of models distilled from DeepSeek-R1, which serves as a first-generation model in DeepSeek LLM models [32]. The model is trained by directly applying Reinforcement Learning (RL) without applying supervised fine-tuning [32]. Several small models were created as distilled versions from the DeepSeek-R1 model [32]. These distilled models showed exceptional reasoning ability [32]. The parent model achieved exceptional results in multiple domains over its market competitors [32]. However, the official documentation does not state the expertise in the medical domain, and we assume that the model has the ability to generate output similar to general-use models.

6.2.1 Performance of DeepSeek-R1-Distill-Qwen-7B

Figure 17 shows that the performance of this model improved significantly when we introduced the model inside the RAG framework. The model benefits from an increase in the amount of context provided when compared to its performance without the context. A similar trend is seen for both accuracy and F-1 score as shown in Figure 17 and 18. The accuracy and F-1 scores shoot up when the model is provided with more than 4 documents each.

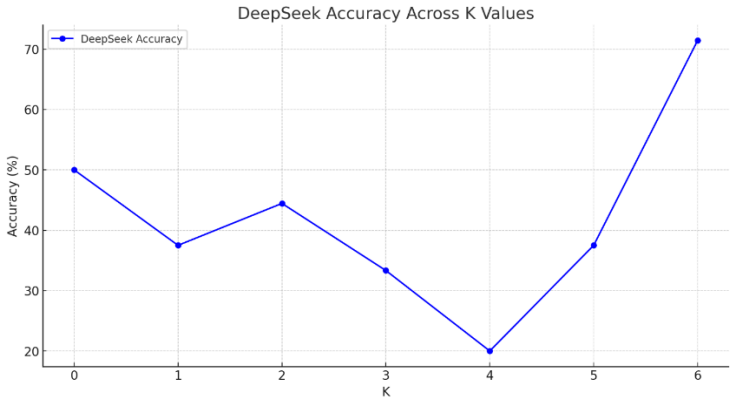

Fig. 17 DeepSeek accuracy on MedQA

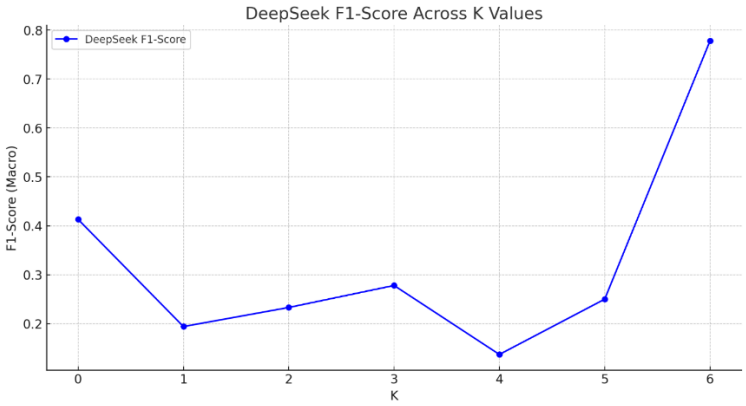

Fig. 18 DeepSeek F-1 score on MedQA

6.3 Mixtral-8x7B

Mixtral-8x7B is a sparse mixture of expert models (SMoE) with open weights [33]. It is a robust open-source model with an overall superior cost/performance trade-off [33]. It competes with and outperforms GPT-3.5 on most of the standard benchmarks [33]. The model is well known for its fast inference and outperforms bigger models in less amount of time. It has the ability to handle a context of 32k tokens and is fine-tuned towards an instruction-following model [33].

6.3.1 Performance of Mixtral-8x7B

Figure 19 demonstrates the superior performance of Mixtral on medical queries even without the introduction of context. The model is trained to be an expert in multiple domains and demonstrates strong expertise in the field of medical diagnosis. Moreover, when the context is introduced, the performance of the model drops initially. However, the performance improves with an increase in the number of documents provided as context. Additionally, Figure 20 shows that while accuracy shoots up, the F-1 score remains pretty much constant. This increase is observed in steps, owing to the size of the dataset at hand. However, in our experiments, the model never reaches the heights of metrics achieved when tested outside the RAG framework.

