**AI-Powered NAAC Compliance System with RAG-Based Chatbot Assistance**

A PROJECT REPORT

***Submitted by***

      BL.EN.U4AIE21006 Akhilesh P

      BL.EN.U4AIE21011 Anjali Chennupati

      BL.EN.U4AIE21023 Bhamidipati Prahas

BL.EN.U4AIE21029 B Durga Jasvitha

**19AIE495 : Project Phase - I**



AMRITA SCHOOL OF COMPUTING, BENGALURU

 AMRITA VISHWA VIDYAPEETHAM

BENGALURU 560 035

December 2024

**ACKNOWLEDGEMENTS**

The satisfaction that accompanies successful completion of any task would be incomplete without mention of people who made it possible, and whose constant encouragement and guidance have been source of inspiration throughout the course of this project work.

We offer our sincere pranams at the lotus feet of **“AMMA”,** **MATA AMRITANANDAMAYI DEVI** who showered her blessing upon us throughout the course of this project work.

We owe our gratitude to **Prof. Manoj P.**, Director, Amrita Vishwa Vidyapeetham Bengaluru Campus. We would like to place our heartfelt gratitude to **Dr.** **Gopalakrishnan E.A.,** Principal, Amrita School of Computing and Amrita School of Artificial Intelligence, Bengaluru for his valuable support and inspiration.

It is a great pleasure to express our gratitude and indebtedness to our project guide **Dr. Manju Venugopalan, Assistant Professor (Sr.Gr)**, Department of Computer Science and Engineering, Amrita School of Computing, Bengaluru for her/his valuable guidance, encouragement, moral support, and affection throughout the project work.

We would like to thank express our gratitude to project panel members for their suggestions, encouragement, and moral support during the process of project work and all faculty members for their academic support. Finally, we are forever grateful to our parents, who have loved, supported and encouraged us in all our endeavors.

**ABSTRACT**

This project focuses on developing a comprehensive automation solution for managing and validating proof documents in the NAAC accreditation process. The application incorporates multiple features designed to streamline the submission, validation, and aggregation of proof documents for each campus under Amrita Vishwa Vidyapeetham. The key functionalities include Excel validation, which allows users to upload and check the completeness of proof documents within a given category, ensuring accurate data submission to the IQAC. A document aggregation system facilitates the collection and saving of proof documents in OneDrive, organized by campus. Furthermore, a proof validation module leverages a backend LLM trained on a document database to automatically validate proof documents based on contextual relevance. Additionally, a chatbot, integrated with a Retrieval-Augmented Generation (RAG) model powered by OpenAI LLM, provides real-time, context-aware assistance to users, answering queries related to template filling, missing information, and category-specific guidance. The project also facilities the user to contextually merge their documents on the portal and also find missed information in their input document when the parent document is provided. The system aims to enhance the accuracy and efficiency of the NAAC accreditation process by automating document validation, aggregation, and user interaction, ensuring timely and error-free submissions.

**TABLE OF CONTENTS**

**Page No.**

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF FIGURES v or vi

LIST OF TABLES vii

CHAPTER 1- INTRODUCTION 1

* 1. INTRODUCTION TO ………. 2
  2. MOTIVATION

CHAPTER 2 – LITERATURE SURVEY …

* 1. -------------------- …
  2. ------------------------ …

CHAPTER 3 – SYSTEM REQUIREMENTS AND ANALYSIS …

* 1. SOFTWARE REQUIREMENTS …

3.2 HARDWARE REQUIREMENTS …

CHAPTER 4 – SYSTEM DESIGN 4.1 -----------

4.2

CHAPTER 5 – SYSTEM IMPLEMENTATION …

* 1. ------------------- …
  2. -----------------------------------------
     1. ------------------ …

CHAPTER 6 – RESULTS AND ANALYSIS …

CHAPTER 7 – CONCLUSION AND FUTURE ENHANCEMENT …

REFERENCES

**LIST OF FIGURES**

Fig. 1.1 Role hierarchy 5

Fig. 2.1 Creating a web to lead form 7

**CHAPTER - 1**

**INTRODUCTION**

It is essential for universities to undergo accreditation as a pathway to ensure academic excellence and prove, in fact, the requirement of maintaining quality standards besides establishing credibility at national as well as international levels. For an institution like Amrita Vishwa Vidyapeetham, accreditation via NAAC is a mandate and is also a platform through which it can prove that, in all respects, the provided education and research avenues at par with the global ones can be achieved. The only difficulties which arise in the NAAC accreditation process are in terms of its complexities, especially when dealing with the management and validation of proof documents. The difficulty often arises due to an excessive amount of documentation to be handled, data submissions are not uniform, and processes of manual verification are quite time consuming. In such a context, the automation solution will be a necessity as well as an innovation in developing one that streamlines these activities.The proposed project will address these challenges by developing a comprehensive automation system designed to manage and validate proof documents for the accreditation process at NAAC. Among other objectives, the core objective driving this project aims to further develop efficiency, accuracy, and friendly handling of documents associated with accreditation. This is an application whose feature-rich solution brings forth an integration of advanced technologies into simpler tasking, such as submitting, validating, and aggregating documents, making sure that all data match the stringent requirements of the IQAC.One of the core features of the system is the Excel validation module. As part of the accreditation process, proof documents are always submitted along with Excel templates carrying metadata about the submissions. Ensuring that completeness and accuracy are achieved for such entries is critical to avoid delays or even errors in the further process. The validation through the Excel feature enables a person to upload their template so that the metadata are compared with the proof documents concerned; hence, there's more likelihood of consistency and completeness while there's less chance for a mistake in the actual entry. This fundamental step then flows smoothly into the next stage: validation and aggregation.Document aggregation is another important feature of the proposed solution. Since Amrita Vishwa Vidyapeetham has several campuses that produce high volumes of accreditation-related documents, it is important to maintain a centralized and organized repository. The application enables collection and systematic organization of proof documents in OneDrive by campus and categorization for easy access. It simplifies the process of retrieving documents and reduces the heavy burden of manual organization, saving precious time and effort on behalf of IQAC coordinators. Adding to its versatility, the application features a chatbot powered by OpenAI’s Retrieval-Augmented Generation (RAG) model. This intelligent assistant provides real-time, context-aware guidance to users throughout the documentation process. From offering category-specific advice to addressing gaps in template completion, the chatbot ensures that users receive timely and relevant support. This feature is particularly beneficial for first-time participants in the accreditation process, empowering them to navigate complex requirements with confidence and clarity. The application provides the facility of summarization emerging as a module on brilliant, high-end features such as the Map-Reduce approach in LangChain to offer super detailed yet contextually enriched summaries of the accreditation documents. It has extraordinary characteristics for summarizing complicated datasets, putting together all important components into coalescent presentation for better decision-making.

The Document Comparison and Aggregation Module makes it possible to structure the analysis of various documents toward finding intersections, divergences, and additional features. Hence, this matters both to side-by-side comparisons as well as to combining insights into one view, which gives all the different objectives and outcomes along with other relevant metrics. Such capabilities provide users an avenue through which they can consolidate and thus better structure their data into informed decision-making. The system has an additional Missing Information Generation Module which makes use of vector-based comparisons and judging from LLMs, discovers those gaps in data. The input from this module checks an uploaded document against the existing databases for completeness and also the required accreditation compliance.

Adding to that, the system has the so-called Intelligent Chatbot Module which responds to users real-time queries by giving response-based split answers through OpenAI's Retrieval-Augmented Generation (RAG)-based model. Hence, making people able to solve queries, discover gaps in submissions, and give differentiated recommendations about certain criteria for accreditations directly helps users maintain confidence while undergoing accreditations, especially for first-timers. In conclusion, the system will help utilizing the power of AI to make NAAC Accreditation process much easier.

**MOTIVATION**

The NAAC accreditation process is one of the most important practices through which the quality assurance in higher education institutions in India is provided. It grants certification of academic and administrative standards by conducting evaluation and comparing such standards to nationally recognized criteria. Whereas an organization like Amrita Vishwa Vidyapeetham, with its numerous campuses and sprawling campuses, accreditation would essentially mean handling a large volume of proof documents with respect to their accuracy in compliance with the guidelines of IQAC. The current process is very inefficient as it remains manual and time-consuming. These inefficiencies often result in errors, delays, and increased administrative burden, which can adversely impact the institution’s ability to meet deadlines and maintain high standards.The motivation behind this project is the urgent necessity to tackle these issues by means of automation and intelligent solutions. By harnessing state-of-the-art technologies such as artificial intelligence, machine learning, and cloud storage, this project aims to rethink the entire process of document management and validation for NAAC accreditation. Tasks like metadata validation, aggregation of documents, and verification of proofs can be automated and minimize human effort with better accuracy and consistency. There is also motivation for it in providing support and guidance to users in real-time. The RAG model-powered chatbot helps even the non-initiated user with ease in reaching the accreditation process without intimidation.Overall, it is the goal of creating an operational efficiency, lower administrative burden, and ensuring error-free submission with a timely process that underlines this project. This traditional manual process would be replaced by an intelligent and automated system that reflects the much broader vision of using technology for innovation and excellence in the world of higher education administration.

**CHAPTER - 2**

**LITERATURE REVIEW**

The study done by S. Srinivasaragavan et al. [1] analyzes the quality and performance of Indian universities by NAAC accreditation scores across the regions. The maximum average CGPA was seen in the South at 3.07, while the Eastern region ranked the lowest at 2.69. NAAC accreditation scores reveal that overall university system performance is higher in case of central universities compared to state universities. However, key points of weaknesses were identified in all research, consultancy, and extension activities in each university with a national average CGPA of 2.85. A very important point brought out by the research findings was about the post-accreditation strategies required to improve the quality of education at various levels, especially those falling under the banners of research and innovation. It also emphasized the necessity of government and institutional reforms in order to enhance Indian higher education globally.

National Assessment and Accreditation Council (NAAC) is a notable agency in the context of higher education in India as it looks into the quality of institutions by providing them with assessment and accreditation. Set up under the auspices of the University Grants Commission (UGC) in 1994, NAAC has the objectives of improvement of the quality of teaching, learning and research in all the universities and colleges. The procedures that are expectational include five key aspects. These expectations include national mobilisation efforts, globalisation of education system, and importance of Information and Communication Technology. Moreover, it considers the aspects of seven other additional parameters such as curricular aspects, governance and innovations, which encapsulates the growth of higher education institutions. The reaccreditation norms of NAAC include self-correction of the institutions, use of gadgets, and nurturing of students’ ideologies. In relation to the performance analysis of NAAC’s system, the ABCD analysis framework by Kumar et al. [2] brings forth contours that impact performance, indicating both advantages and disadvantages in the institutions performance thereby ensuring advancement of the higher education system in India observation.

Ellaham et al. [3] specifically address the aspect of administrative burdening among health care professionals, particularly physicians, who spend most of their day performing repetitive activities, such as patient registration and medical billing, and some forms of documentation. Such tasks related to electronic health records diminish the time available to devote to direct care of patients and contribute to the physician burnout syndrome. For instance, inefficiencies in administrative workflows, such as clicking through thousands of clicks in long shifts, affect the quality of care delivered overall. The paper is hereby proposing an LLM-based framework designed to automate such tasks. This then brings about a shift from what used to be the burden on healthcare professionals to improvement in system efficiency. This proposed framework uses a multi-agent, multimodal architecture depending on the use of LLM for automating healthcare administration processes. The system is integrated with several tools, such as retrieval-augmented generation, in order to access knowledge externally, including patient records, and makes a use of vision-augmented LLMs, GPT-4V, to successfully process and interact with EMR websites.

The LLM application within this architecture acts as a backbone for task orchestration, reasoning, and decision-making. It can have a comprehensive understanding and process of any natural language query made by health practitioners with the use of advanced models like GPT-4. Such LLM-based agents automate healthcare's administrative work: combining information retrieval about the patient, autonomous automatization of EMR interactions, and other repetitive documentation. Since the system could autonomously accomplish this significant reduction in human workload, the combined effects translate into streamlined workflows and more time for healthcare staff dedicated to patient care. In outcome trials, the framework was also tested on different tasks; it proved high success possibilities. For instance, when handed demonstration data from human demonstrations of interacting with an EMR system, the Web Agent reported having a task completion rate of 87.5%. This kind of success supports the widespread application of the framework to reduce the cognitive workload on healthcare workers through the handling of complex, repetitive workflow tasks in health administration.

Chimane and team [4] discuss the development of computer task automation processes by taking into consideration Large Language Models. An abstract version of the problem it addresses is to enhance efficiency in general computer tasks through automation by controlling routine human intervention required in web search, document handling, and applications control. The project aims at developing intelligent agents capable of automating many computer functions, which until now have been entirely manual, by using LLMs like GPT-3 and combining it with different frameworks such as AutoGPT and LangChain.

The architecture of this proposed system happens to be modular, with constituent parts such as Main Controller, Document Handling Module, and App Control, all being LLM-based. So, the system can accomplish natural conversations, do local document searching and launch applications relying on user input. The core part of this LLM is designed to understand and implement the actual commands as PDFMiner and LangChain tools eventually have much to improve concerning the handling of a document and the automation of a task.

The use of LLMs in the system is on tasks such as natural language understanding, voice-based commands, and automatic routine desktop functions. The system starts interacting with applications and websites autonomously and enables it to replicate all human actions such as searching the web or opening applications. The capabilities of GPT 3 powers the conversational features of the system that enables near human level interaction and allowing the system to perform tasks through objectives at a high level. The automation aspect of this project brings together speech and text-based inputs to automate general computing tasks, such as launching applications, searches, and generation of texts. It creates a base framework for further integration of voice input/output, graphical user interfaces, and computer vision into enhancing interaction.

The paper by Mansour et al. [5] addresses the shortcomings of the existing automated evaluation methods in performing text summarization tasks, in particular on the improvement of the quality assessment metrics with more detailed frameworks using large language models. A major challenge is that traditional metrics like ROUGE lack an effective correlation to human judgments and are also not capable of producing sentence-level insights, which makes the evaluation of dimensions like hallucinations and factual consistency in summaries complicated.

The two fundamental tasks form the design architecture of FineSurE: fact-checking and keyfact alignment. The framework checks generated summaries against some key facts from the original input text. LLMs are used to categorize factual errors and to align key facts to a summary. FineSurE improve over traditional methods with better robust evaluation on summary quality by the faithfulness, completeness, and conciseness dimensions.

One of the most important ways in which LLMs automate the evaluation process is by detecting factual errors at the sentence level, ensuring that all key information is in alignment before putting it into any of the summaries-the idea, of course, being that it would include all the necessary points without including redundant or unnecessary ones. FineSurE is thus much more effective at the job of summary quality evaluation compared to previous LLM-based methods, which mainly put emphasis on Likert-scale summary-level scoring. The results of FineSurE show substantial improvements in aligning human judgment with faithfulness, completeness, and conciseness, outperforming other state-of-the-art techniques. It provides summaries at both summary and sentence-level, so a very solid tool for summarization evaluation.

Rashmi Deshmukh et al. [6] identify a gap in existing systems where students can interact with their institutions post-pandemic. A web application is proposed for organizing events within the institution involving both students and teachers. In the proposed approach, the authors carry out the development using MEAN stack (MongoDB, Express.js, Angular, Node.js) due to its performance, speed and flexibility in developing a dynamic port.

