SRH HOCHSCHULE HEIDELBERG

MASTERS THESIS

AI-Driven Medical Chatbot for Medical Students and Patients: An LLM and Embedding-Based Approach

Author:

NATARAJU TUMAKURU SHREESHYLESHA

Matriculation Number: 11027742

Supervisors:

PROF. DR. CHANDNA SWATI

A thesis submitted in fulfilment of the requirements for the degree of Master Of Science In Applied Data Science and Analytics

in the

School of Information, Media and Design

25th February, 2025

Declaration of Authorship

I, **Nataraju Tumakuru Shreeshylesha**, declare that this thesis titled, *"AI-Driven Medical Chatbot for Medical Students and Patients: An LLM and Embedding-Based Approach"* and the work presented in it is my own. I confirm that this work submitted for assessment is expressed in my own words and is my own. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are appropriately acknowledged at any point of their use. A list of the references employed is included.

- This work was done wholly for a research degree at this University.
- It has been clearly stated in this work if any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution.
- Where I have consulted the published work of others, credit is always given to the concerned work.
- Where I have quoted from the work provided by others, the source is always given unless it is entirely my work.
- I have acknowledged all main sources that supported me and helped during this work.

Signed: _____

Date: _____

Acknowledgements

I am deeply grateful for the support and guidance of numerous individuals who have contributed to the completion of this thesis, and I would like to express my heartfelt appreciation to all of them.

First and foremost, I would like to extend my sincere gratitude to my supervisor, Prof.Dr.Chandna Swati, for her invaluable guidance, expertise, and unwavering support throughout this research. Her mentorship has played a crucial role in shaping my ideas and refining my work.

I am also grateful to the faculty and staff of the School of Information, Media and Design at SRH Hochschule Heidelberg for their continuous encouragement and for providing an enriching environment for research and learning.

Special thanks to my colleagues and friends for their motivation, insightful discussions, and unwavering support. Their camaraderie has made this journey more fulfilling and enjoyable.

I am profoundly indebted to my family for their unconditional love, patience, and encouragement throughout this journey. Their unwavering belief in me has been my greatest source of strength and perseverance.

Lastly, I extend my gratitude to everyone who directly or indirectly contributed to the successful completion of this thesis. Your support and encouragement mean the world to me.

Abstract

Artificial intelligence (AI) is rapidly transforming the landscape of medical education and patient support. This thesis proposes an AI-driven medical chatbot system designed to serve two main user roles—medical students and patients—via distinct modes of operation. For medical students, the system provides an interactive knowledge platform with detailed explanations of complex topics, whereas for patients, it offers symptom analysis, healthcare provider recommendations, and even autonomous appointment booking through Agentic AI capabilities.

The architecture comprises several key components. First, textual data (such as PDFs of medical articles or learning materials) undergoes text extraction and chunking before being transformed into embeddings by a dedicated model. These embeddings are then indexed in a vector database, enabling efficient similarity-based retrieval. When a user submits a query, the system identifies and retrieves the most relevant text segments and subsequently augments the prompt for a Large Language Model (LLM), which can be tailored to specific roles: a learning AI assistant (e.g., Llama 2, 13B) for medical students and an agentic diagnostic AI (e.g., Mistral 13B) for patients.

In student mode, the chatbot focuses on explaining medical concepts and providing in-depth, evidence-based knowledge. In patient mode, the chatbot employs symptom analysis to generate preliminary diagnostic suggestions, seamlessly directing users to the HealthCare Provider Recommendation System. Harnessing its Agentic AI feature, the system can autonomously schedule appointments if necessary, reducing administrative overhead and expediting patient care. The entire process is orchestrated through a secure login mechanism, ensuring that only authorized users can access specific functionalities suited to their roles.

Experimental evaluations draw upon real-world medical texts and feedback from both medical students and healthcare practitioners. Early results indicate that this approach effectively balances accurate medical guidance, educational depth, and practical utility in assisting patients. Future enhancements include expanding multi-language support, improving the LLM's medical domain expertise, and integrating real-time health data streams to strengthen diagnostic accuracy.

Keywords: AI-driven Medical Chatbot, Role-based LLM, Patient Symptom Analysis, Healthcare Provider Recommendation, Autonomous Appointment Booking, Embedding Model, Vector Database

Contents

List of Figures

List of Tables

Listings

Abbreviations

CTM	Correlated Topic Model
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
PLSA	Probabilistic Latent Semantic Analysis
NMF	Non-Negative Matrix Factorization
TF-IDF	Term Frequency-Inverse Document Frequency
SVD	Singular Value Decomposition
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
HAC	Hierarchical Agglomerative Clustering
BERT	Bidirectional Encoder Representations from Transformers
NLP	Natural Language Processing
BIRCH	Balanced Iterative Reducing and Clustering Hierarchies
BOW	Bag-of-words
GloVe	Global Vectors for Word Representation
ML	Machine Learning
SBERT	Sentence-Bidirectional Encoder Representations from Transformers
GBERT	German-Bidirectional Encoder Representations from Transformers
T5	Text-To-Text Transfer Transformer
RoBERTa	Robustly Optimized BERT
UMAP	Uniform Manifold Approximation and Projection
PCA	Principal Component Analysis
T-SNE	T-Distributed Stochastic Neighbor Embedding
LLM	Large Language Models
GPT	Generative Pre-trained Transformer
LLaMA	Large Language Model Meta AI
JSON	JavaScript Object Notation

Chapter 1. Introduction

1.1 Introduction

AI (Artificial Intelligence) has seamlessly integrated with modern medicine and medical education, with the potential to improve patient outcomes and successfully train the next generation of physicians. Traditionally, patient-oriented solutions have either been rule-based systems or static information portals, and generally missed the capacity to comprehend context comprehensively, to adjust in a fluid manner, and to execute driving operations without human supervision. Likewise, learning tools in medicine have largely been passive, forcing users into static content repositories or low-interactivity platforms that fail to capture the complexity of medical concepts. This thesis addresses these limitations by proposing a role-based **AI-driven medical chatbot** capable of serving two distinct user groups:

- **Medical Students:** Learners find themselves in an interactive learning atmosphere, which allows students to ask about medical topics and get real-time, evidence-based answers.
- **Patients:** A simple interface for automatic appointment booking, healthcare provider suggestions, and symptom evaluation.

The system's architecture harmonizes several advanced technologies:

1. **Data Ingestion and Chunking:** Relevant medical texts (e.g., PDFs, articles, or curated training materials) are split into smaller, semantically consistent segments for informative retrieval and information reduction.

2. **Embedding and Vector Storage:** Embed every item under a model, say an LLM-based encoder employing numerical embedding and vector storage. By use of a vector database indexing, these embeddings exhibit fast similarity. Underline quickly sections from the database most likely appropriate for a user's search.

3. **Retrieval-Augmented Generation (RAG):** The system searches the top-matching chunks from the vector database upon user query submission, hence augmenting the generation. The language model (LLM) then makes use of these acquired segments as background, therefore improving the validity and accuracy of its produced answers.

4. **Role-based Large Language Models:** The chatbot distinguishes between "patient mode" (giving symptom-based guidance and possible next actions) and "student mode," which provides comprehensive medical knowledge and conceptual explanations. This role-specific strategy guarantees patients have easily available assistance and in-depth answers for pupils.

5. **Agentic AI for Appointment Booking:** The system combines an agentic component that enables autonomous scheduling or rescheduling of appointments. This function simplifies patient travels and lowers administrative overhead for physicians, therefore maybe improving access to care.

From profound academic support for medical students to pragmatic, real-world patient treatments, the suggested chatbot therefore covers a wide spectrum of capabilities. The architecture addresses current gaps in both user groups' experiences by combining embedding-based retrieval, role-aware LLMs, and autonomous scheduling: medical students acquire a dynamic tutor that can recall and contextualize complex material, while patients receive an accessible, on-demand assistant that not only provides initial information but also handles logistical tasks like finding the right specialist and booking an appointment.

Furthermore, ensuring a robust, user-centric design requires addressing core challenges. These include:

- **Data Quality and Scope:** The medical knowledge base must be reliable, up-to-date, and appropriately chunked to avoid misinformation.

- **Model Accuracy and Safety:** Large language models can exhibit "hallucination" or produce erroneous conclusions; thus, retrieval of verifiable source chunks is critical.

- **Ethical and Regulatory Considerations:** Especially with managing personal health information and schedule details, patient data protection, informed permission, and adherence to healthcare standards remain first priorities.

- **User Experience Across Roles:** The interface has to be able to easily differentiate between student and patient use, give students advanced conceptual detail and reduce medical language for patients.

This thesis assesses the performance of the suggested chat-bot architecture in fulfilling these objectives by means of methodical experimentation involving real-time input from both patients and healthcare practitioners. The next chapters cover the fundamental technologies, design approaches, system assessment criteria, and future developments that together define an advanced, role-based AI solution bridging medical education and patient-centric healthcare.

1.2 Motivation

Modern healthcare deals with a constant scarcity of medical experts as well as rising patient expectations at once. On one side, patients may need quick attention and easy scheduling systems to lower missed diagnosis, improve therapy compliance, and ease anxiety. To remain current, medical students and early-careers doctors must negotiate the complexity of specialized knowledge-based, hands-on practice and always changing policies. Static textbooks and passive lectures are among the conventional teaching strategies that might not be sufficient for equipping students to manage challenging, real-time situations.

Concurrently, traditional patient-facing instruments as rule-based triage systems or simple symptom checks seldom offer enough background or tailored advice. They also lack direct connection with hospital resources for appointment scheduling. Patients could thus be unsure about the severity of their symptoms or spend significant time researching several, dubious web sites.

These challenges point toward a critical need for a comprehensive solution that offers:

- **On-demand access to credible information:** This translates for patients into effective triage and clear symptom interpretation. It means interactive learning tools for pupils that replicate reasonable, sophisticated searches.

- **Reduced administrative barriers:** Automating chores like appointment scheduling guarantees that patients get timely treatment and frees the time of health professionals.

- **Seamless retrieval of validated data:** Whether it's localised healthcare provider information or current medical guidelines, the system has to combine pertinent data and provide it in context.

- **Adaptability for different user roles:** Advanced question-answering for students is somewhat different from simple patient scheduling and symptom analysis. One chatbot has to be able to manage these several modes fluidly.