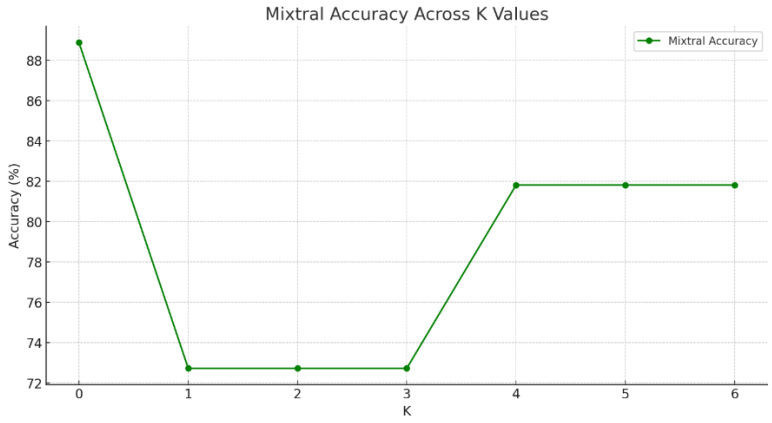

Fig. 19 Mixtral accuracy on MedQA

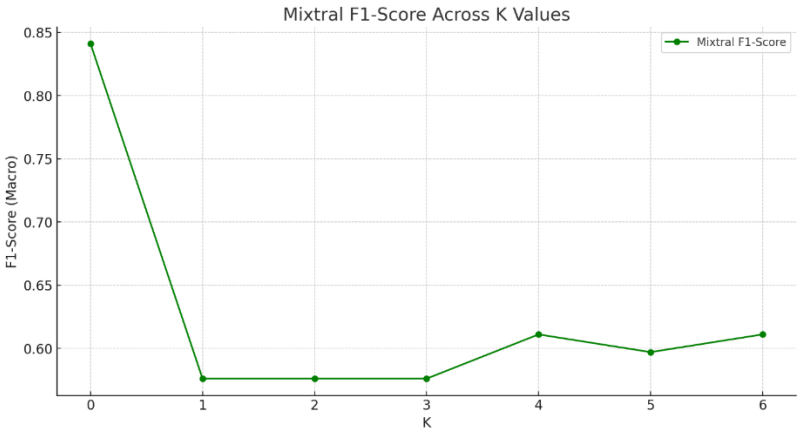

Fig. 20 Mixtral F-1 score on MedQA

6.4 Medalpaca-7b

Medalpaca is a family of fine-tuned open-source LLMs on medical datasets [11]. It achieves competitive diagnostic performance across USMLE questions [11]. Consists of a dataset called Medical Meadow. It consists of 2 categories: a collection of NLP tasks and a collection of medical data from the internet [11].

6.4.1 Performance of MedAlpaca

MedAlpaca demonstrates a reasonable improvement in the introduction of the RAG framework. However, the performance disintegrates severely once the size of the context increases. Figures 21 and 22 demonstrate a significant drop in both accuracy and F-1 score for MedAlpaca when the K-value increases beyond 3. However, it is interesting to note that the peak performance is observed when the model is subjected to the RAG framework rather than when outside the RAG framework.