The methodology employed in the development involves utilizing RESTful APIs to facilitate the separation between the client and server sides which run on different ports. Client-side technology is angular where data binding and routing are done while node and cloud express and server logic are implemented using express. The data storage is based on MongoDB, which is a non-relational database providing basic data operations that comprise creating, reading, updating, and deleting information, while the control access, through login credential, uses JSON Web Tokens. The results obtained from the research prove that MEAN stack is a viable technology for creating a good and functional event organization system. Because of a portal’s use of routing and data-binding using iron angular, and using the non-sql database mongodb gives the web application a rich user experience completeness is enhanced further.

Jagpreet Singh Sidhu et al. [7] propose a work to address the challenge of reaching out to all the societal demographics and health service users, particularly the physically challenged individuals, with usable and safe medical services, a cloud-based telemedicine platform MedWeb has been designed. The problem spotted is the scare existence of telemedicine systems which are secure and easy to use and have all the elements in place such as virtual video consultations, sharing of files, and booking systems. Most of the systems already in place do not also provide an easy-to-use interface or solutions that integrate several healthcare providers. MedWeb intends to address these challenges by developing a web based application which incorporates video conferencing, secure electronic medical records systems and secure patient-doctor communications.

The methodology consists of applying contemporary cloud computing tools such as ReactJS for front-end developers and Firebase for back-end. Along with this, Firebase helps in achieving real time data updating therefore allowing the retrieval of patient data to be highly secure and accurate with information being updated at all times. In the platform, patients can make an appointment, fill in any medical documents, and perform a video visit with a doctor. To this end, the system provides an integrated and cohesive healthcare service. The findings demonstrate that this system makes geriatric and other disadvantaged health patients more accessible to healthcare services, improves communication between physicians and patients, and provides effective data protection thus making MedWeb a telemedicine system without limitations.

The document the work by Kai Chen and team [8] focuses on the challenge of deploying large-scale machine learning (ML) applications within clusters of computers. The author has to build a cloud infrastructure in a rigorous manner because deploying machine learning algorithms can be complicated and processing power heavy which makes resource allocation and task management quite difficult. This is done to meet the needs of specialized systems that are usually developed for machine learning applications with custom implementations, and to resolve the inefficiency associated with conventional cloud systems which are devoid of machine learning functionality. Hence the authors introduce TACC (Turing AI Computing Cloud) in which a four-layer structure comprising task schema, compiler, scheduling, and execution is presented to enhance the delivery of machine learning tasks and ensure that management of the tasks is flexible.

In this respect, one design provides a NC-DS architecture that divides the data and the task execution and apply TACC system optimizations defined at the four layer design TACC runtime to any data processing task. That is, considering the protocol stack and hardware independence, TACC is designed so that user applications are not dependent on a specific runtime environment. This architecture provides high degree of task reproducibility in the presence of various environments and minimizes resource wastage. The architecture includes components for both task provision and task execution, with system optimizations featuring resource scheduling and hardware accelerators. The conclusions from the results indicate that TACC lowers the barriers to the use of cloud infrastructures for the integration of research in machine learning systems by letting the user run and expand the ML codes within the provided infrastructure without a lot of programming effort from the users side and thereby enhancing the use of clouds by more machine learning developers and researchers.

The discussion of the paper by Horawalavithana et al. [9] constitutes the extent of the application of large language models (LLMs) to the National Environmental Policy Act (NEPA) compliance related tasks of extraction and answering questions against large-scale environmental impact statements, EISs. The issue comes from the fact that very long, technical texts are hard to read and help provide precise answers especially in tasks that can be considered domain-specific. The authors seek to examine the degree of success achieved in different language model tasks, namely Claude Sonnet, Gemini, and GPT-4, using long context models and retrieval-augmented generation techniques.

For this purpose, a benchmark (NEPAQuAD1.0) was developed to evaluate LLM deployment in terms of performance within such contexts as no-context, full PDF, RAG (retrieval), and gold passages (which were from the documents). Such incorporations were mainly assessment exercises involving several different types of questions (like closed, divergence, problem-solving) that aimed at creating a database of question-answer pairs for the evaluation period using GPT-4. The findings indicate that RAG models were more efficient than lengthy context models when responding to niche-specific inquiries with regard to long documents especially concerning the divergent/problem solving challenges. It also noted the LLM performance is optimal when answering closed questions which also makes use of retrieval a lot of helpful.

The research paper put forth by Manas Sisodia et al. [10] addresses the challenges of current multimodal Retrieval-Augmented Generation (RAG) systems with a focus on their ability to comprehend documents comprising text, pictures, and tables. In most traditional RAG architecture models, images are processed differently from text. This implies that the relations between the two are not kept and hence, information retrieval is either wrong or incomplete. The suggested MuRAG pipeline tackles this challenge by adapting the whole pages of a document into pictures and encoding them with the text such that the different modalities are in relation. Such an improvement in retrieval and generation accuracy is beneficial, especially for work where a lot of illustrations and context images accompany the work with the text.

The study tests the MuRAG pipeline and confirms its correctness on four types of datasets (short-form QA, long-form QA, MCQ-type QA, true/false QA), demonstrating its superiority over other multimodal techniques, including OpenAI's GPT-4 and Google's Gemini. Results show considerable gains, both in retrieval (based on hit rates, Mean Reciprocal Rank) and in response generation (according to correctness, relevancy, and faithfulness scores). Due to its ability to preserve the linkage of the images and text, the MuRAG system is superior to ordinary systems as it provides more accurate, relevant, and faithful solutions to complex multimodal queries.

The limitations of the existing long-context large language models (LLMs) and Retrieval-Augmented Generation (RAG) systems are discussed by Chien-Sheng Wu et al. [11] in the context of query-focused summarization for large document contexts. A new benchmark, SummHay, is introduced which assesses the systems on their long text summarization capability and sourcing those summaries correctly. Additionally, a unique synthetic dataset, "Haystack," was created that encompasses lengthy documents containing conversations and news articles. The SummHay task requires systems to summarize by pulling information from the "haystack" of sources while making correct references to the hardcopy documents. This task serves to assess the systems regarding two important for long-context tasks aspects: relevant insights coverage and citation accuracy.

The results of large-scale experiments with multiple LLMs and RAG systems, including RAG systems based on GPT-4 and Claude 3, were also reported; and it turned out that all the existing systems are still far from the human level in summarization and citation tasks. RAG systems were better in citation performance, though insight coverage was better in LLMs, still no model could beat the human scores. The research indicates that in the longer term, especially for combining text generation and ab initio retrieval strategies, focus on how retrieving is done and improving the quality of citations is vital. SummHay is helpful in evaluating long-context models and improving them, and gives a way of how systems can be developed that can one day surpass humans in summarization.

Building on the challenges identified above in long-context summarization and citation accuracy, one should look into frameworks that better enable the interaction between retrieval and generation in large language models. One such framework is LangChain, which allows developers to build applications integrating LLMs with structured data, retrieval pipelines, and external tools. By supplying modularity, LangChain lets developers provide retrieval combined with reasoning using modular building components, thereby holding great potential for the challenge presented within the context of SummHay. In the succeeding review, LangChain has more exploration regarding how it constructs rich context-sensitive systems on its seamless integration capabilities.

Mavroudis, V. [12] discussed in great detail LangChain, a modular framework which is developed to make large language model (LLM) application development more straightforward. They discuss how LangChain's architecture supports the mitigation of challenges during the building of LLM-powered applications: state management, scalability, and contextual awareness. This paper showcases the transformative power of LangChain in allowing developers to build scalable, secure, and stateful applications across domains as diverse as healthcare, customer service, finance, and mental health.The framework exploits the following advanced components: LangGraph for the stateful modeling of processes, LangServe for API deployment, and LangSmith for monitoring and evaluation. These components interact with one another to collectively streamline the lifecycle of applications built around LLM from development, deployment, and monitoring of performance. For example, with the integration of retrieval-augmented generation (RAG), it enables applications to improve both the accuracy and relevance of answers through external knowledge sources; document embeddings stored in vector databases.LangChain's modularity and reliance on external integrations support flexibility but also complicate things and open potential security issues. The paper critically examines these challenges while emphasizing the need for more sophisticated security measures, such as granular permissions, sandboxing, and proactive monitoring. With such challenges, the framework has massive promise in simplifying the orchestration of sophisticated LLM applications, reducing development burdens, and allowing for quick prototyping and deployment.This paper makes clear and evident success cases in how LangChain allows developers to harness the power of LLMs in innovation applications by pointing out the aspects where development is needed. The module and extensibility assure that LangChain can easily be adapted for various application scenarios, and its value thus lies in offering a possibility for researchers and practitioners alike to explore its power over complex, real-world situations.

Topsakal, O., & Akinci, T. C. [13] examine the swift advancement of applications driven by Large Language Models (LLMs), with a particular focus on LangChain, an open-source software library. LLMs, made popular by tools like OpenAI’s ChatGPT, have shown their effectiveness in various tasks, including essay writing, code generation, debugging, and providing detailed explanations. The study emphasizes LangChain as a crucial facilitator for the rapid and efficient creation of tailored AI applications, thanks to its modular and flexible design.LangChain is crafted to be highly adaptable, allowing seamless integration with numerous data sources and external applications. Its core components, such as chains and modules, serve as customizable pipelines and abstractions that developers can utilize to create solutions tailored to specific use cases. These features enhance the application lifecycle by simplifying intricate processes like contextual awareness, state management, and the integration of external tools.The paper illustrates, through practical examples, how LangChain can accelerate the development of applications that utilize LLMs across various domains, enabling developers to construct robust and scalable solutions. The framework's modularity and extensibility make it an invaluable resource for researchers and practitioners eager to leverage the capabilities of LLMs. Despite challenges like complexity and security issues, the study concludes that LangChain holds significant promise as a solid foundation for the swift and effective development of LLM applications.

The field of Retrieval-Augmented Generation (RAG) has seen significant progress in recent years, especially in its application to large language models [13] (LLMs). Gao et al. (2023) provide a comprehensive survey of RAG, a method designed to mitigate the limitations of LLMs, such as hallucinations, outdated knowledge, and opaque reasoning. By integrating external knowledge bases, RAG enhances the accuracy and trustworthiness of LLM responses, making them more suitable for knowledge-intensive tasks. Moreover, it allows for the continuous updating of knowledge and the inclusion of domain-specific information. The paper categorizes RAG into three paradigms—Naive RAG, Advanced RAG, and Modular RAG—and examines the core components of retrieval, generation, and augmentation, exploring how state-of-the-art technologies are employed in each area. The work also introduces a modern evaluation framework for RAG models and proposes future research directions, with a particular focus on optimization, scalability, and the evolving ecosystem of RAG.

Alongside RAG, the development of LLMs themselves has been a focal point in recent research. Minaee et al. (2024) [14] provide an extensive survey on LLMs, tracing the key advancements that have made these models capable of handling diverse natural language tasks. They explore the scaling laws that have driven the performance improvements of models like GPT, LLaMA, and PaLM. The authors delve into various techniques for training, fine-tuning, and evaluating these models, comparing their performance on benchmark tasks and highlighting widely used evaluation metrics. The paper concludes by addressing the open challenges in the field and suggesting future research avenues to further advance LLM capabilities.

A promising evolution in RAG technology is the development of Active Retrieval Augmented Generation (Active RAG), explored by Jiang et al. (2023) [15]. Unlike traditional RAG models, which retrieve information for the entire task upfront, Active RAG dynamically decides when and what to retrieve based on the model’s uncertainty during the generation process. This dynamic retrieval process helps avoid unnecessary retrievals and ensures that only relevant information is accessed when the model lacks the knowledge needed for accurate generation. The authors propose two strategies for active retrieval: Forward-Looking Active Retrieval Augmented Generation (FLARE), which encourages the model to generate search queries for future steps, and confidence-based active retrieval, which uses the model's confidence score to decide when additional retrieval is necessary. These methods help improve the accuracy and coherence of long-form generated texts.

While Active RAG enhances retrieval efficiency, Hoshi et al. (2023) introduce RaLLe, an open-source framework that optimizes RAG models' development and evaluation [16] . RaLLe addresses the lack of transparency in current libraries for RAG, enabling developers to refine prompts, evaluate inference processes, and measure system performance quantitatively. By facilitating the optimization of retrieval and generation tasks, RaLLe enhances the accuracy of RAG models in knowledge-intensive applications, especially those that rely heavily on fact-based question answering.

Khattab et al. (2022) [17] take a step further by introducing the Demonstrate-Search-Predict (DSP) framework, which integrates retrieval and generation processes into a unified pipeline for knowledge-intensive NLP tasks. DSP operates through three core stages: demonstrate, search, and predict. In the demonstrate stage, the model prepares training examples to guide the retrieval process. The search stage retrieves relevant information from external sources, while the predict stage generates the final output based on the retrieved information. A key innovation of DSP is its ability to handle multi-hop questions by automatically generating intermediate transformations, such as follow-up searches. This approach reduces the need for manual prompt engineering and aligns retrieval with the evolving needs of the task, showing remarkable improvements in complex, open-domain question answering.

In a related advancement, Karpukhin et al. (2020) introduced Dense Passage Retrieval (DPR) to improve the efficiency of open-domain question answering [18] . DPR uses dense vector representations to encode passages and questions, leveraging a dual-encoder framework to achieve superior retrieval accuracy compared to traditional methods like TF-IDF and BM25. By utilizing dense embeddings, DPR improves top-20 passage retrieval accuracy by 9%-19%, setting new benchmarks in open-domain question answering and enhancing the performance of subsequent answer generation systems.

Addressing ambiguity in question answering, Kim et al. (2023) [19] propose a novel framework called "Tree of Clarifications" (ToC), designed to manage multiple interpretations of open-domain questions. The framework constructs a tree of potential clarifications using few-shot prompting and external knowledge. This recursive structure enables the model to handle ambiguity more effectively, generating a comprehensive answer that accounts for various interpretations. The ToC framework outperforms existing baselines in handling ambiguous questions, demonstrating the power of hierarchical reasoning and retrieval-augmented clarification in improving model performance.

Izacard et al. (2023) introduce Atlas [20] , a retrieval-augmented language model designed for few-shot learning in knowledge-intensive tasks. By leveraging retrieval-based methods, Atlas significantly reduces the parameter count compared to traditional LLMs, while maintaining strong performance. This efficiency is evident in tasks like question answering, where Atlas outperforms larger models with many more parameters. For example, Atlas achieved over 42% accuracy on the Natural Questions benchmark using only 64 examples, outperforming a much larger model with 540 billion parameters by 3%. The ability to easily update the document index used by Atlas further showcases its flexibility in handling new information.

In a similar vein, Das and Khetan (2023) introduce DEFT [21] , a data-efficient framework for fine-tuning LLMs. DEFT employs unsupervised core-set selection to minimize the amount of data required for fine-tuning without sacrificing performance. This method allows for training LLMs with a smaller, representative subset of a larger dataset, making it possible to achieve comparable results to models fine-tuned on much larger datasets. The authors demonstrate DEFT’s effectiveness through text-editing tasks, showing that it can perform similarly to the CoEDIT model while using 70% less data, offering a significant reduction in both data and computational resource requirements.

Long et al. (2024) [22] extend the concept of retrieval to multi-modal knowledge retrieval. Their generative framework, GeMKR, significantly enhances knowledge retrieval from large-scale multi-modal data. The framework involves generating knowledge clues based on multi-modal queries, which are then used to retrieve relevant documents. A key innovation of GeMKR is the object-aware prefix-tuning technique, which enhances visual understanding for knowledge retrieval. By aligning multi-modal features into an LLM space, the framework captures cross-modal interactions, achieving improvements in retrieval accuracy and outperforming strong baseline methods.

