Incorporating LLM-based Interactive Learning Environments in CS Education: Learning Data Structures and Algorithms using the Gurukul platform

Ashwin Kedari Rachha

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science and Applications

Mohammed S. Seyam, Chair

Clifford A. Shaffer

Dwayne C. Brown

May 7th, 2024

Blacksburg, Virginia

Keywords: Large Language Models, Retrieval Augmented Generation, Guardrails,

Computer Science Education, Adaptive Learning Technologies

Incorporating LLM-based Interactive Learning Environments in CS Education: Learning Data Structures and Algorithms using the Gurukul platform

Ashwin Kedari Rachha

ABSTRACT

Large Language Models (LLMs) have emerged as a revolutionary force in Computer Science Education, offering unprecedented opportunities to facilitate learning and comprehension. Their application in the classroom, however, is not without challenges. LLMs are prone to hallucination and contextual inaccuracies. Furthermore, they risk exposing learning processes to cheating illicit practices and providing explicit solutions that impede the development of critical thinking skills in students. To address these pitfalls and investigate how specialized LLMs can enhance engagement among learners particularly using LLMs, we present Gurukul, a unique coding platform incorporating dual features - Retrieval Augmented Generation and Guardrails. Gurukul's practice feature provides a hands-on code editor to solve DSA problems with the help of a dynamically Guardrailed LLM to prevent explicit code solutions. On the other hand, Gurukul's Study feature incorporates a Retrieval Augmented Generation mechanism that uses OpenDSA as its source of truth, allowing the LLM to fetch and present information accurately and relevantly, thereby trying to overcome the issue of inaccuracies. We present these features to evaluate the user perceptions of LLM-assisted educational tools. To evaluate the effectiveness and utility of Gurukul in a real-world educational setting, we conducted a User Study and a User Expert Review with students (n=40) and faculty (n=2), respectively, from a public state university in the US specializing in DSA courses. We examine student's usage patterns and perceptions of the tool and

report reflections from instructors and a series of recommendations for classroom use. Our findings suggest that Gurukul had a positive impact on student learning and engagement in learning DSA. This feedback analyzed through qualitative and quantitative methods indicates the promise of the utility of specialized LLMs in enhancing student engagement in DSA learning.

Incorporating LLM-based Interactive Learning Environments in CS Education: Learning Data Structures and Algorithms using the Gurukul platform

Ashwin Kedari Rachha

GENERAL AUDIENCE ABSTRACT

Computer science education is continuously evolving with new technologies enhancing the learning experience. This thesis introduces Gurukul, an innovative platform designed to transform the way students learn Data Structures and Algorithms (DSA). Gurukul integrates large language models (LLMs) with advanced features like Retrieval Augmented Generation (RAG) and Guardrails to create an interactive and adaptive learning environment.

Traditional learning methods often struggle with providing accurate information and engaging students actively. Gurukul addresses these issues by offering a live code editor for hands-on practice and a study feature that retrieves accurate information from trusted sources. The platform ensures students receive context-sensitive guidance without bypassing critical thinking skills.

A study involving students and faculty from a public university specializing in DSA courses evaluated Gurukul's effectiveness. The feedback, based on qualitative and quantitative evaluations, highlights the platform's potential to enhance student engagement and learning outcomes in computer science education. This research contributes to the ongoing development of educational technologies and provides insights for future improvements.

Dedicated to Virginia Tech.

Acknowledgments

I would like to acknowledge the guidance and support of my advisor, Dr. Mohammed Seyam, in my graduate studies and completing my thesis for his invaluable patience and feedback.

I would also like to thank my committee members, Dr. Clifford Shaffer and Dr. Chris Brown, for their support and feedback on my thesis defense.

Special thanks to Dr. Mohammed Farghally and Patrick Sullivan for providing their invaluable insights on Gurukul.

Heartfelt appreciation goes to my family members - Anaya and Anvika my beautiful nieces and Dr Swapnil, Chanda, Kedari and Sneha Rachha.

I want to thank Sage Holden and all my friends for their support.

Lastly, I am deeply indebted to the Virginia Tech CS department for their generous support, without which this endeavor would not have been possible.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
CS	Computer Science
CSed	Computer Science Education
DSA	Data Structures and Algorithms
GAI	Generative Artificial Intelligence
LLMs	Large Language Models
NLP	Natural Language Processing
PBL	Problem Based Learning
RAG	Retrieval Augmented Generation
SEO	Search Engine Optimization
SSG	Static Site Generation
SSR	Server Side Rendering
ZPD	Zone of Proximal Development

Chapter 1

Introduction

1.1 DSA Education

Data Structures and Algorithms (DSA) are the cornerstone of computer science education, equipping students with essential skills for solving complex real-world problems efficiently. Mastery of DSA is crucial for academic success and professional competence in software development and computational problem-solving. However, the abstract nature of DSA concepts poses significant challenges for both teaching and learning. Traditional educational methods, which often rely heavily on lecture-based instruction and static textbooks, tend to be less effective in fostering a deep understanding of these complex topics. Students frequently struggle to link theoretical principles with practical applications, leading to a gap in their comprehension and retention of material.

1.2 Challenges in Traditional Education

1.2.1 Abstract Nature of DSA Concepts

The inherent complexity and abstract nature of DSA concepts make them difficult for students to grasp without tangible examples and practical applications. Traditional methods, while comprehensive, often fall short in addressing the dynamic inquiries that naturally arise

in students' minds as they delve into DSA topics. Without immediate clarification tools and hands-on practice, retention and application of these concepts can be elusive [2].

1.2.2 One Size Fits All Approach

The one-size-fits-all approach is a significant drawback of traditional education in DSA learning. In a typical classroom, the pace and complexity of instruction are generally uniform, failing to account for the diverse learning speeds and styles of students. Fast learners may find the material repetitive and slow, while others may struggle to keep up, leading to a disparity in learning outcomes and overall dissatisfaction among students. Moreover, traditional assessment methods, such as standardized tests and assignments, often do not accurately reflect a student's practical ability to implement DSA in real-world scenarios [3].

1.2.3 Reduced Levels of Practical Engagement

Traditional pedagogy, with its unidimensional focus, does not adequately cater to modern learners' needs for interactivity and feedback. Aspects of DSA, such as algorithm efficiency and the choice of appropriate data structures, are best understood through trial and error, experimentation, and iterative learning—elements that conventional classroom settings are ill-equipped to facilitate. The delayed feedback loop, where students must wait for graded assignments or exams to gauge their understanding, impedes the correction of misconceptions and hinders the iterative learning process. Immediate feedback is essential for reinforcing learning, encouraging experimentation, and fostering a deeper understanding of complex concepts through trial and error [**?**].

1.2.4 Gaming the System

While Large Language Model (LLM) tools have brought various resources to enhance learning, they have also introduced challenges that can potentially undermine the educational process. One such challenge is "gaming the system" where students exploit LLM tools and systems to achieve high scores or complete assignments without genuinely understanding the material [4, 5]. This approach to learning raises concerns about the depth of learning and the development of critical thinking skills. While these AI systems can provide immediate answers and solutions, reliance on them can deter students from engaging deeply with DSA concepts, hindering their ability to apply these principles in novel or complex situations. The crux of mastering Data Structures and Algorithms lies in finding solutions and navigating the thought process that leads to these solutions, which includes understanding the problem, conceptualizing algorithms, and choosing suitable data structures based on efficiency and applicability [6].

1.3 Emergence of Large Language Models (LLMs) in Education

The advent of LLMs, such as ChatGPT [7], Claude [8], and Llama [9], has transformed the educational landscape. These advanced AI tools offer the potential to enhance the learning experience by providing personalized, context-aware assistance that traditional educational methods often lack. In computer science education, LLMs can help bridge the gap between theoretical concepts and practical applications by providing interactive coding assistance, troubleshooting, and instant clarification of doubts. However, the application of LLMs is not without challenges. Two primary issues inherent to LLMs are their propensity for

"hallucinations"—generating plausible but incorrect or irrelevant content—and contextual inaccuracies. These issues can mislead learners, especially in complex subjects like DSA, where precision is critical. Additionally, LLMs can be easily manipulated to provide direct answers, bypassing the critical thinking and problem-solving processes essential in education.

1.4 Addressing Challenges with Specialized LLM-Based Educational Tools

In response to these challenges, we present the Gurukul platform, designed to explore patterns and concepts for creating specialized LLM-based applications that enhance engagement and learning experiences in DSA courses. Gurukul's theoretical foundation is built on the premise that modern educational tools should provide interactive, personalized learning experiences that actively engage students and adapt to their individual needs. The platform introduces two key features of LLM enhancement - the Study section and the practice section which incorporates Retrieval Augmented Generation (RAG) and Guardrails within its LLMs respectively. The platform is designed with the integration in mind that the theoretical knowledge gained in the study section is immediately reinforced through practical application in the practice section. This unified approach addresses the common challenge of disconnect between learning and application, fostering a deeper understanding of DSA concepts. This approach is meant to encourage the development of critical thinking skills rather than complete reliance on LLMs, fostering a more balanced and effective learning environment.

1.5 Research Questions

This study is guided by two principal research questions:

- What are the user perceptions towards using Gurukul as an educational tool, integrated with specialized LLMs regarding learning comprehension and engagement in DSA courses?
- How do Guardrails and Retrieval Augmented Generation work towards enhancing engagement and learning productivity, and what could be the design principles to consider while employing such applications?

By addressing these research questions, this study aims to provide insights into the effectiveness and utility of LLM-based educational tools in enhancing student learning and engagement in DSA courses.

Chapter 2

Review of Literature

The utility of large language models like ChatGPT and GPT-4 [7, 10] has opened up multiple avenues for educational use. In Computer Science Education (CSEd), recent studies have demonstrated the potential of these models to improve learning and teaching experiences in university-level programming courses [11, 12, 13]. Studies on the utilization of Artificial Intelligence have been an area of active research for many years. However, the investigation of Large Language Models in educational settings has emerged more recently [14, 15]. As we examine the technological landscape in education, particularly within the confines of Computer Science and DSA, it is imperative to critically analyze existing digital tools and educational platforms that cater to these specialized subjects. Through such an evaluation of the existing rich literature, one may discern the limitations inherent within these technologies and speculate on the subsequent evolutionary steps necessary to refine and evaluate the educational experiences they provide.

2.1 AI in Education

Historically, AI in Education has seen a proliferation of interest, particularly with the advent of Intelligent Tutoring Systems and Chatbots well-versed in human language [16]. As shown in Figure 2.1, the Google Trends data for the term 'AI in Education' demonstrates a significant increase in interest from 2004 to 2024. AI has progressively become a central

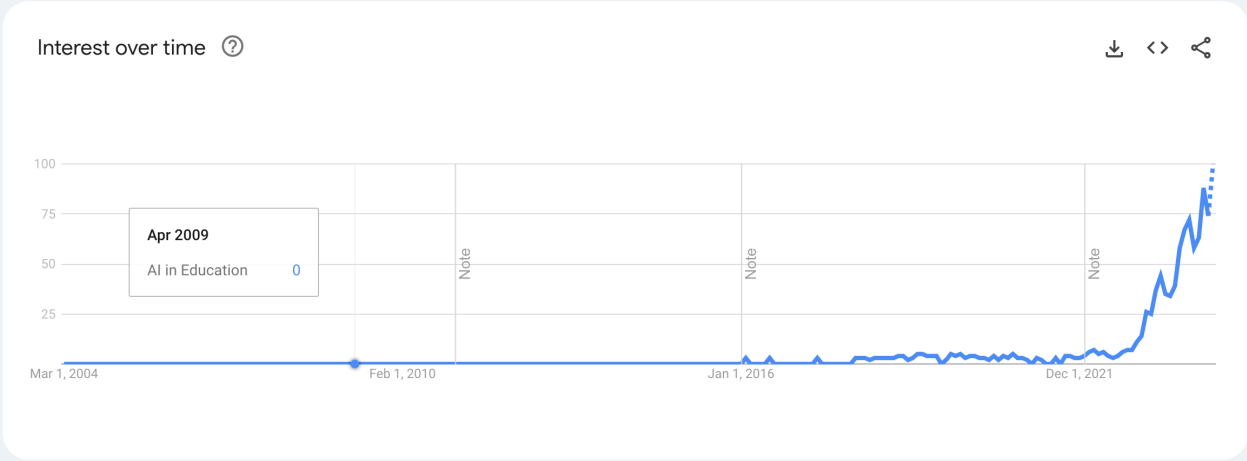

Figure 2.1: Google Trends results for the term 'AI in Education' from the year 2004-2024 [1]

player in transforming educational paradigms, offering unprecedented personalization and adaptability across diverse learning environments [17]. The inception of AI in education traces back to intelligent tutoring systems and computer-assisted instruction, which aimed to mimic one-on-one instruction provided by human tutors [18, 19, 20]. Over the years, the scope and sophistication of AI applications have broadened, encompassing everything from administrative automation to personalized learning experiences and intelligent content delivery [21, 22]. Recent studies [23] have demonstrated significant improvements in student engagement and personalized learning outcomes through AI-driven analytics.

A seminal paper by Lievertz (2019) illustrates AI's journey in education, emphasizing its role in revolutionizing traditional structures through enhanced flexibility and customization [24]. This narrative is further explored by Chen et al. (2020), who detail the multi-faceted impact of AI in administrative, instructional, and learning processes within educational institutions [25]. Their study underscores how AI technologies have evolved from basic computer-assisted technologies to sophisticated systems capable of executing complex educational tasks such as grading and personalized content delivery.

The narrative of AI's evolution is enriched by more detailed studies, such as that by Hamal

et al., who discuss the intersection of AI with modern educational technologies against the backdrop of contemporary challenges such as mobile learning, augmented reality, and educational games [26]. They provide a comprehensive look at the diverse applications of AI that span from enhancing personalized learning to fostering collaborative environments through social media platforms. Moreover, the integration of AI in education is not without challenges and considerations, which are thoroughly examined by Vasiliev and Eremeeva (2023). They critically assess both the advantages and potential drawbacks of AI, such as privacy concerns and the risk of dependency on technological solutions [27]. Their insights call for a balanced approach to integrating AI in educational settings, ensuring that these technologies supplement rather than supplant traditional educational methods. Holmes et al. (2022) all deal with the ethics of AI in education and counter the challenges of ensuring AI fairness in diverse educational settings [28]. These insights are critical for the Gurukul app, which leverages AI to enhance educational interactions while safeguarding ethical standards.

Traditional approaches to teaching Computer Science (CS) face numerous challenges that can hinder the effectiveness of learning, especially in foundational courses like Data Structures and Algorithms (DSA). These challenges primarily arise from the abstract nature of the content, the diversity of student backgrounds, and the conventional methods employed in teaching these complex subjects.

Traditional CS education often relies heavily on lecture-based delivery, where theoretical concepts are presented in a passive learning format. This method can be particularly challenging in CS due to the abstract nature of the content, which requires practical application to fully grasp. Studies such as those by Yadav et al. (2016) and Knobelsdorf and Vahrenhold (2013) highlight the isolation and lack of adequate CS background among teachers as significant barriers, impacting their ability to effectively convey complex CS concepts to students [2].

The specific challenges in traditional CS education include a lack of engagement, difficulty in understanding abstract concepts, and a one-size-fits-all approach that does not cater to individual learning paces or styles. Sentance and Csizmadia (2017) discuss how teachers face intrinsic and extrinsic challenges, including limited professional development and difficulties in adapting teaching strategies to meet diverse student needs in computing education [3].

Moreover, the traditional curriculum often lacks practical application, which is crucial for understanding and retaining complex CS concepts. This gap between theoretical knowledge and practical skills is further elaborated by Webb et al. (2017), who argue for a curriculum that integrates real-world applications to better prepare students for professional challenges in CS [29].

To address these educational challenges, innovative approaches such as Problem-Based Learning (PBL) have been introduced. PBL shifts the focus from passive reception of information to active problem-solving, helping students develop practical skills and a deeper understanding of the material. Kay et al. (2000) describe their implementation of PBL in foundation CS courses, highlighting its effectiveness in improving student engagement and learning outcomes [30].

Traditional approaches to data structures and algorithms (DSA) education, in particular, often rely heavily on theoretical lectures, which may not effectively address the learning needs of all students, especially given the abstract nature of many DSA concepts. This section discusses the prevalent teaching methods for DSA and their inherent limitations, as well as specific challenges that hinder effective learning.

Traditional DSA education frequently employs lecture-based teaching, where theoretical concepts are explained without sufficient practical application. This method can lead to a lack of engagement and poor retention of complex information. According to Tamassia (1998), the

challenge is not only in conveying abstract concepts effectively but also in developing correct and efficient implementations that students can understand and replicate [31]. Furthermore, Steingartner et al. (2019) note that traditional methods often fail to account for the wide content scope and variability in student backgrounds, which can lead to inconsistencies in learning outcomes across different institutions [32].

One of the primary challenges in traditional DSA education is the difficulty students face in visualizing and understanding how data structures and algorithms work under the hood. Su et al. (2021) discuss how students find DSA concepts challenging partly due to their abstract nature and also because of the traditional teaching methods that do not foster motivation or deep understanding [33].

Additionally, traditional approaches often do not provide adequate tools for students to experiment and see the immediate results of manipulating data structures or modifying algorithms. This lack of interactive learning tools can prevent students from fully grasping the implications of different design choices and understanding the efficiency of various algorithms. Tang et al. (2012) emphasize the importance of incorporating experimental teaching and using campus networks to enhance student engagement and learning efficiency in DSA courses [34].

2.2 Applications of Large Language Models in CS Education

Large Language Models (LLMs) have emerged as transformative tools within computer science education, offering novel approaches to both teaching and learning. These models are being increasingly utilized to enhance educational outcomes through the generation of

interactive learning materials and the provision of personalized assistance.

LLMs, such as GPT-3 and Codex, have been instrumental in reshaping the pedagogical landscape of computer science by facilitating the automatic generation of programming content, quizzes, and even entire courses tailored to individual learning needs. The ability of these models to process and generate human-like text has opened up new possibilities for educators to create dynamic and responsive learning environments. For instance, the work of Macneil et al. (2022) highlights the potential of LLMs to generate both code and corresponding explanations, significantly aiding in the learning process by allowing students to interact with complex code in more intuitive ways [35].

One compelling application of LLMs in CS education is their use in creating personalized learning experiences that adapt to the pace and style of individual students. In a case study presented by Krüger and Gref (2023), LLMs were utilized within a computer science degree program to simulate real-world programming tasks. The models not only facilitated learning but also engaged students through interactive problem-solving scenarios. The study concluded that LLMs could significantly enhance the learning process by providing a hands-on approach that traditional methodologies often lack [36].

Another example is the integration of LLMs to support faculty in developing curriculum materials. Tran et al. (2023) explored how LLMs could assist in the design of culturally responsive projects for K-8 computer science education, demonstrating that these models can significantly reduce the workload on educators while enhancing the cultural relevance of the curriculum [37].

While Tran et al. worked on incorporating curriculum materials in a CS course, Macneil et al. (2023) reported on their experiences generating MCQ explanation types using LLMs and integrating them in an interactive e-book on web software development. Results showed that

the three levels of explanations based on their verbosity granularity were viewed by students as helpful to them [38].

Glynn et al., in their seminal work CAET: Code Analysis and Education Tutor [39], propose an integrated system that utilizes Generative Artificial Intelligence (GAI) to provide personalized assistance in early computer science education focusing on programming tasks and maintaining the integrity of students' learning.

While Large Language Models (LLMs) such as GPT and BERT have revolutionized several aspects of artificial intelligence and have found increasing utility in educational settings, they come with a set of inherent drawbacks that need careful consideration.

LLMs are computationally expensive to train and operate, requiring substantial hardware and energy resources, which can limit their accessibility and scalability, particularly in resource-constrained environments. Additionally, these models often require extensive data to train, raising concerns about data privacy and the potential for training on biased data sets. This can lead to models that perpetuate or even amplify existing biases, as discussed in studies such as those by Meyer et al. (2023), who emphasize the ethical concerns surrounding the deployment of LLMs [13].

In educational contexts, the use of LLMs raises several specific challenges. One major concern is the accuracy and reliability of the content generated by LLMs. These models, while proficient in generating human-like text, can produce outputs that are factually incorrect or misleading, potentially confusing learners or leading them astray. The editorial by Hamaniuk (2021) highlights the importance of careful oversight when using LLMs as educational tools to ensure that the information they provide is accurate and contextually appropriate [40].

Moreover, the deployment of LLMs in classrooms could potentially diminish the role of teachers, reducing valuable human interaction that is crucial for the development of critical

thinking and interpersonal skills. Macneil et al. (2022) discuss the need for a balanced approach that uses LLMs to enhance, rather than replace, traditional teaching methods [41]. In their work, Joseph et al. (2023) use integrative literature review to identify and synthesize a Critical AI Literacy framework that would enable language teachers to adopt LLM-based tools to enhance their instructional strategies [42].

Furthermore, there is the issue of academic integrity. LLMs can easily generate essays and answers to complex questions, which could be used by students to cheat on assignments and exams. This raises significant concerns about the assessment of student learning and the development of genuine understanding.

While LLMs offer exciting possibilities for enhancing educational experiences through personalized learning and content generation, their drawbacks necessitate careful management and regulation. Ensuring that these powerful tools are used responsibly in educational settings is crucial to harnessing their benefits while mitigating risks related to accuracy, bias, and the undermining of traditional educational values. Addressing these challenges will be key to the successful integration of LLMs into the educational landscape.

2.3 Retrieval Augmented Generation and Its Uses in Education

Retrieval Augmented Generation (RAG) represents a significant advancement in the integration of AI with educational technologies, particularly in enhancing the personalization and accuracy of content delivery in learning environments.

RAG involves enhancing language models by enabling them to retrieve information from external data sources in real time to augment their responses, making them more relevant

and accurate. This approach combines the generative capabilities of models like GPT with retrieval functions similar to traditional search engines, allowing the model to pull in information as needed during the generation process. The technology not only increases the factual accuracy of the outputs but also significantly reduces the common issue of hallucination—where models generate plausible but incorrect or irrelevant information. A detailed exploration of this concept is provided by Jiang et al. (2023), who describe a method for actively deciding what and when to retrieve, improving the relevance and accuracy of generated content in educational settings [43]. Wang et al. (2023) in their work introduce Chat-Ed, a chatbot architecture that combines ChatGPT with a retrieval-based chatbot framework to provide enhanced student support in higher education [44].

In educational settings, RAG has been used to create more dynamic and responsive learning materials. For instance, Li et al. (2022) survey various applications of RAG in education, highlighting its use in dialogue response generation and machine translation, which are critical for creating interactive learning experiences and multilingual educational content [45].

Another impactful application is described by Mao et al. (2020), who developed a Generation-Augmented Retrieval system for open-domain question answering. This system augments queries with generated contexts that enrich the semantics of the queries, thereby enhancing the retrieval accuracy and making the responses more relevant and tailored to educational needs [**?**].

Furthermore, Manathunga and Illangasekara (2023) discuss the use of RAG in medical education, where it helps integrate up-to-date medical knowledge into learning platforms, assisting in the training of healthcare professionals by providing them with the latest information and clinical guidelines [46].

2.4 LLMs with Guardrails and Their Uses

Large Language Models (LLMs) have seen widespread adoption in various fields, including education. However, their potential to generate unintended or inappropriate content necessitates the implementation of guardrails. These guardrails ensure that LLM outputs remain within ethical boundaries and align with educational goals.