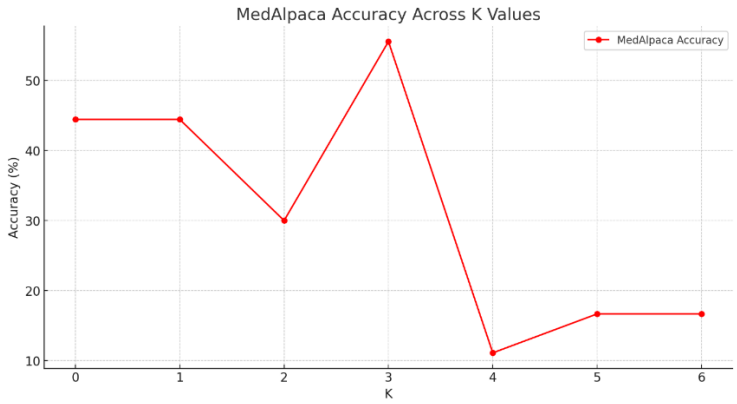

Fig. 21 MedAlpaca accuracy on MedQA

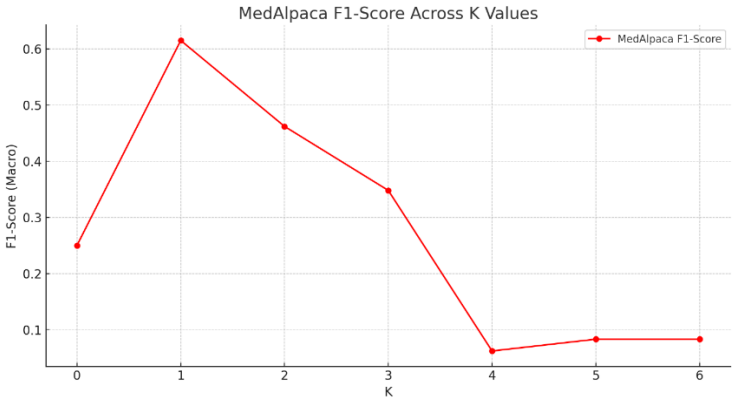

Fig. 22 MedAlpaca F-1 score on MedQA

6.5 Performance Comparison

Figures 23 and 24 show a detailed performance comparison of all the models. They highlight the facts not observed in individual comparison. Among all the models, Mixtral performs the best in answering medical queries. However, it performs best outside the RAG framework. Unlike other models that achieved peak performance scores with context in the RAG framework, Mixtral experiences a dip in performance with the introduction of context. However, interestingly, the performance improves with an increase in the number of RAG documents. Therefore, it cannot be determined that Mixtral suffers from context; rather, we can derive a conclusion that the model needs enough context to improve its performance. DeepSeek experiences a continuous increase in performance similar to Mixtral with an increase in the number of documents. However, MedAlpaca does not demonstrate the same behaviour, rather, it experiences a severe dip in performance when the k value is more than three. The dip in performance of the model on the introduction of the RAG framework aligns with research in [34].