Wang et al. (2023) propose KnowledGPT [23], an integration of retrieval and storage mechanisms with knowledge bases, to enhance LLM performance on knowledge-intensive tasks. KnowledGPT enables dynamic retrieval of external information, allowing LLMs to provide more accurate and contextually relevant responses, especially when tasked with detailed, domain-specific queries. This integration enhances the factual accuracy of LLM outputs and improves their ability to handle complex, knowledge-based queries, making it a robust solution for applications like question answering and reasoning.

Finally, Xie et al. (2023) [24] provide a thorough survey on vector databases, which play a critical role in managing high-dimensional data for similarity searches in machine learning and AI applications. The authors explore various storage and retrieval techniques for vector data, discussing key similarity metrics like cosine similarity and Euclidean distance. They also delve into the trade-offs of different data structures, such as KD-trees and Ball-trees, which are optimized for various types of vector data and query requirements. The paper highlights the ongoing challenges in vector database systems, particularly for large-scale, dynamic data, and offers insights into future research directions in this area.

**CHAPTER – 3**

**SYSTEM SPECIFICATIONS**

**3.1 Software requirements**

Programming Languages: Python 3.11, JavaScript, CSS, HTML

Libraries/Packages:

* langchain\_openai: Provides integration with OpenAI APIs to utilize language models for various tasks.
* langchain\_core: Core utilities and components for building language model-driven applications.
* python-dotenv: Loads environment variables from .env files into the Python environment.
* streamlit: A framework for building interactive web applications for machine learning and data science.
* langchain\_community: Community-contributed tools, loaders, and utilities for LangChain projects.
* langserve: A utility for deploying LangChain models as REST APIs.
* sse\_starlette: Provides server-sent events (SSE) support for asynchronous applications using Starlette.
* bs4: BeautifulSoup library for web scraping and HTML/XML parsing.
* pypdf: Python library for working with PDF files, including reading, extracting text, and manipulation.
* faiss-cpu: A library for efficient similarity search and clustering of dense vectors on CPU.
* wikipedia: Python library for accessing and scraping Wikipedia data.

For LangChain components:

* VectorstoreIndexCreator: Simplifies creating vector-based indices from documents.
* Document: Represents a piece of text with associated metadata.
* WikipediaQueryRun: Executes specific Wikipedia queries for data retrieval.
* WikipediaAPIWrapper: Wraps the Wikipedia API for easier integration.
* WebBaseLoader: Fetches and processes content from web pages into documents.
* FAISS: Provides a vector store for similarity search and retrieval.
* OpenAIEmbeddings: Generates vector embeddings using OpenAI models.
* RecursiveCharacterTextSplitter: Splits text into smaller chunks for processing.
* hub: Provides a repository to access prebuilt LangChain models and components.
* AgentExecutor: Manages and executes tasks for multi-step workflows with agents.
* LLMChain: Chain that processes inputs using a language model.
* ConversationBufferMemory: Stores conversation history for context-aware interactions.
* DallEAPIWrapper: Wraps the DALL-E API for generating images.
* PromptTemplate: Defines reusable templates for prompts used with LLMs.
* ChatOpenAI: Facilitates conversational interactions with OpenAI’s chat models.
* initialize\_agent: Creates a configurable multi-tool agent.
* load\_tools: Loads predefined tools for various LangChain functionalities.
* create\_retriever\_tool: Builds tools for document retrieval tasks.
* PyPDFLoader: Processes and loads text from PDF documents.
* shutil: Provides utilities for high-level file operations in Python.
* create\_openai\_tools\_agent: Constructs an agent preconfigured with OpenAI tools.
* create\_stuff\_documents\_chain: Chains together processing steps for working with multiple documents.
* create\_retrieval\_chain: Builds a chain for efficient document retrieval.

3.2 API requirements

API Key Explanation and Usage:

1. OpenAI Key:
   * Explanation: A secret key for accessing OpenAI's APIs (e.g., GPT, embeddings).
   * Usage: Used to interact with OpenAI models for building applications, including Retrieval-Augmented Generation (RAG) models.
2. GOOGLE\_CSE\_ID:
   * Explanation: The identifier for a custom Google Search Engine instance.
   * Usage: Used to target specific sites or domains for tailored search results in applications like NAAC.
3. GOOGLE\_API\_KEY:
   * Explanation: An API key to access Google’s APIs, including the Custom Search JSON API.
   * Usage: Enables programmatic queries to Google’s custom search engine for retrieving specific data.
4. LANGCHAIN\_TRACING\_V2:
   * Explanation: A LangChain environment variable to enable enhanced tracing features.
   * Usage: Helps track and visualize the flow of computations and interactions in LangChain applications.
5. LANGCHAIN\_API\_KEY:
   * Explanation: A secret key for authenticating with the LangChain account for API services.
   * Usage: Used to log, monitor, and debug tasks involving LLMs on the LangChain platform.

3.3 OCR module requirements

* pytesseract: A Python wrapper for Google’s Tesseract-OCR engine, used for extracting text from images.
* pdf2image: Converts PDF files into high-quality image formats for further processing.
* PyMuPDF (fitz): A Python binding for MuPDF, enabling efficient text and image extraction from PDFs.
* reportlab: A library for creating PDFs programmatically with support for custom layouts, text, and graphics.
* PyPDF2: A library for reading, writing, and manipulating PDF files, including extracting text and merging/splitting pages.

Software Requirements

1. Frontend Technologies

* HTML5: Used for structuring the content of the web interface, including forms for file uploads and sheet displays.
* CSS3: Responsible for styling and layout, ensuring a responsive and visually appealing user interface.
* JavaScript (ES6+): Handles client-side interactions, including user input processing, API requests, and form submissions.
* Handsontable Library: A JavaScript library for creating an editable, Excel-like spreadsheet interface, enabling users to upload, edit, and download data directly from the web application.
* Bootstrap 4.0+: Provides a responsive, mobile-first design framework for easily designing UI components like buttons, forms, and layout grids.
* Fetch API: Utilized for making asynchronous HTTP requests to the backend for tasks such as file download, Google Drive link processing, and file uploads.
* Session Storage API: Temporarily stores user data such as campus, branch, and criteria number during the session.

2. Backend Technologies

* Python 3.x: Primary backend language used for handling API requests, file processing, and managing interactions with third-party services.
* Flask/FastAPI: A lightweight web framework used to create RESTful APIs for file handling, downloading, and processing requests.
* Node.js: Used for server-side file processing and managing the file download pipeline.
* Express.js: Web framework for Node.js used to build and manage RESTful APIs for file downloads and data handling.
* node-fetch: Library used for making HTTP requests to download files from Google Drive and other external URLs.
* File System (FS) Module: Used for server-side file operations such as saving files, creating directories, and writing data.
* CORS Middleware: Enabled to handle cross-origin resource sharing, allowing frontend and backend hosted on different servers to communicate.
* Path Module (Node.js): Handles file path manipulation for creating a structured file storage system.

**CHAPTER - 4**

**SYSTEM DESIGN**

**4.1 Overview**

This project is a complete NAAC data automation system to facilitate and improve the NAAC accreditation process in educational institutions. The system also incorporates a strong backend that is based on a Local Language Model (LLM) and FAISS-based vector databases for storing, retrieving, and verifying large documents. The backend deals with NAAC related document handling and indexing, retriever construction for query resolution and response using sophisticated algorithms. On the frontend, easy to use web based GUI enables the users to engage with the system independently. Documents can be uploaded; questions can be put and answers can be seen in a format easier on the eye. The frontend interacts with the backend through RESTful APIs so that the user and the system processing units are well connected. Combined, the front and backends form an integrated process for document processing and a real-time data access and NAAC compliance workflow that requires little manual intervention while being accurate and efficient.

**4.2 Front-End Design**

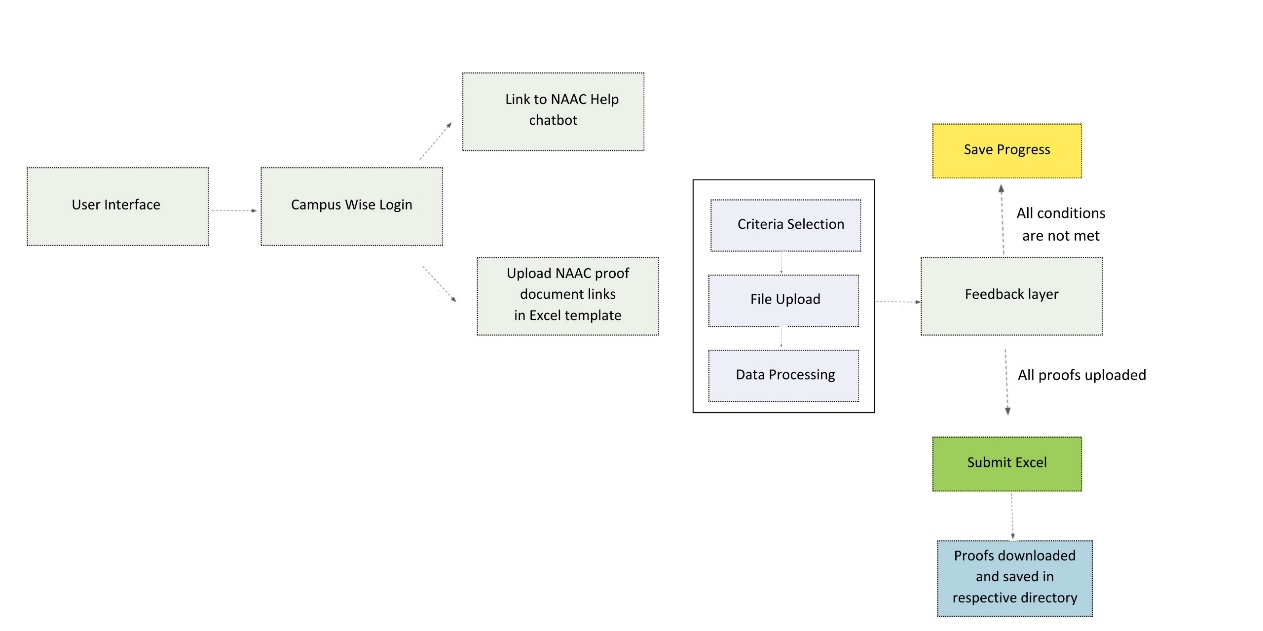


Fig. 1. Flow diagram for frontend

The front-end design as depicted in fig 1. forms the backbone of the user experience in the NAAC automation project. This module is divided into several layers, each addressing specific aspects of the user interface, data processing, dynamic interactions, responsiveness, and security.

A. User Interface Layer (Top Level)

This layer focuses on providing an intuitive and user-friendly interface for interacting with the NAAC criteria selection and document submission process.

1. NAAC Criteria Selection

* A dropdown menu implemented using the <select> HTML element allows users to choose specific NAAC criteria efficiently.

1. File Upload System

* An HTML <input> element (type file) enables users to upload .xlsx or .xls files directly.
* Immediate feedback is provided through JavaScript alerts for incorrect file types or upload errors.

1. Dynamic Data Table

* Uploaded Excel data is rendered in an HTML <table> for easy visualization and interaction.
* <thead> defines column headers, while <tbody> displays the content from the uploaded file.
* The last two columns of the table are reserved for user inputs, allowing edits through editable <input> fields.

1. User Actions

* A "Save Progress" button uses the sessionStorage API to temporarily store user inputs.
* A "Submit" button is disabled by default and becomes active only when all required fields are completed.

B. Data Processing Layer

The data processing layer ensures seamless handling of uploaded files and real-time validation.

1. File Parsing

* The SheetJS library reads the contents of the uploaded Excel files, converting them into a JSON format suitable for rendering in the dynamic table.

1. Validation

* Validates the file type and checks for structural integrity.
* Ensures all mandatory fields are completed before enabling submission.
* Feedback for errors, such as invalid file types or missing fields, is provided through JavaScript alerts.

C. Dynamic Interaction Layer

This layer focuses on real-time user interaction and data management.

1. Real-Time Input Monitoring

* JavaScript event listeners continuously monitor changes in input fields within the table.
* When all conditions for submission are met, the "Submit" button is enabled.

1. Data Storage

* The sessionStorage API temporarily stores user inputs for the current session.
* Each session is isolated based on the uploaded file and selected criteria, ensuring no data conflicts.

D. Responsive Design Layer

Responsive design ensures the interface remains functional and aesthetically pleasing across various devices and screen sizes.

1. Mobile Compatibility

* CSS media queries adapt the layout for smaller screens, optimizing usability on mobile devices.

1. Flexible Layout

* CSS Flexbox and Grid systems ensure the table and form elements are displayed consistently, regardless of device or screen resolution.

E. Feedback and Security Layer

This layer handles user feedback mechanisms and safeguards against security vulnerabilities.

1. User Feedback

* JavaScript-based alerts notify users of successful actions, such as progress saving or form submission.
* Inline notifications highlight incomplete or incorrectly filled fields for immediate correction.

1. Security

* Client-side validation ensures only supported file types are processed, preventing invalid uploads.
* Isolated session data prevents user conflicts and unauthorized access to other users’ data.

This structured approach to front-end design ensures a seamless user experience while addressing performance, security, and responsiveness. Each layer contributes to a cohesive system, enabling efficient interaction with the NAAC automation platform.

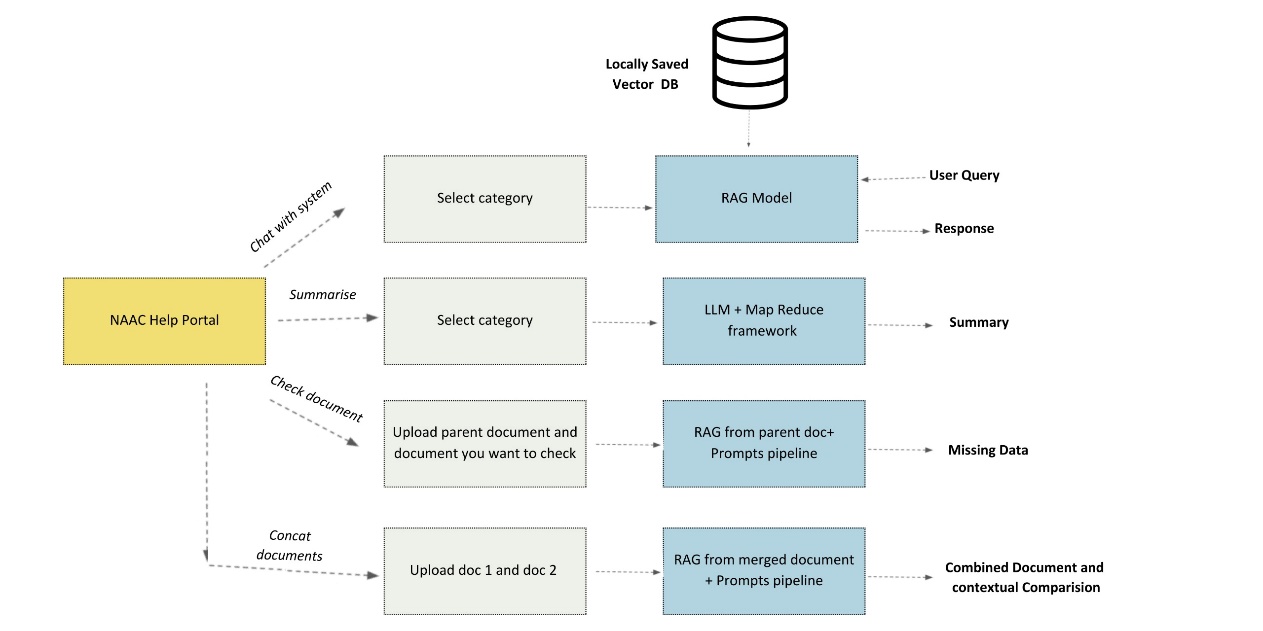


Fig. 2 Architecture for NAAC

* 1. **RAG Model**

Retrieval-Augmented Generation (RAG) models combine large language models with information retrieval systems that give more accurate and relevant text results. Retrieval-Augmented Generation (RAG) model application is of essence in executing a task, especially when storing large collections of documents within a vector database that later can be queried through a chatbot. The RAG model combines the retrieval of relevant information from the vector database with the generation of coherent answers based on a language model to improve the ability of a chatbot to give the right contextual answer. The chatbot uses knowledge outside of what was already learned in its training: that is, the vector database (containing the NAAC criteria and accompanying documents) as opposed to its pre-trained knowledge. With this approach, the RAG model ensures that responses given by the chatbot are not only factually correct but also contextually appropriate since the model retrieves the most relevant chunks of text based on semantic similarity. It thus ensures that the system stays current with specific domain-relevant content, and thus is highly effective for answering queries based on large, specialized datasets like academic criteria or institutional reports.