In their work, Liffiton et al. (2023) explore the implementation of CodeHelp, an LLM-powered tool that provides on-demand programming assistance without directly revealing solutions. This work emphasizes the importance of incorporating guardrails in LLMs to strike a balance between providing assistance and mitigating over-reliance on LLMs [47].

Guardrails are essential for maintaining the integrity and safety of LLM applications, particularly in sensitive areas such as education. They prevent the model from generating harmful, biased, or incorrect content that could mislead learners or propagate misinformation. Liffiton et al. (2023) discuss the implementation of a tool called CodeHelp, which uses LLMs with guardrails to provide scalable support in programming classes without revealing solutions directly to the students. This approach not only helps in maintaining academic integrity but also supports personalized learning by prompting students towards the correct solution path rather than providing outright answers [47].

2.4.1 Review of Different Approaches and Technologies Used to Implement These Guardrails

Different methodologies have been explored to effectively implement guardrails in LLMs. One approach involves the use of "Contrastive and Scenario-Guided Distillation" methods, which tailor LLM outputs to adhere strictly to predefined rules. Sun et al. (2023) describe

how their CONSCENDI process uses scenario-guided examples to train guardrail models, ensuring that the outputs do not violate specific guidelines. This method is particularly useful in educational settings where maintaining the accuracy and appropriateness of the content is crucial [48].

Additionally, Rebedea et al. (2023) introduce NeMo Guardrails, a toolkit for adding programmable guardrails to LLM applications. This toolkit allows developers to set user-defined rules that are not only embedded during the training of the model but can also be adjusted in real-time, providing flexibility and control over the model's outputs in educational applications [49].

In conclusion, the integration of guardrails in LLM applications is crucial for their safe and effective use in education. By ensuring that outputs are aligned with ethical standards and educational objectives, these guardrails help in leveraging the benefits of LLMs while minimizing risks. The ongoing development of innovative guardrail technologies and methodologies will continue to enhance the reliability and safety of LLM applications in educational settings.

Chapter 3

Methodology

This section delineates the system architecture and implementation of the application, touching upon the abstract theoretical aspects and core functionalities.

3.1 Application Design

The Gurukul application is divided into two main features–the Study section and the Practice section. The study section focuses on providing students with context specific information about a DSA topic retrieved from relevant sources using an LLM with Retrieval Augmented Generation (RAG), while the practice section enables students to practice coding problems in a live interactive code editor with an LLM with Guardrails as a teaching assistant.

3.1.1 Study Section

The study section of the Gurukul application is a dedicated space where students can explore and learn DSA concepts interactively through a prompt and learn method. This section leverages the feature of RAG to enhance LLM capabilities to provide accurate and contextually relevant information.

When a student queries a specific DSA topic or problem the system uses RAG to fetch information from a pre-processed and vetted knowledge base - the OpenDSA [50]. The

retrieved information is then presented to the student in a coherent and understandable format, facilitating deeper comprehension of complex concepts.

Key features of the study section include:

Prompt-Based Interaction: The use of natural language prompts intended to make communication intuitive and accessible.

Dynamic Information Retrieval: Fetches relevant information in real-time from a vast database based on the user's query.

Verified Sources: Provides information from authoritative and reliable references to ensure credibility.

The Study section's interface is visually demonstrated in Figures 3.1 and 3.2, which illustrate the RAG mechanism at work, showcasing how students interact with the system to retrieve and comprehend DSA concepts effectively.

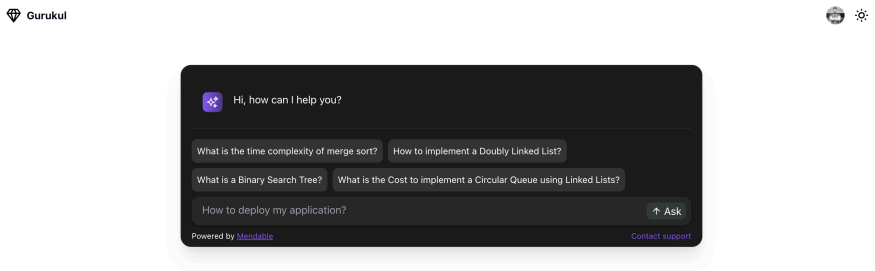

Figure 3.1: Study section RAG demonstration - 1

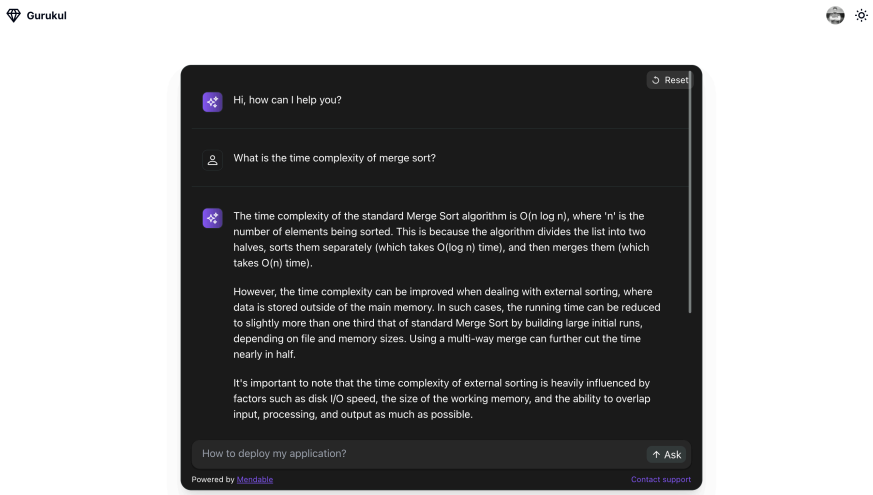

Figure 3.2: Study section RAG demonstration - 2

3.1.2 Practice Section

The practice section of the application is designed to provide students with a hands-on coding experience, allowing them to apply the DSA concepts they have learned in the Study section. This section integrates a live code editor and an LLM based assistant with Guardrails to create an interactive and supportive learning environment. As shown in Figure 3.3, the Practice section includes a live code editor that presents students with coding problems and enables them to write, test, and debug their code. This integration with the LLM assistant ensures that students receive instant feedback and guidance while maintaining a focus on problem-solving rather than merely providing solutions.

Key features of the Practice section include:

Live Code Editor: The practice section features a live code editor where students can write, test, and execute their code. The editor includes several functionalities to enhance the coding experience:

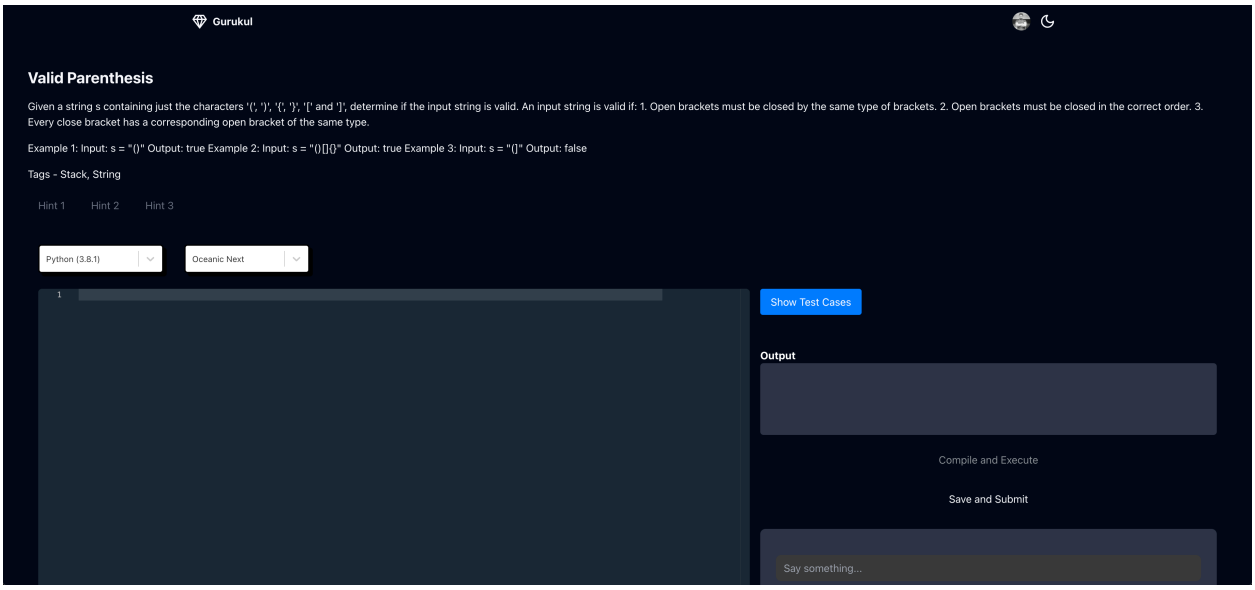

Figure 3.3: Code editor with sample problem

- **Syntax Highlighting**: Helps students read and understand their code more easily by visually distinguishing different elements.

- **Error Detection**: Identifies syntax and logical errors in the code, providing immediate feedback.

- **Multiple Language Support**: Students can choose from multiple programming languages, including Python, Java, C++, C, Go, and JavaScript, to solve the problems.

Interactive LLM Assistant with Guardrails: The practice section also integrates an LLM assistant to provide instant feedback, hints, explanations, and alternative solutions in the form of pseudocode. This interactive component ensures that students receive immediate reinforcement and consolidation of new knowledge. The LLM assistant is embedded next to the code editor for easy access, enabling students to ask questions and receive help without leaving the coding environment. Guardrails are implemented to ensure the assistant does not provide direct answers, encouraging students to develop their problem-solving skills.

Hint System: Three static hints generated by ChatGPT are provided in the code editor to help students understand how to solve problems. Hints are ordered in increasing levels of verbosity, with hint1 providing the least information and hint3 providing the most. This system is designed to guide students through the problem-solving process without giving away the complete solution.

Problem Sets and Challenges: The practice section includes a diverse set of problems taken from Leetcode, ranging from easy to medium (under Leetcode's categorization). Problems are designed to help students prepare for real-world coding challenges found in interviews and class tests.

User-Friendly Interface: The interface is designed to be intuitive and user-friendly, with a clean layout that facilitates ease of navigation.

3.2 Large Language Model

The LLM used in the Gurukul application is the GPT-4 model through the OpenAI API. GPT-4 is a state-of-the-art language model developed by OpenAI. It provides the underlying intelligence for both the Study and the Practice sections enhancing the learning experience by offering contextually relevant information, guidance and support to students.

GPT-4 leverages the Transformer architecture, a breakthrough in deep learning models. Transformers are a type of deep learning models introduced by Vaswani et al in their seminal paper "Attention is all you need" [51]. GPT-4 is the latest iteration in the Generative Pre-trained Transformer series by OpenAI. It follows its predecessors, GPT-1, GPT-2, GPT-3 each increasing in scale and capability [52]. GPT-4 builds on advancements with several key enhancements compared to previous models.

In terms of scale, GPT-4 has rumored 1.76 trillion parameters, a substantial increase from GPT-3, allowing it to model complex patterns and nuances in data more effectively. Unlike its predecessors, GPT4 is multimodal, meaning it can process both text and image inputs. This enables it to handle more diverse tasks such as describing images and interpreting visual data. GPT-4 offers context windows of up to 32,768 tokens significantly higher than GPT-3's 4096 tokens. This allows it to maintain coherence over longer texts and handle more extensive interactions.

3.3 Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) forms the pedagogical backbone of the study feature. RAG leverages the power of Large Language Models (LLMs) by augmenting them with an external knowledge base. The purpose of RAG is to address the limitations of LLMs that are trained on static datasets dated to a specific point in time. This can make the LLM information outdated or lack specific information during inference. RAG synthesizes the retrieval of information from a vast corpus of data with the generative capabilities of advanced language models to offer explanations, solutions, and clarifications tailored to students' needs.

3.3.1 Mechanism of Action

The various phases involved in Retrieval Augmented Generation are:

1. **Document Ingestion:** The first step in RAG is document ingestion, where a wide range of documents and data sources are collected to serve as the knowledge base for the model. These documents can vary greatly in format and content, ranging from

technical manuals and scientific articles to news stories and general web content. The key objective here is to amass a comprehensive repository of information that the model can retrieve from when generating responses to queries. This step is foundational, as the diversity and quality of the ingested documents directly influence the model's ability to provide accurate and relevant information.

2. **Document Chunking:** Once the documents are ingested, they are chunked into smaller, more manageable pieces. Document chunking involves breaking down large documents into segments or paragraphs that are easier for the model to process. This step is crucial for enhancing the efficiency of the retrieval process, as it allows the model to quickly identify and retrieve the most relevant snippets of information related to a given query. Chunking thereby ensures that the system can operate with a high degree of precision and speed, enabling real-time responses to user inquiries.

3. **Converting Documents to Vector Databases:** The chunked documents are then converted into a format that can be easily and effectively used by the model, typically through a process known as "embedding". This involves transforming the textual data into mathematical vectors using complex machine learning models. These vectors, or embeddings, represent the semantic and contextual essence of the text, allowing the retrieval model to effectively identify relevant data points within the vast knowledge base. Vector databases are then employed to store these embeddings, enabling rapid and accurate retrieval of information during the RAG process.

4. **Using Embeddings for Semantic Search:** Embeddings are instrumental in enabling semantic search within the RAG framework. Unlike traditional keyword-based search, semantic search uses vector embeddings to understand the intent and contextual meaning behind queries. This allows the system to fetch information that is semantically related to the query, even if the exact keywords are not present in the

text. Semantic search significantly enhances the model's ability to provide contextually relevant and nuanced responses, making the system far more effective in addressing complex and specific user inquiries.

5. **Using LLMs with Vector Databases for RAG:** Finally, the LLMs are integrated with the vector databases to execute the RAG process. When a query is received, the LLM first formulates it into a numeric format that can be understood by the retrieval component. The retrieval model then uses the embeddings stored in the vector databases to find and fetch the most relevant information related to the query. This information is then passed back to the LLM, which synthesizes it with its internal knowledge to generate a comprehensive, accurate, and contextually enriched response. The integration of LLMs with vector databases for RAG thus enables a dynamic and intelligent system capable of leveraging external knowledge to enhance the quality of its output.

In the Gurukul application, RAG was implemented in the 3 pivotal steps. Fig 3.4 depicts the flow diagram of the various phases of RAG:

1. **Data Crawling and Preprocessing:**

   - OpenDSA, developed at Virginia Polytechnic Institute and State University, was selected as the primary source of documents for the LLM to refer to, providing a repository of educational content. OpenDSA's eTextbooks cover a wide variety of topics related to Computer Science, including Data Structures and Algorithms, Formal Languages, and Programming Languages. These resources are designed to enhance computer science education through interactive components such as algorithm visualizations, proficiency exercises, and multiple-choice questions.

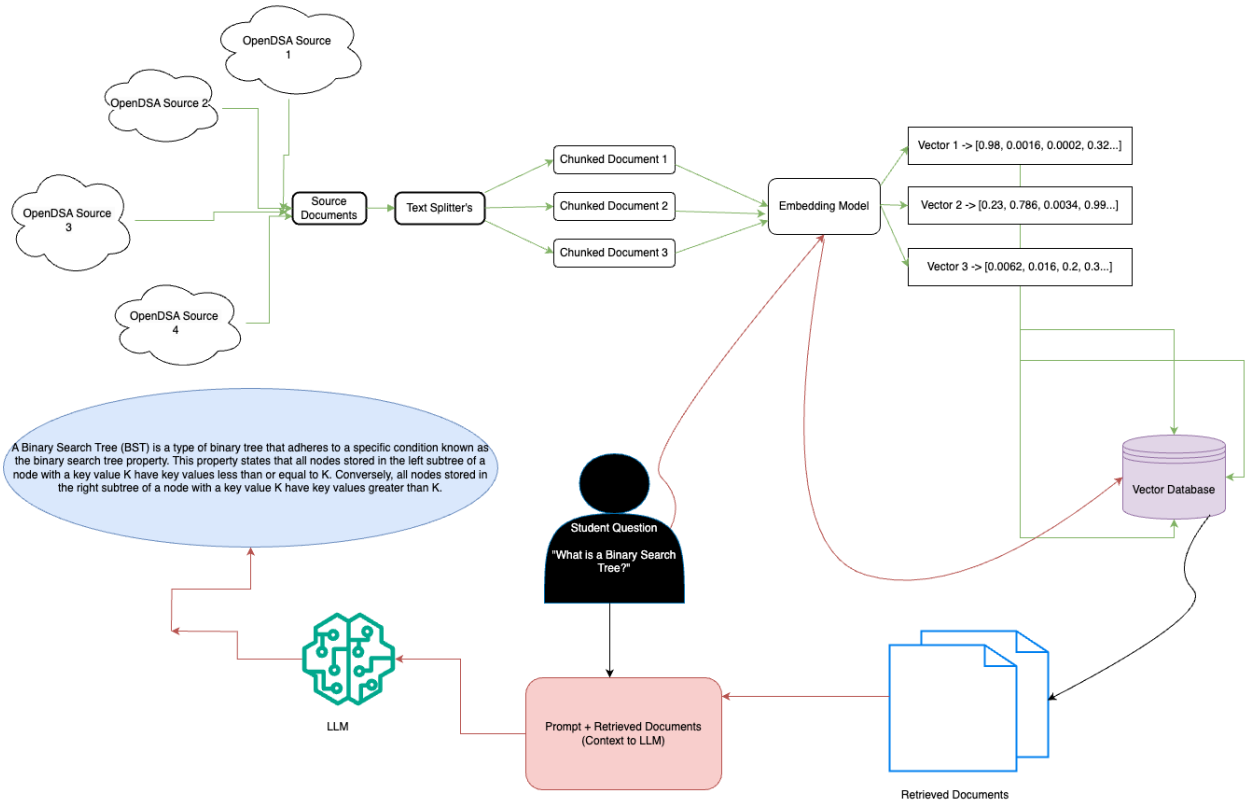

Figure 3.4: Code editor with sample problem

- The crawled HTML documents from OpenDSA are converted into dense word vectors and stored in Supabase's PostgreSQL-based vector database (pgvector) [53]. This process involves running the data through an embedding model to convert it into vectors, representing the meaning of the input text in a dense numerical format.

2. **Vectorization:**

- An embedding model, OpenAI's "text-embedding-ada-002" was used to convert the chunked documents into a vector database.

3. **Retrieval and Generation:**

- When a learner poses a question within Gurukul's Study section, the system initiates the retrieval phase of the RAG process. The user's query is converted into a high-dimensional vector using an embedding model. This vector represents the semantic essence of the query in mathematical space.

- The system then searches its VectorDB for the most relevant documents or pieces of information that match the query. The retrieval mechanism employs algorithms to identify the vectors in the VectorDB that are most similar to the query vector. These vectors correspond to chunks of educational content that are likely to contain the information the learner is seeking.

- After identifying the most relevant documents or content pieces, Gurukul's system retrieves this information and presents it to the learner within the Study Section. This integration can take various forms, from directly displaying text extracts to summarizing the content or generating new, contextually enriched responses that combine insights from multiple sources. The goal is to provide learners with accurate, comprehensive, and easily understandable answers that enhance their grasp of the subject matter.

By employing RAG, Gurukul ensures that students receive up-to-date and contextually relevant information, significantly enhancing their learning experience. The integration of RAG into the Gurukul platform bridges the gap between static knowledge bases and dynamic, real-time educational needs, providing a powerful tool for mastering complex domains like Data Structures and Algorithms.

3.4 Guardrails

Guardrails are critical components in the Gurukul application, designed to ensure the integrity, safety, and educational value of interactions between students and the Large Language Model (LLM). These guardrails maintain the ethical and pedagogical standards of the platform by preventing the model from generating inappropriate, biased, or harmful content and ensuring that the educational experience remains focused and productive.

Conceptual Framework

Guardrails in the context of LLMs like GPT-4 are mechanisms that control and guide the model's behavior, ensuring that its outputs align with predefined ethical and educational guidelines. These mechanisms are essential in educational settings to prevent the dissemination of incorrect or inappropriate information and to encourage constructive learning practices.

Implementation of Guardrails

In the Gurukul application guardrails are implemented using a system prompt integrated through the OpenAI API [10]. These guardrails are configured to ensure that the LLM behaves in a manner consistent with the educational goals of the platform. The model in use is the GPT-4 model.

System Prompts and Behavioral Guidelines: The system prompts define the scope and nature of the LLM's responses. For example, a system prompt is used to guide the LLM to focus exclusively on providing educational content related to Data Structures and Algorithms. The prompt explicitly instructs the LLM not to generate any executable code, thus preventing the model from offering direct solutions to coding problems. Instead, it encourages the model to assist students through explanations, pseudocode, and conceptual

guidance.

Implementation Details

The following code snippet demonstrates the implementation of the guardrails in the Practice section using the OpenAI API:

Listing 3.1: LLM Assistant implementation in the practice section.

```
1
2 import OpenAI from 'openai';
3 import { OpenAIStream, StreamingTextResponse } from 'ai';
4 import { initSupabase, insertChatInteraction } from '@/utils/supabaseUtils';
5
6 const openai = new OpenAI({
7   apiKey: process.env.OPENAI_API_KEY,
8 });
9
10 export const runtime = 'edge';
11 export async function POST(req) {
12
13   const { messages, userId } = await req.json();
14   const latestUserInput = messages[messages.length - 1]?.content;
15
16   // system message for guardrails
17   const systemMessage = {
18     role: ``system",
19     content: ``You are a benevolent helping Teaching Assistant in Data
         Structures and Analysis. You only Answer Questions Related to Data
         Structures and Algorithms and Test Cases and Imaginary Test Cases and
         Everything related to the Question at Hand. You DO NOT OUTPUT ANY CODE
         "
20   };
```

```
21
22    messages.unshift(systemMessage);
23
24    const response = await openai.chat.completions.create({
25      model: 'gpt-4',
26      stream: true,
27      messages,
28    });
29
30    const currentDate = new Date();
31    const currentDateTimeString = currentDate.toLocaleString();
32
33    // console.log(response);
34
35    const supabase = initSupabase();
36
37    const stream = OpenAIStream(response, {
38      onCompletion: async (completion) => {
39        // saving the response
40        await insertChatInteraction(supabase, userId, latestUserInput,
              completion);
41      }
42
43    });
44
45    return new StreamingTextResponse(stream);
46  }
```

This code snippet demonstrates how the server-side logic for handling responses is encapsulated in the POST function for implementing the Guardrailed LLM API endpoint. It

prepares the messages for the API call, including a system prompt that guides the AI's responses. The 'OpenAIStream' function processes the streaming response from the OpenAI API, ensuring that the AI's outputs are saved and conform to the guardrails set by the system prompts. This setup enables a real-time, interactive chat experience while ensuring the LLM's behavior remains aligned with the platform's educational goals.

By implementing these guardrails, the Gurukul application ensures that students receive accurate, relevant, and ethically aligned educational support. This approach not only enhances the quality of the learning experience but also fosters a safe and productive environment for mastering complex subjects like Data Structures and Algorithms.

3.5 Implementation

The implementation of the Gurukul application encompasses the integration of various technologies and systems to create a cohesive and effective learning platform. This section outlines the detailed steps and components involved in bringing the Gurukul application to life.

3.5.1 System Architecture

1. **Hybrid Client-Server Architecture:**

    - The Gurukul platform is built on a hybrid client-server architecture, utilizing Next.js [54] for its flexibility in rendering strategies, such as Server Side Rendering (SSR) and Static Site Generation (SSG).
    - This architectural choice ensures rapid loading times and effective Search Engine Optimization (SEO) while maintaining rich interactivity.

- The backend is powered by Node.js, which efficiently handles API requests, user authentication, and database operations.