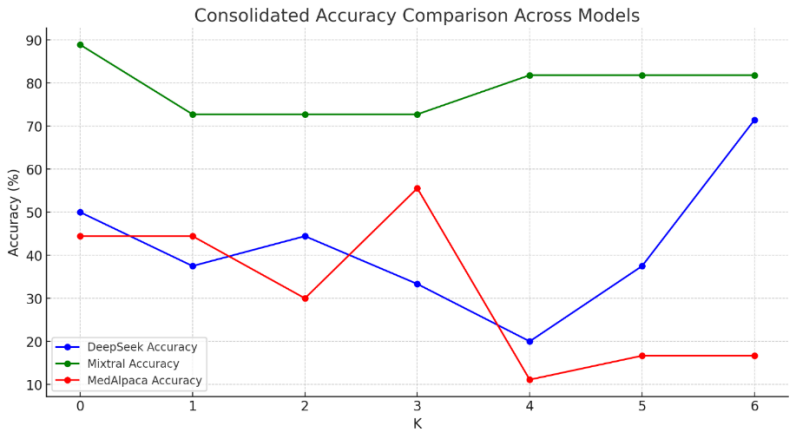

Fig. 23 Comparison of models on accuracy scores

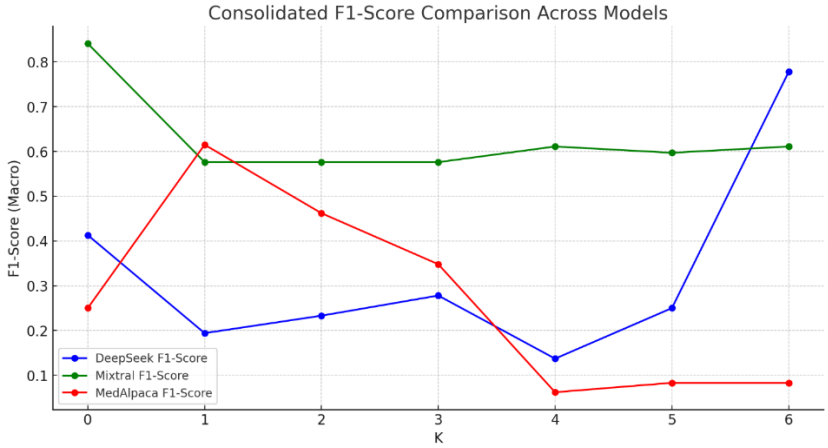

Fig. 24 Comparison of models on F-1 scores

Chapter 7

Conclusion and Future Work

Through this research, we have identified the problems faced by contemporary AI healthcare models. The literature survey underlines the limitations of current diagnostic models and establishes four main components required to build an acceptable disease diagnosis agent using LLMs. Contemporary models are decoupled from the knowledge base, which makes them not only harder to maintain but also inefficient in responses. Moreover, current models are more focused on accuracy and results, often neglecting the importance of explainability and reasoning. This research explains the importance of foundational knowledge in the form of a knowledge base to drive the responses and keep the model up to date. Additionally, it underlines the need for differential diagnosis and chain-of-thought inference to gain confidence in the performance of the model. Lastly, it calls for an open-source framework to engage the community, keep ethical concerns in check, and provide complete transparency over the development of the model.

The research establishes a framework to develop a robust medical diagnosis agent. The RAG framework helps establish a robust knowledge base. From the experiments performed, we have observed that the performance of the model predictions improves with an increase in the number of documents in using our RAG framework. This is a general trend observed in the models considered; however, the trends observed are not the same throughout. Models specialized in the medical domain do not show a significant gain in performance when compared to general-purpose models. We observe a similar trend for the Mixtral model as observed by G. Xiong et al. in [34], where the peak accuracy is achieved when the model is used outside the RAG framework. Similar to their study, we observe a dip in accuracy scores for the Mixtral model with the introduction of

documents in the RAG framework. Whereas a more general model like DeepSeek shows a higher increase in accuracy scores inside the RAG framework.

The experiment setup does not test the quality of reasoning and differential diagnosis due to the lack of appropriate standards to test these metrics. However, future iterations of this framework can introduce a mechanism that tests the model on the entire output generated.

REFERENCES

[1] H. Wang, S. Zhao, Z. Qiang, N. Xi, B. Qin, T. Liu, “Beyond Direct Diagnosis: LLM-based Multi-Specialist Agent Consultation for Automatic Diagnosis,” 2024, . [Online]. Available: https://doi.org/10.48550/arXiv.2401.16107 (accessed Dec. 8, 2024)

[2] M. Rosendal, D. E. Jarbøl, A. F. Pedersen, and R. S. Andersen, "Multiple Perspectives on Symptom Interpretation in Primary Care Research," *BMC Family Practice*, vol. 14, no. 167, Nov. 2013. [Online]. Available: https://bmcprimcare.biomedcentral.com/articles/10.1186/1471-2296-14-167 (accessed Dec. 8, 2024)

[3] A. F. Tchango, R. Goel, J. Martel, Z. Wen, G. M. Caron, and J. Ghosn, "Towards Trustworthy Automatic Diagnosis Systems by Emulating Doctors’ Reasoning with Deep Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 35, 2022. [Online]. Available: https://arxiv.org/abs/2210.07198 (accessed Dec. 8, 2024)

[4] M. Abbasian, I. Azimi, A. M. Rahmani, and R. Jain, "Conversational Health Agents: A Personalized LLM-Powered Agent Framework," *arXiv preprint arXiv:2310.02374*, Sep. 2024. [Online]. Available: https://arxiv.org/pdf/2310.02374 (accessed Dec. 8, 2024)

[5] K. Singhal, et al., “Towards Expert-Level Medical Question Answering with Large Language Models,” *arXiv e-prints.* 2023. DOI: 10.48550/arXiv.2305.09617
Link - https://doi.org/10.48550/arXiv.2305.09617 (accessed Dec. 8, 2024)