Medical education and patient-centric services may be united on a single platform by using sophisticated embedding-based retrieval systems, agentic artificial intelligence capable of autonomous interaction with scheduling APIs, and large language models (LLMs). This coherent strategy provides quick advantages:

1. **Enhanced Learning:** At every degree of complexity, students may ask questions of the system and get thorough, evidence-based answers and case studies, therefore bridging the gap between theoretical knowledge and practical experience.

2. **Better Patient Outcomes:** Clear symptom expression and early appointment scheduling serve to minimize treatment delays, therefore lowering the possibility of issues and so decreasing patient stress.

3. **Administrative Efficiency:** Automated booking solutions simplify staff effort and reduce the chance of scheduling errors or gaps, therefore helping healthcare organizations to reduce their over-heads.

Developing a dual-mode medical chatbot is ultimately driven by transcending the constraints of single-purpose teaching platforms or basic symptom checks. Rather, an integrated AI-driven framework is ready to provide customized user experiences, guarantee flawless data flow, and create a transforming learning environment for medical students as well as better healthcare accessible for patients.

1.3 Problem Statement and Challenges

Two issues characterize modern healthcare: patients need consistent advice when confronted with contradicting symptoms; medical students need basic access to updated, accurate material to increase their core knowledge and be ready for clinical practice. Often lacking the timeliness, complexity, and contextual clarity required for complicated medical conditions are current solutions such as basic symptom checklists or static web-based systems. Moreover, these stand-alone technologies hardly provide for quick scheduling or referral, therefore leaving patients wondering about their next course of action on their medical path.

Regarding medical students particularly, the challenge is in Five Short Notes Names. Negotiating vast numbers of research publications, medical textbooks, and clinical case reports may be difficult, especially when doing so in real time or when referring to other fields (pharmacy and pathology, for example). Conventional e-learning systems lack effective Q and A systems suitable to user questions and hardly connect academic information with practical, scenario-based learning.

Timeliness and Accuracy: The pace of development of medical technology is so rapid that even conventional resources find it impossible to survive. Students take the risk of selecting obsolete techniques without fast, modern resources.

With respect to the patients, the struggles are equally important: people can sometimes struggle with knowing whether their symptoms warrant routine care or a trip to an urgent treatment center, which can lead to anxiety and potentially a delay in treatment.

Obstacles blocking healthcare access: Usually taking multiple phone calls or online searches, scheduling a specialist or follow-up visit slows patients from getting appropriate treatment especially those confused about the type of expert to see.

Unverified or contradicting information: Patients might be driven to self-diagnose via dubious internet sources as standard symptom-checking websites could be insufficient or not tailored.

Thus, the main challenge is designing a single, synthetic intelligence-powered system that satisfies both sets of needs:

1. A comprehensive teaching tool for medical students responding in-depth, contextually sensitively to specific questions derived from properly chosen clinical and scholarly sources. It allays any concerns, rapidly links patients to pertinent medical experts, and offers initial direction based on a patient-facing symptom analysis and triage mechanism.
2. Agentic appointment management helps to reduce administrative load and fill the information gap between pragmatic healthcare measures by enabling the system to autonomously organize or postpone patient appointments.

These underlines the requirement of including Large Language Models (LLMs), embedding-based data retrieval, and user role-aware interactions. One such strategy is: Abbreviations 6 covers the academic demands needed for medical school as well as the pragmatic, immediate concerns patients have when presented with health issues. The next parts will address how this integrated chatbot idea may provide medical students and patients accurate, ethically acceptable, contextually relevant advice.

1.4 Research Questions

Modern healthcare faces two major challenges. First, patients often receive inconsistent advice when confronted with ambiguous symptoms, which can lead to anxiety and delays in seeking proper care. Second, medical students frequently struggle to access the most current and accurate information needed to build their knowledge, especially when they have to sift through vast amounts of literature and clinical reports. In addition, existing systems rarely integrate practical features such as automated appointment scheduling or provider recommendations, leaving both patients and healthcare professionals with fragmented support.

This thesis aims to address these challenges by exploring the development of an integrated medical assistant tool that serves both patients and medical students. In particular, the study is guided by the following questions:

1. **Supporting Medical Education:** How can a system be built to provide medical students with timely, thorough, context-rich responses from current clinical literature thereby strengthening their basis for practice?

2. **Improving Patient Guidance:** How can the tool clearly advise whether a routine consultation or urgent care is suitable and fairly evaluate patient-reported symptoms?

3. **Healthcare Provider Recommendations:** Based on a patient's particular symptoms and location, how may the system efficiently suggest surrounding healthcare providers?

4. **Autonomous Appointment Management:** How can the technology be turned on to automatically arrange and control patient visits, therefore lowering administrative load and closing the distance between diagnosis and treatment?

Together, these questions aim to develop a unified solution that not only enhances the educational experience for future healthcare professionals but also offers practical, immediate support for patients.

1.5 Thesis Structure

This thesis is organized into six main chapters, as outlined below:

1. **Introduction:** This chapter defines the issue statement, introduces the topic, clarifies the driving force of the effort, and lists the research questions thereby setting the scene. It also describes the difficulties contemporary healthcare encounters with regard to medical education and patient treatment.

2. **State of the Art:** Reviewed in this chapter are present literature and accepted methods pertinent to the project. Topics cover traditional approaches and their constraints as well as the usage of Large Language Models and embedding techniques in healthcare. The backdrop this study offers helps one to grasp the setting of the suggested remedy.

3. **Design and Methodology:** This chapter describes the general system architecture and the techniques applied to build the medical chatbot. It addresses preprocessing and data collecting; it integrates language models with embedding-based data retrieval; it clarifies the agentic aspect of autonomous appointment management.

4. **Implementation:** The technological features of the project are then under discussion. The chapter details the tools, programming models, and procedures followed in system development. It also clarifies the way several modules were carried out and combined.

5. **Evaluation:** This chapter offers qualitative as well as numerical evaluations of the system. To show the success of the suggested strategy, it comprises performance criteria, user testing comments, and analogies with current solutions.

6. **Conclusion and Future Work:** The last chapter reviews the main conclusions of the studies, emphasizes the thesis's contributions, and addresses potential areas for next development.

Additional sections, such as the Appendices and the Bibliography, provide supplementary material, including source code, detailed experimental results, and a comprehensive list of references.

Chapter 2. State-of-the-Art

2.1 Introduction

Modern artificial intelligence, enormous data availability, and changing clinical demands taken together have revolutionized healthcare and medical education. Large language models (LLMs) and advanced embedding methods have become transforming technologies allowing dynamic patient involvement and inter-active educational support in recent years. These technologies today support conversational systems able to provide contextually complicated replies and manage challenging, unstructured clinical data. Covering historical advances, technical breakthroughs, distributed data structures, and assessment techniques, this chapter offers an all-inclusive survey of the state-of- the-art methods in the field. We build our dual-mode architecture meant to serve medical students and patients by combining ideas from two key studies—Paper 1 and Paper 2.

We arrange our review into various areas. Section 2.2 reviews the literature and traces the evolution of LLMs in healthcare; Section 2.3 explores the methodologies, including retrieval-augmentated generation, advanced embedding, dimensionality reduction, and clustering techniques; Section 2.4 addresses distributed architectures and privacy-enhancing technologies; Section 2.5 outlines evaluation strategies and benchmarks; and Section 2.6 discusses the system architectures that support these functionalities. Section 2.7, which looks at the difficulties and next avenues of study, closes us.

2.2 Literature Review

2.2.1 Evolving Role of LLMs in Healthcare and Medical Education

Early healthcare information systems used to be built on rigid databases with limited adaptation and rule-based expert systems. Data driven techniques took front stage when statistical language models and then deep learning approaches emerged. By allowing models like BERT and GPT to detect long-range relationships and contextual subtleties, transformer architectures introduced in 2017 transformed natural

language processing (NLP). This development opened the path for models capable of producing very fluid and context-sensitive answers.

Recent innovations best shown by GPT-4 and open-source models such as Meta's LLaMa series have opened fresh opportunities for clinical decision assistance and medical education. Paper 2 shows how remarkably fluidly LLMs may now create teaching materials and patient-tailored discourse. To guarantee factual correctness, both publications do point out that such systems need thorough domain-specific adaption and strong retrieval methods.

2.2.2 Decentralized Architectures for Data Privacy

The major privacy issues present in conventional, centralized healthcare systems are discussed in this part together with how distributed solutions might help. Data breaches and illegal access expose centralized systems, therefore raising ethical and legal questions. By keeping their own sensitive medical data in a Personal Data Store (PDS), distributed architectures let every patient to keep control over it.

Processing and filtering data locally often via de-identification and edge-processing mechanisms only non-sensitive, anonymized information is communicated externally (for example, with huge language models used in chatbot systems). This approach guarantees compliance with strict rules including GDPR and HIPAA in addition to lowering the possibility of privacy invasions. Furthermore, these designs improve general data security by eliminating single points of failure and spreading data management, thereby building more user confidence.

2.2.3 Conversational AI for Patient Engagement and Education

Recent research has demonstrated the transformative potential of conversational AI systems in healthcare. Paper 2 outlines a multi-layered conversational framework where patient inputs are first processed by specialized modules—ranging from ad-hoc parsers for vital signs to intent recognition systems (e.g., Wit.AI)—before being passed to an LLM for free-form dialogue. This tiered approach is designed to protect sensitive clinical data while enabling rich, empathetic interactions.

For medical students, similar systems provide interactive tutoring that offers in-depth, evidence-based explanations and simulates realistic clinical scenarios. Tailoring

responses based on user roles significantly enhances the system's utility, meeting the divergent needs of patients and students.

Both Paper 1 and Paper 2 highlight the use of retrieval-augmented generation (RAG) as a means to ground the generative process in verifiable sources. By indexing trusted medical texts in a vector database and retrieving contextually relevant passages, these approaches improve both accuracy and reliability in clinical applications.

2.3 Methodologies

The approaches followed in this part to create and assess AI based chatbot systems in the medical field. The method combines two complimentary points of view: one on data privacy and decentralization and the other on using large language models (LLMs) to improve patient involvement.

2.3.1 Decentralized Data Management for Enhanced Privacy

Building on current developments in distributed architectures, the suggested system uses a design whereby every patient has a Personal Data Store (PDS). This method guarantees that sensitive medical data is kept dispersed instead of in one, centralized repository, therefore reducing data breach and unauthorized access concerns. The essential actions consist in:

- **Data Ingestion and Local Processing:** Medical texts, vital sign measurements, and other clinical data are ingested and split into semantically consistent segments. Sensitive information is filtered and de-identified at the edge, ensuring that only non-sensitive, anonymized data is transmitted for further processing.