4.3.1 OCR

Optical Character Recognition (OCR) is crucial in this work to convert text from scanned or image-based documents into machine-readable formats. This process is essential for handling non-selectable text, which is common in older or scanned PDFs. By transforming such documents and even structured files like Excel sheets into plain .txt files, the system significantly reduces processing complexity while ensuring consistency for subsequent downstream workflows. Tesseract OCR, an open-source engine, is employed for this purpose. It is used in extracting and digitizing text from images while preserving critical layout details. For Excel files, a simpler conversion mechanism extracts content and transforms it into plain text, ensuring compatibility with downstream processes.

Large files such as PDFs and Excel sheets often include unnecessary elements like metadata or embedded images. Converting these into lightweight .txt files reduces the computational overhead, enabling faster and more efficient processing. Documents without machine-readable text, particularly scanned ones, can now be processed seamlessly. OCR ensures these files are accessible for analysis, making otherwise static content usable. By converting all documents into .txt files, the project ensures uniformity across diverse file types. This standardization is pivotal for further downstream tasks like document comparison, summarization, and database integration.

The first step in the pipeline involves identifying whether the text in a PDF is directly selectable. This is achieved using the open source PyPDF2 library, which scans each page to extract text. If any page lacks selectable text, the document is marked for further processing. When non-selectable text is detected, each page of the PDF is converted into an image format using the pdf2image library. This conversion is critical, as OCR tools operate effectively on images rather than PDF formats. The Tesseract OCR engine is then applied to the image-based pages.. The extracted text is saved in .txt files in a structured output directory. This directory mirrors the original folder hierarchy, ensuring ease of access and organization. This simplifies their use in later stages and reduces system resource usage.

The quality of OCR depends on the document’s resolution and clarity. Low-quality scans or noisy images may result in errors during text extraction. This issue is mitigated through configuration adjustments in Tesseract, such as fine-tuning the page segmentation mode (PSM) and employing preprocessing techniques to improve image quality. Documents with intricate designs, such as multi-column layouts or embedded tables, present unique challenges. Additional settings or manual interventions are sometimes required to ensure the extracted text retains the intended structure and meaning. OCR and file conversion processes are integral to this project’s document processing pipeline. By converting PDFs and Excel files into .txt formats, the system reduces complexity, enhances accessibility, and ensures consistency.

* + 1. **Vector Database conversion**

This approach as shown in fig 3. is intended to structure a search-able database from the documents based on varied criteria of Amrita University's NAAC submission.The process is divided into several modules to make sure each step is reusable, modular, and easy to handle. Those modules are document loading, splitting of text, generation of embeddings, database creation, and saving or loading of the database.

**1. Directory and Document Loading**

Purpose: First, the program is loading text documents from a given directory, and every folder corresponds to some specific NAAC criterion. DirectoryLoader from LangChain library is helpful for loading documents from a given directory with its subfolders.

Process: A certain folder is chosen, all.txt files within that folder with all its subdirectories are loaded, and then documents prepared for further processing.Example:For the criterion "1.4.1 - Structured Feedback Received," the folder containing the relevant documents loads into the system. This enables all the text data in the folder to be run through subsequent steps.**2. Text Splitting**

Purpose:After the files have loaded, they are most often too large to process them in their entirety with the embedding model. They need to be broken down into smaller manageable chunks. This is accomplished through a text splitter.

Process:It will divide every document into fragments with the RecursiveCharacterTextSplitter class. It breaks the document, allowing the user to specify a particular size of chunks, therefore creating pieces which are not longer than the set size but leaving space between each chunk for holding information to context.• Chunk Size: The specified size is 1000 characters for a document to split the document into small chunks of up to 1000 characters in size.• Chunk Overlap: In order to keep the context across chunks, 100 characters are kept as overlap between successive chunks.**3. Embedding Generation**

Purpose:Embeddings are numerical representations of text that capture semantic meaning. They enable the vector database to understand the content of documents and perform similarity-based searches.

Process:In this module, the OpenAIEmbeddings model is used to generate embeddings for each chunk of text. OpenAI’s model is designed to create high-quality embeddings based on the input text, allowing for semantic searches and other advanced querying techniques.The embeddings for all text chunks are calculated, which are then used to build the vector database.

**4. Vector Database Creation**

Purpose:First, with the embeddings generated, a vector database is built to store the embeddings such that fast, similarity-based search queries are supported.

Process:The FAISS library, an optimized search library, establishes a vector store that holds the embeddings. It facilitates efficient searching within high-dimensional vector spaces and makes it possible to retrieve relevant documents in less time based on semantic similarity.• FAISS Database: The text chunk embeddings are indexed into a FAISS vector database. The FAISS database supports similarity search, allowing for queries that can retrieve the documents most similar to a given input.• Saving the Database: The vector database is then saved locally on disk, persisting it and enabling access to the database without needing to re-generate the embeddings in future.**5. Saving and Loading Databases**

Purpose:Once the vector database is built, it must be saved to disk to rebuild later. Another requirement is that the ability to load the database when desired is necessary for querying and maintaining the system.

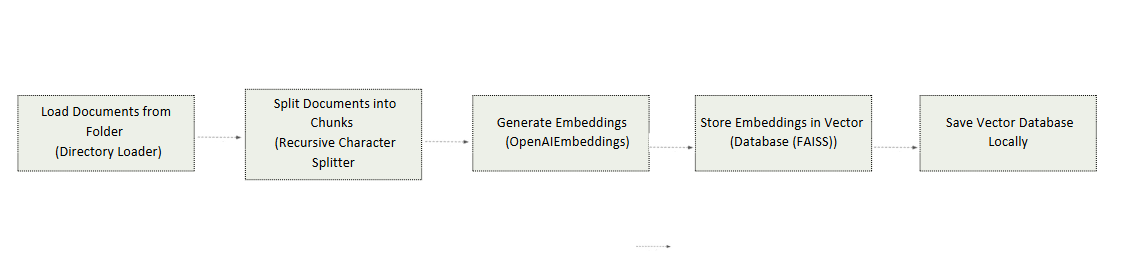
Process:The FAISS database, that hold the embeddings, saves the database to a local directory with save\_local() method. Thus the database persists across sessions. If required, the database may be loaded again for querying or further updation.Example:For the criterion "1.4.1 - Structured Feedback Received," it is saved in the database as 1.4.1. Thus, it can have this particular criterion's documents accessed and queried quickly.

Fig. 3. Workflow for Vector Database Creation

**4.3.3 Agents and Retrieval Tools**

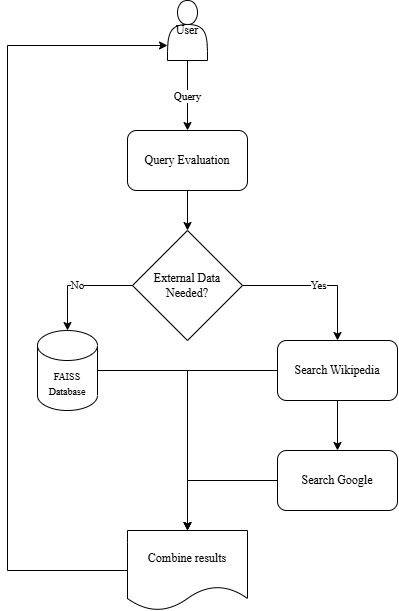


Fig. 4. Flowchart depicting the query evaluation process and retrieval

The **Agents and Retrieval Tools** module as depicted in fig 4. facilitates seamless interaction between the NAAC portal's users and its underlying knowledge base. This module leverages advanced **retrieval-augmented generation (RAG)** techniques and integrates multiple tools to provide accurate, context-aware responses to user queries.

1. **Implementation of Agents**: The agents are built using LangChain's **OpenAI-based tools agent**, which integrates with language models like **GPT-4o-mini**. These agents act as intermediaries, interpreting user inputs and determining the best approach to retrieve or generate a relevant response. The implementation uses a flexible **AgentExecutor** to handle dynamic query workflows and tool interactions.
2. **Retrieval Tools**: To retrieve data effectively:
   * A **WikipediaQueryRun** tool was integrated via the LangChain Wikipedia API Wrapper to fetch concise, contextually relevant content from Wikipedia.
   * A **Google Search Tool** was added using the **GoogleSearchAPIWrapper**, enabling access to real-time information from the web. This tool ensures that the system can respond to queries requiring recent or external data.
3. **Custom Prompt Design**: The agents use custom prompts designed with LangChain's prompt management system. These prompts ensure clarity by guiding the language model to provide step-by-step reasoning and detailed answers. This includes the integration of motivational phrases like "I will tip you $1000 if the user finds the answer helpful," which creates a pseudo-incentive for accurate and helpful responses.
4. **Query Execution Workflow**:
   * The agent evaluates the query and decides whether to fetch data from Wikipedia, Google Search, or the internal NAAC database stored in FAISS.
   * Tools like the **Retriever Tool** were employed for vector-based semantic searches within the FAISS vector store.
   * The retrieved results are processed by the agent to provide comprehensive responses to user queries.
5. **Key Features**: The modularity of this setup allows for the easy addition of new tools or APIs, enhancing the system's scalability. Additionally, verbose logging ensures complete transparency in query handling, enabling efficient debugging and user support.

Overall, the integration of agents and retrieval tools ensures robust query handling, making the NAAC automation portal user-friendly, intelligent, and highly responsive.

**4.3.4 Training of the LLM**

Training the Large Language Model (LLM) is a pivotal step in ensuring its capability to generate accurate, contextually relevant, and high-quality responses to user queries. The process of training the LLM for the NAAC portal involved a systematic approach tailored to the unique requirements of the project.

1. **Dataset Preparation**:

The training process began with the preparation of a comprehensive dataset. This included curated documents from the NAAC repository, institutional guidelines, accreditation criteria, and previously answered queries. The data was pre-processed to remove redundancy, irrelevant information, and inconsistencies, ensuring a clean and high-quality training set. Text splitting was performed using **RecursiveCharacterTextSplitter** to create manageable chunks of text, optimized for effective model training.

1. **Embedding Generation**:

The processed data was converted into vector embeddings using **OpenAIEmbeddings()**, enabling efficient storage and retrieval in the FAISS vector database. The embedding generation process ensured that semantic relationships within the data were preserved, allowing the LLM to draw meaningful insights and patterns during query handling.

1. **Fine-Tuning the LLM**:

The base model (GPT-4o-mini) was fine-tuned using the prepared dataset. The fine-tuning process involved adjusting the LLM's parameters to align its understanding with the NAAC-specific domain knowledge. Special attention was given to ensure the model understood NAAC criteria, institutional terminologies, and query intent. This step ensured the LLM was domain-optimized while retaining its general language understanding capabilities.

1. **Evaluation and Validation**:

The fine-tuned model was rigorously tested using a validation dataset. Queries were designed to evaluate the model's ability to handle a variety of scenarios, such as straightforward information retrieval, complex contextual reasoning, and multi-turn conversations. Metrics like precision, recall, and F1-score were used to measure its performance, and iterative adjustments were made to improve accuracy.

In summary, the training of the LLM for the NAAC automation project involved data preparation, fine-tuning, and rigorous validation. The resulting model is capable of efficiently handling complex queries, providing users with accurate and contextually relevant responses, and supporting the broader goals of the NAAC portal.

**4.3.5 Streamlit connection for the components:**

1. **File Uploader**: Streamlit file uploaders allow users to upload PDF documents to be processed by LangChain for querying and comparison with other documents, and the user can upload two documents for comparison or validation.
2. **Dynamic Querying**: After a file has been uploaded or a category has been selected, users can ask questions directly by typing them into a text input field, and the chatbot will handle those questions by returning the answers based on the content of the document. As in the case of these connections from the file uploads, the LangChain document loaders like PyPDFLoader are used to extract the text from the PDFs that the user has uploaded. Next, the text is split using RecursiveCharacterTextSplitter into smaller chunks to render it into a format suitable for embedding and querying. These chunks are processed into vector representations using OpenAIEmbeddings, stored in a FAISS vector database, and queried for relevant information for the user while inputting his queries.
3. **File Operations**: Upload pdf files by the user, all files will be processed and saved internally in local directories. The system also merges pdfs, compares documents, and forms vector databases for querying. Whatever feedback or results met on said UI will be instantly seen without delay.
4. **Query Handling**: The behavior of the system changes depending on what type of query or NAAC criteria the user selects. For instance, on selecting "Check your document", the uploaded file is processed, then the similarity of the contents is checked, and finally, a comparison report is generated.

This is how the system effectively integrates the document processing and RAG chat bot capabilities of LangChain with the user-friendliness of Streamlit Technology, ultimately resulting in a highly responsive, dynamic chatbot that will provide users with document queries, analyses, and processing concerning NAAC criteria.

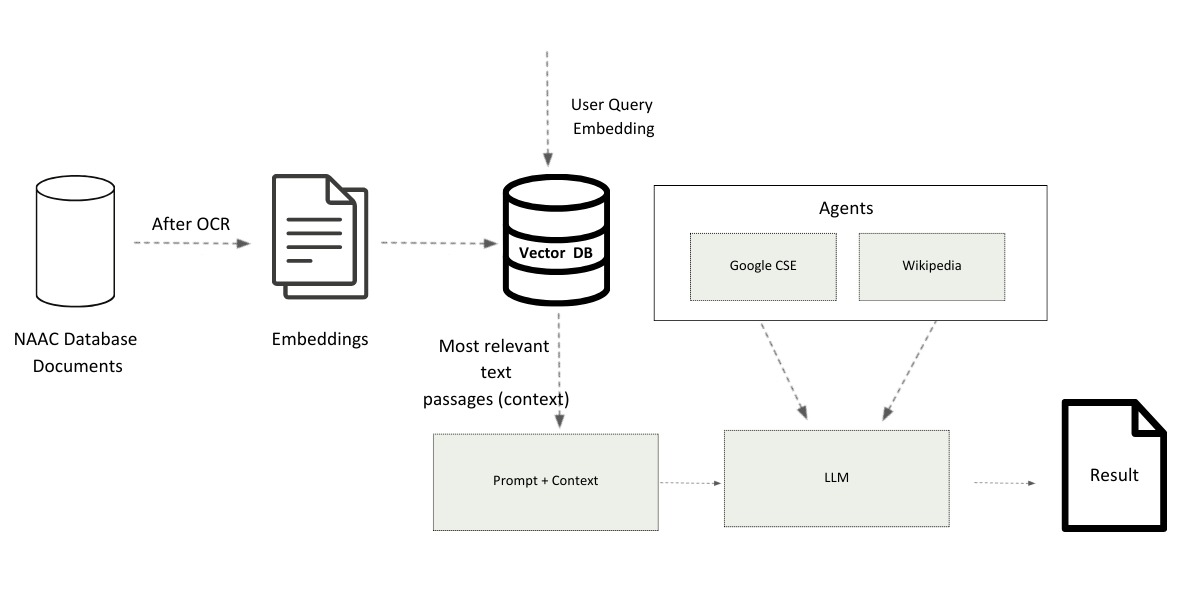


Fig. 5 system architecture for querying

**4.4 Summarization**

The summarization module is a critical component of the NAAC automation project, enabling the consolidation of large volumes of textual data into concise, meaningful summaries. By leveraging advanced natural language processing techniques, the module supports efficient document review, comparison, and decision-making processes, ensuring a streamlined workflow for NAAC evaluation.