3.5.2 Study Section Implementation

1. **Data Crawling and Preprocessing:**

   - Educational content from sources like OpenDSA is crawled and preprocessed to convert it into dense word vectors.
   - OpenDSA, developed at Virginia Tech, provides a comprehensive repository of educational content that covers a wide variety of topics related to Computer Science, including Data Structures and Algorithms, Formal Languages, and Programming Languages. These resources are interactive, combining text, image algorithm visualizations, proficiency exercises, multiple-choice questions, and programming exercises to enhance learning.
   - The HTML documents from OpenDSA are converted into dense word vectors using an embedding model and stored in Supabase's PostgreSQL-based vector database (pgvector). This process ensures that the LLM can refer to a structured and comprehensive knowledge base during inference.

2. **Document Ingestion and Chunking:**

   - A wide range of documents and data sources are ingested to build a robust knowledge base. The ingestion process includes collecting various formats of documents, ranging from technical manuals and scientific articles to news stories and general web content.
   - These documents are then chunked into smaller, manageable pieces to enhance the efficiency of the retrieval process. Chunking involves breaking down large

documents into segments or paragraphs, allowing the model to quickly identify and retrieve the most relevant snippets of information related to a given query.

3. **Vectorization:**

   - An embedding model, which is a type of LLM, converts the chunked data into vectors: arrays, groups, or numbers that capture the semantic and contextual essence of the text.
   - These vectors are stored in a specialized vector database optimized for high-dimensional vector data, facilitating efficient retrieval of information based on similarity searches.

4. **Semantic Search and Retrieval:**

   - When a student poses a query within the Study section, the system initiates the retrieval phase of the RAG process. The query is vectorized using an embedding model, representing the semantic essence of the query in mathematical space.
   - The system then searches the VectorDB for the most relevant documents or pieces of information that match the query. The retrieval mechanism employs algorithms to identify the vectors in the VectorDB that are most similar to the query vector. These vectors correspond to chunks of educational content that are likely to contain the information the learner is seeking.
   - The retrieved information is then synthesized by the LLM to generate accurate, contextually enriched responses, which are presented to the student in a coherent and understandable format.

3.5.3 Practice Section Implementation

1. **Live Code Editor Integration:**

    - The Practice section features a live code editor where students can write, test, and debug code in a real-world coding environment.
    - The Monaco editor component is used as the code editor, providing a robust and flexible coding environment.
    - Additional features such as syntax highlighting, error detection, and code completion are implemented to enhance the user experience. The editor supports multiple programming languages, including Python, Java, C++, C, Go, and JavaScript.

2. **Interactive LLM Assistant with Guardrails:**

    - The LLM assistant is integrated within the code editor to provide real-time support to students. It offers instant feedback, hints, explanations, and alternative solutions in the form of pseudocode.
    - Guardrails are implemented to ensure the assistant does not provide direct answers, encouraging students to develop their problem-solving skills.
    - The LLM assistant uses system prompts to restrict responses to educational content only, ensuring that it supports the learning process without providing complete solutions.

3. **Hint System and Problem Sets:**

    - The Practice section includes a diverse set of problems taken from platforms like Leetcode [55], ranging from easy to medium levels.

- Three static hints generated by ChatGPT are provided in the code editor to help students understand how to solve problems. Hints are ordered in increasing levels of verbosity, with hint1 providing the least information and hint3 providing the most.
- The problems are designed to help students prepare for real-world coding challenges found in interviews and class tests.

4. **User-Friendly Interface:**

   - The interface is designed to be intuitive and user-friendly, with a clean layout that facilitates ease of navigation.
   - Students can easily switch between different problems and view their progress.

3.5.4 LLM and RAG Integration

1. **System Prompts and Guardrails Configuration:**

   - The LLM is configured with specific system prompts that restrict its output to educational content only.
   - Guardrails are implemented to prevent the LLM from generating harmful, biased, or inappropriate content.

2. **Data Processing and Vectorization:**

   - Textual data is transformed into vectors using embedding models, capturing the semantic meaning of the content. These vectors are stored in a specialized vector database optimized for high-dimensional vector data, facilitating efficient retrieval during the RAG process.

3. **Semantic Search and Response Generation:**

    - The LLM utilizes semantic search techniques to understand the intent behind queries. It performs searches within the vector database to retrieve the most relevant information, which is then synthesized into coherent and contextually accurate responses.

Chapter 4

Results

The evaluation of Gurukul was done using a two-pronged approach. We conducted a User Expert Review with professors from Virginia Tech and a User Study involving students interested in or taking a Data Structures and Algorithms course. This mechanism for evaluation was adopted to perform a comprehensive and holistic evaluation from both sides—the pedagogical or instructor viewpoint and the student viewpoint of usage. This form of evaluation was performed to mitigate potential biases and validate findings through corroborating evidence by triangulating data from multiple sources and employing different data collection methods.

4.1 User Expert Review

A User Expert Review was conducted to assess the Gurukul application's usability, accessibility, and overall user experience from an educator's perspective. Two seasoned professors from Virginia Tech critically analyzed the app's educational content, usability, user interface, and interactive features to gauge its efficacy and potential integration into academic curricula.

4.1.1 Review Methodology

Two interviews were conducted, each lasting one hour and structured as follows:

1. Introduction (5 minutes): The review began with a clear articulation of the objectives and structure, setting the stage for a focused assessment.

2. Cognitive Walkthrough (5 minutes): Participants were given a guided demonstration on navigating and utilizing the app, with key functionalities highlighted.

3. Independent Exploration (10-20 minutes): The professors independently explored the app's features, allowing them to form an unbiased assessment of its interface and content.

4. Structured Interview (25 minutes): The professors participated in a structured interview, answering questions designed to gather detailed feedback on specific aspects of the system.

5. Quantitative Assessment (5 minutes): After the interview, the experts completed a quantitative form [56] designed to numerically rate various aspects of the application.

4.1.2 Objectives of the Expert Review

- **Content Accuracy and Relevance of RAG**: Evaluating the pedagogical soundness and alignment of the app's content with academic standards was the paramount objective of the expert review. The review particularly scrutinized the Retrieval Augmented Generation (RAG) for its ability to provide accurate, context-aware responses based on the OpenDSA framework and how it prevented hallucinations.

- **Usability and Interface Design**: The application's design and user interface were assessed for their intuitiveness, aesthetic appeal, and functionality. This included an

evaluation of navigation ease, information architecture, and overall user engagement potential.

- **Interactive Features and Virtual Assistant**: The functionality and educational value of the app's interactive features, including the live code editor, use of static hints and AI-enabled Assistant, were critically analyzed. The aim was to determine their effectiveness in enhancing learning without fostering dependency. The reviewers also analyzed the output from the Guardrailed LLM to assess its efficacy to not output any code while aiding students with Pseudocode for solving the problems.

- **Persuasion and Adaptability**: The adaptability of the Gurukul app for integration into existing curricula and its scalability for diverse educational settings were also evaluated.

- **Accessibility Review**: Ensuring that the app was accessible to all users, including those with disabilities, was a key objective. This involved assessing compliance with accessibility standards such as the Web Content Accessibility Guidelines (WCAG) [57].

4.1.3 Key Features Evaluated

First Impressions

The professors found the platform visually appealing and professionally designed. They noted that the clean and structured layout made it inviting and easy to navigate, which could potentially reduce the learning curve for new users significantly. Both professors advocated for simplicity in User Experience and remarked that the platform illustrated simplicity and ease of use. Furthermore the app's interface was reported to be intuitive with well-delineated sections for theory, practical exercises, and interactive tools, which facilitated an easy learning

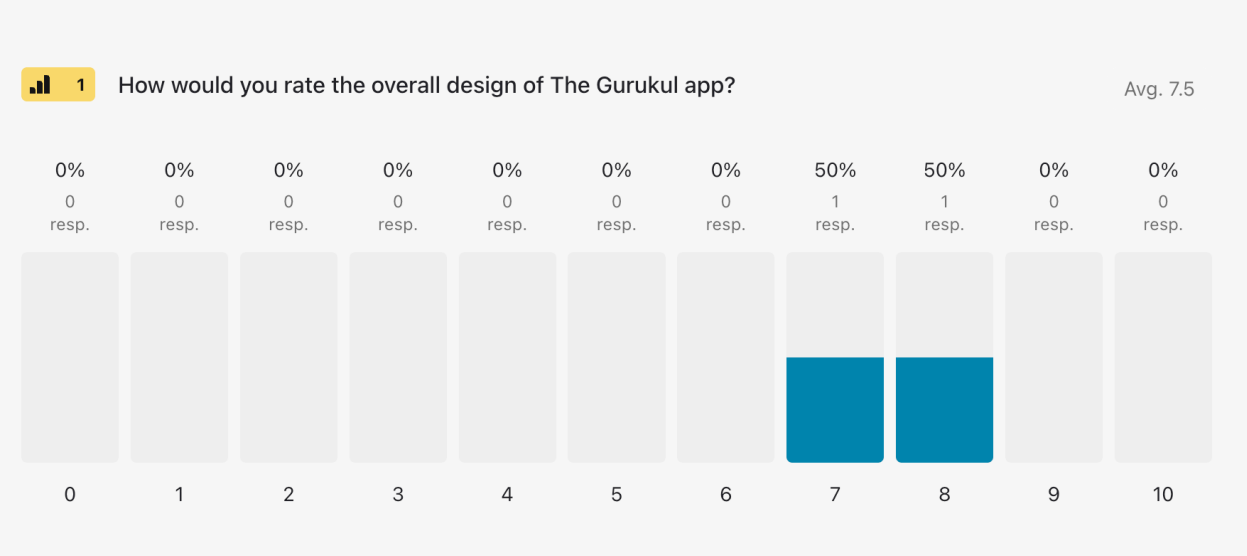

Figure 4.1: Expert reviews for overall design of the app.

flow. The design and aesthetics were pleasing, although there were suggestions to enhance the font size and contrast to aid users with visual impairments and to accommodate different screen sizes. The reviewer particularly noted the error messages and popups to be hard to understand. Some error messages were found to be overly verbose and seemed to include the entire stack trace. The alternative that was suggested for this was providing a summary of the message or the end message where the error occured.

The straightforward sign-up process and the benefits of account creation, like personalized tracking of progress and preference saving, were noted as positives. These first impressions are visually summarized in Figure 4.1, which reflects the expert reviews for the overall design of the app.

Site Navigation

The logical flow between different sections of the app received positive feedback, with users appreciating the ease with which they could find features and content. However, there were

Figure 4.2: Expert issues while navigating the application.

recommendations for better labeling of interactive elements and possibly a more detailed tutorial for first-time users to familiarize themselves with the navigation. While the app was generally accessible, there was a call for more robust accessibility features, especially to aid users with disabilities such as color blindness or hearing impairments. Suggestions included text-to-speech capabilities, better color contrasts, and more comprehensive keyboard navigability. One expert particularly noted that the main page could use a bit more information for onboarding. There were some layout and display issues while interacting with the live code editor for this reviewer. These issues and feedback are depicted in Figures 4.2 and 4.3.

Content Evaluation

The experts were satisfied with the use of RAG leveraging the OpenDSA material to generate dynamic, context-aware educational content. During the review, the experts found out

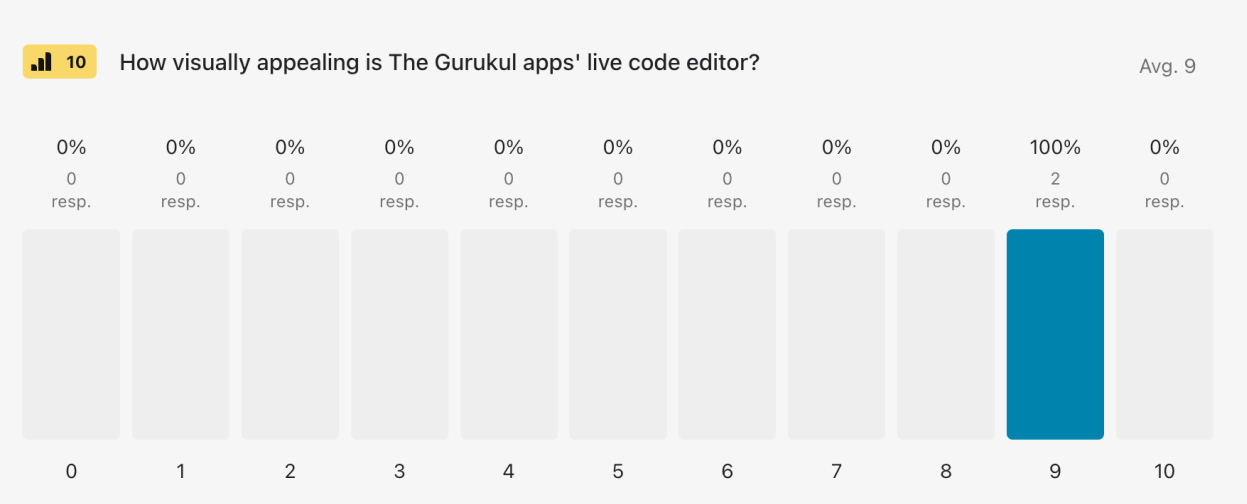

Figure 4.3: Figure denoting how visually appealing the experts found the application.

that the LLM output an answer that would be helpful for the students to refer to. The professors noted that the LLM mostly output correct and detailed answers for more well defined prompts. For prompts that were ambiguous for eg - "What is the time complexity of Binary Search", the LLM responded with a generic answer but the professor expected it to be precise, the professor however expected the LLM to output the best case, worst case and average case time complexity for the question. It was found out that The RAG effectively filters and synthesizes information from OpenDSA ensuring the LLM focuses on delivering content that is not only contextually accurate but also beneficial for learning DSA comprehensively. During one question that was input regarding queues, one expert surmised if the code was generated by the underlying LLM itself without refering to any sources and upon inspecting the sources it was found out that the LLM indeed referred to the said sources to extract out the code. The reviewers highlighted the reliability of the LLM to counter hallucinations.

A crucial aspect observed was the system's ability to reject queries outside its scope (e.g., asking about "Alexander the Great" by stating a lack of relevant information in verified sources. Another query which was rejected was "What is the Cost to implement a Cir-

cular Queue using Linked Lists?" for which there seemed to be no source associated with OpenDSA hence the LLM responded with - "I could not find the answer to this in the verified sources." This feature prevents the model from "hallucinating" or presenting false facts, thus maintaining the integrity of the educational content.

The experts commended the verified sources that get displayed for each response, which not only adds a layer of trust by disclosing the origin of the information but also educates students on recognizing credible information sources.

A potential improvement identified through the review involved the integration of a memory component in the LLM assistant, allowing it to recall previous interactions. This enhancement could lead to more personalized and contextually enriched experiences for users.

Another recommendation was to fine-tune the balance between using OpenDSA and external knowledge bases. This tuning would ensure that while the system uses the extensive information available from OpenDSA, it can also harness broader knowledge as necessary, without compromising the educational goals.

The reviwers noted and commended the availability of a comprehensive set of problems ranging from basic to advanced difficulty levels, equipped with a live code editor. During the UX expert review, the practice section was thoroughly evaluated for its ability to provide a comprehensive and effective learning environment for students practicing DSA. This session is crucial as it allows users to apply theoretical knowledge in practical scenarios, enhancing their understanding and proficiency in DSA. The practice section of Gurukul was observed to offer a vast array of problems varying from basic to advanced levels, which the experts deemed instrumental for catering to students at different stages of their learning curve. The experts pointed out the utility of the live code editor in solving DSA problems with a hands-on practice mode. The experts liked the inclusion of multiple programming language

support.

The utilization of OpenDSA materials through Retrieval Augmented Generation was highly praised. The dynamic generation of content based on user queries and interaction history was seen as a revolutionary approach to personalized learning. The accuracy and relevance of the content provided by RAG impressed the professors, as it adhered closely to the trusted OpenDSA materials, which are widely respected for their clarity and depth.

The comprehensive set of DSA problems, ranging from basic to advanced, equipped with a live code editor, was lauded for its design and utility. The live editor supports multiple programming languages, which broadens accessibility and provides a real-world coding experience. The interactive environment, with instant feedback on code compilation and execution, enhances learning by allowing students to learn from mistakes in real-time. These insights are captured in Figures 4.4, 4.5, 4.6, and 4.7, which depict the expert evaluations of the study and practice sections and their suggestions for improvement.

The LLM assistant, designed to aid learning without enabling dependency, was highlighted as a beneficial feature. It prompts users to engage in critical thinking and problem-solving rather than providing direct answers. However, there were concerns about the depth of learning achieved through the assistant's interactions, suggesting a potential area for further refinement to ensure the LLM challenges users adequately.

Persuasion

The content's alignment with current DSA curricula was confirmed to be very beneficial for learners. The structure and depth of the material prepare students effectively for academic and professional challenges in data structures and algorithms. Both professors saw significant potential for Gurukul to be integrated into classroom settings as a supplementary

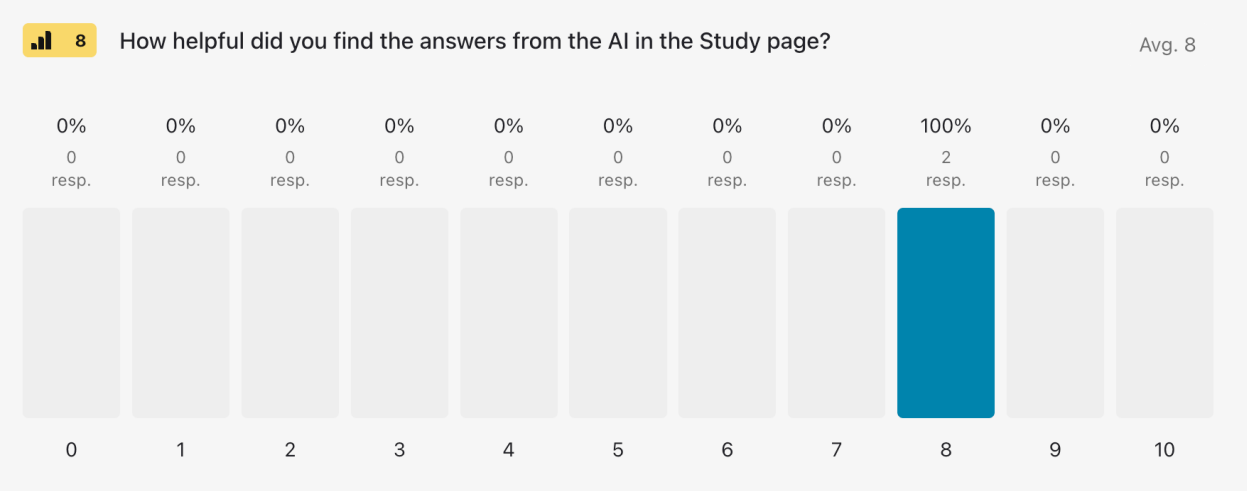

Figure 4.4: Figure denoting how helpful the experts found answers from the RAG system.

tool that could support traditional learning methods. They appreciated the potential for Gurukul to provide hands-on practice and immediate feedback, which are often missing from conventional educational environments.

4.2 User Study

The User Study was designed to evaluate the effectiveness, usability and educational impact of Gurukul on students. An Institutional Review Board application was submitted (see IRB approval document) and participants were selected from a diverse pool of students and professionals intersted in enhancing their understanding of DSA. Each participant was given access to the Gurukul platform along with a video tutorial that explained the various features of the application and the tasks they were expected to perform. The study was conducted remotely, with participants using the application independently and at their convenience.

After interacting with the application, 40 participants completed a detailed survey designed to capture their experiences, satisfaction levels and the applications impact on their learn-

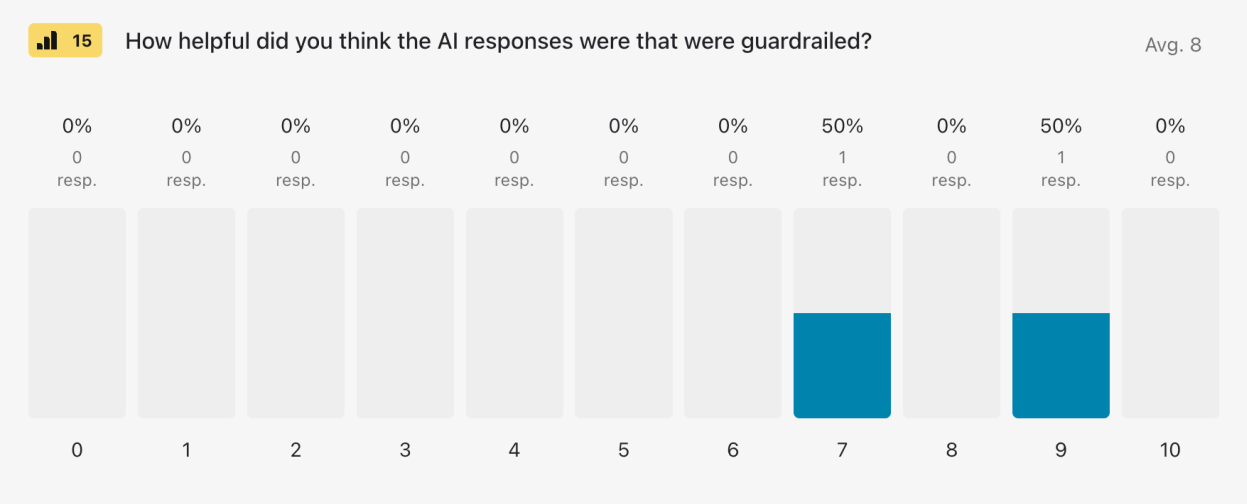

Figure 4.5: Figure denoting how helpful the experts found the answers from the LLM that were guardrailed.

ing. The survey included both a quantitative ratings and open-ended questions to gather comprehensive feedback. An additional Quantitative analysis was performed with the user engagement logs collected from the users.

4.3 Overview of Results

This section presents an analysis of data collected from the Gurukul app, including user feedback, performance metrics, and system interactions. The analysis methods include descriptive statistics, user surveys, and comparative studies to evaluate the app's effectiveness in enhancing Data Structures and Algorithms education.

4.3.1 Data Collection Methods

Surveys: Collected detailed responses about user satisfaction, educational impact, and overall feedback for improvement [58].