- **Personal Data Manager:** Each PDS is managed by a dedicated component that handles read/write operations and basic data aggregation (e.g., computing average values over specified time periods). This manager also enforces user control over access permissions, allowing patients to authorize healthcare professionals to view or modify their data.

- **Access Control Mechanisms:** Although not necessarily relying on blockchain in our implementation, the system uses a robust, smart-contract-based access control list to log and manage permissions transparently. This ensures compliance with stringent data protection regulations such as GDPR and HIPAA.

2.3.2 Leveraging Large Language Models for Patient Engagement

To improve patient engagement and support self-management, the system integrates advanced LLMs for natural language understanding and generation. The methodology here is structured around a multi-layered natural language processing pipeline:

- **Multi-tier NLP Pipeline:**
  - *Ad-hoc Interpretation:* Custom heuristics catered for the clinical setting help to first parse user inputs (e.g., vital sign data).
  - *Intent Recognition with Wit.AI:* Inputs not addressed by the ad-hoc parser are passed to Wit.AI for domain-specific intent identification, in which the system is trained on utterances connected to healthcare.
  - *Fallback via LLMs:* For searches that remain unsolved or call for a more conversational approach, a high-capacity LLM (such as GPT-4 or an open-source version) is called upon to provide sympathetic, context-aware re-responses.
- **Case Study Implementations:** Several case cases illustrating the LLM-based approach show:
  - Examining conversations on mental health to spot risk factors and language trends.
  - Creating customized chatbots for senior cognitive engagement.
  - Pairwise assessment frame-works help to summarize medical discussions.

  - Creating a patient interaction solution driven by artificial intelligence that connects with healthcare processes for automatic appointment scheduling

- **Evaluation and Ethical Considerations:** Using both quantitative measures such as victory rates from paired model comparisons and qualitative assessments by professional evaluators a strong evaluation methodology is applied. Throughout the development process, ethical issues like data privacy, bias, openness, and regulatory compliance are taken under discussion.

2.3.3 Integration and Synergy

Combining distributed data management with sophisticated conversational artificial intelligence helps the entire system design to fulfill two primary goals:

1. **Enhanced Privacy and Trust:** Patients have ownership of their own data, therefore guaranteeing compliant, safe, and open data handling.

2. **Improved Patient Engagement:** By use of LLMs, the system may provide dynamic, sympathetic replies, present individualized information, and offer timely advice depending on real-time patient contacts.

This combined approach not only solves the technical issues of privacy protection and safe data handling but also makes use of LLM transforming power to support improved therapeutic results and patient empowerment.

2.4 System Architecture

Our system design harmonizes strong data privacy with cutting-edge conversational artificial intelligence to enable doctor supervision and patient self-management. Inspired by approaches from distributed data management and digital health chatbots, the architecture is arranged into the following main elements:

2.4.1 Decentralized Data Management

The system uses a distributed method to protect patient privacy and follow laws including GDPR and HIPAA:

- **Personal Data Store (PDS):** Every patient receives a PDS, a safe haven for their private medical information. This architecture reduces the chances connected to centralized data leaks.

- **Personal Data Manager:** A dedicated component manages data ingestion, de-identification, and aggregation (e.g., computing average vital sign measurements over time). This manager ensures that only non-sensitive, anonymized data is transmitted for further processing.

- **Access Control Mechanisms:** Robust, policy-driven controls are implemented to allow only authorized healthcare professionals to access or update patient data, thus reinforcing user sovereignty over personal information.

2.4.2 Conversational AI Engine

The core of the system is its conversational AI engine, which leverages large language models (LLMs) to engage with patients effectively:

- **Multi-Tier Natural Language Processing:**
  - *Ad-hoc Interpretation:* Custom heuristics catered for the clinical setting help to first parse user inputs (e.g., vital sign data).

  - *Intent Recognition with Wit.AI:* Inputs not addressed by the ad-hoc parser are passed to Wit.AI for domain-specific intent identification, in which the system is trained on utterances connected to healthcare.

  - *Fallback via LLMs:* For searches that remain unsolved or call for a more conversational approach, a high-capacity LLM (such as GPT-4 or an open-source version) is called upon to provide sympathetic, context-aware re-responses.

- **Personalized Interaction:** The AI engine tailors its responses based on patient profiles, ensuring that conversations are both clinically relevant and emotionally supportive.

2.4.3 Integration with Clinical Workflows

The architecture is designed to bridge patient interactions with clinician oversight:

- **Patient Interface:** Easily entered data, health monitoring, and real-time chatbot engagement are made possible by the patient interface accessible via mobile applications, SMS, or web platforms.

- **Clinician Dashboard:** To enable quick clinical interventions, healthcare providers use a web-based dashboard, summary reporting, and real-time patient data analysis.

2.4.4 Data Flow and Security

First data is gathered via patient inputs and handled locally within the PDS. After that, securely sent non-sensitive, de-identified data goes to the conversational artificial intelligence engine to generate responses. Strong encryption and access control policies are used all throughout the system to guarantee data integrity, confidentiality, and regulatory standard compliance.

Using the benefits of distributed data management and cutting-edge conversational artificial intelligence, this integrated architecture creates a safe, patient-centric platform that increases clinical decision support, engagement, and finally helps to provide improved health outcomes.

2.5 Challenges

Using a safe and efficient AI-driven healthcare chatbot system presents various technological, ethical, and operational difficulties across several spheres. These difficulties result from the distribution of sophisticated large language models (LLMs) as well as from the distributed data management architecture:

- **Data Privacy and Security:** First and most importantly is safeguarding private patient information. To stop unwanted access and breaches, distributed architectures call for strong encryption, efficient data de-identification, and exact access limits. Following laws like HIPAA and GDPR adds even another level of complication.

- **Scalability and Performance:** Maintaining low latency for real-time interactions is difficult when the system expands to serve a rising user population and increased data volume. Sustaining performance depends critically on effective processing, storage, and retrieval of high dimensional embeddings.

- **Integration Complexity:** Combining conversational artificial intelligence engine with distributed data management calls for careful system architecture. Perfecting data flow among Personal Data Stores (PDS), natural language processing modules, patient interfaces, and clinician dashboards calls for strong APIs and exact synchronizing.

- **Reliability and Accuracy of LLMs:** Although LLMs like as GPT-4 have strong natural language processing and generating ability, their outputs have to be therapeutically appropriate and contextually precise. Especially in high-stakes healthcare environments, it is imperative to address problems including factual errors, possible biases, and erratic responses.

- **Regulatory Compliance and Ethical Considerations:** The system has to negotiate moral questions and complicated legal systems. This covers guaranteeing data sovereignty, getting informed permission, keeping openness in AI-driven judgments, and assigning unambiguous responsibility for automated activities.

- **User Trust and Adoption:** Patients and healthcare professionals alike have to see the system as dependable, safe, and user-friendly if adoption is successful. Establishing this confidence not only solves technological problems but also efficiently presents the data protection policies and artificial intelligence decision-making procedures of the system.

2.6 Evaluation Strategies

2.6.1 Quantitative Metrics

A comprehensive evaluation employs multiple quantitative metrics:

- **Semantic Similarity Scores:** Cosine similarity measures the closeness between generated responses and gold-standard references.

- **Retrieval Metrics:** Metrics like Precision@k and Mean Reciprocal Rank (MRR) assess the effectiveness of the retrieval module.

- **Clustering Validity Indices:** Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index evaluate the coherence and separation of clusters.

- **Pairwise Comparison:** Evaluate user satisfaction and system performance over time to appraise the long-term effects.

2.6.2 Human-Centric Evaluations

Human evaluations are important, in addition to numerical metrics:

- **User Studies and Surveys:** Ask medical students, patients, and doctors about clarity, empathy, and clinical relevance.

- **Expert Panels:** Have medical experts verify the accuracy of the chatbot outputs and usefulness

- **Task-Based Evaluations:** Design informative clinical scenarios and training simulations for performance evaluation

- **Longitudinal Studies:** Monitor user satisfaction and system performance over time to evaluate long-term impact.

2.6.3 Comparative Benchmarking

Comparative studies are essential to contextualize performance:

- **Traditional vs. Modern Approaches:** Benchmark the LLM-based approach against conventional methods (e.g., LDA, NMF).

- **Cross-Model Evaluations:** Compare outputs from different LLMs (e.g., GPT-3.5, Llama2-70B, Mistral-7B) under identical conditions.

2.7 Future Directions

Building on the current system architecture and methodologies, several avenues for future work can further enhance both the technical performance and clinical utility of AI-driven healthcare chatbots:

- **Multimodal Data Integration:** Future systems might combine electronic health records (EHRs), sensor data from wearable devices, and medical imaging to provide a more complete picture of patient health. Using real-time physiological data would provide more accurate and tailored responses.

- **Enhanced Model Fine-Tuning:** LLMs must be kept constantly fine-tuned on specific medical corpora and clinical conversations. Particularly for important decision-making in healthcare, this involves domain-specific training to increase the factual correctness, contextual relevance, and empathy of created replies.

- **Improved Data Privacy Techniques:** Data privacy is still a major issue, so more study on advanced de-identification techniques, federated learning, and safe multi-party computing can assist to guarantee that private patient data stays encrypted without endangering system performance.

- **Scalability and Real-Time Performance:** Future research should concentrate on maximizing scalability of system components. Examined are methods including distributed computing, pruning, and model compression to keep real-time responsiveness as the user base and data volume increase.

- **User-Centric Design and Usability Studies:** It is important to have constant comments from patients as well as from doctors. Larger and more varied user groups might be part of future research to hone user interfaces, increase conversational dynamics, and raise general system usability and accessibility.

- **Integration with Broader Healthcare Ecosystems:** Increasing the system's compatibility with current clinical management systems and healthcare IoT devices would help to provide a more flawless integration into daily clinical procedures.

This covers standardizing communication channels and data formats for more general use.

- **Ethical and Regulatory Frameworks:** Further research is needed to create and improve ethical rules and regulatory requirements as artificial intelligence-driven products find increasing presence in clinical practice. Establishing clear assessment criteria and responsibility systems should be the main priorities of further studies to guarantee responsible application.