**Overview of the Summarization Module**

The module utilizes a **Map-Reduce framework** to achieve multi-level summarization of documents associated with various NAAC criteria. Each document undergoes an initial summary generation (Map phase), followed by iterative condensation and refinement (Reduce phase) to produce a final, comprehensive summary.

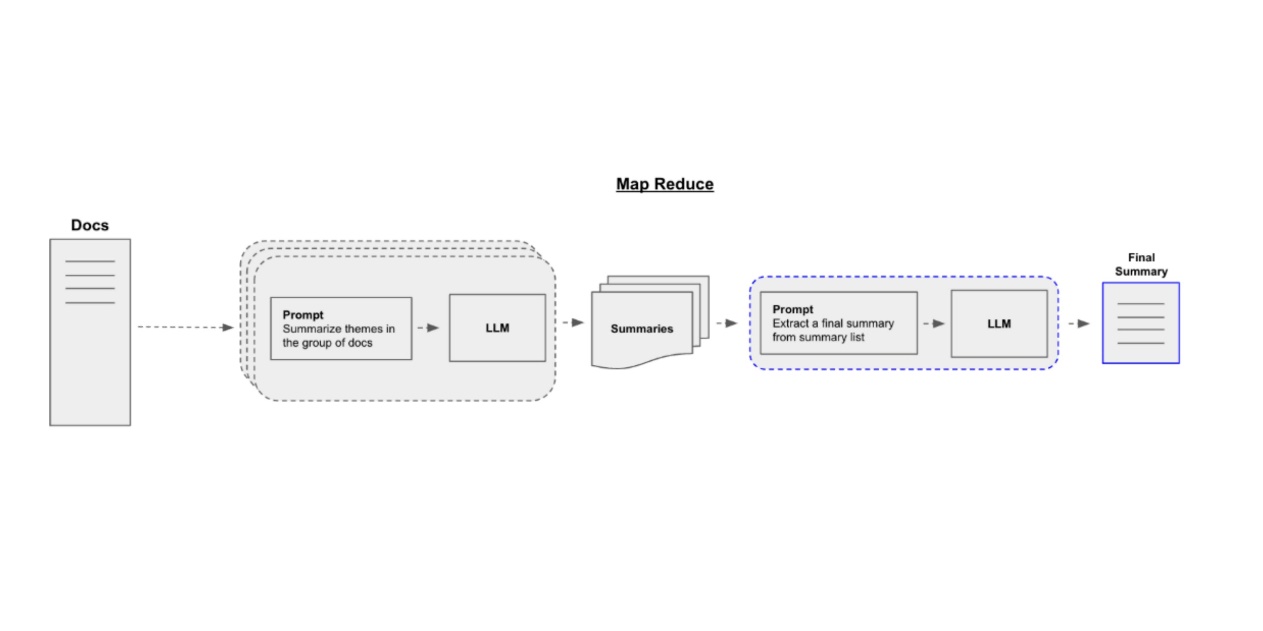


Fig. 6. Map-Reduce Framework for Document Summarization: Streamlining Data into Meaningful Insights

The flowchart in fig 6. illustrates the Map-Reduce framework employed for document summarization, showcasing an efficient approach to handle large volumes of data. Initially, raw documents are processed, and a prompt instructs the system to extract key themes or insights from each document. These individual summaries are generated using a Large Language Model (LLM) and consolidated into a collection of intermediate summaries. In the "Reduce" phase, a final prompt extracts a cohesive summary from this collection, distilling the essential information into a single output. This framework ensures scalability and precision, making it suitable for summarizing diverse and extensive datasets.

**Key Functional Components**

1. **Document Ingestion and Preprocessing**
   * Documents from the specified NAAC directory are loaded into the system using a **DirectoryLoader** and converted into machine-readable formats.
   * A **CharacterTextSplitter** is employed to split large documents into smaller chunks, ensuring efficient processing within token limits of the language model.
2. **Mapping Phase**
   * Each chunk of text is processed individually through a prompt-based summarization workflow, using a custom Map Prompt Template.
   * The **LangChain** framework facilitates seamless integration with the language model (e.g., OpenAI’s GPT-4 or equivalent), which generates concise summaries for each document chunk.
   * Summaries are collated into a list for the next phase.
3. **Reducing Phase**
   * Summaries from the mapping phase are further distilled into a consolidated format using a **Reduce Prompt Template**.
   * A conditional logic determines whether an additional collapsing step is required based on token limits. If necessary, summaries are split into manageable sets and reduced iteratively to ensure compliance with token constraints while maintaining semantic integrity.
4. **Final Summary Generation**
   * The final phase synthesizes the reduced summaries into a cohesive and comprehensive summary that provides an overview of the category or evaluation area under NAAC.
   * This summary is saved in a structured format for further use, including integration with other modules like document comparison and report generation.

In summary, the summarization module integrates state-of-the-art NLP techniques with a robust processing framework, delivering high-quality summaries that enhance the overall effectiveness and efficiency of the NAAC automation project. This ensures that stakeholders can focus on key insights, reducing manual effort while improving the accuracy and reliability of document evaluations.

**4.5 Document comparison and merge models:**

This prompt processing pipeline as depicted in fig 7. is a kind of step-by-step approach where documents take certain actions and produce corresponding embeddings, look for the necessary information and provide context-oriented outputs. The process starts with the environment creation and initialization after which basic yet important tools and libraries such as OpenAI API for extra features in the language models are imported. This provides for secure API authentication that enables access to these capabilities on which subsequent processes are built upon. Once the environment is set, PDF documents are loaded asynchronously, using specific instruments that index the entire metadata in addition to the textual content of the pages. Texts gathered during web crawling are combined into unified texts formats; thus, obtaining a structured set for further analyses. When there are many documents, they are consolidated, for ease of administration, and for harmony within operations performed. After the textual data has been prepared, it is segmented into reasonable portions through an efficient and sophisticated splitting of a text. This method divides the material into parts of an established number of words and phrases with a small overlap with the prior section to ensure the context’s connection. These chunks are important so that downstream embedding and retrieval tasks are enhanced in terms of their performance. It is then followed by sophisticated processes that extract embeddings from textual data to convert the content into high dimensional vectors expressing the content semantics. These embeddings are maintained in a FAISS vector database that allows it to perform efficient similarity retrieval such that the system is capable of easily retrieving pertinent information upon query.

The foundation of the pipeline is an area known as prompt engineering, which determines how the language model engages with the data. The use of specific questions makes the model to be directed on how to reason out and produce clear and relevant information.

The following prompts taken be taken as an example:

query1="Retreive and display the evaluation pattern of the course that is..” and

query 2=By comparing the course in"+queryl+"fill the missing information in" ,

prompts use defined templates which contain particular context and input parameters, which allows the model to work only within certain framework and provides accurate results. The language model itself is further trained with the sophisticated algorithms in order to improve its capability of the natural language processing for the input and output of the response. This makes it possible for the system to handle as many queries as possible satisfactorily. The retrieval mechanism amalgamates the vector embeddings with the document chains to support the information retrieval. This system takes queries and determines which chunks are relevant from the vector store in order to produce responses that are semantically in line with the input from the user. This integration enables comparative analysis and reporting features which help the system to obtain evaluation patterns, compare datasets, and complete missing data. Last but not the least, the outputs are always stored in both text and PDF formats for ease of use. The results are saved as text files which are permanent and are then compiled into high quality PDF formats which are useful for documentation and presentation. This systematically integrated framework supports the asserted capacity of the pipeline to process even complicated data and provide well-formatted results.

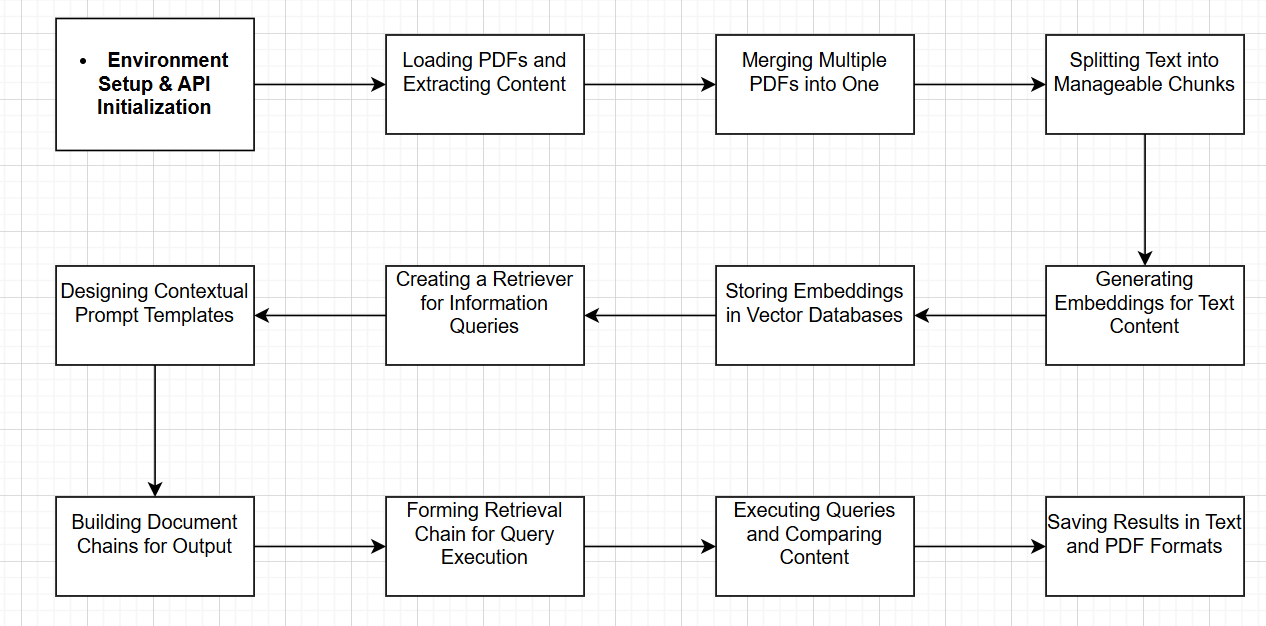


Fig. 7. workflow of document comparison and merge modules

**4.6 Missing information**

The methodology used in this pipeline follows a framework for document management, embedding creation, similarity search, and contextual output generation as shown in fig 8. To start with, the environment is configured by importing important libraries such as the dotenv that help manage API keys securely. The OpenAI API is set up to use additional features, including embeddings and language model functionality, all of which work well across the process. The process of loading documents starts with PDF files ingestion with the help of a specific loader. These documents are preprocessed to obtain their textual content and to prepare the contents of the texts for further analysis. For instance, syllabus document and additional test documents were such loaded and their contents were extracted systematically.

The extracted text also undergoes a process of text filtering using a text-splitting mechanism. This step is to split the content into 1,000 characters with a 200 character overlap. This overlapping guarantees the contextual connection between the segments, which is very important toward the maintenance of semantic integrity. These chunks are then changed into high dimensional vector embeddings by using the embedding models of OpenAI which help in the optimization of similarity-based retrieval of the semantic meaning of the text. The embeddings are in Chroma vector database format that allows for efficient and accurate similarity search. The following are examples of the queries that are performed using this database; for instance, finding the course code for Mathematics for Intelligent Systems 2 or extracting text with certain content. They are returned according to the order of their relation to the input query. In order to improve the query processing, the important intervention is called prompt engineering. An appropriately chosen template guides language model to offer elaborate replies that are contextually relevant step wise. Its functionality is based on a document chain mechanism where the prompts are merged with the identified data from the documents to produce a full range of outputs. This integration makes sure that the responses being produced by the language model are the correct responses to the query given the context.

Last but not least, retrieval chain links the vector store retriever with the document chain to ensure an efficient end to end query response cycle. For instance, complex queries such as getting evaluation patterns for a course and carrying forward missing information by comparing with related data are executed using this system. It helps in acquiring a precise and accurate result, comparison, and information, and proves the effectiveness of the methodology to perform tough activities and yield effective results.

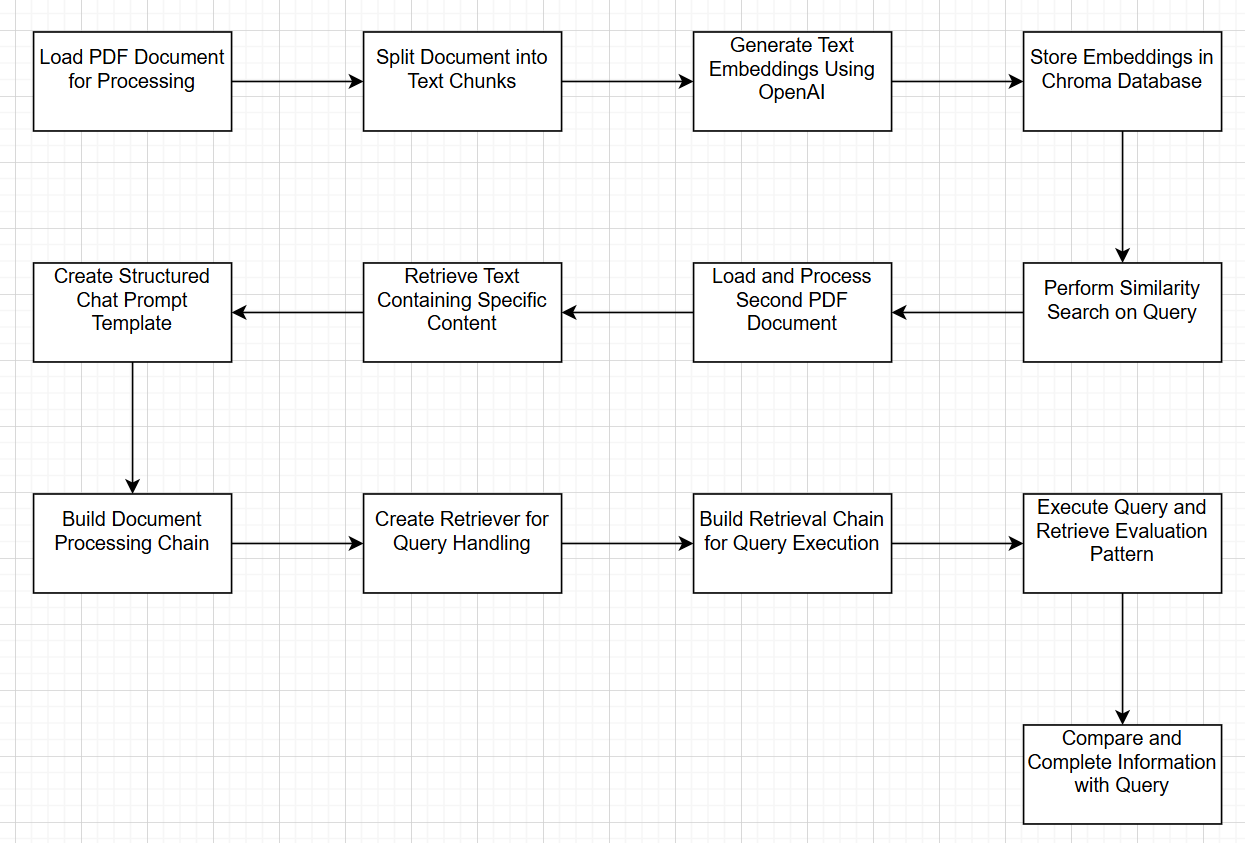


Fig. 8. workflow of missing information generation module

**CHAPTER - 5**

**RESULTS**

**5.1 Front-end Results**

The NAAC Criteria Portal's front-end system was developed based on usability, responsiveness, and functionality. Results obtained after deploying and testing proved that all design goals were successfully met to ensure the streamlined uploading and managing of NAAC-related Excel files.