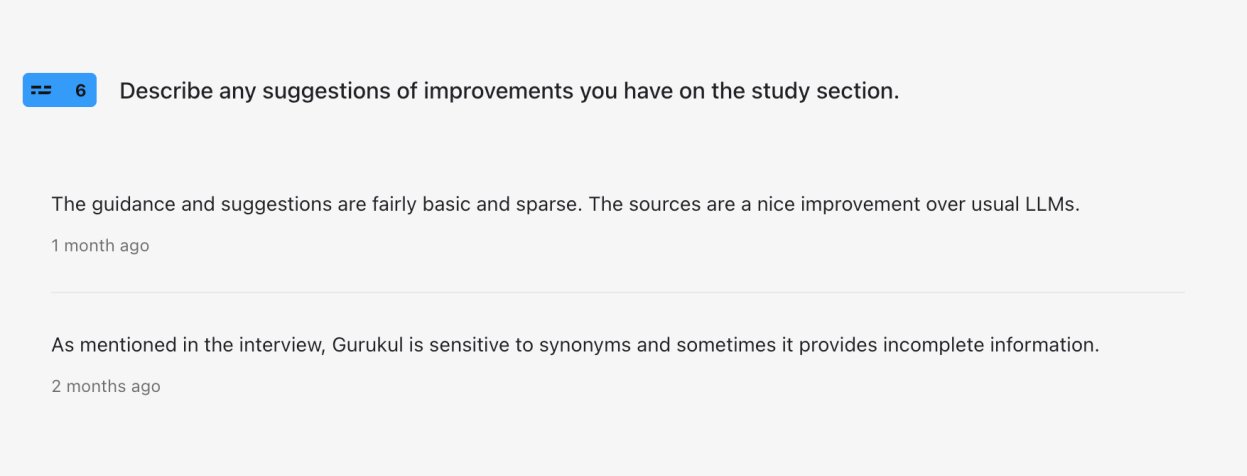

Figure 4.6: Suggestions for improvements from the experts for the study section

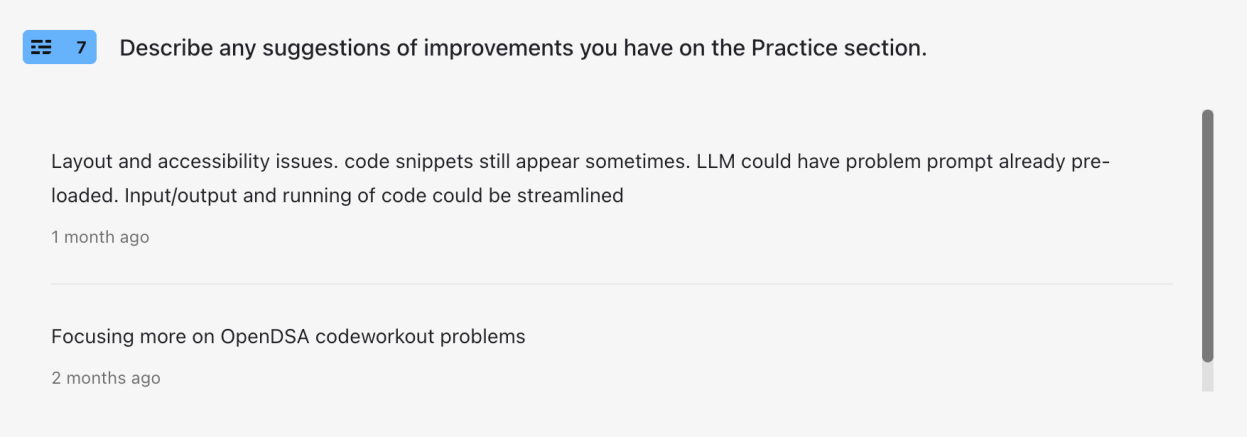

Figure 4.7: Suggestions for improvements from the experts for the practice section

Usage Analytics: Recorded metrics like the number of test cases passed, count of particular tagged problems solved, languages used, execution time, etc. [53].

4.4 Qualitative Evaluation

Users highlighted the educational value of RAG (Retrieval Augmented Generation), emphasizing its effectiveness in delivering theoretical knowledge. The inclusion of references in AI responses was appreciated for adding credibility, but it received mixed reactions when

perceived as overused. This sentiment reflects a broader pattern observed in user feedback, where clarity and thoroughness were often praised, yet the performance speed of the system was noted as a significant area needing improvement. Some users pointed out that the system's speed was slower compared to other generative AI tools, which could detract from the overall experience, especially when quick, responsive feedback is expected in a learning environment.

Gurukul's focus on specific DSA topics provided a more targeted learning experience than broader AI systems like ChatGPT, making the platform more relevant to their learning goals. This relevance was particularly appreciated in the context of handling different levels of difficulty, where users found that the system effectively supported a range of complexity from basic to advanced problems. However, users also consistently desired more content depth and a greater variety of problem types, expressing a wish for more challenging questions to cater to different skill levels.

While the user interface was generally well-received for its clean design, there was a call for better labeling of interactive elements and more detailed tutorials for first-time users. Users also expressed a preference for a blend of direct instructional content alongside the interactive AI-driven query system. This feedback aligns with suggestions for improving the accessibility of the platform, such as increasing font size for readability and enhancing the onboarding process to better guide new users.

Feedback on the platform's impact on users' motivation and confidence varied. Some users reported a boost in motivation and confidence, praising the immediate and interactive feedback, which allowed them to engage deeply with the content. However, others saw no noticeable change, indicating that while the platform is effective for some users, there is a need for further refinement to ensure it can consistently motivate and build confidence across a broader user base.

The practice section received both positive and negative feedback. Users appreciated the integration of AI, which allowed them to ask questions and receive help while practicing, drawing similarities to platforms like LeetCode. This functionality was particularly valued for its educational impact, helping users bridge the gap between theoretical understanding and practical application. However, some users found the AI to be limited, and the coding area too small for effective practice. Issues with frequent popups during code compilation and a lack of clear feedback were also noted, which could disrupt the learning flow and reduce the overall effectiveness of the practice environment.

Users provided mixed feedback on the system's performance in terms of speed, clarity, and completeness. While the clarity and completeness of responses were generally seen as strengths, the slower performance detracted from the user experience. Additionally, while most users found the AI's tone and personality appropriate for an educational setting, some noted occasional lapses into artificiality, which could undermine the naturalness and engagement of the interactions.

The qualitative feedback also revealed concerns about the system's data security, with some users questioning the safety of their data and whether it could be accessed by others. This highlights the importance of clearly communicating the platform's data handling and privacy policies to users, ensuring they feel confident and secure when using the system.

In summary, while users found the practice section helpful and appreciated the system's targeted focus on DSA topics, there are clear areas for improvement. The need for more challenging content, better user interface design, and faster system performance were recurring themes. Furthermore, the occasional artificiality in responses and the need for better data security communication underscore the importance of ongoing refinement to enhance the overall user experience.

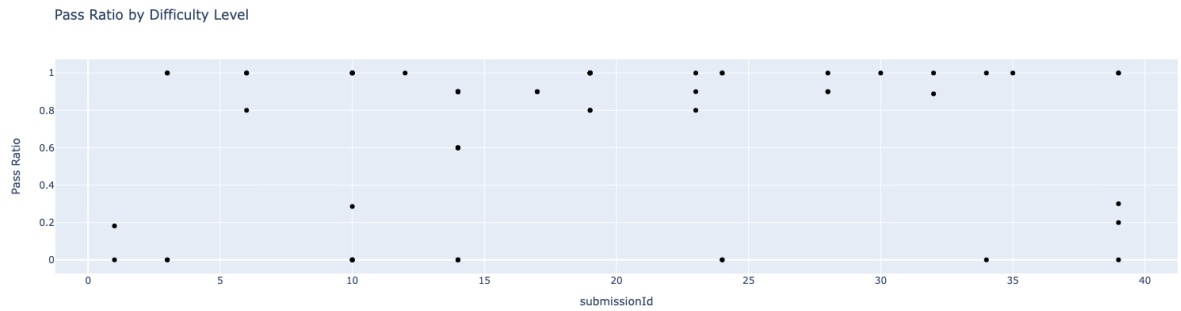

Figure 4.8: Reviews for Overall Design of the App

Figure 4.8 reflects user reviews of the app's overall design, showing satisfaction with the platform's professional layout, though highlighting the need for a more intuitive interface

4.4.1 Quantitative Evaluation

Detailed charts for quantitative analysis are provided in the appendix.

Study Section

The study section of the Gurukul app, utilizing Retrieval-Augmented Generation (RAG), received varied feedback from users. Students appreciated the comprehensive explanations and tailored learning resources, which helped deepen their understanding of theoretical concepts. The system's ability to reference credible sources added trust, although some users felt references were overused. The alignment of the study section with curriculum objectives and its ability to address learning challenges are depicted in Figures 4.10 and 4.11.

The study section's effectiveness in matching learning styles varied among users is illustrated in Figure 4.9.

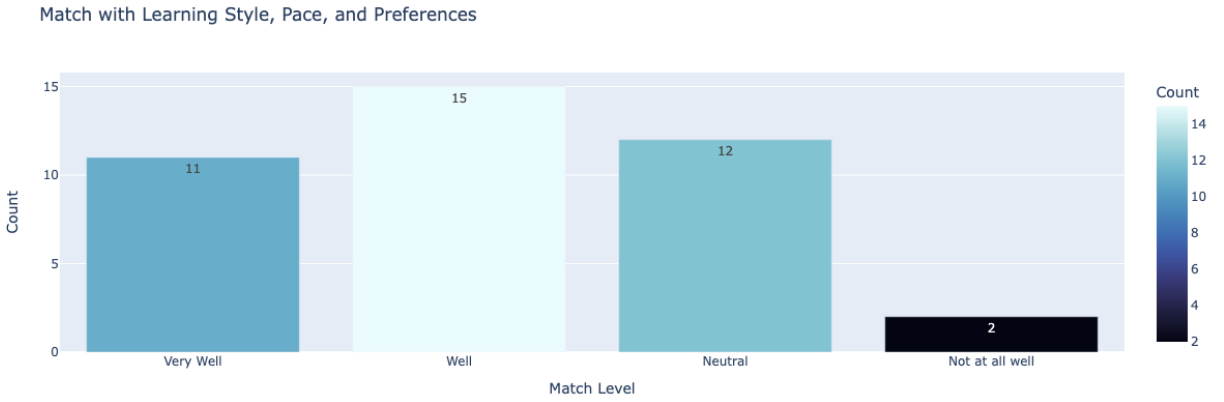

Figure 4.9: Responses to the question : "How well did the study section of the system match your learning style, pace, and preferences?"

Practice Section

Figure 4.12 illustrates how well users were able to apply the knowledge and skills they acquired from the study section to the practice section or other tasks. The results indicate that while many users found the practice section effective for reinforcing their learning, there is still room for improvement to ensure that all users can seamlessly transition from theoretical understanding to practical application.

Figure 4.13 highlights the ease of use of the practice section, revealing that some users experienced difficulties navigating the interface and interacting with the system. This aligns with the feedback regarding the need for a more user-friendly design and better guidance throughout the practice activities.

Finally, Figure 4.14 shows users' ratings of the usefulness of the practice section for their learning goals. The responses suggest that while the practice section was generally considered beneficial, enhancements in usability and interactivity could further increase its effectiveness and user satisfaction.

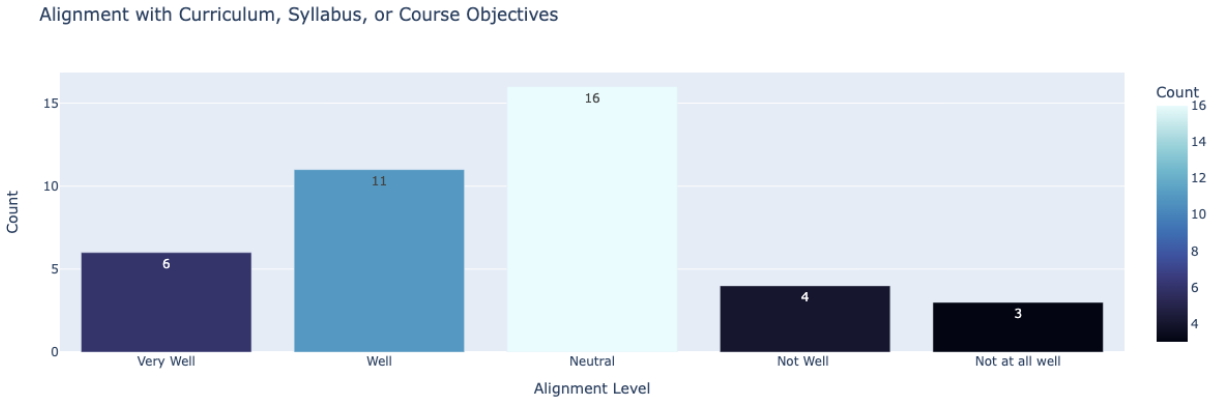

Figure 4.10: Responses to the question "How well did the study section of the system align with your curriculum, syllabus, or course objectives?"

Table 4.1: Performance of LLM Across Various DSA Domains

Domain	Very Well (%)	Well (%)	Just Fine (%)	Not Well (%)
Arrays	48.28	44.83	3.45	3.45
Linked Lists	41.38	34.48	20.69	3.45
HashMaps	20.69	41.38	31.03	6.90
Dynamic Programming	17.24	44.83	34.48	3.45
Sliding Window	17.24	44.83	31.03	6.90
Trees	37.93	41.38	13.79	6.90
Binary Search Trees	24.14	48.28	24.14	3.45
Graphs	24.14	34.48	27.59	13.79
DFS + BFS	20.69	48.28	24.14	6.90

The table titled "Performance of LLM Across Various DSA Domains" provides a detailed breakdown of how well the LLM (Large Language Model) performed across different Data Structures and Algorithms (DSA) domains, including Arrays, Linked Lists, HashMaps, Dynamic Programming, Sliding Window, Trees, Binary Search Trees, Graphs, and DFS + BFS.

Complementing this evaluation, Figure 4.15 shows the acceptance rates by programming language, indicating that some languages may be better supported, which could guide future

Figure 4.11: Responses to the question : "How well did the study section of the system address your learning needs, challenges, and difficulties?"

enhancements to ensure consistency across different coding environments. Figure 4.16 details execution times for submissions, emphasizing the importance of optimizing performance, particularly for slower languages, to maintain a responsive user experience. Together, these figures underscore the strengths and areas for refinement in both the functionality and design of the Gurukul platform.

Submission Analysis

The analysis of submission patterns in the practice section provides insights into user engagement and the effectiveness of the problems presented. Submissions are uniformly distributed across various coding challenges, indicating balanced interest. Most submissions have a status of "success," suggesting the level of problem difficulty is well-matched to the user's skill level.

Analyzing user activity metrics provided insights into user engagement, efficiency of practice sessions, and overall system usage patterns. Figures 4.17, 4.18, and 4.19 illustrate the count of solved questions by DSA tag, the count of submissions by programming language, and

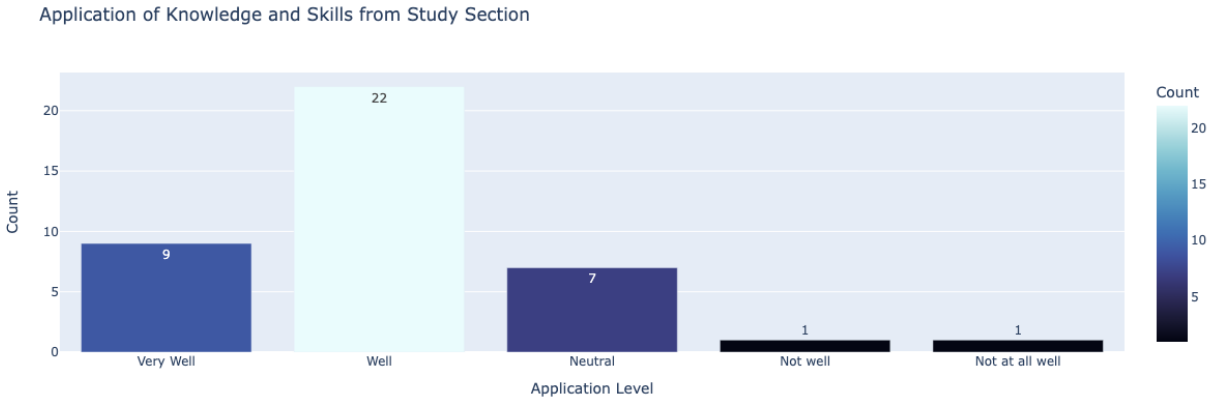

Figure 4.12: Responses to the question : "How well did you apply the knowledge and skills learned from the study section of the system to the practice section or other tasks?"

the distribution of acceptance rates, respectively, which are crucial for understanding user performance and the effectiveness of the practice section.

Problem Complexity and User Performance

The data shows a wide range of problem complexities. Performance metrics, such as execution time and success rate, vary across different problem types. Problems categorized under "Dynamic Programming" and "Graphs" show longer execution times, indicating higher complexity.

Interaction Patterns

Interaction patterns reveal user behavior and system responsiveness. Users frequently attempt problems multiple times, indicating persistence in learning. The system's immediate feedback on submissions supports iterative learning. The spread of execution times suggests the system handles a range of problem complexities without significant delays.

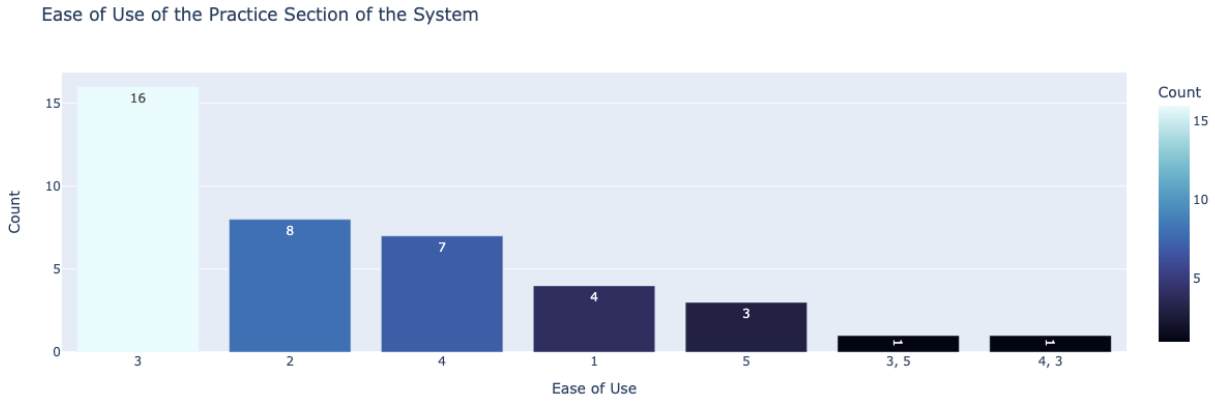

Figure 4.13: Responses to the question : "How easy or difficult was it to use the practice section of the system?"

These insights highlight both strengths and areas for improvement within the practice section, with significant satisfaction in ease of use, enjoyment, and learning utility, yet mixed feelings on recommending the section and varied experiences in overall quality and comprehension of user needs.

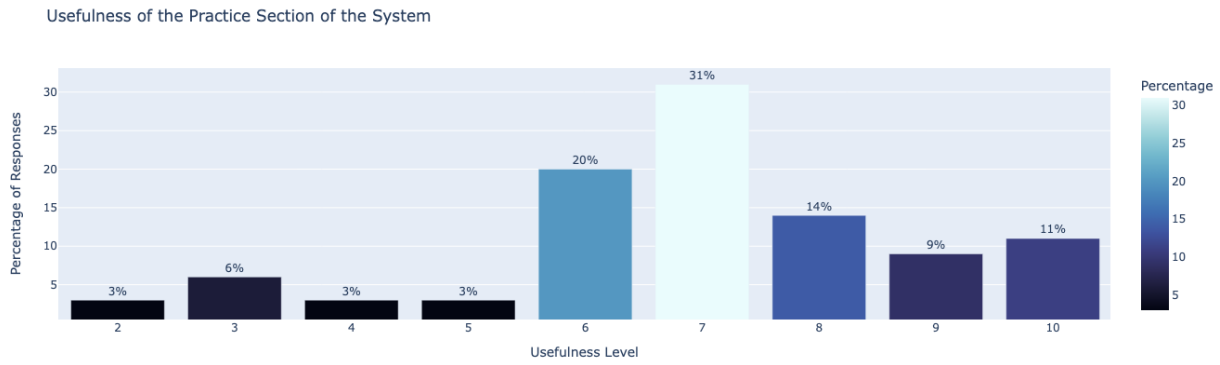

Figure 4.14: Responses to the question : "How useful was the practice section of the system for your learning goals?"

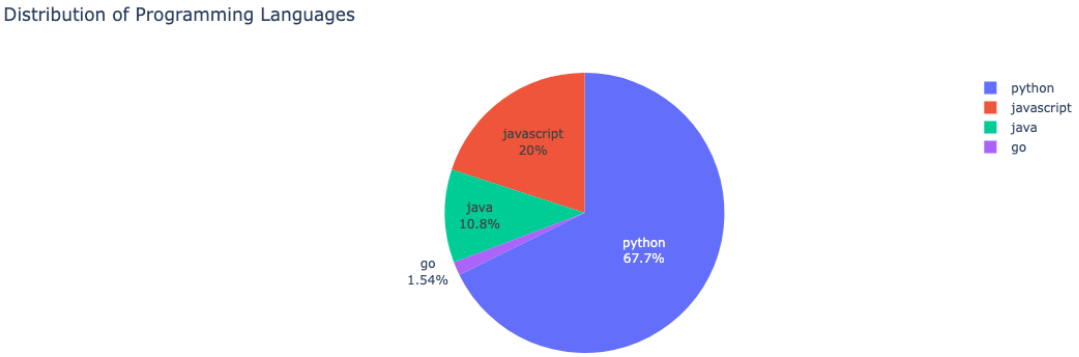

Figure 4.15: Acceptance rate by programming language

Figure 4.16: Execution times for submissions

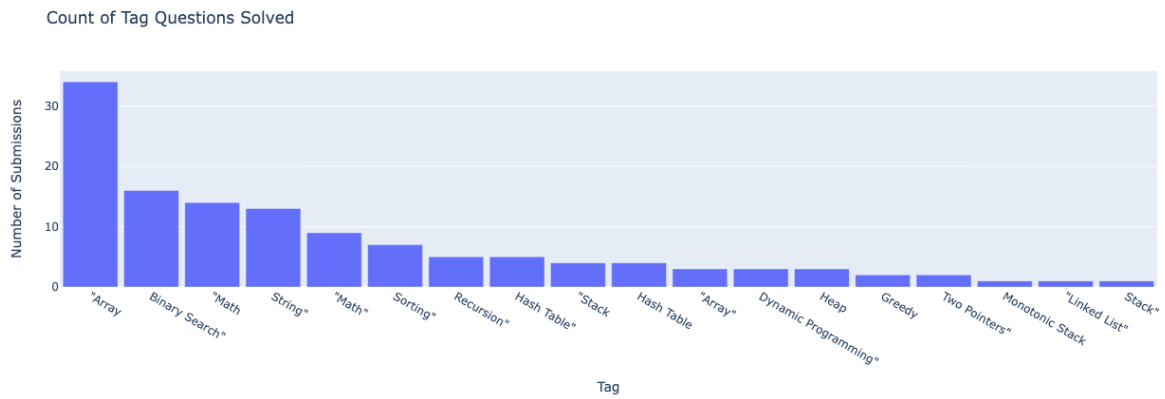

Figure 4.17: Count of solved questions by DSA tag.