2.8 Summary

The state-of- the-art in LLM driven systems for healthcare and medical education is fully reviewed in this chapter. From early transformer models to contemporary systems providing improved fluency and contextual awareness, we tracked the development of LLMs. Along with an in-depth study of distributed architectures and privacy-enhancing technologies as described in Paper 1, further talks on retrieval-augmented generation, sophisticated embedding techniques, dimensionality reduction, and resilient clustering

Our suggested dual-mode system is built on the multi-layered system architecture integrating data intake, embedding generation, retrieval, and LLM-driven response generating. Our thesis tries to close the distance between clinical practice and medical education by tackling important issues such factual correctness, data privacy, computational efficiency, and assessment complexity.

Future avenues of study include improving model dependability, merging multimodal data sources, enlarging distributed architectures, and standardizing assessment techniques. Development of next generation AI-driven medical chatbots that enhance patient involvement and educational results depends on these initiatives.

Chapter 3. Design and Methodology

This chapter includes a comprehensive review of the design and methods used adopted for developing the AI-driven medical chatbot. The system is tailored to cater to two distinct cohorts: patients and students of medicine, approached from an educational standpoint and pragmatic support including symptom evaluation and independent management of appointments. The complete system flow and design are shown in Figure 3.1.

3.1 High-Level Architecture

The system is structured into several interconnected modules, as shown in Figure 3.1.

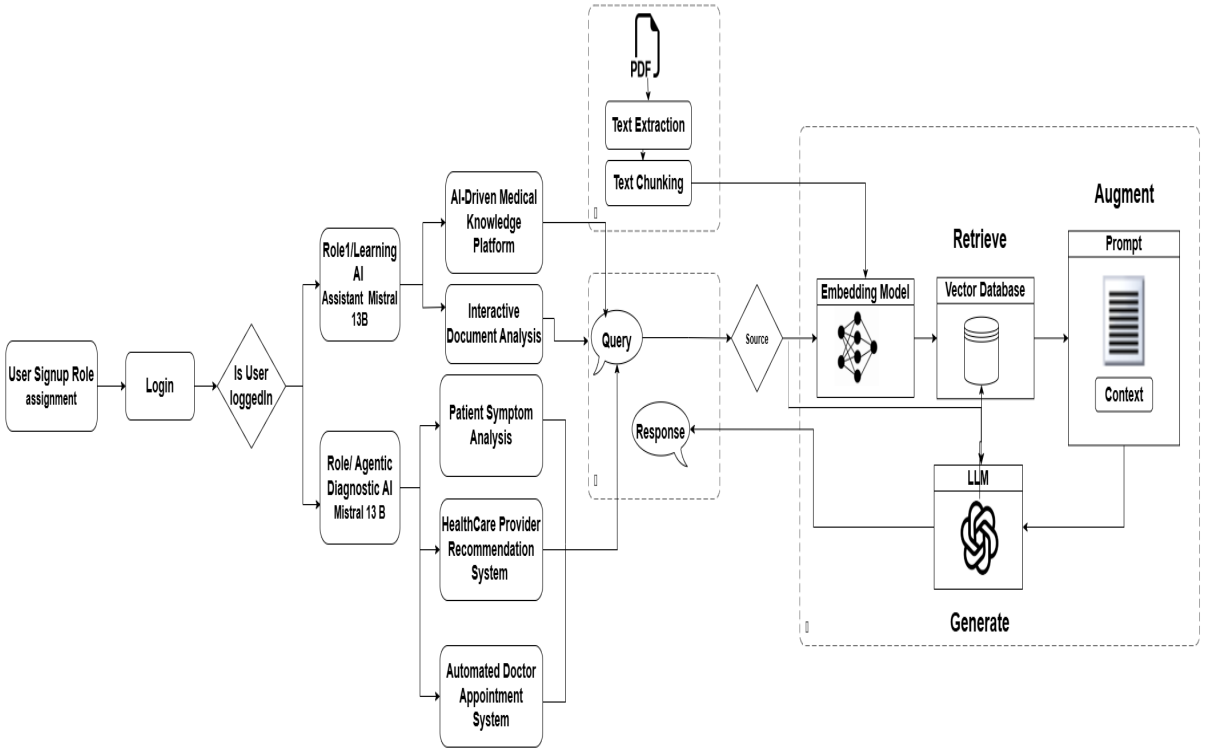

FIGURE 3.1: Thesis implementation phases

1. **User Authentication and Role Identification:**
    - Upon user login, the system authenticates and decides the individual's role as either medical student or patient.
    - Role-based access controls the assignment of users to a particular dashboard and capabilities.

2. **Role-Based Interaction:**
    - *Medical Student Mode:* Medical Student Mode: Provides exhaustive study aids such as PDF files with scholarly question-and-answer sessions.
    - *Patient Mode:* Gives a symptom analysis with suggestions for location based doctor recommendation with agentic appointment system.

3. **PDF Processing and Text Chunking:**
    - Extracts text from uploaded PDFs and segments them into manageable chunks for embedding and retrieval.

4. **Embedding and Vector Storage:**
    - Uses `all-mpnet-base-v2` for semantic embeddings.
    - Stores embeddings in a vector database (*Pinecone*) for efficient similarity-based retrieval.

5. **Retrieval-Augmented Generation (RAG):**
    - User queries are embedded and matched against stored vectors to retrieve relevant chunks.
    - The retrieved context is then used to generate accurate responses using the **Mistral 13B Quantized Model**.

6. **LLM Response Generation:**
    - The Mistral 13B Quantized Model generates context-aware, role-specific answers.
    - Responses are tailored differently for educational depth (students) and practical advice (patients).

7. **Agentic Appointment Booking:**
    - For patients, the system autonomously schedules appointments by integrating healthcare provider data.

3.2 Mistral 13B Quantized Model

The **Mistral 13B Quantized Model** serves as the core LLM for generating responses. It is chosen for its balance between performance and computational efficiency.

3.2.1 Overview and Architecture

The Mistral 13B Quantized Model is a variant of the Mistral architecture known for:

- **Parameter Efficiency:** With 13 billion parameters, it achieves high performance while maintaining manageable resource requirements due to quantization.

- **Quantization Technique:** Utilizes Q3_K_M quantization, reducing memory footprint and speeding up inference without significantly sacrificing accuracy.

- **Transformer Architecture:** Built on a decoder-only architecture with multi-head self-attention and feed-forward layers, optimized for generative tasks.

3.2.2 Contextual Response Generation

- The model generates responses by leveraging the context retrieved through the *RAG* pipeline.
- Role-specific prompts guide the model in generating:
    - *Educational Explanations* for medical students with detailed references.
    - *Simple, Actionable Advice* for patients, ensuring clarity and minimizing jargon.

3.2.3 Role-Specific Prompting

Different prompts are crafted based on user roles:

- **For Medical Students:**
  - Requests comprehensive explanations with references to academic sources.
  - Example: "Explain the mechanism of action of beta-blockers with clinical implications."

- **For Patients:**
  - Prompts the model to generate concise, easy-to-understand guidance.
  - Example: "What should I do if I experience chest pain?"

3.2.4 Additional Explanation of the Mistral 13B Quantized Model

The Mistral 13B Quantized Model has been specifically chosen for this system due to its excellent balance between computational efficiency and high-quality response generation. The quantization process significantly reduces the model's memory footprint and speeds up inference times, which is critical for delivering real-time responses in a production environment. Despite its compact size, the model maintains robust performance across diverse tasks, making it well-suited for both the detailed educational requirements of medical students and the clear, actionable advice needed by patients.

3.3 Embedding and Vector Database

3.3.1 Embedding Model: `all-mpnet-base-v2`

`all-mpnet-base-v2` is employed for converting text chunks and queries into 768-dimensional embeddings. Key features include:

- **Contextual Semantic Representation:** Captures contextual meaning, enhancing query relevance.

- **Performance:** Consistently outperforms older models (like BERT) in semantic search tasks.

- **Compatibility:** Embeddings are stored in *Pinecone* for rapid cosine similarity-based searches.

3.3.2 Vector Database: Pinecone

Pinecone is chosen as the vector database for its robust performance in managing high-dimensional embeddings and its seamless integration within the retrieval pipeline. Key features include:

- **Scalability:** Pinecone is designed to efficiently index and manage millions of embeddings, ensuring low-latency retrieval even as the data volume grows. This scalability makes it ideal for applications with rapidly expanding datasets.

- **Metric Support:** By utilizing cosine similarity as the primary metric for relevance ranking, Pinecone delivers highly accurate and meaningful search results, which are critical for ensuring that the most contextually relevant information is retrieved.

- **High Performance:** The database is optimized for speed, enabling real-time searches and updates. This is essential for applications requiring immediate responses, such as the medical chatbot system.

- **Integration with Retrieval Pipelines:** Pinecone seamlessly integrates with frameworks like LangChain, simplifying the process of embedding storage and retrieval. This tight integration helps maintain a smooth workflow from query embedding to final response generation.

- **Reliability and Consistency:** Pinecone offers a reliable infrastructure that ensures data consistency and durability, which is critical for systems handling sensitive and large-scale information.

- **Ease of Use:** With a user-friendly API and robust documentation, Pinecone enables developers to implement and manage the vector database with minimal overhead, facilitating faster development cycles.

3.4 Agentic Appointment Booking

3.4.1 Triggering Conditions and Workflow

The system autonomously handles appointment booking by:

1. Detecting user intent through keywords (e.g., "book an appointment").
2. Collecting details like location and specialty.
3. Querying a dynamic dataset of 7,000+ healthcare providers scraped from Bing Maps.
4. Generating a booking link via Doctolib and sending it through Twilio SMS integration.

3.5 User Interface Approaches

3.5.1 Flask UI

Flask is used for:

- **User Authentication:** Role-based access control for students and patients.
- **File Upload:** Allowing PDF uploads for medical students.
- **Query Form:** Basic input form for submitting questions.

The Flask UI is designed to handle user authentication, registration, and file uploads seamlessly. It also integrates with a separate Chainlit process that provides a chat-centric interface. Upon successful login, Flask starts Chainlit in a new thread, passing the user's email via environment variables so that Chainlit can fetch role-specific details from the database. This integration ensures that users are quickly redirected to a dynamic and responsive chatbot interface after authentication.

3.5.2 Chainlit UI

Chainlit provides a chat-centric interface designed to enhance user engagement through interactive and dynamic conversation flows. Key features include:

- **Role-Specific Chat Profiles:** The system supports different conversation modes tailored for medical students and patients. Each profile presents customized

prompts and responses, ensuring that the dialogue remains relevant to the user's context—whether that involves detailed academic discussions or clear, actionable health advice.