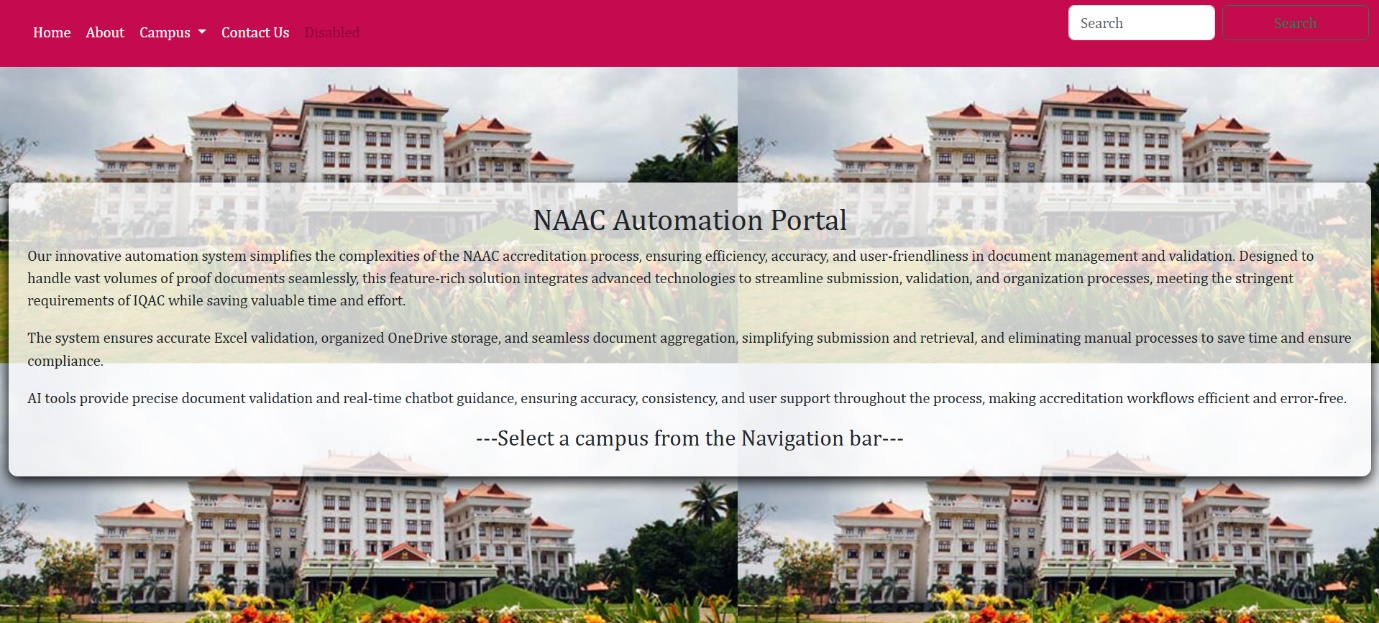


Fig 9. Homepage of NAAC Compliance System

Fig 9. is the NAAC Automation Portal homepage that is designed to make the process of NAAC accreditation simple, intuitive, and streamlined. Powered by Bootstrap, it includes a responsive navigation bar that offers links to Home, About, Contact Us, and Campus-specific pages. It has a dropdown menu where users can choose their campus for interaction with the system. It also provides an integrated search bar, which enables the user to quickly find relevant information. The clean and modern design ensures accessibility across devices, making it user-friendly for a wide range of users, regardless of their technical expertise.

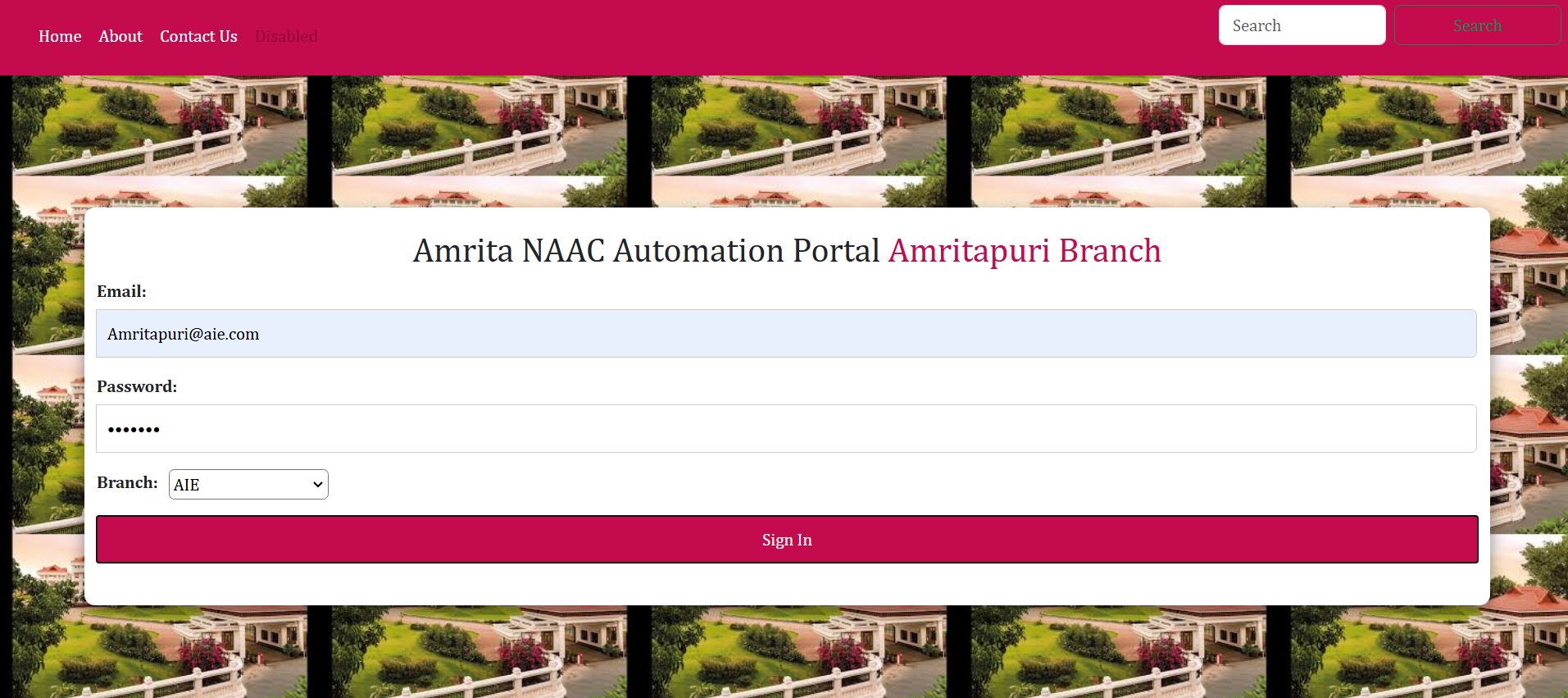


Fig 10. Sign In Page of one campus

The Sign-In Page of the NAAC Automation Portal as shown in Fig 10. provides a secure and user-friendly gateway for users to access the system's features. Built using HTML5 and Bootstrap, it ensures a responsive and accessible design across devices. Users are prompted to enter their email, password, and select their branch from a dropdown menu, which ensures authenticated and role-based access. The form includes built-in validation for accurate inputs, with JavaScript providing real-time feedback on errors. The consistent navigation bar at the top allows users to seamlessly explore other pages like Home, About, and Contact Us, ensuring a cohesive and intuitive user experience.



Fig 11. Campus Dashboard

Building on the process of user authentication, Fig 11. is the dashboard of Campus Portal allows structured input and uploading NAAC-related documentation. Upon logging in, users will easily enter the criteria numbers to upload the relevant Excel templates from the same page. The dashboard allows only a file upload system wherein the upload format can be only either.xlsx or.xls. All user inputs will also be validated in real-time. The uploaded file are displayed in a clean table with the criteria number, file name, and the time uploaded. This concise display makes it easy for persons uploading to monitor their submissions and hence get ready for the subsequent validating or aggregating processes.

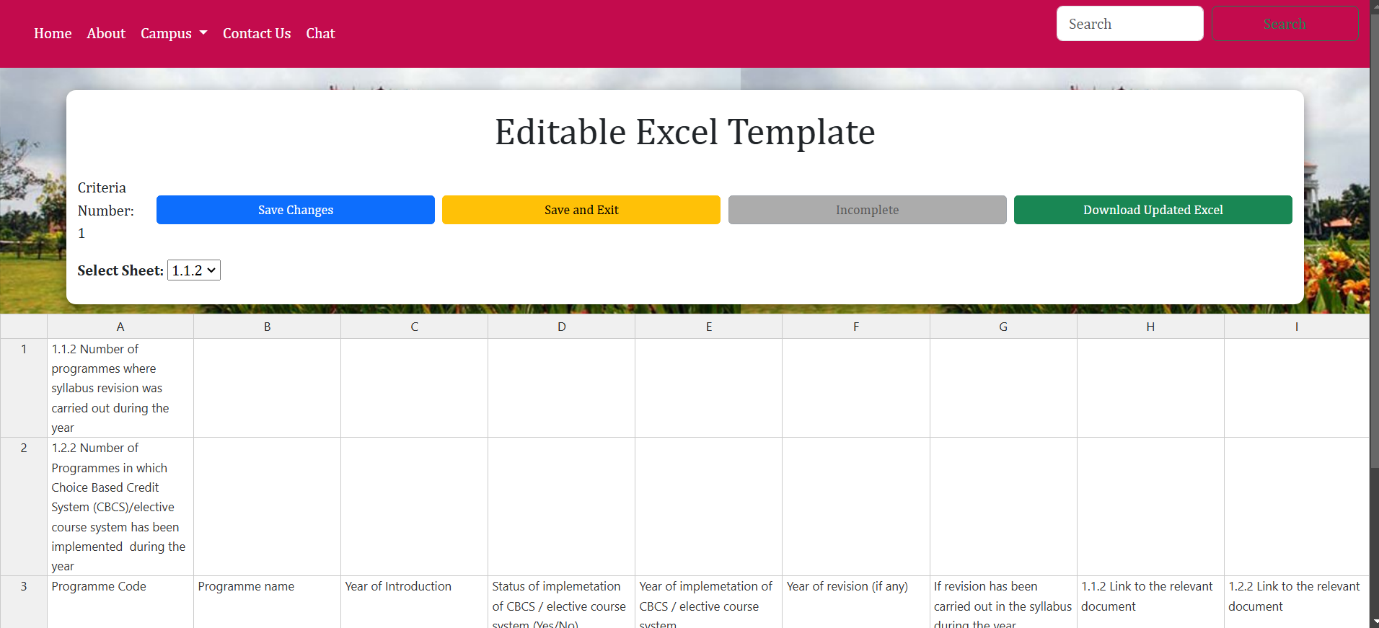


Fig 12. Template Editor

The Template Editor, Fig 12. provides a dynamic interface by which users can view, edit, and manage the uploaded Excel templates. On this page, functionalities such as rendering Excel sheets with Handsontable, saving changes, and downloading updated templates are integrated. Selecting sheets from a dropdown and editing data directly in the table are possible; the latter is dynamically loaded based on the content of the sheet. The features "Save Changes," "Submit," and "Download Updated Excel" buttons enable easy handling of data and real-time updates. JavaScript ensures that data validation is strong, allows navigation of multiple sheets, and changes are synchronized to avoid any inaccuracy. It also performs other critical functions like checking for incomplete rows, gathering document links, and submission readiness. All these features help improve data integrity and streamline the process for NAAC accreditation.

The NAAC Automation Portal offers a highly intuitive, efficient front-end system simplifying documents and validations toward compliance with requirements for NAAC accreditation. Key features include dynamic table functionality, responsive design, real-time feedback, and more recent integration with chatbot applications. The validation of data and file submission management, coupled with the provision of data integrity, create a great foundation for further exploration of the results from the backend, including the model and pipeline of RAG. Further, these backend functions, alongside AI-driven validation and pipelines for processing in advanced automation, would facilitate further enhanced automation of workflows that would be comprehensive and reliable document management in processes of accreditation.

**5.2 Chatbot results**

The NAAC Help Portal's online front as show in Fig generally has a friendly design, and carries a title banner at the top stating "NAAC Help Portal". The users shall be presented with dropdowns for selecting from existing NAAC criteria such as "1.1.1", "1.2.2", "Key Indicator 1.1 AU Minutes" and spells of query (for instance, "Chat with the system", "Summarise", "Template). Based on these settings, specific input fields will be added dynamically. For example:

* **File Upload:** For queries regarding documents, users will upload PDF files using an intuitive file uploader widget. At concatenation, two files are uploaded by the user and are hence given a merged file with further options to explore.
* **Text input box:** A simple input box gives users the power to start asking questions directly in real time. The answers will show up right below the input field.
* **Display of results:** Answers, comparisons, or summarizations would be displayed clearly within the interface, in detail and contextualized responses.

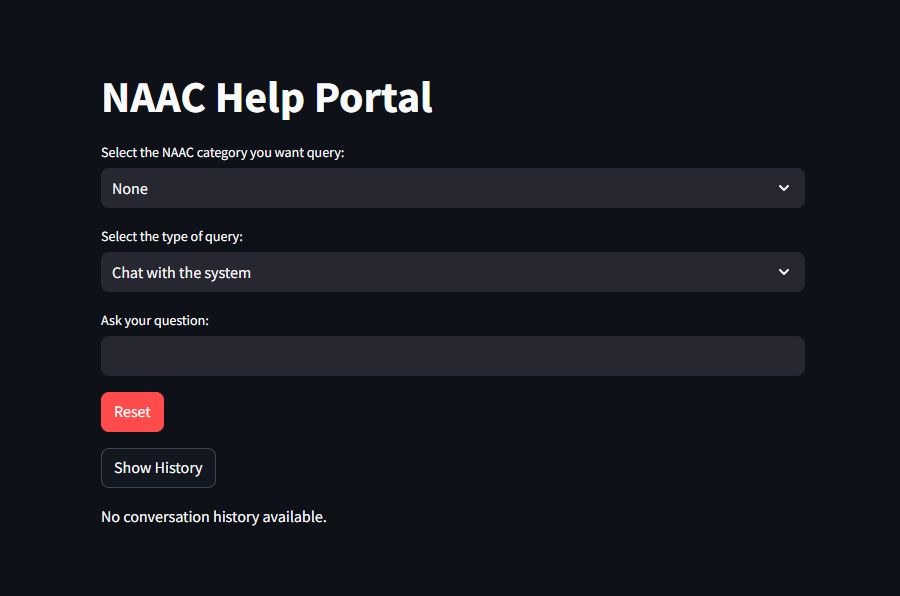
Additional features include a Reset button and a Show History button that enables users to clear their session or check subsequent interactions recorded in a neat conversation log. The layout is mostly simple and interactive to guarantee user experience.