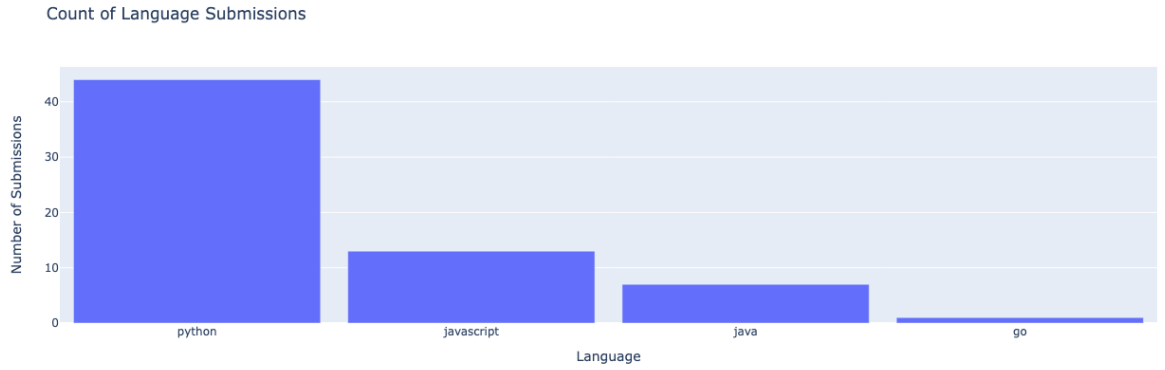

Figure 4.18: Count of submissions by language

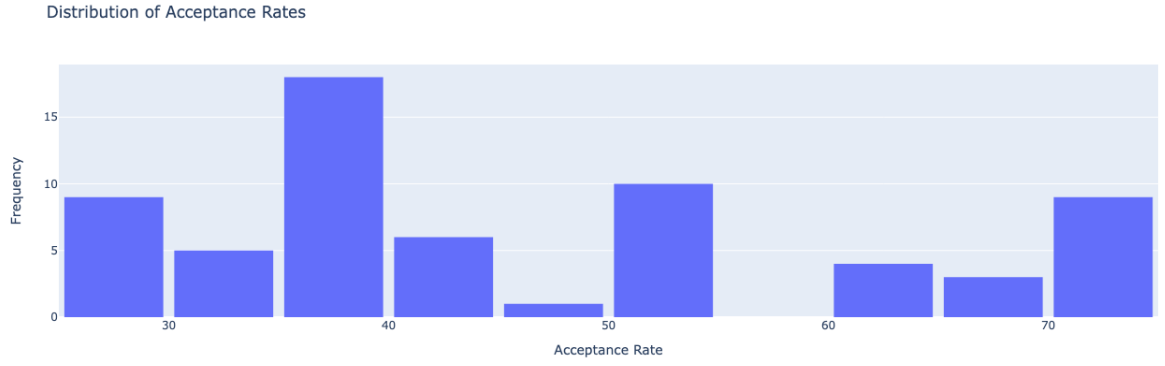

Figure 4.19: Distribution of Acceptance Rates

Chapter 5

Discussion

5.1 Overview

The implementation and evaluation of the Gurukul platform have provided significant insights into the potential and challenges of using specialized LLMs in educational settings. This section discusses the key findings from the user expert review and user study, contextualizing them within the broader landscape of computer science education and LLM-based learning tools.

5.2 User Expert Review Insights

The user expert review highlighted several critical aspects of the Gurukul platform, providing valuable feedback on its usability, content quality, and potential for integration into academic curricula [56].

5.2.1 Usability and Design

The professors from Virginia Tech found the Gurukul platform visually appealing and professionally designed. They appreciated the clean and structured layout, which made the platform inviting and easy to navigate. However, suggestions were made to enhance font

size and contrast for users with visual impairments and to improve the clarity of error messages and pop-ups. These recommendations underscore the importance of accessibility and user-centered design in educational tools. Furthermore, while the logical flow between different sections of the app was praised, there was a clear call for better labelling or annotation of interactive elements and more detailed tutorials for onboarding first-time users. Enhancing site navigation and onboarding processes can significantly improve the overall UX.

5.2.2 Content Quality and Relevance

The experts were satisfied with the use of RAG leveraging OpenDSA material to generate dynamic, context-aware educational content. They commended the system's ability to reject queries outside its scope and prevent the model from hallucinating or presenting false facts. The verified sources displayed for each response added a layer of trust, educating students on recognizing credible information sources. However, concerns were raised about the depth of learning achieved through the LLM assistant's interactions. The experts recommended integrating a memory component in the LLM assistant to recall previous interactions, leading to a more personalized and contextually enriched experiences for users. This feedback highlights the need for continuous improvement and fine-tuning of LLM capabilities to meet educational standards effectively.

5.2.3 Practical Application and Engagement

The comprehensive set of DSA problems, ranging from basic to advanced, equipped with a live code editor, was lauded for its design and utility. The live editor supports multiple programming languages, providing a real-world coding experience. The interactive environment, with instant feedback on code compilation and execution, enhances learning by

allowing students to learn from mistakes in real-time.

The LLM assistant was highlighted as beneficial for aiding learning without enabling dependency. However, concerns about the assistant's interactions suggested a need for further refinement to ensure it challenges users adequately and promotes critical thinking and problem-solving skills.

5.3 User Study Insights

The user study involving students provided a broader perspective on the effectiveness, usability, and educational impact of Gurukul.

5.3.1 User Perception of the Study Section

Users generally appreciated the study section for its thoroughness and clarity, but there were clear indications that its content could be overwhelming and would benefit from more concise explanations and interactive elements.

Analysis of Responses

Students appreciated the retrieval-augmented generation (RAG) mechanism, which helped them access accurate and contextually relevant information from the OpenDSA material. The clarity and completeness of the information provided by the study section were particularly well-received, with several users noting that the content was thorough and aligned well with their learning needs. However, some users felt that the depth of information provided could be overwhelming at times, suggesting a need for more concise explanations or the option to explore topics in layers of increasing complexity. This layered approach could

allow students to gradually delve deeper into complex topics, which would be particularly beneficial for learners at different stages of their educational journey.

Additionally, while the study section was praised for its ability to deliver trustworthy content, some users expressed a desire for more interactive elements, such as quizzes or flashcards, to reinforce the material learned. These interactive components could help students consolidate their knowledge and ensure they are retaining the information effectively.

The naturalness of the AI's responses in the study section was generally seen as positive, with users finding that the interactions felt smooth and realistic. However, there were a few instances where responses were perceived as somewhat artificial, which could detract from the overall learning experience. This suggests that while the AI is largely effective in maintaining a natural tone, continuous refinement is necessary to ensure that all interactions feel authentic and engaging.

Some users also noted that the system was slower compared to other generative AI tools, which could detract from the overall experience. This feedback points to the importance of optimizing system performance to ensure that users remain engaged and do not become frustrated by delays. Users specifically mentioned that while the clarity and detail provided were valuable, the slower response times could disrupt the learning flow, particularly when they were attempting to quickly access information or clarification on complex topics.

Moreover, there were suggestions for the study section to include more varied types of content and instructional strategies. For example, while the existing material was helpful, some students expressed a desire for more diverse problem sets and additional examples that illustrate key concepts in different contexts. This feedback highlights the need for continuous expansion and diversification of the study material to cater to a wider range of learning preferences and needs.

Lastly, a few users indicated that the current format of the study section, while comprehensive, could benefit from being more descriptive and user-friendly. This includes providing clearer explanations and potentially reformatting the content to be more visually engaging. These enhancements could help reduce cognitive load and make the material more accessible, especially for students who may struggle with more abstract concepts.

5.3.2 User Feedback on the Practice Session

Users generally appreciated the practice section, but there were clear indications that the ease of use and enjoyability could be enhanced.

Analysis of Responses

Users generally appreciated the practice section for its role in applying theoretical knowledge to practical problems, which was a key strength of the Gurukul platform. The integration of AI, which allowed users to ask questions and receive real-time help while practicing, was particularly valued. This functionality provided an interactive and supportive environment that helped bridge the gap between understanding concepts and applying them in real-world scenarios.

However, several users noted that the system's speed was slower compared to other generative AI tools, which could detract from the overall experience. The slower response times were particularly frustrating during practice sessions, where users expected immediate feedback to maintain their momentum and focus. This highlights the importance of optimizing the system's performance to keep users engaged and ensure a smooth learning experience.

In terms of usability, while the practice section was generally well-received, there were clear indications that ease of use and overall enjoyability could be enhanced. Users found the

interface somewhat challenging to navigate, with specific issues related to the size of the coding area and the frequency of popups during code compilation. These popups were seen as disruptive, interrupting the flow of practice and making it difficult for users to concentrate on solving problems. Additionally, some users expressed a desire for clearer and more detailed feedback, particularly when errors occurred. The current feedback mechanism was viewed as insufficient for guiding users through the problem-solving process, which could lead to frustration and reduced learning effectiveness.

There were also suggestions for improving the interactivity of the practice section. Users recommended the inclusion of more gamification elements to make the practice sessions more engaging and fun. For example, introducing challenges, rewards, or competitive elements could increase motivation and make the practice experience more enjoyable. Furthermore, some users proposed enhancing the practice section by cross-referencing problems with real-world applications or assignments. This approach could help contextualize the practice problems, making them more relevant and demonstrating their practical utility.

The handling of different levels of difficulty within the practice section was another point of discussion. While many users felt that the practice section effectively catered to varying levels of complexity—from easy to advanced problems—there was feedback indicating that the harder problems could be better supported. Users suggested that additional guidance or hints for more complex problems could help prevent them from feeling overwhelmed or stuck.

Finally, users provided feedback on the overall design and functionality of the practice section. While some appreciated the thoroughness of the problems and the support provided by the AI, others felt that the practice section could benefit from more independent problem sets that do not rely too heavily on the AI for solutions. This would encourage users to develop their problem-solving skills more robustly and reduce dependency on the AI assistant.

Chapter 6

Conclusion

The Gurukul platform represents a significant advancement in the integration of Large Language Models (LLMs) into computer science education, specifically in the teaching and learning of Data Structures and Algorithms (DSA). By addressing the challenges inherent in traditional DSA education, such as the abstract nature of concepts, the one-size-fits-all approach, and the need for immediate feedback, Gurukul has demonstrated potential to enhance student engagement and learning outcomes.

6.1 Key Findings

1. **Enhanced Learning Experience:** The combination of Retrieval Augmented Generation (RAG) and Guardrails within the Gurukul platform has shown to provide a more interactive and personalized learning experience. The Study section, powered by RAG, offers contextually accurate and relevant information, while the Practice section, enhanced with Guardrails, ensures that students engage in critical thinking and problem-solving without relying on direct answers from the LLM.

2. **Positive User Perception:** Both the user expert review and the user study indicated a generally positive reception towards the platform. Educators appreciated the professional design and structured layout, while students found the practice section beneficial for their learning goals. However, there were notable suggestions for im-

provement, particularly in enhancing the user interface and increasing the engagement level of the interactive elements.

3. **Areas for Improvement:** Despite its strengths, the Gurukul platform has areas that require refinement. The ease of use and enjoyability of the practice section received lower ratings, indicating a need for a more user-friendly interface and engaging elements. Additionally, the depth of learning interactions facilitated by the LLM assistant needs further enhancement to ensure it adequately challenges users and promotes critical thinking.

6.2 Contributions

1. **Bridging Theory and Practice:** Gurukul successfully bridges the gap between theoretical knowledge and practical application in DSA education. By providing a platform where students can practice coding problems with immediate feedback and access contextually relevant information, Gurukul enhances the understanding and application of DSA concepts.

2. **Personalized Learning:** The platform's ability to adapt to individual learning styles and needs through personalized feedback and dynamic content generation marks a significant step towards more effective educational tools.

3. **Engagement and Motivation:** The inclusion of interactive and gamified elements, such as coding challenges and achievement badges, helps to maintain student engagement and motivation, critical factors for successful learning in complex subjects like DSA.

6.3 Future Work

1. **User Interface and Experience Enhancements:** Based on the feedback from the user expert review and user study, future iterations of Gurukul should focus on simplifying navigation, improving error message clarity, and enhancing the overall user interface to be more intuitive and engaging.

2. **Integration of Memory Component in LLM:** Implementing a memory component in the LLM assistant to recall previous interactions can lead to more personalized and contextually enriched experiences for users, improving the depth of learning interactions.

3. **Scalability and Performance Optimization:** As the platform scales to accommodate a larger user base, optimizing the server infrastructure and database management to handle increased demand without compromising performance will be crucial.

4. **Expanded Problem Sets and Adaptive Learning:** Incorporating a broader range of problems with varying difficulty levels and adapting the learning path based on individual progress can further enhance the learning experience. Leveraging adaptive learning technologies can help tailor the content and feedback to each student's needs, ensuring a more effective learning journey.

5. **Comprehensive Accessibility Features:** Ensuring the platform is accessible to users with disabilities by incorporating features such as text-to-speech capabilities, better color contrasts, and comprehensive keyboard navigability can make Gurukul more inclusive.

6. **Longitudinal Studies and Real-World Application:** Conducting longitudinal studies to assess the long-term impact of Gurukul on student learning and performance

in real-world settings can provide deeper insights into its effectiveness. Additionally, exploring its application in different educational contexts and subjects can extend its utility beyond DSA.

6.4 Final Thoughts

Gurukul represents a promising step towards the future of computer science education. By leveraging the capabilities of LLMs and integrating advanced educational technologies, it offers a robust solution to the challenges of teaching and learning DSA. Continuous improvement, guided by user feedback and technological advancements, will ensure that Gurukul remains at the forefront of innovative educational tools, contributing to the development of proficient and adaptive computer science professionals.

Appendices

Appendix A

Implementation Considerations and Code snippets

A.1 System Design

- In this section, we provide a high-level overview of the system design architecture of Gurukul, built on the NextJS framework with a hybrid client-server architecture that supports both server-side rendering (SSR) and static-site generation (SSG).
- Pre-requisites:
  - Functional Requirements:
    * Support for user authentication and management.
    * Content management.
    * Interactive problem solving.
    * Real-time inference.
  - Non-Functional Requirements:
    * High Availability.
    * Security.
    * Performance.

    - Compliance with Educational Standards.
    - Intuitive User Experience.
  - Technology Stack:
    - NextJS for frontend.
    - NodeJS for backend.
    - Supabase PostgreSQL for database management.
    - Clerk for User Authentication.
    - OpenAI as LLM provider.

- User Authentication and Management:
  - User authentication and management are handled by Clerk, a third-party library that supports multiple authentication strategies.
  - Clerk provides various options for setting up a sign-up and sign-in flow.
  - User Flow:
    1. Initiation: The user initiates the authentication process by selecting either the sign-in or sign-up option.
    2. Data Entry: In the sign-up phase, users are prompted to enter essential information such as email, username, and password. They have the alternative option to sign up with Google Provider.
    3. Validation and Verification: Upon form submission, the entered data undergoes validation. For new users, this might include email verification processes to confirm the authenticity of the provided email address.
    4. Authentication: This critical phase leverages the '@clerk/nextjs' package where the system interacts with Clerk's APIs to authenticate the user. Clerk

handles the heavy lifting of verifying credentials, managing sessions, and ensuring security standards are met.

5. Session Management: Upon successful authentication, a session is created for the user. This session is key to maintaining the user's logged-in status and is used to manage user-specific information and preferences.

Listing A.1: Sign-In Component

```
1
2 // Example
3
4 import { type Metadata } from "next"
5 import { SignIn } from "@clerk/nextjs"
6
7 export const metadata: Metadata = {
8   metadataBase: new URL("https://chef-genie.app"),
9   title: "Sign In",
10  description: "Sign in to your account",
11 }
12
13 export default function SignInPage() {
14   return (
15     <div className="mt-2 md:mt-8">
16       <SignIn />
17     </div>
18   )
19 }
```

The sign-in page is designed to be straightforward and user-friendly, focusing solely on the sign-in process. It utilizes the @clerk/nextjs package for handling user authentication, indicating a reliance on Clerk as a third-party service for managing user sessions and authentica-

tion flows. This choice suggests an emphasis on security and efficiency in user management. The metadata associated with this page includes a title and description purely for sign-in purposes, which is standard for authentication pages but also indicates a thoughtful consideration of SEO and user experience.

Listing A.2: Sign-Up Component

```
1
2 import { type Metadata } from "next"
3 import { SignUp } from "@clerk/nextjs"
4
5 export const metadata: Metadata = {
6   metadataBase: new URL("https://chef-genie.app"),
7   title: "Sign Up",
8   description: "Sign up for an account",
9 }
10
11 export default function SignUpPage() {
12   return (
13     <div className="mt-2 md:mt-8">
14       <SignUp />
15     </div>
16   )
17 }
```

Similarly, the sign-up page adopts a minimalist approach, employing the @clerk/nextjs package to facilitate the creation of new user accounts. Like the sign-in page, it includes metadata specifying the page's purpose (in this case, user registration), ensuring clarity for users and search engines alike. The use of Clerk for both sign-in and sign-up functionalities signifies a cohesive and secure strategy for user management within Gurukul, streamlining the process for users to create and access their accounts.

Listing A.3: Authentication Middleware

```
1
2 [
3 frame=lines,
4 framesep=2mm,
5 baselinestretch=1.2,
6 bgcolor=Gray,
7 fontsize=\footnotesize,
8 linenos
9 ]
10 {javascript}
11 import { NextResponse } from "next/server"
12 import { authMiddleware } from "@clerk/nextjs"
13
14 export default authMiddleware({
15   publicRoutes: [
16     "/",
17     "/sign-in(.*)",
18     "/sign-up(.*)",
19     "/dashboard(.*)",
20     "/dashboard",
21     "/sign-out",
22     "/api(.*)",
23     "problem",
24     "problemstable",
25     "/api/submit(.*)",
26     "/api/chat(.*)"
27
28   ],
29   async afterAuth(auth, req) {
30     if (auth.isPublicRoute) {
```

```
31        //  For public routes, we don't need to do anything
32        return NextResponse.next()
33      }
34
35      const url = new URL(req.nextUrl.origin)
36
37      if (!auth.userId) {
38        //  If user tries to access a private route without being authenticated,
39        //  redirect them to the sign in page
40        url.pathname = "/sign-in"
41        return NextResponse.redirect(url)
42      }
43    },
44  })
45
46  export const config = {
47    matcher: ["/((?!.*\\..*|_next).*)", "/", "/(api|trpc)(.*)"],
```

The following code block demonstrates the usage of a middleware file to intercept user requests to before accessing server side functionalities. The matcher configuration ensures the middleware intercepts requests intended for specific routes on the application. This excludes public routes and focuses enforcement on areas containing sensitive data or user-specific functionality.

The afterAuth function is core of the middleware's logic. It checks if a user has been authenticated with the auth.userId keyword.

If users attempt to access a protected route without being logged in, they are redirected to the sign-in page. This prevents unauthorized access to private sections of the site.

A.2 RAG Implementation

For building the Retrieval Augmented Generation powered the application utilizes the Mendable API [59] as the central element of the retrieval mechanism, enabling dynamic querying of the OpenDSA documentation. The app/qa/page.jsx file you provided suggests the incorporation of Mendable AI using a component called MendableInPlace. This component is configured with properties like anonkey, darkMode, and accentColor to tailor the appearance and functionality of the Mendable AI within your application. The specific setup indicates a focus on providing a search and interactive chat interface within the Gurukul app, leveraging Mendable's capabilities to access and query educational content dynamically.

Design Principles

Data Integration and Processing: The RAG system integrates data from OpenDSA, a comprehensive repository of educational materials on data structures and algorithms. It preprocesses this content to optimize retrieval efficiency and accuracy. Dynamic Retrieval: Utilizing state-of-the-art vectorization techniques, the system converts text content into semantic vectors. These vectors are stored in a dedicated database, enabling rapid and precise retrieval based on query relevance. Generative Response Modeling: Once relevant data is retrieved, the system utilizes LLMs to generate educational and context-aware responses that are tailored to the user's queries, ensuring that the information is not only accurate but also pedagogically aligned.

Key Technologies

Vector Database: Utilizes an optimized vector database to store and retrieve information efficiently. LLMs: Employs advanced LLMs for dynamic response generation, ensuring that the

outputs are contextually relevant and educationally beneficial. Interaction Flow Query Input: Students input their queries related to data structures and algorithms. Data Retrieval: The system retrieves relevant information from the vector database. Content Generation: LLMs generate explanatory content that integrates the retrieved information, providing the students with understandable and accurate responses.

Listing A.4: RAG Component implementation

```
1
2 //---> app/qa/page.jsx
3 'use client'
4 import React from 'react';
5 import { MendableInPlace } from "@mendable/search"
6
7 const MyMendableSearchBar = () => {
8
9   const mendableStyles = {
10     width: '1600px', // adjust as needed
11     height: '800px', // adjust as needed
12   };
13
14   const containerStyles = {
15     display: 'flex',
16     justifyContent: 'center',
17     alignItems: 'center',
18     height: '100vh',
19   };
20
21   return (
22     <div style={containerStyles}>
23       <div style={mendableStyles}>
```

```
24          <MendableInPlace anon_key={process.env.NEXT_PUBLIC_MENDABLE_ANON_KEY}
25            darkMode={true}
26            accentColor=" 123456"
27            messageSettings={{ openSourcesInNewTab: true }}
28          />
29        </div>
30      </div>
31    );
32 };
33
34 export default MyMendableSearchBar;
```

The user interface is meticulously designed to provide an intuitive and efficient user experience. The design is centered around the MendableInPlace react component, which is integrated in the NextJS server-side rendered application to offer in-place search functionality. This approach allows. The styling of the MendableInPlace component is carefully crafted to align with the overall aesthetic of the application, employing a combination of width, height and container styles to achieve a visually appealing and functional layout. The use of 'darkMode' and 'accentColor' settings further personalizes the user experience, providing a customizable interface that can adapt to individual user preferences.

In the Mendable Console we can configure the LLM that is responsible for the Generation part. In this example we are making use of the GPT4 model with a System prompt already embedded in the application. We can also control the length of characters the system can output and what starter questions are required to be provided can also be input. The selection of GPT4 was made since it outperforms all current benchmarks in LLM evaluation and has the largest context window. This is especially important in RAG applications as this expands the capabilities for a larger context to be input into the LLM.

A.3 Practice Section

A.3.1 Problems Table

As soon as users click the Practice tab on the homepage, they are forwarded to the Problems Table. The Problems Table is simple table component that renders problems with the following columns:

1. Title - Title of the Problem to solve. 2. Difficulty - A difficulty tag between Easy, Medium, Hard. 3. Tag - A tag associated with the appropriate Data Structure or Algorithm required to solve the question. 4. Acceptance - A number collected from the Leetcode Database signifying the percentage of users on the platform that attempted and successfully solved the question.

A.3.2 Design Components

Live Code Editor: Facilitates real-time coding, testing, and debugging by students, providing a hands-on experience with immediate feedback. Problem Repository: A diverse set of problems varying in complexity is available for practice. Each problem is tagged with specific learning objectives and difficulty levels. Guardrails: Implement guardrails using AI to ensure that while students receive hints and guidance, they do not get direct solutions to problems, fostering learning and understanding rather than rote memorization.

A.3.3 System Functionality

Interactive Problem Solving: Students interact with the system by solving problems, submitting code, and receiving feedback. Adaptive Learning Paths: The system adapts to the

student's learning pace and style, offering problems that gradually increase in difficulty. Instant Feedback and Hints: Provides hints and feedback based on the student's submissions, helping them understand mistakes and learn effectively. Technology Stack Backend: Node.js for handling logic and database operations. Frontend: React powered by NextJS for a dynamic and responsive UI. Database: PostgreSQL for storing user data, submissions, and problem sets. Authentication: Clerk for secure user authentication and session management.

Listing A.5: Fetch Problems function

```
1
2 interface Problem {
3   id: string;
4   Title: string;
5   Difficulty: string;
6   Tags: string;
7   Acceptance: number;
8 }
9
10 const ProblemsTable: FC = () => {
11   const [problems, setProblems] = useState<Problem[]>([]);
12   const [loading, setLoading] = useState<boolean>(true);
13
14   useEffect(() => {
15     const supabase = initSupabase();
16     const tableName = 'LCDB';
17
18     const fetchProblemsFromSupabase = async () => {
19       try {
20         const data = await fetchProblems(supabase, tableName);
21         setProblems(data);
22         setLoading(false);
```

```
23        } catch (error) {
24          console.error("Failed to fetch problems:", error);
25        }
26      };
27
28      fetchProblemsFromSupabase();
29    }, []);
30
31    // code for rendering component
32  };
33
34  export default ProblemsTable;
```

The fetchProblems function encapsulates the logic required to retrieve the problem data from a designated table, referred to within the code as 'LCDB'. It utilizes an asynchronous pattern, returning a promise that either resolves to the data set or throws an exception that is caught and logged, indicating a failure in the data retrieval process.