[6] C. Wu et al., "A Medical Diagnostic Assistant Based on LLM," *Communications in Computer and Information Science, Singapore*, Springer Nature Singapore, pp. 135-147, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-981-97-1717-0_12 (accessed Dec. 8, 2024)

[7] Z. Chen *et al.*, "MEDITRON-70B: Scaling Medical Pretraining for Large Language Models," *arXiv preprint arXiv:2311.16079*, Nov. 2023. [Online]. Available: https://arxiv.org/pdf/2311.16079 (accessed Dec. 8, 2024)

[8] B. Yang *et al, “*DrHouse: An LLM-empowered Diagnostic Reasoning System through Harnessing Outcomes from Sensor Data and Expert Knowledge*,”, 2024.* DOI: 10.48550/arXiv.2405.12541

[9] M. Jin *et al*, “Health-LLM: Personalized Retrieval-Augmented Disease Prediction System,” 2024. DOI: 10.48550/arXiv.2402.00746.
Link - https://arxiv.org/abs/2402.00746 (accessed Dec. 8, 2024)

[10] U. Naseem, A. Bandi, S. Raza, J. Rashid, and B. R. Chakravarthi, "Incorporating Medical Knowledge to Transformer-Based Language Models for Medical Dialogue Generation," in *Proceedings of the 21st Workshop on Biomedical Language Processing*, Dublin, Ireland, May 2022, pp. 110–115. [Online]. Available: https://www.researchgate.net/publication/361321718_Incorporating_Medical_Knowledge_to_Transformer-based_Language_Models_for_Medical_Dialogue_Generation (accessed Dec. 8, 2024)

[11] T. Han *et al.*, "MedAlpaca: An Open-Source Collection of Medical Conversational AI Models and Training Data," *arXiv preprint arXiv:2304.08247*, Oct. 2023. [Online]. Available: https://arxiv.org/pdf/2304.08247 (accessed Dec. 8, 2024)

[12] A. Toma *et al.*, "Clinical Camel: An Open Expert-Level Medical Language Model with Dialogue-Based Knowledge Encoding," *arXiv preprint arXiv:2305.12031*, Aug. 2023. [Online]. Available: https://arxiv.org/pdf/2305.12031 (accessed Dec. 8, 2024)

[13] T. Tao et al. “Towards Conversational Diagnostic AI,” *ArXiv* abs/2401.05654 (2024). DOI: 10.48550/arXiv.2401.05654

[14] T. Abdullahi, R. Singh, and C. Eickhoff, "Learning to Make Rare and Complex Diagnoses With Generative AI Assistance: Qualitative Study of Popular Large Language Models," *JMIR Medical Education*, vol. 10, p. e51391, Feb. 2024. [Online]. Available: https://mededu.jmir.org/2024/1/e51391/PDF (accessed Dec. 8, 2024)

[15] Z. Al Nazi & W. Peng, “Large Language Models in Healthcare and Medical Domain: A Review,” *Informatics*. (2024). DOI:11. 57. 10.3390/informatics11030057.

[16] G.K. Gupta, A. Singh, S.V. Manikandan & A. Ehtesham, “Digital Diagnostics: The Potential Of Large Language Models In Recognizing Symptoms Of Common Illnesses,” DOI: 10.48550/arXiv.2405.06712

[17] U. Dhakal *et al.*, "GPT-4's Assessment of Its Performance in a USMLE-Based Case Study," *arXiv preprint arXiv:2402.09654*, Mar. 2024. [Online]. Available: https://arxiv.org/pdf/2402.09654 (accessed Dec. 8, 2024)

[18] D.P. Panagoulias, M. Virvou, G. A. Tsihrintzis, “Evaluating LLM -- Generated Multimodal Diagnosis from Medical Images and Symptom Analysis,” 2024. DOI: 10.48550/arXiv.2402.01730.