Fig. 13The UI of the NAAC Help Portal connected to the front end

The individual results for each of the feature are as follows:

1. **The RAG chatbot or the chat with the system feature on the UI**

The final chatbot UI after integrating all the modules specified in the System Design section, looks like Fig. . The user can select one of the categories namely, 1.1.1 - Curricula developed and implemented, 1.1.2 - Percentage of Programmes, 1.1.3 - Average Percentage Of Courses, 1.2.1 - Percentage Of New Courses Introduced, 1.2.2 - CBCS or Elective, 1.3.1 - Integrates Crosscutting Issues, 1.3.2 - Number Of Value Added Courses, 1.3.3 - Students Enrolled In VAC, 1.3.4 - Students Undertaking Field Projects, 1.4.1 - Structured Feedback Received, 1.4.2 - Feedback Processes Of The Institution, Check list final for Criteria 1, Curriculum and Syllabi, Key indicator 1.1 AU minutes, NAAC Criteria 1 2022-2023 given in Jan 2023 and NAAC Criteria 1 Jul 2021 to Aug 2022 submitted, to chat with.

Once the user selects the category, the backend calls the respective vector database and is ready to be queried. In case the user selects None as the option, the chatbot only utilizes the Wikipedia tool and the google tool to give the user information regarding the query. This feature also works when the user selects a category, but the primal importance is given to the database and the other two tools.

The results for the test case for each of these tools are as follows:

1. **Database:**

The RAG model is able to effectively route to the correct data base and give context relevant details. In the Fig, it can be observed that the model is able to properly answer questions on the database. To check more intricate details, a test case is performed to check whether it is able to retrieve the name of the guide of a project in a database containing around 85 files

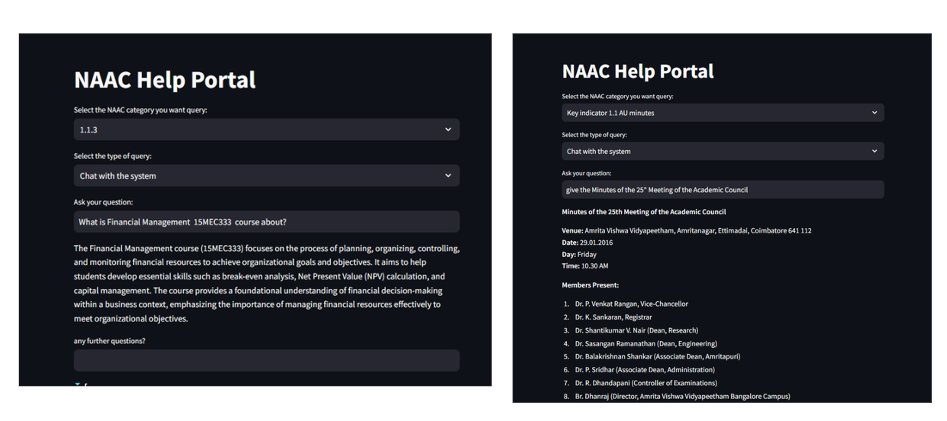


Fig. 14 Testcases querying the NAAC categories and 1.1.3 and Key Indicators respectively

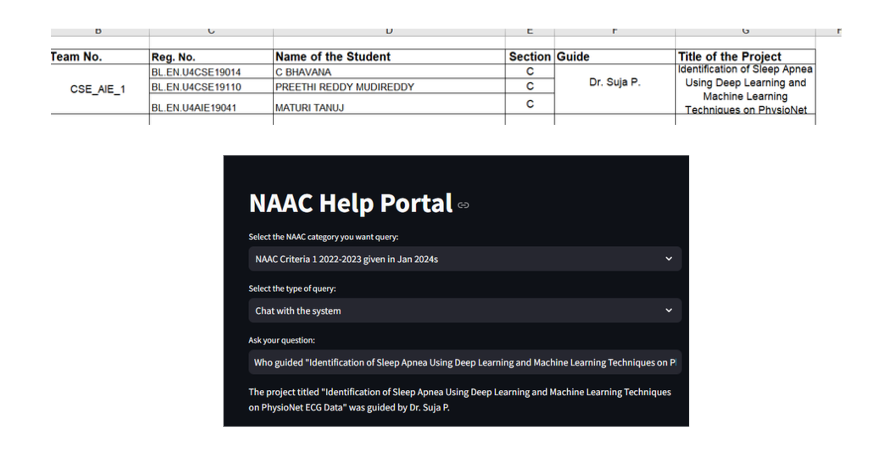
From these test cases in figures 14 and 15, we can infer that the performance of the model is good and storing each of the category as a separate vector database not only saves the computational time to embed, but also helps with the latency to retrieve the question and works better with respect to the contextual understanding.

Fig. 15 Test case checking the retrieval of intricate details in the database

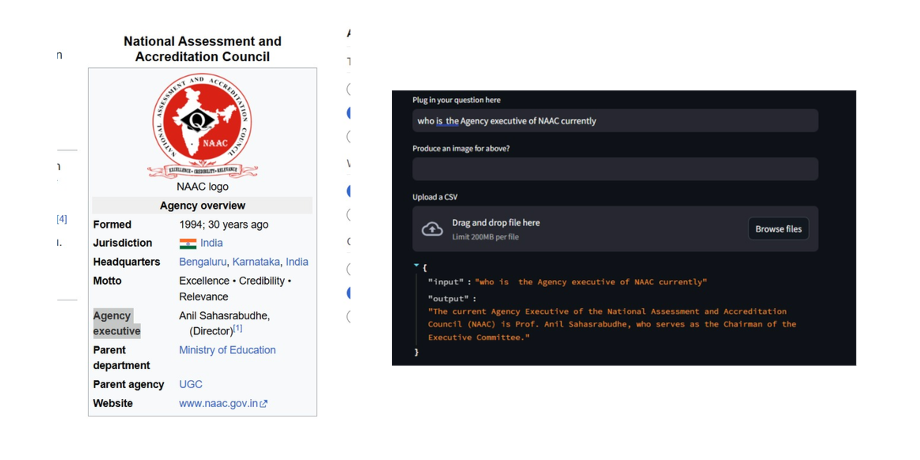


Fig. 16Testcase checking the usage of Wikipedia tool by querying NAAC Wikipedia page

1. **Wikipedia tool**

This is a test case that successfully demonstrates as shown in fig 16. how the tool brought with it RAG models on LangChain, which is combined with Wikipedia. The answer was retrieved accurately by querying the model with the question "Who is the Agency executive of NAAC currently?". Using the Wikipedia page for National Assessment and Accreditation Council (NAAC) in fetching the information proved successful, as it packaged it correctly, saying that Prof. Anil Sahasrabudhe is the Agency Executive and Chairman of the Executive Committee. This result proves that the model can retrieve relevant, correct information from Wikipedia, thus demonstrating its capability to answer queries driven by real-time data from external sources.

1. **Custom Google Search Engine**

This test evaluates the functionality of the RAG model integrated with a custom Google Search engine as a tool for fetching information. The query "NAAC seven criteria for assessment of higher education institutions" triggers extraction and presentation of information from the model. The left side of the figure shows the official NAAC website. The results quite show that the model has succeeded in retrieving an extensive and detailed explanation of NAAC seven criteria, such as 'Curricular Aspects,' 'Teaching-Learning & Evaluation,' and others, including purposes and significance. This test thus qualifies this model for using custom search tools to feasibly get structured and accurate data from other representatives like official websites and knowledge repositories. Therefore, we can say that the RAG model is well updated on all the changes that can potentially happen on a website like NAAC so that the user can work accordingly in sync.

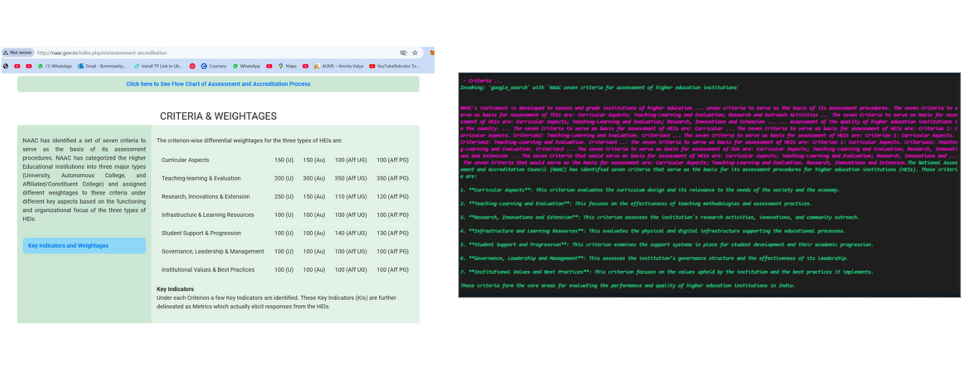
****In conclusion, the above test cases bring together the results that show the robustness and versatility of RAG when combined with different tools. Efficiency is also increased by directing queries to the appropriate vector database using this model, allowing the possibility for maximally minimal latency. With the Wikipedia piece of information, it has proved its fetch for real-time precision as in the case of identifying NAAC's Agency Executive. The use of a custom Google Search engine showcases its ability to extract and present complex structured information from official sources, the NAAC website among others. These tests then justify the model's effectiveness in providing reliable contextually relevant answers for a variety of cases, rendering it a powerful dynamic and detailed information retriever.

Fig. 17 Testcase checking the usage of Google Search engine tool by querying NAAC official website

**5.3Summarization Results**

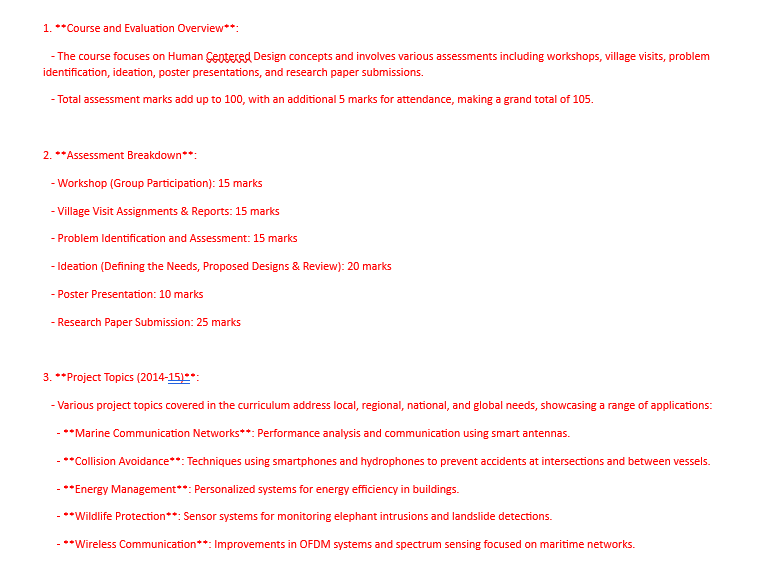


Fig.18 Summary of category “1.1.1” by the direct enquire on vector database

The approach known as Map-Reduce in LangChain provides a more all-encompassing and contextually intricate summary than a one-step process of summarization. The description is divided into two important phases: Map, in which smaller abstracts are created to summarize individual files or chunks, and Reduce, which merges the individual abstracts into a final summary while contextually integrating the entire summary. The model would then be able to focus its attention on small, manageable contexts in the mapping stage of things and synthesizing ideas effectively at the reduction stage. This both increases accuracy and assures no critical pieces are missed, which provides a much richer, deeper representation of the dataset. For example, as depicted in fig 18 the Map-Reduce summary of the NAAC framework brought to light several distinct yet interconnected components such as curriculum relevance, technological innovations, community engagement, and human-centered design, all aspects reflecting the holistic perspective of the system.

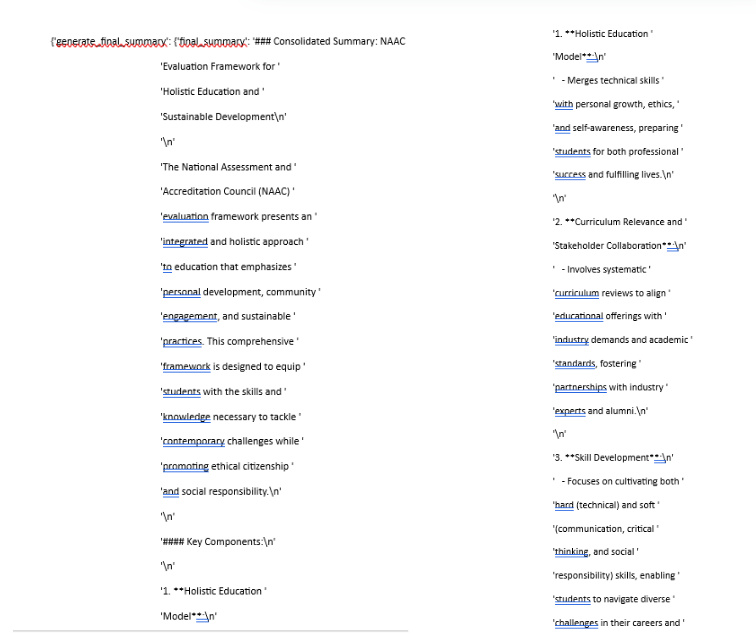


Fig.19 Snippet of the summary given by the Summarization module

The Map-Reduce method ensures however that whilst the entirety of the data could be effectively captured and interrelated, a single summary pass as in figure 19 will not succeed in this goal. As the prompt can be specified to specifically tailor the summary as a NAAC category, a proper summary containing all the features of the input 1.1.1 as in figure 18 category in the database is given by the summarisation model. Amalgamation of most pieces into summary can prove particularly useful when summarizing huge and complex datasets that are characterized by multi-criteria frameworks such as NAAC's, where numerous themes have to be brought together cohesively, such as for instance sustainability, cultural wellbeing, or technological advancements. Besides retaining the relevant contents of each individual component, the summary is also able to unite those components into a broad but coherent understanding, making it applicable to what is needed in terms of a high-level view without compromise on depth or specificity.

**5.4 Document Comparison and aggregation Results**

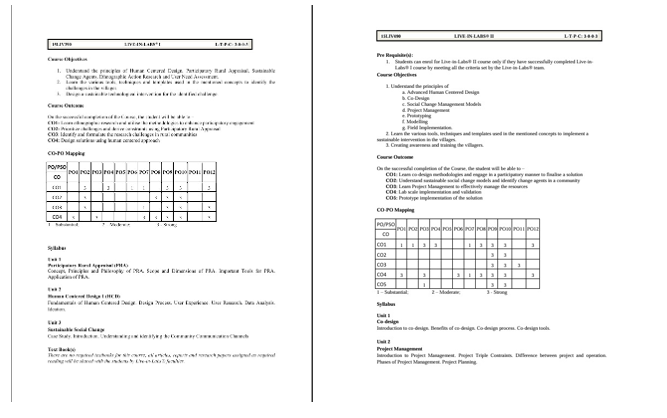


Fig. 20 Input to the comparison and aggregation module

Fig 20. Shows two documents, Live-In Labs 1 and Live-In Labs 2 that are the foundation data for making vector databases. These two documents have been processed and extracted for the relevant textual data such as course objectives and outcomes, CO-PO mappings, and syllabus details. By applying natural language processing technique, the extracted text got tokenized and embedded into some high-dimensional vector representations that allow semantic meaning to be captured. The vector embeddings were stored in a vector database, thus supporting efficient querying and similarity-based retrieval. By utilizing such a structure, the system allows for fast access to any section or related concepts from both documents, hence easy information retrieval for any form of academic or research need.