- **Interactive Conversation Elements:** Chainlit UI incorporates features such as real-time feedback, adaptive conversation threads, and dynamic content loading. This enables users to interact fluidly with the chatbot, with the interface automatically adjusting based on user input and query context.

- **Enhanced User Engagement:** By integrating multimedia elements and interactive buttons, Chainlit helps to create a more engaging experience. For example, users can click on suggested topics or follow-up questions, which makes navigating complex information more intuitive.

- **Seamless Integration with Back-End Systems:** The UI is designed to work in tandem with the underlying retrieval and generation pipelines. It receives context-aware responses from the **Mistral 13B Quantized Model** and displays them in an easily digestible format, while also allowing users to refine or extend their queries interactively.

- **Customization and Scalability:** The modular design of Chainlit allows for easy customization of chat elements and conversation flows. This flexibility supports ongoing improvements and scalability as new features are integrated or as user needs evolve.

3.6 Summary

In summary, this chapter outlines a robust design for an AI-driven medical chatbot that effectively serves two distinct user groups: medical students and patients. The system's modular architecture incorporates user authentication, PDF processing, semantic embedding using `all-mpnet-base-v2`, and a vector database via Pinecone for rapid retrieval. A retrieval-augmented generation pipeline seamlessly integrates the powerful **Mistral 13B Quantized Model** to generate context-aware responses. Furthermore, the solution provides specialized functionalities, such as role-based interaction and autonomous appointment booking, ensuring that educational and practical needs are both addressed efficiently. The integration between the Flask UI and the Chainlit chat interface is a key component, enabling dynamic, real-time engagement that leverages user-specific data to tailor responses and services. The next chapter will delve into the implementation details, system deployment, and further integration strategies.

Chapter 4. Implementation

In this chapter, a comprehensive overview of the libraries utilized throughout and the functions applied for achieving topic modelling, specifying their roles in different sections of the code is given. Through deeper understanding of the inner workings od modelling and explaining the implementation of the algorithms, the thesis is concluded.

For implementation of the thesis, I have used platform like VS Code(Visual Studio Code) and Jupyter Notebook(Jupyter Notebook). VS code is integrated developement environment and Jupyter Notebook is a web based IDE, where both this IDE will support various languages and frameworks for the model implementation and developement.

4.1 Libraries

In this section, we will focus on certain aspects of libraries and their uses in the model implementation process.

Pandas:

Pandas is a library that is commonly used for data analysis and manipulation. It provides different functions and methods that are compatible with structured data such as data frames, CSV files, Excel sheets, etc. This library is very helpful in projects related to data science, machine learning and data analytics because it facilitates tasks such as: data cleaning; pre-processing; manipulation; visualization among others. Python Documentation [2023].

NumPy:

NumPy is a fundamental package for scientific computing in the python which provides a multidimensional array object, derived object (like masked array and matrices) and an assortment of routines for fast operations on arrays which includes mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic algebra, statistical operations, and much more. NumPy have been used in few places in this thesis, like converting the embedded data into an array so that it

can be used for dimensional reduction and to find the total number of clusters formed Python Documentation [2023].

Pickle:

Pickle is one of the most used libraries in python for data serialization and de-serialization as it converts objects into bytes streams for storage and restore them into original form. As pickle can handles objects of python includes list, dictionaries and so on, using different protocols optimized for efficiency. Common use cases include saving objects states, caching, and inter-process communication. The pickle library is used to store the embedded data formed as the data set is huge and running embedding every time takes lot of time Python Documentation [2023].

Json:

The python json module will offers a means of encoding and decoding data in the lightweight, text-based JSON (Javascript Object Notation) format which is frequently used for data transmission. The module allows you to serialize python objects, such as dictionaries, lists into json strings and deserialize json strings into python objects. Json is a human readable and language independent which can be used for web APIs and configuration files. Json format is used for storing the clustered data as this format is human readable and can be used for APIs as the clustered data will be feeded into the large language model Python Documentation [2023].

umap-learn:

umap-learn is an open-source python package will helps to implement the Uniform Manifold Approximation and Projection algorithm for dimensionality reduction. It does this by first learning a high-dimensional graph representation of the data, and thereafter, optimizes a low-dimensional layout which captures the local structure. The umap-learn package offers an easy-to-use interface, in addition to native interfaces to other well-known data science libraries like Pandas, Scikit-learn, and NumPy. It can also deal with supervised and unsupervised learning, which turns out to be versatile for many applications. Of these, a very notable one is dimensionality reduction of feature spaces, hence improving efficiency to downstream tasks like data visualization, classification or clustering Python Documentation [2023].

hdbscan:

hdbscan is a Python package that implements the HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) algorithm. This package is often used in tasks which involves complex data like biology, image processing, grouping the text etc., and this can be easily integrated with the other libraries like Pandas and scikit-learn. This algorithm is used for discovering the patterns in large dataset where number of clusters is unknown Python Documentation [2023].

transformers:

The transformers package is by Hugging Face is a powerful library that allows to access the wide range if pretrained transformers models for many NLP tasks. It supports both PyTorch and TensorFlow and it is easy to use where APIs allows the user to load the models. tokenize data and fine tune them. The package also includes a convenient pipeline API for applying models to various tasks with minimal code. With extensive documentation and a large community, transformers has become a go-to tool for state-of-the-art NLP Python Documentation [2023].

torch:

Troch is a deep learning library that is used for machine learning, and it is core library of PyTorch, a popular open source framework developed by Facebook's AI research lab. PyTorch is widely used for building and training deep learning models due to its speed, flexibility, and ease of use. This package provides a range of functionalities, including tensor operations (Similar to Numpy arrays with GPU acceleration), automatic differentiation, and building neural networks. It also supports GPU acceleration using CUDA, making it suitable for high-performance computing Python Documentation [2023].

Flask:

Flask, being an adaptable and simple web framework helps to create web apps with python. the server-side processing is handled via this tool while it also influences the web request management thanks to its essential utility for forming a web server and directing the requests between the front-end and back-end section of the application Python Documentation [2023].

HTML/CSS:

The application's front-end was developed with HTML for structure and CSS for styling. This combination ensures the user interface is functional and attractive. Hence, it is able to carry out its task without much ado and in a commendable way as well. User input is collected using HTML forms while CSS serves the purpose of improving on webpage aesthetics.

4.2 Code Implementation

This section will explain how the code is implemented in the thesis to achieve topic modelling. This section consist of subsection as follows:

1. Preprocessing
2. Embedding
3. Dimensionality reduction
4. Clustering
5. Large Language Model

4.2.1 Preprocessing

Once the initial data analysis has been done, the data will undergoes preprocessing steps as mentioned below,

LISTING 4.1: Initialization of the list of noise

```
# List of common filler words
filler_words = {"ja", "hm", "mhm", "ach", "gut", "und", "eben", "ne", "ok", "aha", "ach so", "
    nicht", "ja", "nein", "wieder", "schon", "naja", "wieso", "wieso nicht", "wieso nicht"}
```

As there the data is the transcript of the video Interviews, it consists of lot noise in the data. There are lot of shot sentence, noise, repeated small sentences etc., hence, it is necessary to remove them. The below code has list of noise which can be removed from the data. and this is the initialization of the list.

LISTING 4.2: Filtering of noise from the data

```
  def has_repeated_filler_patterns(sentence, filler_words):
    # Check if any filler word appears repeatedly either consecutively or separated by commas
    or spaces in the sentence.
      pattern = r'\b(' + '|'.join(re.escape(word) for word in filler_words) + r')\b(?:[\s,]+)
    +\1\b'
```

```
4
5       if re.search(pattern, sentence):
6           return True
7       return False
8
```

The function `has_repeated_filler_patterns` is useful when talking about text because it helps by filtering out certain sentences from it while breaking the text down into different sentences through regular expressions that look for punctuation marks showing end of sentence. This function checks a variety of factors including minimum word count as well as unique word count. If all these conditions are met, then that particular sentence will go into a filtered sentence list. In addition this could also call has repeated filler patterns to eliminate noise.

LISTING 4.3: Filter sentences from the text

```
1  def filter_short_and_filler_sentences(text, filler_words, min_words, min_unique_words,
       min_characters):
2      # Filter sentences from the text based on length, unique words, minimum characters, and
       absence of repeated filler patterns.
3      sentences = text.split('. ')
4      filtered_sentences = []
5
6      for sentence in sentences:
7          words = sentence.split()
8
9          if len(words) >= min_words and len(set(words)) >= min_unique_words and len(sentence)
       >= min_characters:
10             if not has_repeated_filler_patterns(sentence, filler_words):
11                 filtered_sentences.append(sentence)
12
13     return '. '.join(filtered_sentences)
```

The `filter_short_and_filler_sentences` function will process the text to filter out sentences and it splits the text into individual sentences using the regular expression to check the various punctuation marks that signify the end of a sentence. This function check several conditions like minimum number of words number of unique words and exceed the length in characters once these conditions are satisfied, the sentence will be appended to the list of filtered sentences. Additionally, it will call `has_repeated_filler_patterns` function to identify and remove noise.

LISTING 4.4: Removing columns and combining the sentences to paragraphs

```

    excel_data = pd.read_excel(file_path, sheet_name=None)

    # Combining the sentences to paragraphs and applying the filtering
    combined_paragraphs = {}

    for sheet_name, df in excel_data.items():
        # Drop the 'Timecode' and 'Sprecher' columns
        df = df.drop(columns=['Timecode', 'Sprecher'])

        # Convert the 'Transkript' column to a single string
        df['Transkript'] = df['Transkript'].astype(str)
        paragraph = ' '.join(df['Transkript'].tolist())

        # Filter short sentences and apply filler pattern filtering
        filtered_paragraph = filter_short_and_filler_sentences(paragraph, filler_words,
    min_words, min_unique_words, min_characters)

        # Save the filtered paragraph by sheet name in the dictionary
        combined_paragraphs[sheet_name] = filtered_paragraph

    # The `combined_paragraphs` dictionary now contains the filtered paragraphs for each sheet
     in memory.
    # You can now use `combined_paragraphs` as needed in your code.

    print("Filtered paragraphs have been processed and stored in memory.")
```

The above code will help to read the excel file, remove the columns and combine the sentences to paragraphs, where it reads the data from each sheet in the excel file and removes the columns Timecode and Sprecher. It also combines the data into one single paragraph by adding the sheet name to the start of the paragraph. This code will call the `filter_short_and_filler_sentences` function to filter the sentences and adds the sheet name at the begin of the paragraph and save the combined data to a text file and pickle file for the future use.