Listing A.6: Code Editor

```
1
2  import Editor from "@monaco-editor/react";
3  import React, { useState, useEffect } from "react";
4  const CodeEditor = ({ onChange, language, code, theme, action }) => {
5    const [value, setValue] = useState(code || "");
6    useEffect(() => {
7      setValue(code); // Ensures the editor updates when the code prop changes
8    }, [code]);
9
10   const handleEditorChange = (value) => {
11     setValue(value);
```

```
12      onChange("code", value);
13    };
14
15    return (
16      <div className="overlay rounded-md overflow-hidden w-full h-full shadow-4
            xl">
17        <Editor
18          height="85vh"
19          width={`100%`}
20          language={language || "javascript"}
21          value={value}
22          theme={theme}
23          defaultValue=""
24          onChange={handleEditorChange}
25        />
26      </div>
27
28    );
29  };
30  export default CodeEditor;
```

The code editor in use is the Monaco code editor, powerful web-based code editor developed by Microsoft, into a React application. This component is designed to allow users to edit code within a web application, with support for syntax highlighting, code completion, and other advanced editing features provided by the Monaco Editor

Listing A.7: Code Editor Component

```
1
2   const handleSubmit = async () => {
3     setCode(code);
```

```
4     const codeToRun = code;
5     const input = "";
6     const fileName = fileNameMapping[language.value] || 'Main.txt';
7
8     const options = {
9       method : 'POST',
10      url : process.env.NEXT_PUBLIC_ONECOMPILER_URL,
11      headers : {
12        'content-type' : 'application/json',
13        'X-RapidAPI-Key' : process.env.NEXT_PUBLIC_ONECOMPILER_KEY,
14        'X-RapidAPI-Host' : process.env.NEXT_PUBLIC_ONECOMPILER_HOST,
15      },
16      data : {
17        language : language.value,
18        stdin : input,
19        files : [
20          {
21            name : fileName,
22            content : codeToRun
23          }
24        ]
25      }
26    };
27
28    const tableName = 'LCDB';
29
30    let actualOutput = ''; // Declare actualOutput here
31
32  try {
33    const response = await axios.request(options);
34    console.log(response);
```

```
35      handleCompileResult(response.data);
36      const submissionData = {
37        clerk_user_id : user.id,
38        submission_code: code,
39        language: language.value,
40        stdin: response.data.stdin,
41        status: response.data.status,
42        execution_time: response.data.executionTime,
43        tags: tags,
44        difficult: difficulty,
45        acceptance: acceptance,
46        submissionId: id,
47      }
48      setSubmissionData(submissionData);
49
50      actualOutput = response.data.stdout; // Assign value to actualOutput here
51    } catch (error) {
52      console.error("Error running code:", error);
53      return;
54    }
55
56    try {
57      const expectedOutputData = await fetchTestCasesById(supabase, tableName,
          parseInt(id));
58      const expectedOutputs = expectedOutputData[0].Actual_Output; // Adjust
          according to your data structure
59      const testResults = compareTestCases(actualOutput, expectedOutputs);
60      //console.log(actualOutput);
61      //console.log(expectedOutputs);
62      //console.log(testResults);
63      setTestResults(testResults);
```

```
64
65    } catch (error) {
66      console.error("Failed to fetch expected outputs:", error);
67    }
68
69    }
```

The handleSubmit function is responsible for submitting code for execution in the programming environment. It begins by setting the current code state with the provided code and prepares the code to be run, along with an empty input string and a filename derived from a mapping based on the selected language. The function then constructs an options object for an HTTP POST request, specifying the method, URL, headers, and data payload, which includes the programming language, input, and the code file to be executed. This request is made to an external compiler service, as indicated by the NEXTPUBLICONECOMPILERURL environme variable, using Axios for the HTTP request. Upon receiving a response, the function logs the response, processes the compile result, and sets submission data including the user ID, submission code, language, input, status, execution time, and other relevant details. It also captures the actual output from the execution. Subsequently, the function attempts to fetch expected output data for test cases by ID from a Supabase database and compares the actual output with the expected outputs to determine test results. This process facilitates the execution and evaluation of code submissions, providing feedback on the submission's performance and correctness against predefined test cases. The use of environment variables for sensitive information and the structured approach to handling HTTP requests and responses demonstrate best practices in secure and efficient API interactions within a React application.

Listing A.8: Test Case Comparison

```
1
```

```
function compareTestCases(actual, expected) {
    // Function to safely parse JSON strings
    function safeParseJSON(jsonString) {
      try {
        return JSON.parse(jsonString);
      } catch (e) {

        if (jsonString.toLowerCase() === "true" || jsonString.toLowerCase()
            === "false") {
          return jsonString.toLowerCase();
        }
        return jsonString; // Return the original string if parsing fails
      }
    }

    // Helper function to deep compare two entities
    function deepEqual(a, b) {
      if (typeof a !== typeof b) return false;
      if (typeof a !== 'object' || a === null || b === null) return a === b;

      const keysA = Object.keys(a), keysB = Object.keys(b);
      if (keysA.length !== keysB.length) return false;

      for (const key of keysA) {
        if (!keysB.includes(key)) return false;
        if (!deepEqual(a[key], b[key])) return false;
      }

      return true;
    }

```

```
32      // Parse and compare each test case
33      const parsedActual = actual.split('\n').map(safeParseJSON);
34      const parsedExpected = expected.split('\n').map(safeParseJSON);
35
36      let passedTests = 0;
37      const failedTestCases = [];
38
39      parsedActual.forEach((actualValue, index) => {
40        if (index < parsedExpected.length && deepEqual(actualValue,
              parsedExpected[index])) {
41          passedTests++;
42        } else {
43          failedTestCases.push(index + 1);
44        }
45      });
46
47      return {
48        passedTests,
49        totalTests: parsedExpected.length,
50        successRate: `${(passedTests / parsedExpected.length * 100).toFixed(2)
              }%`,
51        failedTestIndices: failedTestCases,
52      };
53    }
```

The compareTestCases function is designed to compare actual test case outputs against expected outputs, providing a detailed analysis of test results. It employs a custom safeParseJSON function to safely parse JSON strings, handling cases where the input might not be valid JSON, such as boolean values represented as strings. This parsing function attempts to parse the input and, if unsuccessful, checks for boolean values before returning the original string. The core

comparison logic is encapsulated in the deepEqual function, which performs a deep comparison of two entities, capable of handling nested objects and arrays. This function checks for type equality, recursively compares object properties, and ensures that both entities have the same keys and values. The compareTestCases function then splits the actual and expected outputs by line, parses each line using safeParseJSON, and iterates over the parsed actual values, comparing them against the corresponding parsed expected values using deepEqual. It tracks the number of passed tests and failed test cases, calculating the success rate and returning an object with the test results, including the number of passed tests, total tests, success rate, and indices of failed tests. This approach allows for a comprehensive evaluation of test case outputs, supporting complex data structures and providing detailed feedback on test outcomes.

Listing A.9: Code Compilation Logic

```
1
2 const handleCompileResult = (result) => {
3     // Check for successful compilation without errors
4     if (result.status === "success" && !result.exception && result.stderr ===
          null) {
5       // Display a success toast with the output and execution time
6       toast.success(`Compiled Successfully!\nOutput:    {result.stdout}\
            nExecution Time:    {result.executionTime}ms`, {
7         position: "top-right",
8         autoClose: 5000,
9         hideProgressBar: false,
10        closeOnClick: true,
11        pauseOnHover: true,
12        draggable: true,
13      });
14    } else if (result.status === "success" && (result.exception || result.
```

```
        stderr !== null)) {
15        // Display an error toast with the error message
16        toast.error(`Compilation Error:\n {result.exception || result.stderr}`,
              {
17          position: "top-right",
18          autoClose: false, // Keep the toast until closed manually for error
                messages
19          hideProgressBar: false,
20          closeOnClick: true,
21          pauseOnHover: true,
22          draggable: true,
23        });
24      } else {
25        // Handle other statuses or unexpected results
26        toast.info("Unexpected result from compilation.", {
27          position: "top-right",
28          autoClose: 5000,
29          hideProgressBar: false,
30          closeOnClick: true,
31          pauseOnHover: true,
32          draggable: true,
33        });
34      }
35    };
```

The handleCompileResult function is designed to process and display feedback to the user based on the result of a code compilation process. It evaluates the result object to determine the outcome of the compilation attempt. If the compilation is successful without any errors or exceptions, and there is no standard error output (stderr), it displays a success toast notification using the toast.success method from the React-Toastify library. This notification

includes the output of the compilation (stdout) and the execution time, providing immediate feedback to the user. In case of a successful compilation but with an exception or standard error output, it displays an error toast notification using the toast.error method, showing the exception or standard error message. For any other status or unexpected results, it displays an informational toast notification using the toast.info method, indicating an unexpected result from the compilation. Each toast notification is configured with various options such as position, auto-close duration, and user interaction behaviors (e.g., closing on click, pausing on hover, draggable) to enhance the user experience and ensure that the feedback is both informative and user-friendly.

Listing A.10: Submission Logic

```
1
2 const handleSubmissionAndTestCases = async (submissionData, testCasesData) =>
    {
3     try {
4
5       const submissionResponse = await supabase
6       .from('submissionstable')
7       .insert([{
8         clerk_user_id: submissionData.clerk_user_id,
9         submission_code: submissionData.submission_code,
10        language: submissionData.language,
11        stdin: submissionData.stdin,
12        status: submissionData.status,
13        execution_time: submissionData.execution_time,
14        submissionId : id,
15        tags: JSON.stringify(submissionData.tags), // Assuming tags is an
              array or object
16        difficulty: submissionData.difficulty,
17        acceptance: submissionData.acceptance,
```

```
18          passed_tests: testCasesData.passedTests,
19          total_tests: testCasesData.totalTests,
20          failed_indices: JSON.stringify(testResults?.failedTestIndices.slice(0,
                -1)),
21           // Assuming an array
22        }]);
23
24        if (submissionResponse.error) {
25          console.error('Error storing submission:', submissionResponse.error);
26          return { success: false, message: 'Error storing submission.' };
27        }
28
29        return { success: true, message: 'Successfully submitted to DB.' };
30      } catch (error) {
31        console.error('Submission error', error);
32        return {success : false, message : 'Submission failed due to an error'};
33      }
34
35    };
36
37    const handleSaveAndSubmit = async () => {
38      await handleSubmissionAndTestCases(submissionData, testResults);
39
40      toast.success("Successfully submitted to DB", {
41        position: "top-right",
42        autoClose: 5000,
43        hideProgressBar: false,
44        closeOnClick: true,
45        pauseOnHover: true,
46        draggable: true,
47        progress: undefined,
```

```
48      });
49
50    };
```

The handleSubmissionAndTestCases function is an asynchronous function designed to handle the submission of code submissions and their corresponding test case results to a Supabase database. It takes two arguments: submissionData, which contains details about the code submission, and testCasesData, which includes information about the test cases and their outcomes. The function attempts to insert a new record into the 'submissionstable' table in the Supabase database, including fields such as the user ID, submission code, language, input, status, execution time, submission ID, tags, difficulty, acceptance, passed tests, total tests, and failed test indices. It uses the Supabase JavaScript client to perform the insertion, ensuring that the data is correctly formatted and matches the expected schema of the database table. If the insertion is successful, the function returns an object indicating success and a message. In case of an error during the insertion, it logs the error and returns an object indicating failure and an error message. The handleSaveAndSubmit function, which calls handleSubmissionAndTestCases, then displays a success toast notification using the React-Toastify library, providing immediate feedback to the user upon successful submission. This approach ensures that both the code submission and its test case results are securely stored in the database, facilitating further analysis and feedback.

LLM Assistant with Guardrails

Listing A.11: Client component of LLM Assistant

```
1
2 'use client';
3
4 import { useChat } from 'ai/react';
```

```
5  import axios from 'axios';
6  import { useUser } from "@clerk/nextjs";
7  import ReactMarkdown from 'react-markdown';
8  import remarkGfm from 'remark-gfm';
9
10 export default function Chat({ userId }) {
11
12   // const fetcher = (messages) => {
13   //   return axios.post('/app/api/chat', { messages});
14   // };
15
16   //const { messages, input, handleInputChange, handleSubmit } = useChat(
         fetcher);
17
18   const { input, handleInputChange, handleSubmit, messages } = useChat({
19     api: "/api/chat",
20     body: {
21       userId,
22     },
23     initialMessages: [],
24   });
25
26   return (
27     <div className="overflow-x-hidden" style={{
28       width: '100%',
29       backgroundColor: ' 2D3748',
30       padding: '24px',
31       borderRadius: '8px',
32       boxShadow: '0 4px 6px -1px rgb(0 0 0 / 10%)',
33       marginBottom: '24px'
34     }}>
```

```
35        <div style={{
36          overflowY: 'auto',
37          overflowX: 'hidden',
38          maxHeight: '256px',
39          marginBottom: '16px',
40          whiteSpace: 'pre-wrap', // Allows markdown to control formatting
41        }}>
42          {messages.map(m => (
43            <div
44              key={m.id}
45              style={{
46                marginBottom: '8px',
47                padding: '8px',
48                borderRadius: '8px',
49                backgroundColor: m.role === 'user' ? '2B6CC0' : '9F7AEA', //
                      AI messages are now purple
50                color: 'white',
51                overflowWrap: 'break-word',
52              }}
53            >
54              <span style={{ fontWeight: 'bold' }}>{m.role}:</span>
55              <ReactMarkdown remarkPlugins={[remarkGfm]}>
56              {m.content}
57            </ReactMarkdown>
58            </div>
59          ))}
60        </div>
61        <form onSubmit={handleSubmit} style={{ marginTop: '16px' }}>
62          <input
63            value={input}
64            placeholder="Say something..."
```

```
65              onChange={handleInputChange}
66              style={{ width: '100%', padding: '8px 12px', borderRadius: '8px',
                    color: 'black' }}
67            />
68            <button
69              type="submit"
70              style={{ marginTop: '8px', padding: '8px 16px', borderRadius: '8px',
                     backgroundColor: ' 9F7AEA', color: 'white' }}
71            >
72              Send
73            </button>
74          </form>
75        </div>
76    );
77  }
```

The provided code snippet above demonstrates the implementation of the Chatbot interface using a React Client Component. It utilizes the useChat hook from ai/react library for managing chat interactions and the OpenAI API for generating responses. This component is designed to facilitate user-assistant interactions, displaying messages in a styled chat interface and allowing users to input and submit messages.

Listing A.12: LLM Assistant implementation in the practice section.

```
1
2 import OpenAI from 'openai';
3 import { OpenAIStream, StreamingTextResponse } from 'ai';
4 import { initSupabase, insertChatInteraction } from '@/utils/supabaseUtils';
5
6 // Get the clerk User Id
7 const openai = new OpenAI({
```

```
8   apiKey: process.env.OPENAI_API_KEY,
9 });
10
11 export const runtime = 'edge';
12 export async function POST(req) {
13
14   //const { messages } = await req.json();
15     const { messages, userId } = await req.json();
16     const latestUserInput = messages[messages.length - 1]?.content;
17     //console.log(userId);
18     // Add the system message as the first element of the messages array
19     const systemMessage = {
20       role: "system",
21       content: "You are a benevolent helping Teaching Assistant in Data
             Structures and Analysis.You only Answer Questions Related to Data
             Structures and Algorithms and Test Cases and Imaginary Test Cases
             and Everything related to the Question at Hand. You DO NOT OUTPUT
             ANY CODE"
22     };
23     messages.unshift(systemMessage);
24     const response = await openai.chat.completions.create({
25       model: 'gpt-4',
26       stream: true,
27       messages,
28     });
29     const currentDate = new Date();
30     const currentDateTimeString = currentDate.toLocaleString();
31
32     const supabase = initSupabase();
33
34     const stream = OpenAIStream(response, {
```

```
35      onCompletion: async (completion) => {
36        // Save the AI's full response to Supabase
37        await insertChatInteraction(supabase, userId, latestUserInput, completion
              );
38
39      }
40    });
41
42    return new StreamingTextResponse(stream);
43  }
```

The server side logic for handling the response is encapsulated in the POST function in the snippet above. It handles incoming chat messages by interacting with the OpenAI API to generate responses. It prepares the messages for the API call, including a system message -

```
1
2  ```
3  You are a benevolent helping Teaching Assistant in Data Structures
4  and Analysis.You only Answer Questions Related to Data Structures and
       Algorithms
5  and Test Cases and Imaginary Test Cases and Everything related to the Question
       at
6  Hand. You DO NOT OUTPUT ANY CODE"
7  ```
```

to guide the AI's responses and streams the AI's responses back to the client. The OpenAIStream function is used to process the streaming response from the OpenAI API, with the onCompletion callback function saving the AI's full response to a Supabase database using insertChatInteraction utility function. This setup enables a real-time, interactive chat experience, leveraging the capabilities of the OpenAI API for generating contextually relevant

responses to user queries.

Listing A.13: Supabase Database Retrieval

```
1
2 import { createClient } from '@supabase/supabase-js';
3 // import { cookies } from 'next/headers';
4
5 export const initSupabase = () => {
6  return createClient(process.env.NEXT_PUBLIC_SUPABASE_URL, process.env.
       NEXT_PUBLIC_SUPABASE_ANON_KEY);
7 };
8
9 export const fetchProblems = async (supabase, tableName) => {
10  const { data, error } = await supabase.from(tableName).select('*');
11  if (error) throw error;
12  return data;
13 };
14
15 export const fetchStarterCode = async (supabase, language, id) => {
16   const { data, error } = await supabase.from('starter_code').select(language)
       .eq('id', id).single();
17   console.log(data);
18   if (error) {
19     console.error('Error fetching starter code:', error);
20     return { error };
21   }
22   return { starterCode: data ? data[language]: null };
23 }
24
25 export const fetchProblemById = async (supabase, tableName, id) => {
26     const { data, error } = await supabase.from(tableName).select('*').eq('id
```

```
        ', id);
27      if (error) throw error;
28      return data;
29     };
30
31  export const insertIntoTable = async (supabase, tableName, data) => {
32      const { data: result, error } = await supabase.from(tableName).insert([
            data]);
33      if (error) throw error;
34      return result;
35  };
36  export const fetchTestCasesById = async (supabase, tableName, id) => {
37      const { data, error } = await supabase.from(tableName).select('*').eq('id
            ', id);
38      if (error) throw error;
39      return data;
40     };
41
42  export const insertChatInteraction = async (supabase, userId, user_input,
        ai_output) => {
43
44      const payload = {
45        role: userId,
46        prompt: user_input,
47        response: ai_output,
48        timestamp: Date.now()
49      };
50
51      return await insertIntoTable(supabase, 'messages', payload);
52    };
```

Appendix B

Evaluation

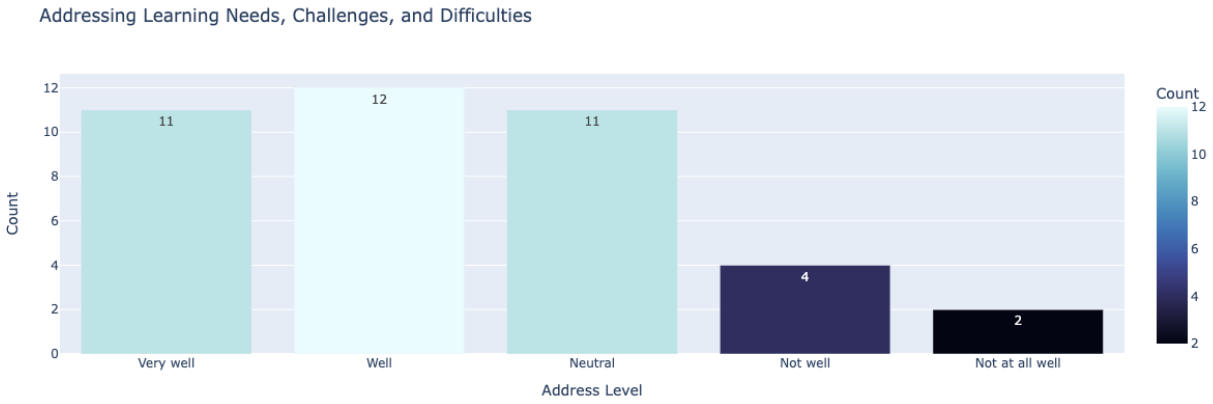

Figure B.1: Adressing learning needs, challenges and difficulties.

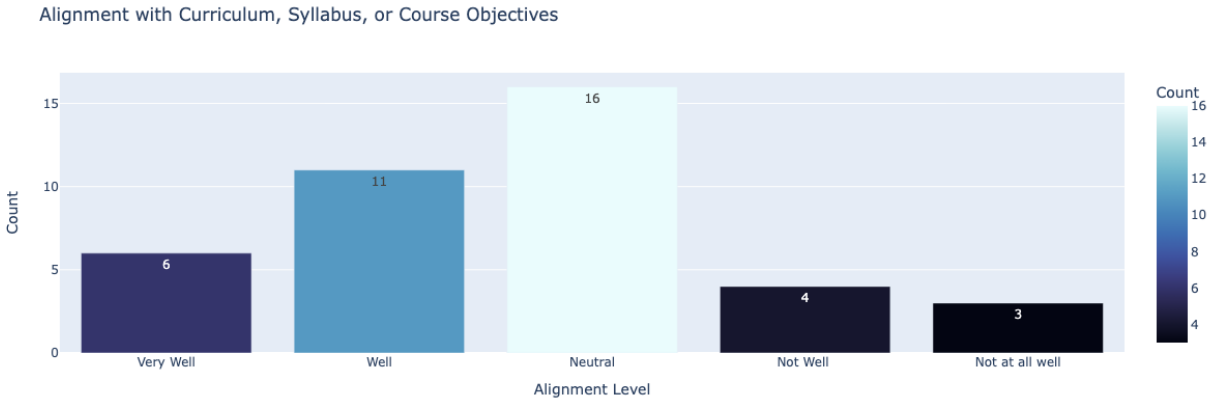

Figure B.2: Alignment with curriculum, syllabus or course objectives.

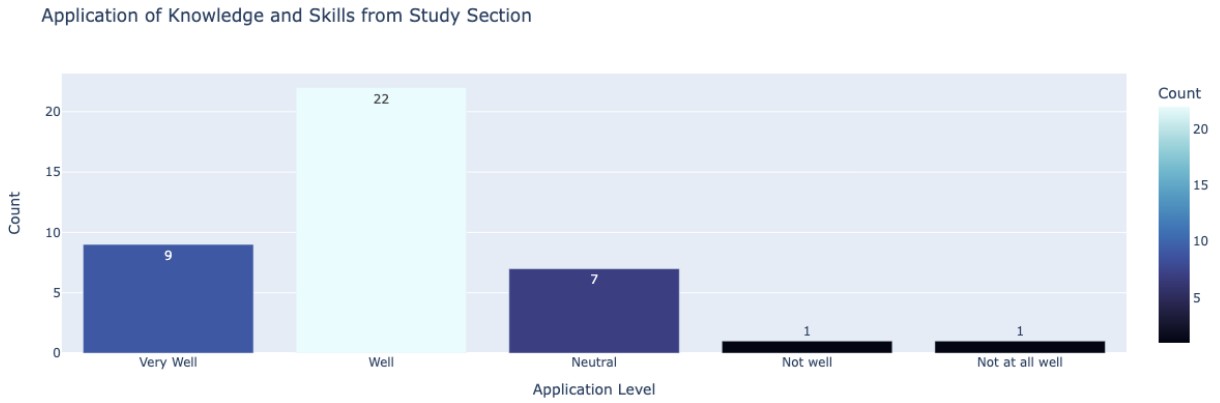

Figure B.3: Application of knowledge and skills from the study section.