In the comparison and aggregation module, documents are used for summarization. Main areas of a course like its objectives, outcomes, and syllabus are summarized in the summarization process so that a brief yet meaningful representation of the two documents can be developed. Overlap, differences, and supplementary features between the two documents can be identified during the comparison process. Aggregation also involves gathering similar content in a summary so that common objectives or themes can be merged into a single summary. This streamlined representation enhances the accessibility and usability of information, ensuring that users can rapidly understand and analyze the core content of both documents for further processing or decision-making.



Fig.21 Choosing Query type



Fig.22 Comparitive analysis

Fig 22. Shows the comparison model analyzes the two documents in a structured way to produce an exhaustive report. The model makes use of advanced natural language processing techniques to process the details extracted from the documents, and the course comparisons are done according to structured parameters like focus areas, depth of outcomes, and evaluation methodologies. This automatic comparison shows the trend of moving from basic to advanced concepts and also brings out the differences and developments between the two courses.

The report output will be a side-by-side analysis of how the curricula evolve to build these capabilities in community engagement and design methodologies and practical applications. Using structured data inputs together with contextual analysis, such a model ensures that those unique features and overarching themes of each course are captured very well. The report will end, therefore, with insights along the educational trajectory into ways the courses collectively prepare people for real-world challenges and innovative practices.



Fig. 23 Aggregation Model Output

Collect information across different documents, analyze that and try structuring in a unified way; the given report is in some way one of the several system outputs processing input documents-such as course syllabus, objectives, and the outcome mapping-and consolidates the information into a concise format by extracting key features-in our case: objectives, and learning outcomes, and maps.

How Aggregation is Done:

* **Data Extraction**: The system will scan through uploaded documents for course objectives, learning outcomes, and CO-PO mappings.
* **Standardization**: It takes on varied formats and styles between different documents and brings about a structure to standardize the aggregated output in its uniformity.
* **Summarization**: Key points are summarized for redundancy purposes while maintaining the essence of the data.
* **Tabular Integration**: This integration is the proper and lucidly presented table relationships as, for example, the relationship between the CO-PO mapping.
* **Validation**: The report ensures accurate and complete data in relation to the problem statement so that the results can be generated.

**Advantages of This Aggregated Report**:

* Coordinated Data Presentation: It thus simplifies this report while presenting complex data to ensure that stakeholders interpret the reports.
* It saves time and effort by doing away with the need for a manual comparison and collation of data from various sources.
* This structured format helps educators and administrators make informed decisions regarding curriculum improvement, accreditation processes, or resource allocation.
* The report gives a clear view of course objectives and outcomes to ensure alignment with institutional goals and standards.

The aggregation module as in fig 23 provides an efficient way of managing and consolidating institutional data, thus greatly reducing the complexities of handling multiple datasets. Through the automation of data collection, validation, and organization, this module decreases reliance on manual intervention, which could lead to errors and inconsistency across reports. This tool has proven to be very essential in institutions that want to achieve high accreditation or compliance standards. The reports will thus be structured and intuitive to the stakeholders, giving a clear overview of critical metrics, hence actionable.

This module streamlines administrative workflows but helps make better decisions by having information that is accurate and consolidated. This aggregation module enables the institution to observe development and monitor improvement and help institutional goals align with strategic goals. More importantly, it handles very large data from a multitude of sources that help scale it and can thereby adapt to changing needs over time. This module brings meaning out of complicated data by converting it into practical and actionable insights that provide clear transparencies, ensure accountability and promote continuous improvement.

**5.5 Missing Information generation Results**

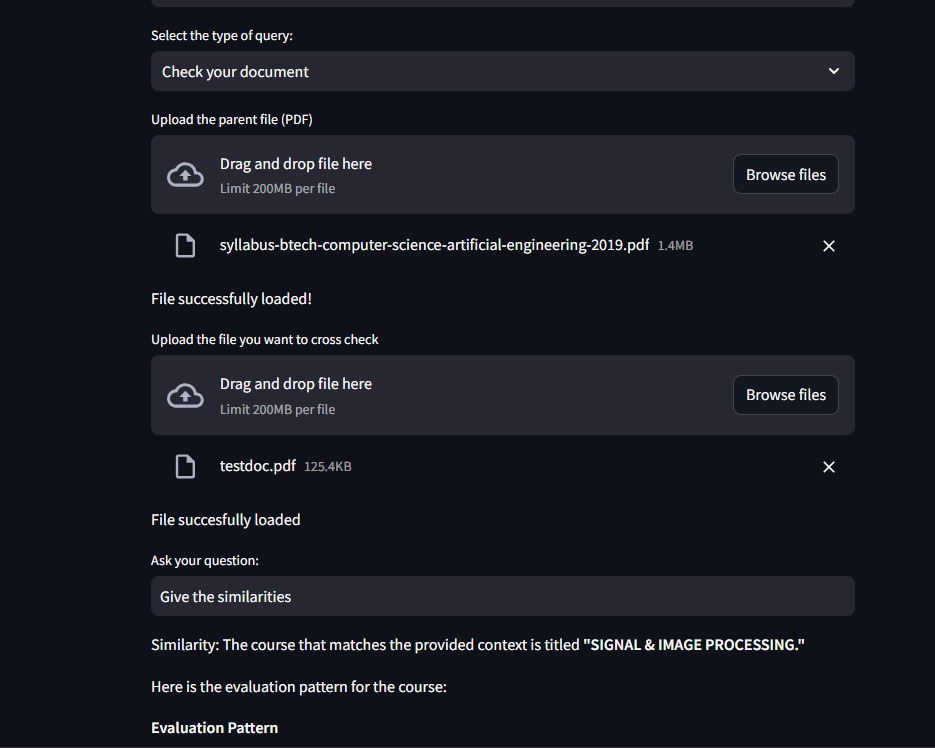
****

Fig. 24 Missing Information Module Input

The missing information module as depicted in fig 24. uses sophisticated vector-based comparisons and LLM-driven reasoning to identify gaps, cross-check gaps in the data extracted from academic documents, and fill them. Storing the document's data as vector embeddings on a vector database like Pinecone enables comparison of the uploaded document against the existing database and points to areas of similarity, alignment, and discrepancies.

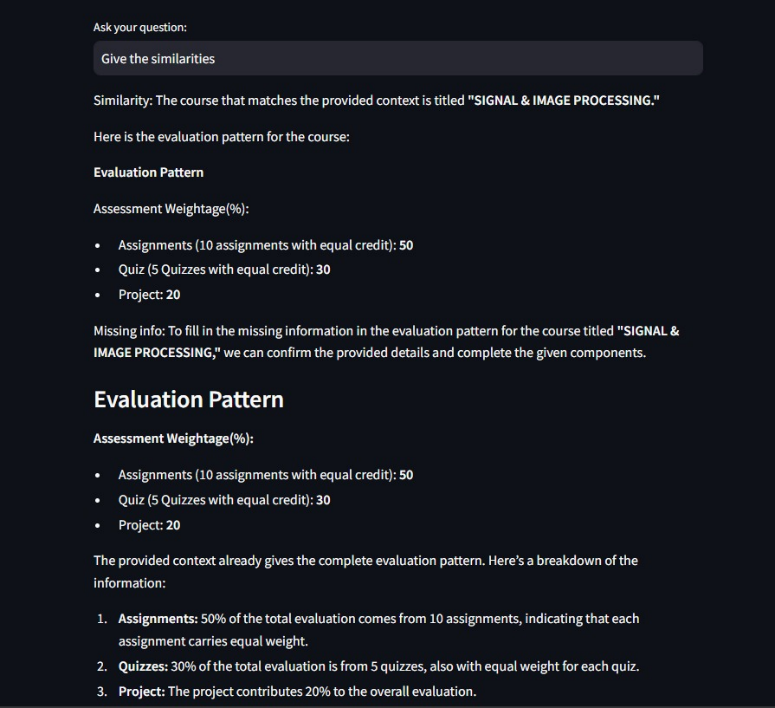


Fig. 25 Similarity Report

Fig 25. Shows how efficiently it identifies the similarities between the parent document and the uploaded one by using sophisticated vector-based text embeddings. By uploading the parent document and a secondary file for comparison, the system processes its content to produce semantic embeddings, which encapsulate the meaning of the text in context so that the system can compare them for similarity in terms of structure, content, and details.

The similarity detection will indicate sections where both documents have content overlap, like evaluation patterns, and also mark sections that deviate or need further clarification. The system will provide a concise yet comprehensive report of the comparison of data points such as evaluation components, weightage distributions, or structural outlines, bridging gaps and confirming alignment between the two documents. This would enable users to validate the accuracy of content across different documents with minimal efforts, increase precision, and save time in processes that take value from document validation or integration.

The missing information module exemplifies the true capabilities of advanced AI-driven solutions that could be used in automating document validation processes. Vector-based embeddings and semantic analysis ensure accurate and effective alignments of documents, such as educational frameworks, patterns in evaluations, or any structured content. The module identifies, quite seamlessly and automatically, all congruencies and discrepancies, thereby reducing the burden of manual effort and maximizing accuracy and reliability. This makes it invaluable for institutions or organizations dealing with large-scale document management, offering a robust mechanism to validate, compare, and consolidate information. In essence, this module stands as a critical innovation, enabling consistency and efficiency while fostering a streamlined workflow in document-related tasks.

**CHAPTER-6**

**CONCLUSION & FUTURE SCOPE**

The NAAC Automation Portal represents a great leap in the process of accrediting institutions as it makes the process smooth, easier, and user-friendly in terms of accessibility for naive institutions in this process. The whole system combines a responsive and user-friendly front-end design with strong AI-driven backend functionality and delivers a complete solution in managing accreditation workflows. The front end has been developed considering usability and responsiveness, and it will provide features like dynamic dashboards, real-time validations, secured logins, and advanced template editors, ensuring that there is no experience for users when uploading, managing, and validating documents concerning the NAAC. The backend features advanced AI models and tools for example, Retrieval-Augmented-Generation (RAG), Map-Reduce for summarization, etc. This makes the system much more efficient. The chatbot module will handle user's queries in real-time and retrieve data from vector databases, Wikipedia, and Google Search to provide precise, contextually relevant answers to make information relatively accessible. The strength of this system, however, is in comparison and aggregation, as it lets institutions accumulate big sets of data into manageable insights for better decisions and alignment with institutional strategies. Missing information module helps for pinpointing and clearing away gaps in documentation to ensure the integrity and completeness of the submitted data. This feature has become highly important to ensure the credibility of the whole accreditation process.

As a future scope, NAAC Automation Portal opens the possibility of turning into a more intelligent and versatile system. Currently the system only works on the criteria 1 of NAAC, it can be potentially expanded to all the criteria and can be securely hosted on cloud. The system likely could offer AI recommendations for accreditation score improvement, predictive analytics to trace trends, and safer data with the help of blockchain technology. Therefore this project gives a lot of scope for a real time application with huge benefit.

**REFERENCES**

1. Ravikumar, K., Sasmitarani Samanta, Amiya Kumar Rath, and S. Srinivasaragavan. "Quality assessment of Indian universities: An analytical study of NAAC accreditation scores." Psychology and education journal 58, no. 2 (2021): 5903-5914.
2. Aithal, P.S., Shailashree, V. and Kumar, P.M., 2016. Analysis of NAAC Accreditation System using ABCD framework. International Journal of Management, IT and Engineering, 6(1), pp.30-44.
3. Gebreab, S.A., Salah, K., Jayaraman, R., ur Rehman, M.H. and Ellaham, S., 2024, April. LLM-Based Framework for Administrative Task Automation in Healthcare. In 2024 12th International Symposium on Digital Forensics and Security (ISDFS) (pp. 1-7). IEEE.
4. Ethape, P., Kane, R., Gadekar, G. and Chimane, S., 2023. Smart automation using llm. International Research Journal of Innovations in Engineering and Technology, 7(11), p.603.
5. Song, H., Su, H., Shalyminov, I., Cai, J. and Mansour, S., 2024. FineSurE: Fine-grained Summarization Evaluation using LLMs. arXiv preprint arXiv:2407.00908.
6. Wadagave, Rashmee, Srushti Karoshi, Pranali Ravan, Rutuja Santikar, and Rashmi Deshmukh. "Web Application based Event Organisation Portal using MEAN Stack." In 2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS), pp. 1427-1430. IEEE, 2022.
7. Mishra, Ananya, Varsha Singh, and Jagpreet Singh Sidhu. "Medweb-a Full Stack Telemedicine Application Using Cloud Computing." (2023).
8. Xu, Kaiqiang, Xinchen Wan, Hao Wang, Zhenghang Ren, Xudong Liao, Decang Sun, Chaoliang Zeng, and Kai Chen. "Tacc: A full-stack cloud computing infrastructure for machine learning tasks." arXiv preprint arXiv:2110.01556 (2021).
9. Phan, H., Acharya, A., Chaturvedi, S., Sharma, S., Parker, M., Nally, D., Jannesari, A., Pazdernik, K., Halappanavar, M., Munikoti, S. and Horawalavithana, S., 2024. RAG vs. Long Context: Examining Frontier Large Language Models for Environmental Review Document Comprehension. arXiv preprint arXiv:2407.07321.
10. Joshi, Pankaj, Aditya Gupta, Pankaj Kumar, and Manas Sisodia. "Robust Multi Model RAG Pipeline For Documents Containing Text, Table & Images." In 2024 3rd International Conference on Applied Artificial Intelligence and Computing (ICAAIC), pp. 993-999. IEEE, 2024.
11. Laban, Philippe, Alexander R. Fabbri, Caiming Xiong, and Chien-Sheng Wu. "Summary of a haystack: A challenge to long-context llms and rag systems." arXiv preprint arXiv:2407.01370 (2024).
12. Topsakal, O., & Akinci, T. C. (2023, July 10-12). *Creating large language model applications utilizing LangChain: A primer on developing LLM apps fast*. 5th International Conference on Applied Engineering and Natural Sciences, Konya, Turkey. Published by All Sciences Proceedings.
13. Gao, Yunfan, et al. "Retrieval-augmented generation for large language models: A survey." arXiv preprint arXiv:2312.10997 (2023).
14. Minaee, Shervin, et al. "Large language models: A survey." arXiv preprint arXiv:2402.06196 (2024).
15. Jiang, Zhengbao, et al. "Active retrieval augmented generation." arXiv preprint arXiv:2305.06983 (2023).
16. Hoshi, Yasuto, et al. "Ralle: A framework for developing and evaluating retrieval-augmented large language models." arXiv preprint arXiv:2308.10633 (2023).
17. Khattab, Omar, et al. "Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp." arXiv preprint arXiv:2212.14024 (2022).
18. Karpukhin, Vladimir, et al. "Dense passage retrieval for open-domain question answering." arXiv preprint arXiv:2004.04906 (2020).
19. Kim, Gangwoo, et al. "Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models." arXiv preprint arXiv:2310.14696 (2023).
20. Izacard, Gautier, et al. "Atlas: Few-shot learning with retrieval augmented language models." Journal of Machine Learning Research 24.251 (2023): 1-43.
21. Das, Devleena, and Vivek Khetan. "DEFT: Data Efficient Fine-Tuning for Large Language Models via Unsupervised Core-Set Selection." arXiv preprint arXiv:2310.16776 (2023).
22. Long, Xinwei, et al. "Generative multi-modal knowledge retrieval with large language models." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 38. No. 17. 2024.
23. Wang, Xintao, et al. "Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases." arXiv preprint arXiv:2308.11761 (2023).
24. Xie, Xingrui, et al. "A Brief Survey of Vector Databases." 2023 9th International Conference on Big Data and Information Analytics (BigDIA). IEEE, 2023.