4.2.2 Embedding

This the code will load the pre-trained Jina embeddings model and tokenizer the pre-processed text data. The model will be used convert the text data into embeddings.

LISTING 4.5: Load Jina AI embedding model and tokenizer

```

# Load the pre-trained Jina embeddings model and tokenizer
model_name = 'jinaai/jina-embeddings-v2-base-de'
model = AutoModel.from_pretrained(model_name, trust_remote_code=True)
```

```
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)

# Assigning the combined_paragraphs to the text variable
text = combined_paragraphs
# Split the text into sentences based on '. ' delimiter
sentences = text.split('. ')
# Define the batch size for processing
batch_size = 32 # can be adjusted based on the computational power
embeddings = []

# Process the sentences in batches
for i in range(0, len(sentences), batch_size):
    batch_sentences = sentences[i:i+batch_size]

    # Tokenize the batch of sentences
    inputs = tokenizer(batch_sentences, return_tensors='pt', padding=True, truncation=True,
    max_length=512)
    # Compute embeddings without tracking gradients
    with torch.no_grad():
        outputs = model(**inputs).last_hidden_state

        # Compute the mean embedding for each sentence in the batch
        batch_embeddings = outputs.mean(dim=1).cpu().numpy()
        # Append the batch embeddings to the list
        embeddings.extend(batch_embeddings)

# Convert the list of embeddings to a NumPy array
embeddings = np.array(embeddings)
```

Load Model and Tokenizer: It loads a pre-trained Jina embeddings model and its corresponding tokenizer. The `trust_remote_code=True` argument ensures that remote code is trusted when loading the model.

Split Text: The data is split into individual sentences using '. ' as the delimiter. As the text dataset has been combined into a paragraph, it is necessary to split the text into individual sentences. As the model can handle sequences length of the embeddings is 8192 and I have reduced to sequence length to 512 as in the future phase as the maximum sequence length need more computational power and memory.

Batch Processing: Sentences are processed in batches of size 32 to manage memory and computational efficiency. This will also reduces the runtime of the embedding process. The batch size can be alter based on the computational power.

Tokenization and Embedding Computation: Each batch of sentences is tokenized using the model's tokenizer. The embeddings are computed without tracking gradients

for efficiency. For every sentence, a fixed size embedding is obtained by calculating the mean of the last concealed state across tokens.

Store Embeddings: The embeddings for each batch are collected and converted into a NumPy array for further use.

4.2.3 Dimensionality Reduction

As the embedding data will be having high dimensional space and it cannot be used for the visualization and clustering directly hence, the UMAP reduction technique is used which will helps to convert the high dimensional space into a low dimensional space.

LISTING 4.6: Dimensionality Reduction

```
import umap.umap_ as umap
# Create a UMAP model instance with specified hyperparameters
umap_model = umap.UMAP(n_neighbors=20, n_components=10, min_dist=0.1, metric='cosine',
    low_memory=True)

# Fit the UMAP model to the filtered embeddings and transform the data
umap_embeddings = umap_model.fit_transform(filtered_embeddings)
```

Initialize UMAP Model: The UMAP model is initialized with the following hyperparameters:

- `n_neighbors`: Number of neighboring points used in local approximations.
- `n_components`: The number of dimensions for the reduced embedding.
- `min_dist`: Minimum distance between points in the low-dimensional space.
- `metric`: Distance metric used to calculate distances between points.
- `low_memory`: Optimizes memory usage by storing fewer intermediate results.

These hyperparameters will help to balance capturing both local and global structure in the data while reducing the dimension of the data. Setting the `n_neighbors` will allows the model to consider neighborhood size preserving both local relationships and broder patterns. The `n_components` which will helps to retains the important information for downstream tasks with any data loss. The `min_dist` will relatively control the distance between groups and tightens the clusters making it easier for grouping the distinct groups. The use of `low_memory=True` will help to reduce the memory usage of the model whenever it is dealing with larger dataset.

Fit and Transform Data: The UMAP model is trained on the `filtered_embeddings` data to reduce its dimensionality from high-dimensional space to a 10-dimensional space. The `fit_transform()` method performs both fitting the model and transforming the data into the new lower-dimensional space. The transformed data is stored in `umap_embeddings`.

4.2.4 Clustering

HDBSCAN is used to cluster the embeddings obtained from UMAP. The clustering results are organized into a JSON file to facilitate easy access and analysis of the clusters and their associated sentences.

LISTING 4.7: HDBSCAN Clustering

```
1
2 # Create a HDBSCAN Clustering instance with specified hyperparameters
3 clusterer = hdbscan.HDBSCAN(min_cluster_size=40, cluster_selection_epsilon=0.4, metric='
      euclidean', cluster_selection_method='eom')
4 clusters = clusterer.fit_predict(umap_embeddings)
5
6 # Print the number of clusters
7 num_clusters = len(set(clusters)) - (1 if -1 in clusters else 0)
8 print(f"Number of clusters: {num_clusters}")
```

Initialize and Apply HDBSCAN: The HDBSCAN clusterer is configured with:

- `min_cluster_size`: Minimum size of clusters.
- `cluster_selection_epsilon`: The parameter to regulate the clusters granularity.
- `metric=`: The distance metric used for clustering.
- `cluster_selection_method`: Method for selecting clusters (excess of mass).

The `fit_predict()` method applies HDBSCAN to the `umap_embeddings` and returns cluster labels.

Count and Print Number of Clusters: The number of clusters is calculated by counting unique cluster labels and adjusting for the label `-1`, which represents noise. The result is printed.

LISTING 4.8: Count and Print Number of Clusters

```
# Create a list to hold clusters with their sentences
clusters_list = []

for cluster in np.unique(clusters):
    # Convert the cluster ID to a regular Python int
    cluster_id = int(cluster)
    # Collect sentences for each cluster
    sentences_in_cluster = [sentence for i, sentence in enumerate(sentences) if clusters[i] ==
     cluster]

    # Create a dictionary for the cluster
    cluster_dict = {
        "cluster_id": cluster_id,
        "sentences": sentences_in_cluster
        }
    # Add the cluster dictionary to the list
    clusters_list.append(cluster_dict)

# Save the clusters to a JSON file
output_json_file = 'clusters.json'
with open(output_json_file, 'w', encoding='utf-8') as f:
    json.dump(clusters_list, f, ensure_ascii=False, indent=4)

print(f"Clusters have been saved to '{output_json_file}'")
```

Organize and Store Clusters: The code creates a list of dictionaries where each dictionary represents a cluster. Each dictionary contains:

- `cluster_id`: The ID of the cluster.
- `sentences`: A list of sentences belonging to that cluster.

Save Clusters to JSON: The list of clusters is saved to a JSON file named `clusters.json`. The `json.dump()` function serializes the list and writes it to the file. A confirmation message is printed indicating successful saving.

4.2.5 Large Language Model

The cluster data which is stored in the JSON file is used to generate the topic names for each cluster. This process involves leveraging the model's ability to process and synthesize information from multiple sentences, producing a coherent and meaningful topic label for each cluster.

Loading the Model and Tokenizer: Accessing the pre-trained language model and tokenizer through Hugging Face model hub with a script that has a name Meta-Llama-version, it is based on an access token (this model is being used because of my compute resources constraints although other models may be possible too).

LISTING 4.9: Loading Llama 3 model

```
import json
import torch
from transformers import AutoModelForCausalLM, AutoTokenizer

# Debugging function to help track where issues might be occurring
def debug_message(message):
    print(f"[DEBUG] {message}")
# Load the model and tokenizer once at the beginning
try:
    debug_message("Loading model and tokenizer...")
    model = AutoModelForCausalLM.from_pretrained("meta-llama/Meta-Llama-3-8B", token="
    HuggingFace access token")
    tokenizer = AutoTokenizer.from_pretrained("meta-llama/Meta-Llama-3-8B", token="HuggingFace
     access token")
    debug_message("Model and tokenizer loaded successfully.")
except Exception as e:
    debug_message(f"Error loading model or tokenizer: {str(e)}")
    raise
```

Generating Topics: A function named `generate_topic` is used to generate a topic name for a set of sentences within a cluster. The sentences are combined into a corpus and fed to the language model using a prompt, which returns a concise and specific topic of up to 5 words. The topic generation process is wrapped in error handling to catch any issues that may arise.

LISTING 4.10: Generate Topic

```
def generate_topic(sentences):
    try:
        # Combine sentences into a single string
        combined_sentences = " ".join(sentences)
        # Create a prompt for the model
        prompt = f"Analysiere die folgenden Saetze und generiere einen praezisen,
    aussagekraeftigen Themennamen von maximal 5 Worten. Der Themenname sollte den Kerninhalt
    erfassen und spezifisch sein. Saetze: {combined_sentences} Thema:"
        # Tokenize input and generate the topic
        inputs = tokenizer(prompt, return_tensors="pt")
        outputs = model.generate(**inputs, max_new_tokens=10, num_return_sequences=1)
        # Decode the generated topic
        return tokenizer.decode(outputs[0], skip_special_tokens=True).strip()

    except Exception as e:
```

```
        debug_message(f"Error during topic generation: {str(e)}")
    return "Topic generation failed"
```

Loading Clusters: The clusters of sentences are loaded from a JSON file (`clusters.json`), which contains a list of dictionaries. Each dictionary represents a cluster and includes a `cluster_id` and the corresponding `sentences`.

LISTING 4.11: Load Clusters from JSON

```
# Load the clusters from the JSON file
input_json_file = 'clusters.json'
try:
    debug_message(f"Loading clusters from {input_json_file}...")
    with open(input_json_file, 'r', encoding='utf-8') as f:
        clusters_list = json.load(f)
    debug_message("Clusters loaded successfully.")
except Exception as e:
    debug_message(f"Error loading clusters: {str(e)}")
    raise

# List to hold the results with generated topics
clusters_with_topics = []
```

Processing Clusters Sequentially: The function `process_clusters_sequentially` iterates through each cluster, skipping any noise clusters (identified by `cluster_id` = -1). For each valid cluster, the function generates a topic, appends the topic to the cluster data, and adds the updated cluster to a list (`clusters_with_topics`). After processing each cluster, the script clears the CUDA memory (if using a GPU) to optimize memory usage.