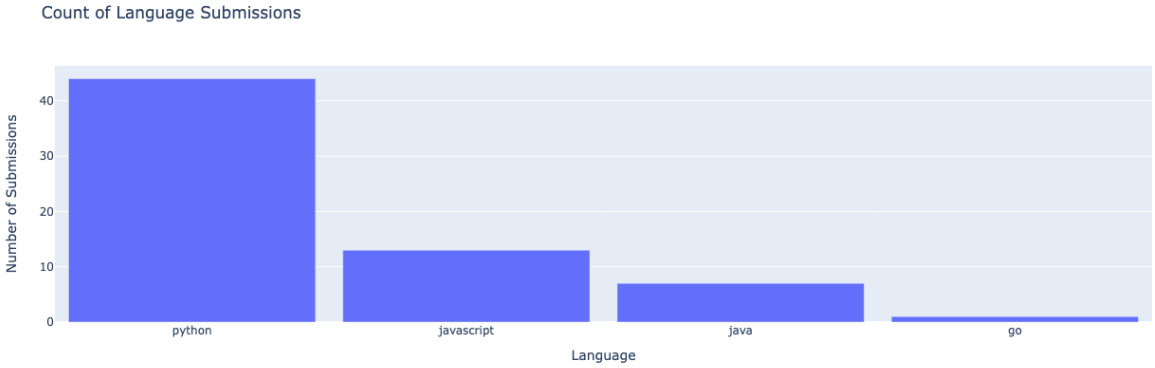

Figure B.4: Count of submissions by language

Figure B.5: Distribution of solution acceptance rates.

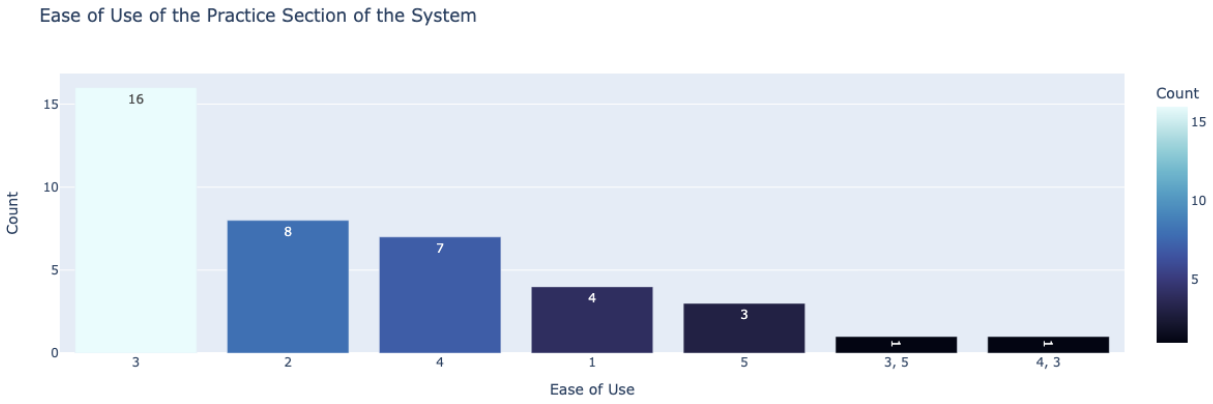

Figure B.6: Ease of use of the practice section.

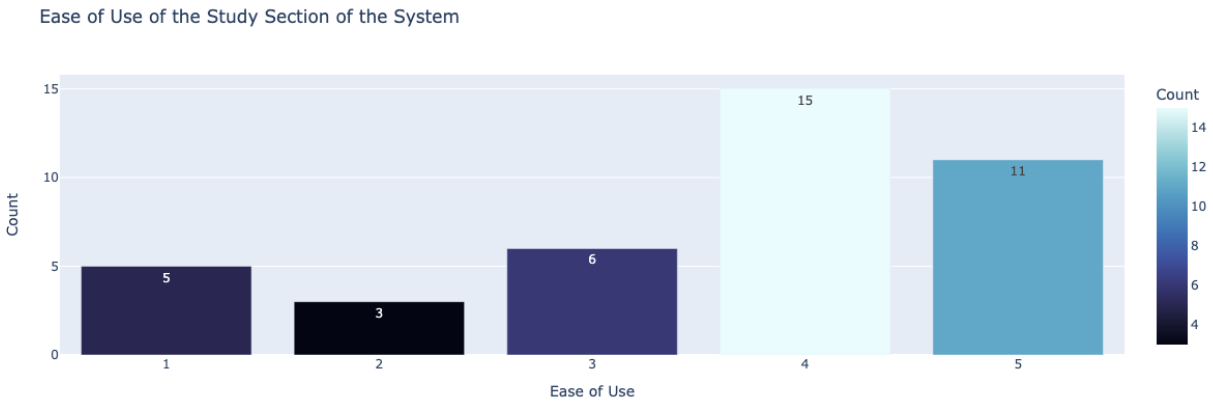

Figure B.7: Ease of use of the study section.

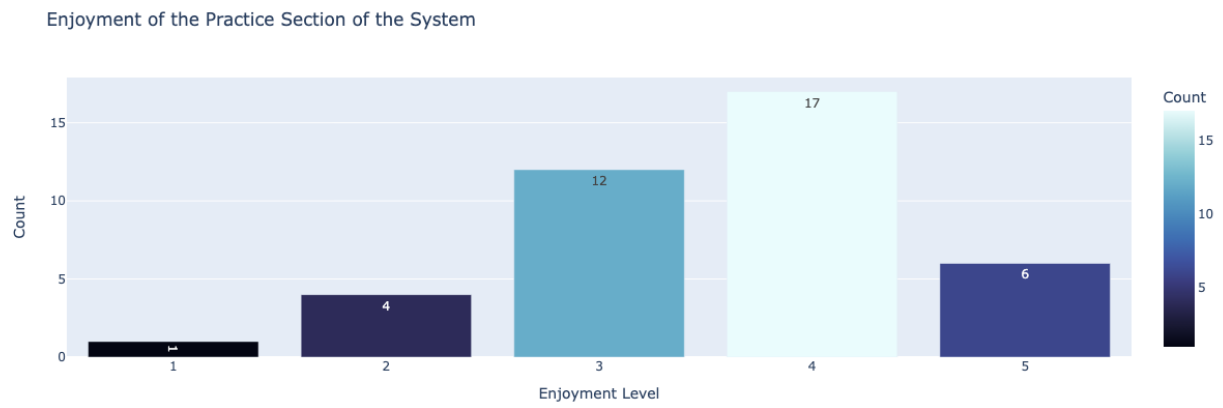

Figure B.8: Enjoyment rating of the practice section.

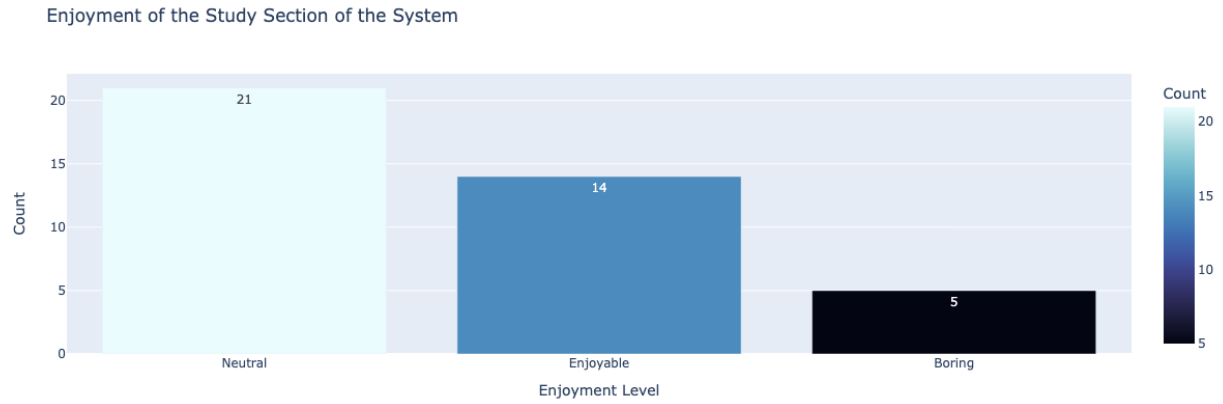

Figure B.9: Enjoyment rating of the study section.

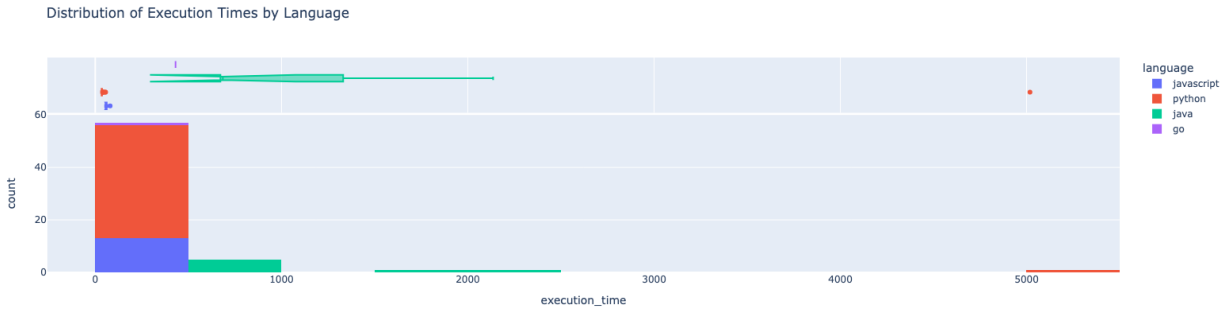

Figure B.10: Distribution of Execution times by language.

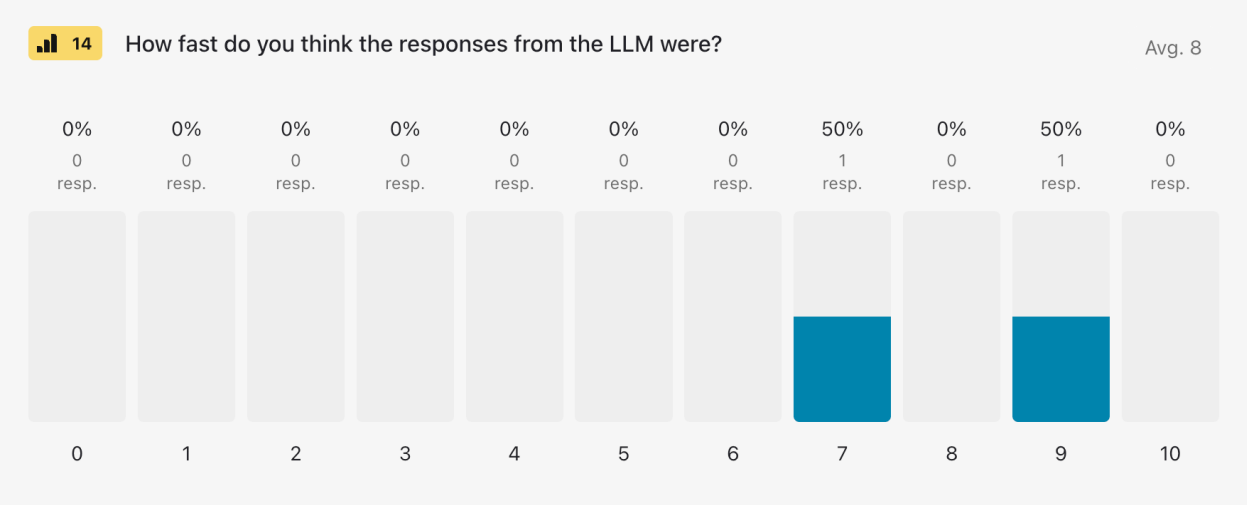

Figure B.11: UX Expert Review - Speed of responses from the LLMs.

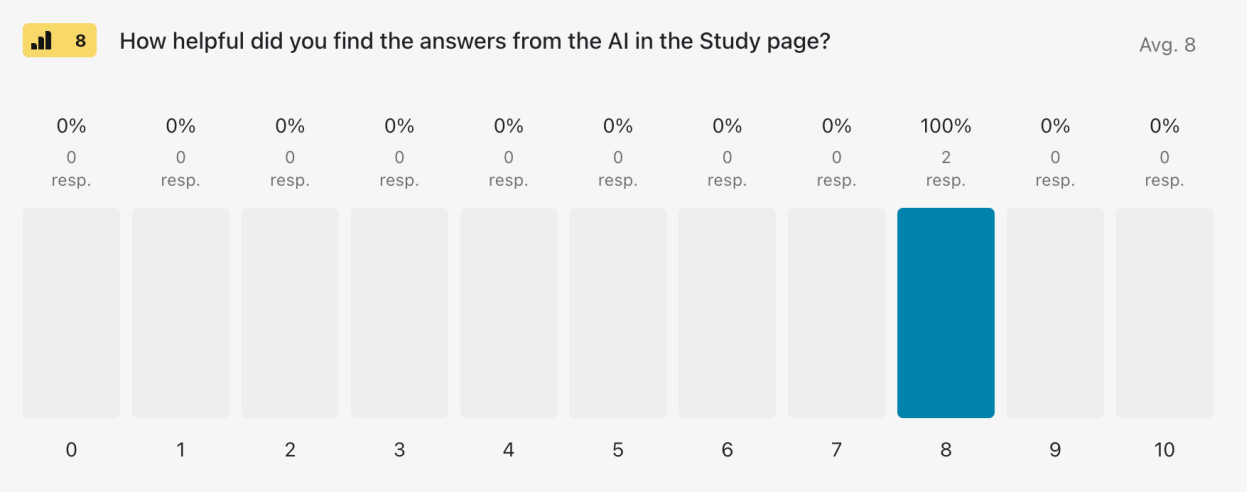

Figure B.12: UX Expert Review - Helpfulness of the RAG AI responses.

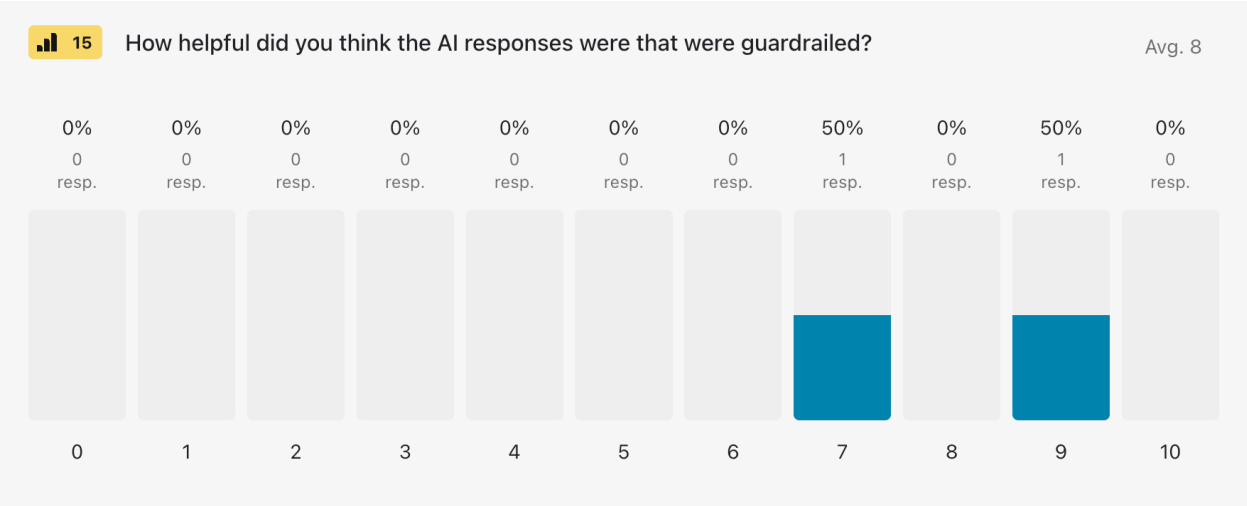

Figure B.13: UX Expert Review - Helpfulness of AI responses that were guardrailed

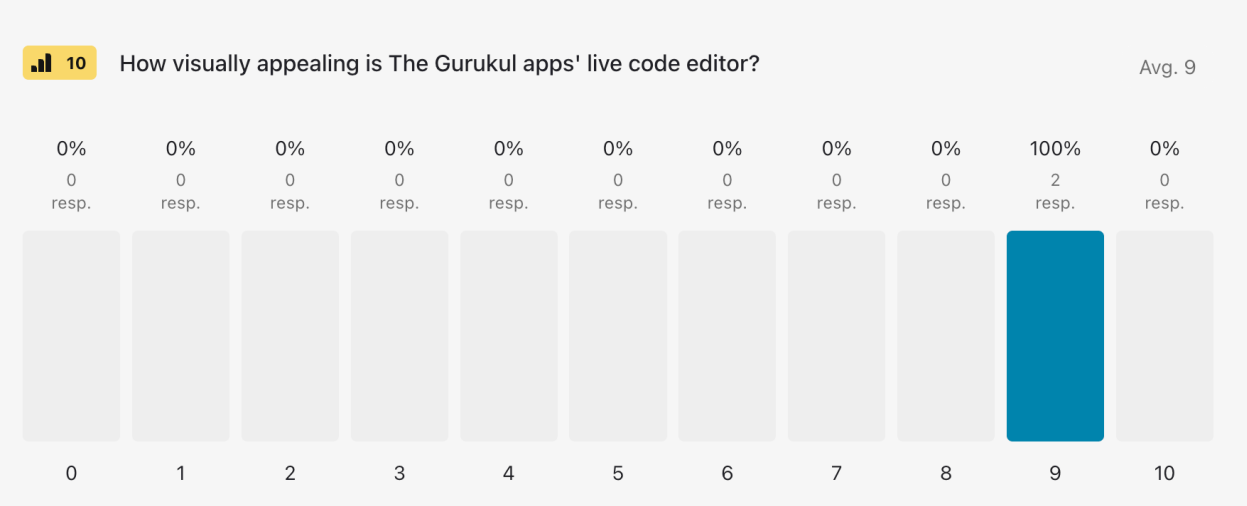

Figure B.14: UX Expert Review - Visual appeal of the application

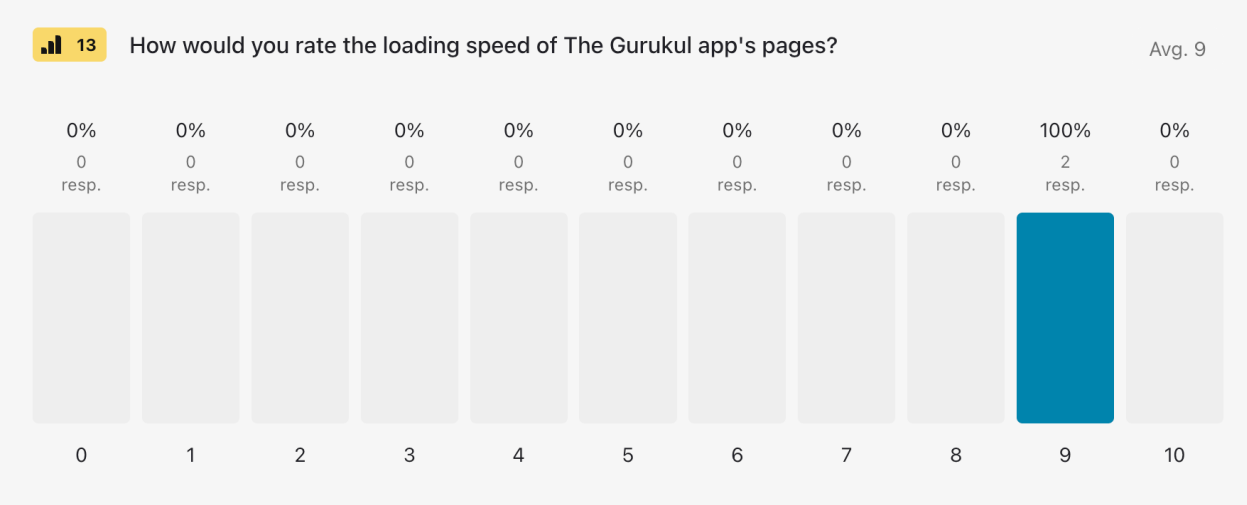

Figure B.15: UX Expert Review - Loading speed of app pages.

Figure B.16: Match with learning style, pace and preferences.

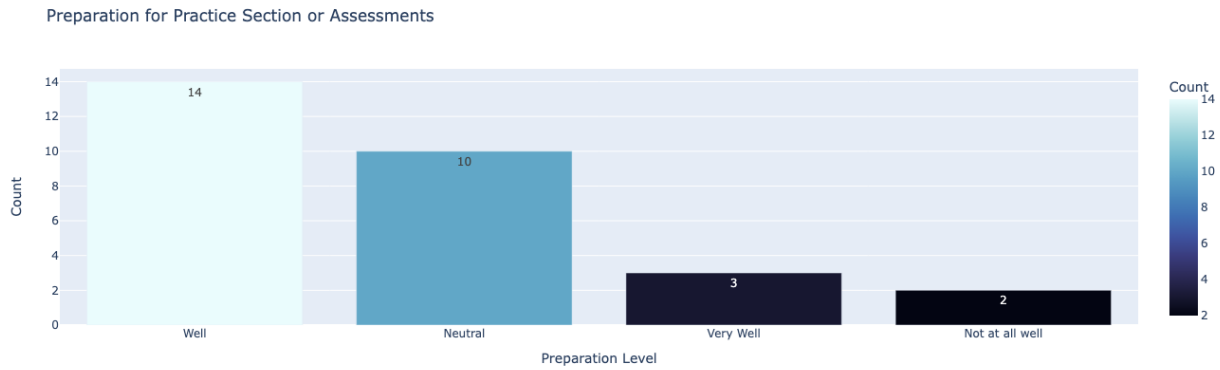

Figure B.17: Preparation rating of practice section for assessments

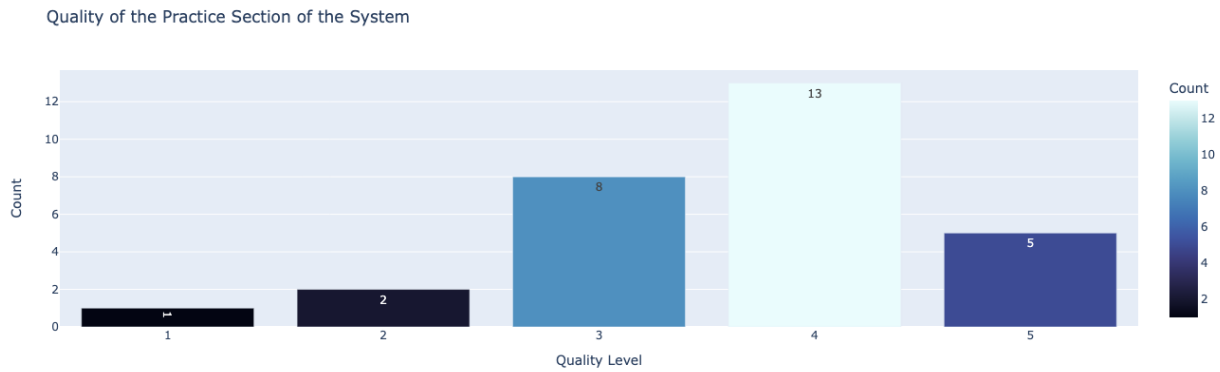

Figure B.18: Quality of the practice section.