[19] A. Choudhury and Z. Chaudhry, "Large Language Models and User Trust: Consequence of Self-Referential Learning Loop and the Deskilling of Health Care Professionals," *Journal of Medical Internet Research*, vol. 26, p. e56764, Apr. 2024. [Online]. Available: https://www.jmir.org/2024/1/e56764/ (accessed Dec. 8, 2024)

[20] Z. Fan *et al*, “AI Hospital: Benchmarking Large Language Models in a Multi-agent Medical Interaction Simulator,” 2024. DOI: 10.48550/arXiv.2402.09742

[21] V. Pimprale, N. Deshmukh, and S. Arora, "Early Prediction of Critical Diseases Using Machine Learning and Image Processing," *African Journal of Biomedical Research*, vol. 27, no. 3, pp. 679–684, Nov. 2024. [Online]. Available: https://www.ajol.info/index.php/ajbr/article/view/283262 (accessed Dec. 8, 2024)

[22] A. Serag *et al.*, "Translational AI and Deep Learning in Diagnostic Pathology," *Frontiers in Medicine*, vol. 6, p. 185, Oct. 2019. [Online]. Available: https://www.frontiersin.org/journals/medicine/articles/10.3389/fmed.2019.00185/full (accessed Dec. 8, 2024)

[23] OpenAI *et al.*, "GPT-4 Technical Report," *arXiv preprint arXiv:2303.08774*, Mar. 2024. [Online]. Available: https://arxiv.org/pdf/2303.08774 (accessed Dec. 8, 2024)

[24] A. Ríos-Hoyo, N.L. Shan, A. Li, A.T. Pearson, L. Pusztai, F.M. Howard, “Evaluation of large language models as a diagnostic aid for complex medical cases,” *Front Med (Lausanne)*. 2024 Jun 20;11:1380148. doi: 10.3389/fmed.2024.1380148. PMID: 38966538; PMCID: PMC11222590.

[25] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020. [Online]. Available: https://arxiv.org/pdf/2005.11401 (accessed Dec. 8, 2024)

[26] J. Wei *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824–24837, 2022. [Online]. Available: https://arxiv.org/pdf/2201.11903 (accessed Dec. 8, 2024)

[27] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *Foundations and Trends in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018. [Online]. Available: https://arxiv.org/pdf/1811.12560 (accessed Dec. 8, 2024)

[28] M. Douze et al., "The Faiss Library," arXiv preprint arXiv:2401.08281, Jan. 2024. [Online]. Available: https://arxiv.org/abs/2401.08281 (accessed Apr 5, 2025)

[29] O. Topsakal and T. C. Akinci, "Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast," *International Conference on Applied Engineering and Natural Sciences*, vol. 1, no. 1, pp. 1050–1056, Jul. 2023. [Online]. Available: https://www.researchgate.net/publication/372669736_Creating_Large_Language_Model_Applications_Utilizing_LangChain_A_Primer_on_Developing_LLM_Apps_Fast (accessed Apr 5, 2025)

[30] C. E. Metz, "Basic principles of ROC analysis," *Semin. Nucl. Med.*, vol. 8, no. 4, pp. 283–298, Oct. 1978, doi: 10.1016/S0001-2998(78)80014-2.

[31] A. A. Taha, "Metrics for evaluating 3D medical image segmentation: analysis, selection, and tool," *BMC Med. Imaging*, vol. 15, no. 29, pp. 1–28, 2015, doi: 10.1186/S12880-015-0068-X.

[32] DeepSeek-AI et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," *arXiv preprint arXiv:2501.12948*, Jan. 2025. [Online]. Available: https://arxiv.org/abs/2501.12948. (accessed Apr 5, 2025)

[33] A. Q. Jiang et al., "Mixtral of Experts," arXiv preprint arXiv:2401.04088, Jan. 2024. [Online]. Available: https://arxiv.org/abs/2401.04088. (accessed Apr 5, 2025)

[34] G. Xiong, Q. Jin, Z. Lu, and A. Zhang, "Benchmarking Retrieval-Augmented Generation for Medicine," *arXiv preprint arXiv:2402.13178*, Feb. 2024. [Online]. Available: https://arxiv.org/abs/2402.13178 (accessed Apr 5, 2025)