LISTING 4.12: Process Clusters Sequentially

```
def process_clusters_sequentially():
    for cluster in clusters_list:
        try:
            # Extract cluster information
            cluster_id = cluster["cluster_id"]
            # Skip the cluster with ID -1 (noise)
            if cluster_id == -1:
                debug_message(f"Skipping noise cluster {cluster_id}")
                continue

            sentences = cluster["sentences"]
            # Debug: Show the cluster info
            debug_message(f"Processing cluster {cluster_id} with {len(sentences)} sentences.")

            # Generate a topic for the cluster
```

```
            topic = generate_topic(sentences)
            # Add the generated topic to the cluster data
            cluster["topic"] = topic
            # Append the updated cluster to the results list
            clusters_with_topics.append(cluster)
            # Clear memory after processing each cluster
            torch.cuda.empty_cache()  # If you're using a GPU, clear the cache
            debug_message(f"Processed cluster {cluster_id}, Topic: {topic}")
        except Exception as e:
            debug_message(f"An error occurred while processing cluster {cluster_id}: {str(e)}"
    )

# Process all clusters sequentially
try:
    process_clusters_sequentially()
except Exception as e:
    debug_message(f"Error during cluster processing: {str(e)}")
    raise
```

Saving Results: Once all clusters have been processed, the updated clusters with generated topics are saved to a new JSON file (`clusters_with_topics.json`) for further use.

LISTING 4.13: Save Clusters with Topics to JSON

```
# Save the clusters with topics to a new JSON file
output_json_file = 'clusters_with_topics.json'
try:
    debug_message(f"Saving clusters with topics to {output_json_file}...")
    with open(output_json_file, 'w', encoding='utf-8') as f:
        json.dump(clusters_with_topics, f, ensure_ascii=False, indent=4)
    debug_message("Clusters with topics saved successfully.")
except Exception as e:
    debug_message(f"Error saving clusters: {str(e)}")
```

Throughout this code, a debugging function (`debug_message`) is used to print out messages for tracking the progress and identifying potential issues.

4.3 Web Application for Smaller Text Corpus

In this section, the development of the a web application designed to process the smaller text data with the above implementation. The primary goal of this application is to provide a user-friendly interface for analysis of the smaller text corpus. This application will also the same workflow but, the data will be directly feed into the embedding model which is followed by clustering then to the LLM model (Llama).

4.3.1 Methodlogy and Technologies Used

To implement this web application the following Technologies are employed, I have used the function which i have used in the previous section along with the flask and HTML/CSS. Here the flask is used to run the web application where is manage the POST request and Get request. The HTML/CSS is used to design the user interface (UI). The data is processed in the same way as in the previous section. Over here instead of loading the files the user can directly paste the text data which we want to do topic modelling. Once the data is loaded then the user can change the minimum size of the cluster and click on process button to process the data. The data will be undergo the same process which is mwntion in thw previous section like embedding, clustering and then to language model to generating the topic.

Below are few snapshots of the web application,

This is the UI of the web application where the user can change the minimum size of the cluster and process the data.

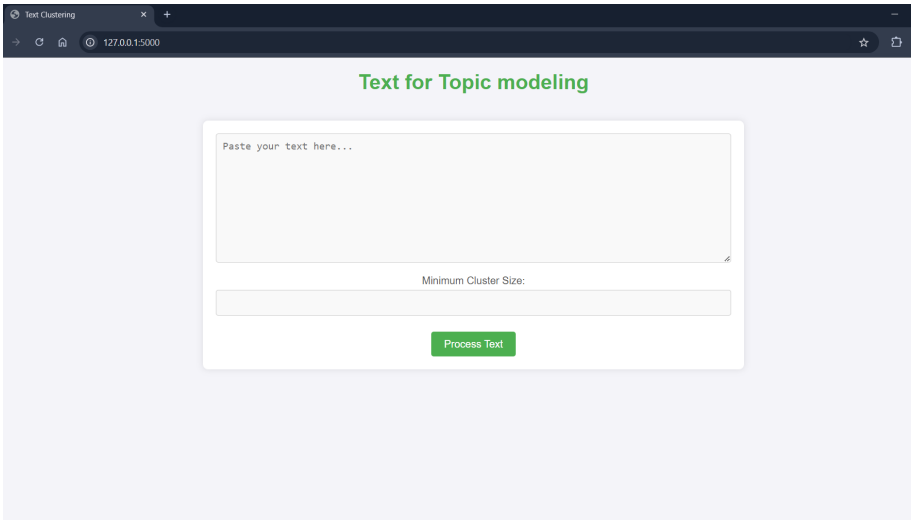

FIGURE 4.1: UI for the web application

The output of the web application looks as attached below,

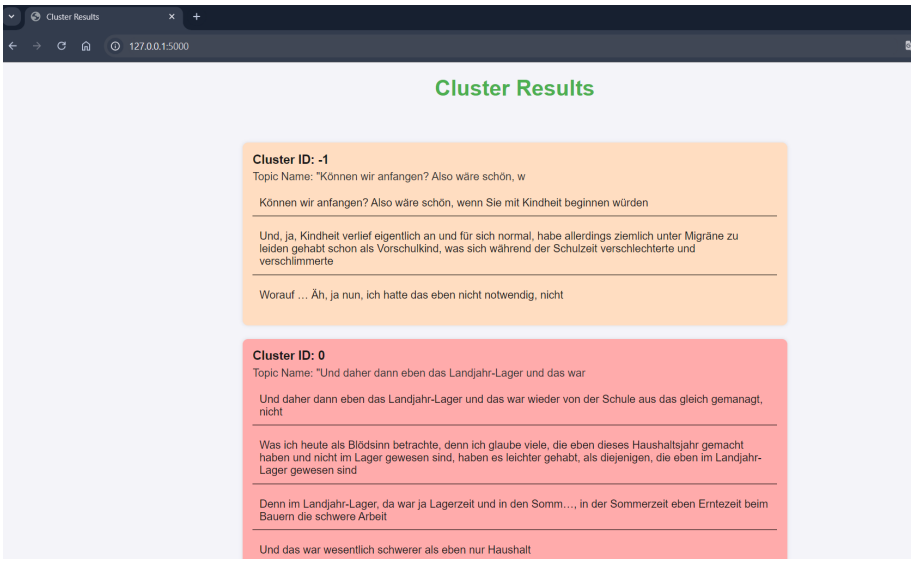

FIGURE 4.2: UI for the web application

This web application is used to process the smaller text data and generate the topic in the same way as the previous section. This will help the user to process the data faster and can be used for both German and English language.

4.4 Summary

This chapter gives the detailed explaination about the various libraries used for the implementation of the topic modelling and along with the web application for smaller text corpus. The process of achieve the topic modelling on the historical biographical interview data using the embedding, clustering and large language model is explained in the previous section. The web application is used to process the smaller text data and generate the topic in the same way as the previous section. With the help of this implementation approach for achieving topic modelling describe how LLMs can be used for topic modelling when the dealing with the larger text documents. With this chapter implementation of the thesis is completed and next chapter will be about checking the performance of the models implemented.

Chapter 5. Evaluation

In this chapter, the evaluation of the model is discussed with various techniques used to analyze how good the clustering has been done and generation of the topics with other LLM models along with the human evalaution.

5.1 Evaluation Design

There are various techniques that can be used to evaluating the model performance and one of the most effective way is through human evaluation. Along with this I have tried to evaluate the model performance by evaluating the clusters formed by the model amd comparing the topic generated by the LLM model with other open-source LLM models. For the human evaluation following of the questions are asked to

1. **How well do you think notebooks effectively explain concepts or content to you?**
   This question was asked to check if the information provided to explain the Python codes for implemented models in the VS code Jupyter notebooks were easy to understand.
2. **How would you rate the cluster formation of the model?**
   The purpose of this question is to determine whether the sentences inside the clusters that the model created were similar to one another or not.
3. **How would you rate the topic generation of the model?**
   This question was asked in order to know if the topics generated by the model were good or not for the cluster sentences so that, the model was able to understand the content of the cluster and generate the proper topic name for the cluster.
4. **How would you rate the LLM model approach when compared to the traditional approach?**
   This question will help us to understand if the LLM model approach is good or not compared to the traditional approach.

5.2 Cluster Evaluation

Cluster evaluation is used to evaluate the quality of the clustering done by the model. There are various techniques that can be used to evaluate and I have considered the below technique. This cluster evaluation is done using the Silhouette Score, Davies-Bouldin Index and Calinski-Harabasz Index which will suitable for evaluating the clusters formed by the clustering algorithm. This will help to understand the quality of the clusters and embedding models. As the data is in the German language, finding the embedding model is very important and the selected embedding model need to cover the entire data so that, the clustering can be done properly. As if chosing a inefficient embedding model, will result in inproper cluster formation and which leds to the poor results. So, by conducting theis evaluation, we can understand the quality of the embedding model along with the cluster formation. The below metrics will give an idea how the evalaution score is calculated is explained below,

1. **Silhouette Score:** Measure of how closely an object is to one's cluster relative to other clusters. The score has a range from -1 to 1, with higher score indicating better clusters. These also give insights into the cluster cohesion and separation.
2. **Davies-Bouldin Index:** ThThis indexes the average similarity between each cluster and its most similar one. Lower Davies-Bouldin values determine a better partition. The compactness of the separation of the clusters will be measured.
3. **Calinski-Harabasz Index:** It measures the ratio of between-cluster scatter to within-cluster scatter. Bigger values are indicative of a better-defined cluster. Index works best with clusters that are convex in shape GeeksforGeeks [2024].

The metrics for each embedding model are shown below,

Embedding Model	Silhouette Score	Davies-Bouldin Index	Calinski-Harabasz Index
Jina AI de	(0.3 - 0.6)	0.95	3459
Cross RoBERTa en-de	(0.1 - 0.4)	1.1	456
BERTopic	(-0.3 - 0.1)	1.5	236

TABLE 5.1: Cluster Evaluation Metrics for HDBSCAN

Based on the above metrics, the Jina AI de model performed well with the hdbscan clustering for the german interview transcript dataset. This metric made me to choose the Jina AI de model for embedding of the interview transcripts.

5.3 Topic Generation with other LLMs

Evaluation includes my using the same LLM model for topic generation by feeding the clusters to the LLM model and prompting the model for generating topics based on the cluster. Since I am using the Llama 3 - 8B model for topic generation, I want to check with other LLM models. The topics generated for each cluster were manually evaluated for whether they are similar or not. Random proverb clusters chosen to prompt models like the GPT 3.5 model by Open AI ChatGPT, Google Gemini model, and a few other LLM models, have been presented here.

The evaluation was performed manually on the generated topics of each cluster to verify if it is similarly trained as the state-of-the-art models. which will give an overview of how good the clustering has been done and what topics are generated.

The output of the complete text based on the sentences belong to the particular cluster will looks as shown in the below attached imaged,

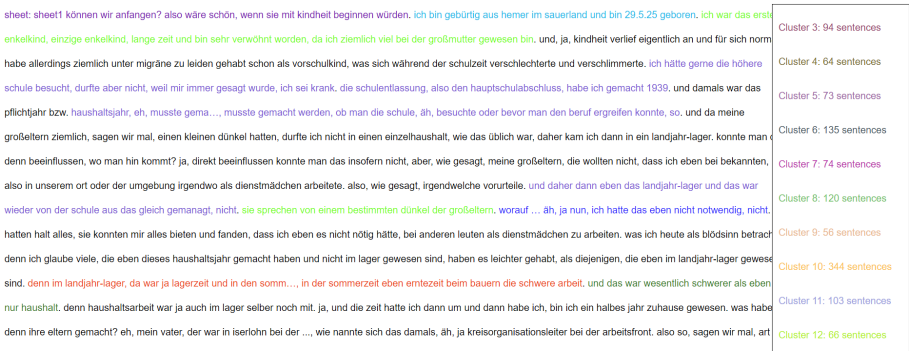

FIGURE 5.1: Clusters sentence distribution w.r.t Original text corpus

This will helps us to get to know how the sentences for each cluster are present and how the topics are distributed accross the documents.

Along with these metrics, I have also compared the clusters formed and the generated topics of the model with human estimation. By taking feedback from humans, on the questions framed in the evalaution design section to understand the model performance.

Human Feedback

On Q1: As jupyter notebooks consist of python code with comment the people with the technical knowledge were able to understand the code without much explanation but, the people with non-technical knowledge were understanding the notebook code based on the comments added. So, the people with non-technical knowledge suggested to having a user interface would be better so that everyone can understand.

On Q2: As the data is huge and there are lot of sentences under each cluster, a random cluster was selected to people to get to know the whether the sentences in the cluster is similar or not and got positive feedback that, the sentence in the cluster is similar to each other and there are lot of sentences in the cluster which has same meaning but expressed in different ways.

On Q3: As a this question is a follow up of Q2, the topic generated by the model for the specific cluster is almost matching and the topic name generated is representing as the title of the cluster which will help to understand related to what domain the cluster belongs to.

On Q4: This question was asked to know if the LLM approach is good or not compared to the traditional approach where I tried to show output of the conventional approach like LDA, NMF etc.., and ask to compare with this thesis approach based on this there was some feedback like the traditional approach has the word representation and this model has a sentence representation. The traditional approach is has grouped the words which are like each other together and the LLM approach has a sentence representation. There was mixed feedback on this question as few people felt traditional approach is better and few people felt the LLM approach is better.

Based on the above feedback, for Question 1, 2 & 3, decide to build a web application which can handle the same text data.

5.4 Discussion

After reciving the feedback from the humans on the evalaution of the models, it was clear that the Jina AI de model was performing well with the HDBSCAN clustering algorithm for the interview transcripts dataset. As the feedback was positive, when it comes to the clustering of the sentences and generation topic name of the cluster made it clear that the technique of achieving the topic modelling using embedding,

cluster and LLM models was good. Eventhrough, the feedback on the comparsiom of the traditional approach and LLM approach was mixed. Based on these feedbacks the idea of building an web application came up and the web aplication was created which can handle the same text data (mentioned in the previous section).

The cluster evaluation plays an important role as it help to understand the quality of the clusters and embedding models. The meteric scores of silhouette score, davies-bouldin score and calinski-harabasz score gave an idea of how good the clustering is done. The evaluation of the topic generation with other LLMs is a manual evaluation which also helped in understanding how good the Llama model has generated the topics and along with the human feedback on the cluster formation and the topic generation help to understand the performance of the model. With the completion of this chapter, the evaluation of the models has been completed.The next chapter will provide a summary of the thesis and also explain the challenges encountered during the research.

Chapter 6. Conclusion and Future Work

This final chapter of the thesis serves to conclude the research, by presenting a comprehensive reflection on the various aspects explored and identifies potential areas for the future research. This chapter will give an quick overview of the thesis by providing the key findings, conclusion and future work. As the main objective of this thesis is to identify an unique approach which can handle the complex relationships among sentences in a text and to achieve the topic modelingusing the latest techniques.

6.1 Conclusion

This thesis proposes a new method of topic modeling, unlike the earlier methods in the forms of LDA, NMF, and LSA. The conventional approaches have great capabilities in topic extraction, but they still strongly depend on word co-occurrence patterns. Besides, they cannot extract the deeper contextual understanding of text data. Meanwhile, BERTopic is a language model-based topic modeling that covers an alternative by making use of embeddings, dimensionality reduction, clustering, and the representation of topics using c-TF-IDF and count vectorizer techniques. Despite such merits, at the same time, BERTopic fails in completely capturing such complex semantic relationships among words in a text. Moreover, performance can be poor for smaller datasets, and results may slightly vary on runs with the same data.

Considering these limitations, a new approach to topic modeling in an attempt to solve some of these challenges. In particular, this thesis focuses on improving topic modeling by providing better semantic understanding of relationships among sentences and extracting more meaningful topics from text. While the novelty of this approach embeds, reduces the dimensionality, and clusters, it makes a further step by feeding the clustered sentences into large language models. Large language models are better equipped to grasp complex semantic relationships and therefore allow for more accurate and insightful topic generation. This, in turn, helps the researchers and analysts in identifying the underlying themes and patterns in text data more appropriately.

As the dataset is in German language, choosing the right embeddings model is a very important part of the process to get the best results. There are many embeddings models which can be used to tackle the problem but, I chose Jina embedding-v2-base-de model because of its capabilities to handle both English and German languages data and gave better clustering results than other models and it uses less computational power and time to process the data. For clustering the embedded data, I have used HDBSCAN algorithm over K-Means as it struggles to handle noise in the data and also need to specify the number of clusters in advance. As the language models are powerful in extracting the semantic relationships and most of the models are multilingual and handle German language feeding the clustered sentences to LLMs model will help to extract more meaningful topics for the text documents.

6.2 Answers to Research Questions

By answering the research questions that were formed earlier in this thesis, we shall be able to appreciate its possibilities and whether its objectives have been met.

1. **Q1: How does topic modelling aid in the extraction of topic process from the dataset of past interviews?**
   The topic modelling approach can be used to extract the topics from the historical interview dataset as this question was made to understand the topic which were already exsited in the past to understand how the topic modeling has been achieved in the past and how it can be improved. This question help to understand the potential of the approaches like LDA, NMF, LSA, BERTopic, prompt-topic-model etc., and how they can be improved in the past to achieve better results. With the help of this question, I was able to understand the workflow which has been used and about its advantages and disadvantages and how has been used in the past which various approaches to achieve better results. This question helped me to understand the potential of the approaches and how it can be used for the interview dataset. This question also helped me to come up the idea of approaching the problem of handling the huge text data by using embedding, clustering and use of large language models.

2. **Q2: How can we use various clustering techniques to extract information and cluster them from the German dataset?**
   This question will help me to understand the various clustering techniques and how they can be used to in this thesis to achieve topic modelling. There are

many clustering techniques like K-Means, DBSCAN, Hierarchical Clustering, HDBSCAN etc., which will help to cluster the data. This research support for choosing the clustering technique suitable for the dataset. As these techniques are effective in grouping the data which are similar to each other. After understnading the pros and cons of various clustering techniques, which helped to achieve better cluster formation for the text data in this thesis. As the number of topics present in the data is unknow and identifying the topics from the data is difficult this question helped to understand how clustering techniques can be used to achieve the topic modelling. After understanding the various aspects of clustering algorithms, this question helped me to choose the HDBSCAN algorithm to achieve better results. As this algorithm is effective in group the similar data point to each other, it can handle the noise in the data and it also doesn't require specifying the number of clusters in advance.

3. **Q3: How can language models be applied for topic modelling of German datasets?**
   This question was asked in order to know how the language models can be applied to the German dataset. As well all know about the language models and how powerfull they are can be used used for multilingual and cross lingual applications. So, the language models can be used for the German language datasets. However, to achieve topic modelling through language models is not an easy task. As the LLM model can't be used to directly extract the topic from the data and this processing of large amount of data cannot be handled by LLM model. So, this question helped to understand the potential of the approaches and as mention in research question 1, about the thesis approach where the data will undergoes embedding and clustering then, the clustered data will be fed to LLM model through the prompt to generate the topic names. In this way the LLM model can be used to extract the topic from the huge dataset. There are many advantages of using the LLM model to extract the topic when compare to traditional approaches, as the traditional methods will extarct the topic based on the occurrence of the words in the data but, the LLM model can understand the context of the sentences in the cluster and generate the topic names. There are many open source LLm model available and can be used based on the computational power of the device.

6.3 Future Work and Challenges

The methods used in this thesis were effective for extracting the topics from the historical interview transcripts along with the corresponding challenges. Firstly, the noise present in the datast can be managed in much better way. The noise are not removed completely which is ending up in forming a seperate cluster which is full of noise. Secondly, with the limited available of the computational power is major challenge faced as the embedding model and language model needs high computational power to process the data. Latest version of LLMs were not used due to the high computational power. Thridly, handling of the noise in the data, as the removal stopwords and normalizing the data will have higher chance of losing the context of the information. Hence, the small amount of noise has been removed after looking into the cluster results but, the noise are not removed completely.

In Future, with the help of better computational resources the challenges of using bigger and better embedding models and language models can be utilized for even better results. The prompting to the language models can be improved which will result in better topic name generation. Hyperparameters can be fine-tuned to get even better results. The web application can also be improved by using the better UI design and better styling.

Appendix A. Source Code

The Source Code is available on GitHub.

GitHub Link: https://github.com/praveenbm1997/Mapping-Memories-Exploring-Topics-in-Historical-Biographical-Interviews

Bibliography

GeeksforGeeks (2024). Clustering metrics.

Python Documentation (2023). Python documentation - the python standard library.