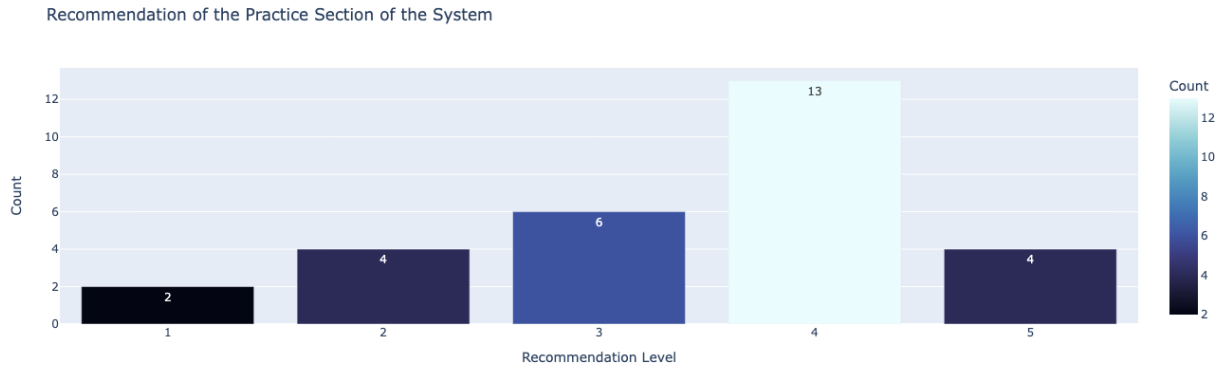

Figure B.19: User recommendation rating for the practice section.

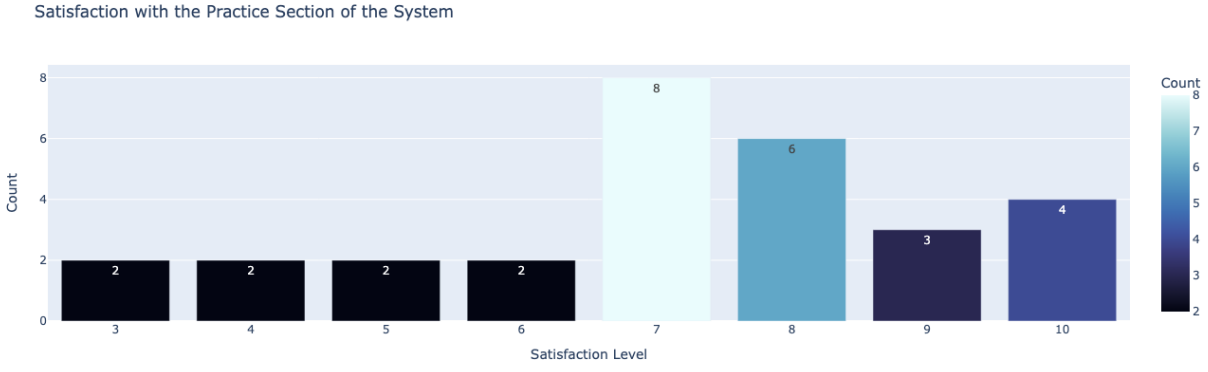

Figure B.20: Satisfaction with the practice section of the system.

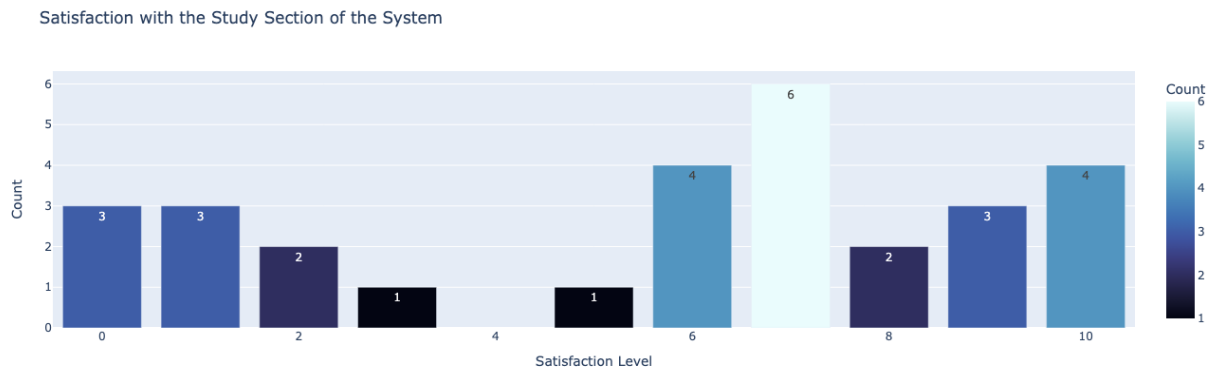

Figure B.21: Satisfaction with the study section of the system.

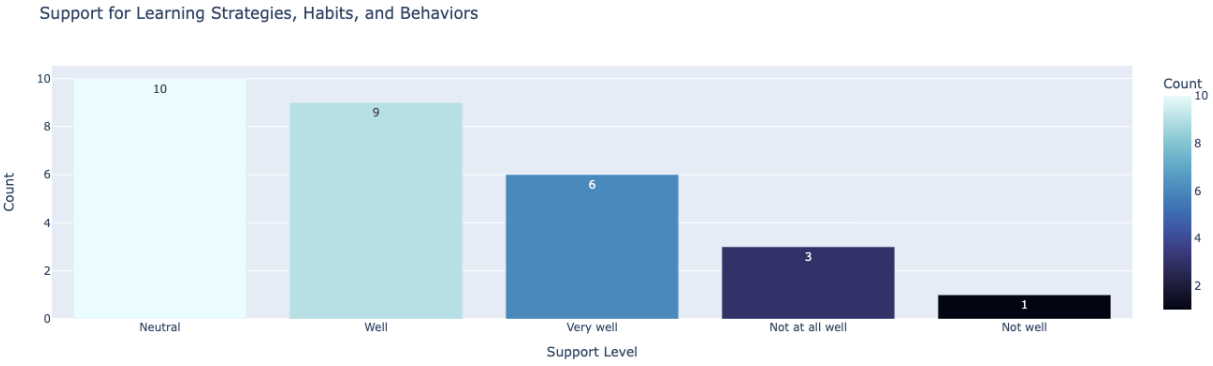

Figure B.22: Support for learning strategoies, habits and behaviors

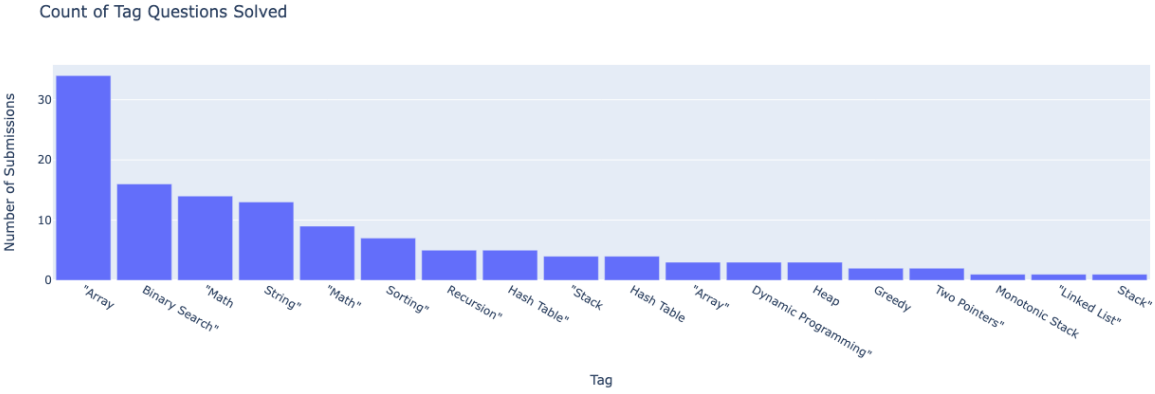

Figure B.23: Number of problems solved by tags.

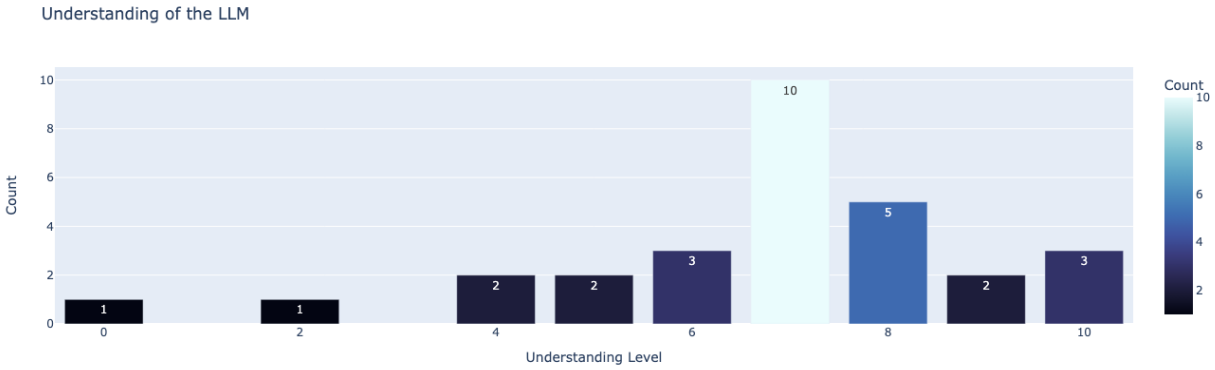

Figure B.24: Rating of understanding of the LLM on a scale of 1-10

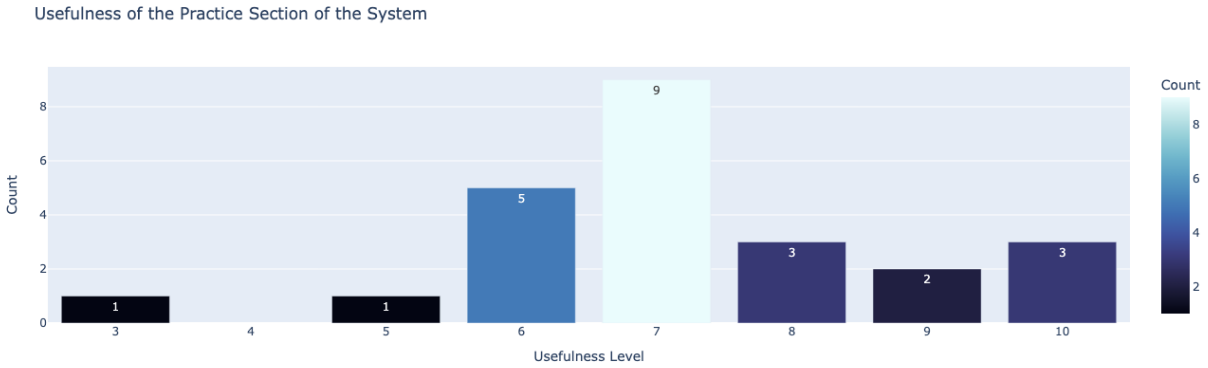

Figure B.25: Usefulness rating of the practice section of the system on a scale of 1-10

Appendix C

Screenshots

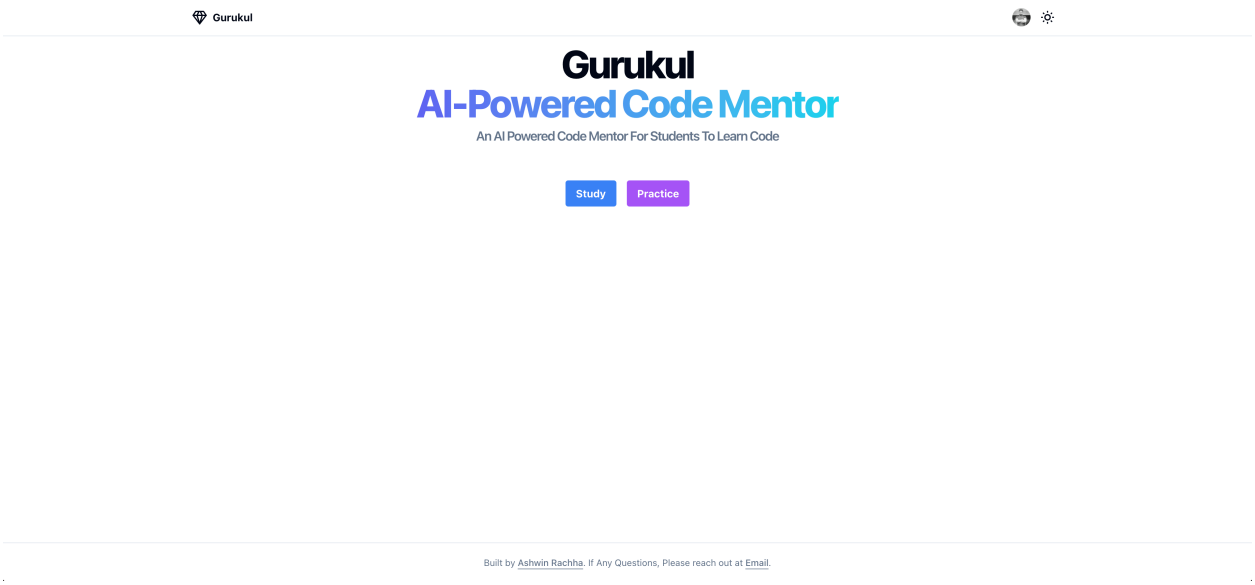

Figure C.1: Landing page of the application.

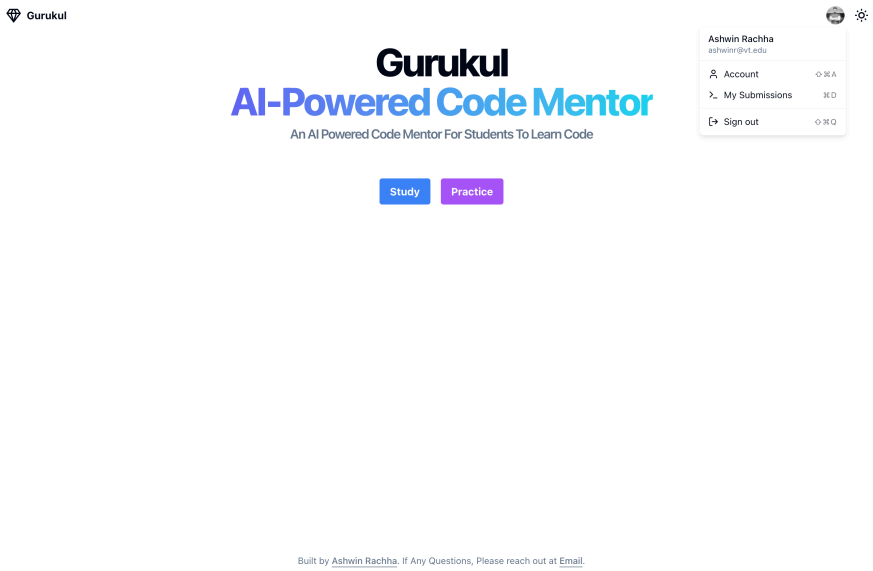

Figure C.2: Landing page with profile popup.

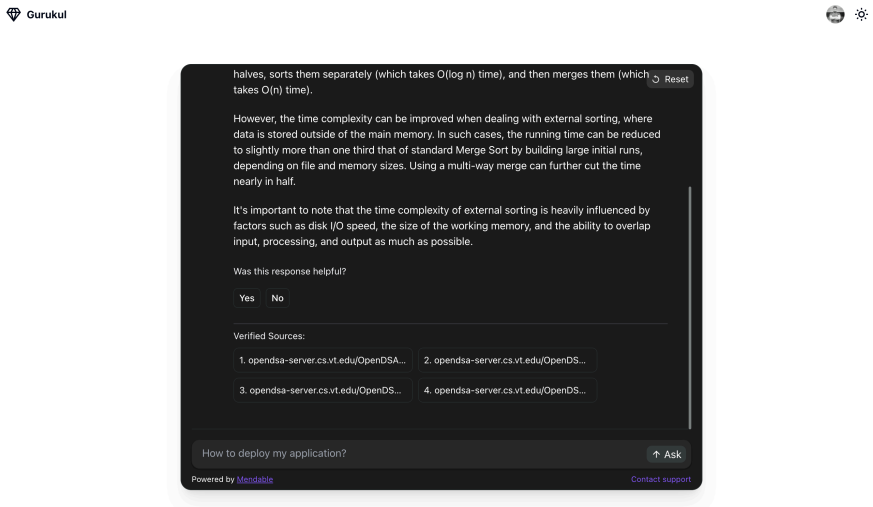

Figure C.3: Study section RAG demonstration - 3

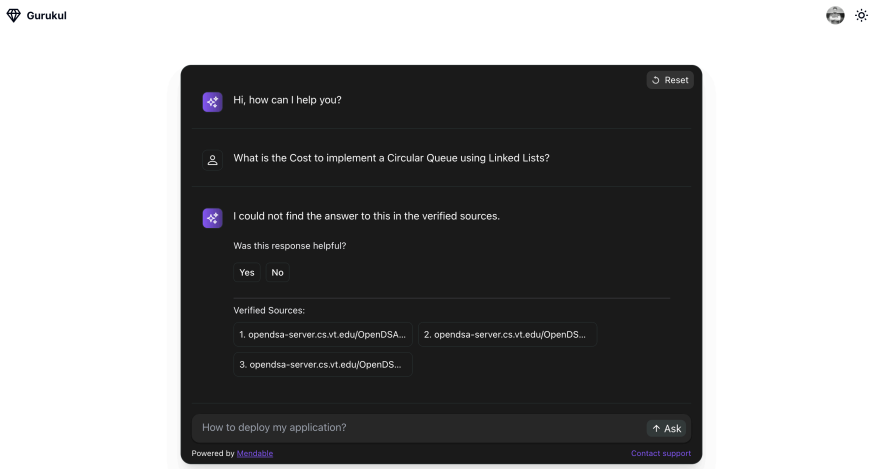

Figure C.4: Study section RAG demonstration - 4.

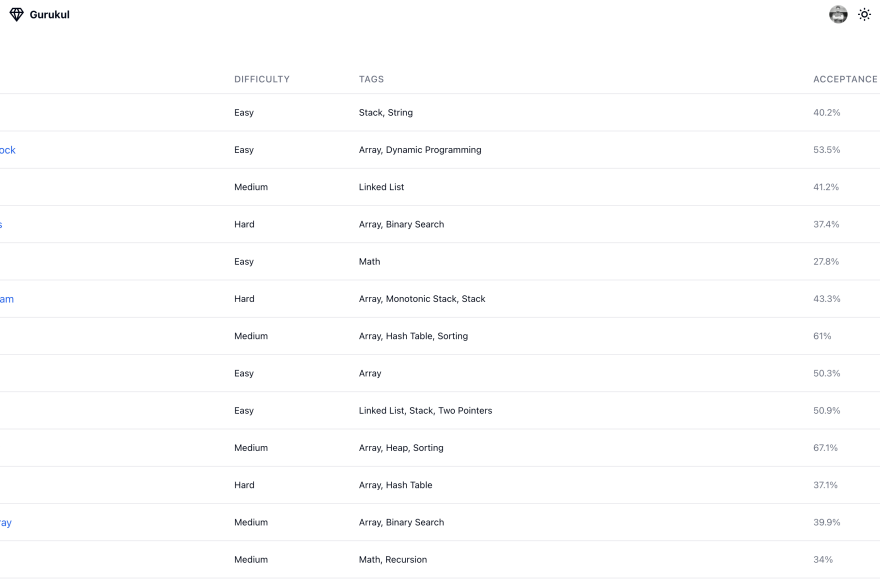

Figure C.5: Problems table component

Figure C.6: Problems table with dark mode for accessibility

Appendix D

IRB Approval Letter

Division of Scholarly Integrity and Research Compliance
Institutional Review Board
North End Center, Suite 4120 (MC 0497)
300 Turner Street NW
Blacksburg, Virginia 24061
540/231-3732
irb@vt.edu
http://www.research.vt.edu/sirc/hrpp

MEMORANDUM

DATE: March 1, 2024

TO: Mohammed Saad Mohamed Elmahdy Seyam, Ashwin Kedari Rachha

FROM: Virginia Tech Institutional Review Board (FWA00000572)

PROTOCOL TITLE: Assessing the Efficacy of Interactive Learning Environments in Coding Education: A Comprehensive Evaluation of the Gurukul Learning Application

IRB NUMBER: **24-065**

Effective January 23, 2024, the Virginia Tech Human Research Protection Program (HRPP) determined that this protocol meets the criteria for exemption from IRB review under 45 CFR 46.104 (d) category(ies) 2(ii).

Ongoing IRB review and approval by this organization is not required. This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made and there are questions about whether these activities impact the exempt determination, please submit an amendment to the HRPP for a determination.

This exempt determination does not apply to any collaborating institution(s). The Virginia Tech HRPP and IRB cannot provide an exemption that overrides the jurisdiction of a local IRB or other institutional mechanism for determining exemptions.

All investigators (listed above) are required to comply with the researcher requirements outlined at:

https://secure.research.vt.edu/external/irb/responsibilities.htm

(Please review responsibilities before beginning your research.)

PROTOCOL INFORMATION:

Determined As:	**Exempt, under 45 CFR 46.104(d) category(ies) 2(ii)**
Protocol Determination Date:	**January 23, 2024**

ASSOCIATED FUNDING:

The table on the following page indicates whether grant proposals are related to this protocol, and which of the listed proposals, if any, have been compared to this protocol, if required.

Bibliography

[1] Google, "Google trends - explore data for "ai in education"," 2023.

[2] A. Yadav, S. Gretter, S. Hambrusch, and P. Sands, "Expanding computer science education in schools: understanding teacher experiences and challenges," *Computer science education*, vol. 26, no. 4, pp. 235–254, 2016.

[3] S. Sentance and A. Csizmadia, "Computing in the curriculum: Challenges and strategies from a teacher's perspective," *Education and information technologies*, vol. 22, pp. 469–495, 2017.

[4] A. E. Wolf, "The plagiarism question of ai: How teachers have responded to llms in the classroom," 2024.

[5] S. Balloccu, P. Schmidtová, M. Lango, and O. Dušek, "Leak, cheat, repeat: Data contamination and evaluation malpractices in closed-source llms," *arXiv preprint arXiv:2402.03927*, 2024.

[6] K. Z. Zhou, Z. Kilhoffer, M. R. Sanfilippo, T. Underwood, E. Gumusel, M. Wei, A. Choudhry, and J. Xiong, "" the teachers are confused as well": A multiple-stakeholder ethics discussion on large language models in computing education," *arXiv preprint arXiv:2401.12453*, 2024.

[7] OpenAI, "Chatgpt (mar 14 version)," 2023. Large language model.

[8] Anthropic, "Claude: Large language model by anthropic," 2024. Accessed: 2024-08-18.

[9] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample,

"Llama: Open and efficient foundation language models." https://ai.facebook.com/blog/large-language-model-llama-meta-ai/, 2023. Accessed: 2024-08-18.

[10] OpenAI, "Gpt-4 (specific version)," 2023. Large multimodal model.

[11] M. Abdullayeva and Z. M. Musayeva, "The impact of chat gpt on student's writing skills: An exploration of ai-assisted writing tools," in *International Conference of Education, Research and Innovation*, vol. 1, pp. 61–66, 2023.

[12] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, *et al.*, "Chatgpt for good? on opportunities and challenges of large language models for education," *Learning and individual differences*, vol. 103, p. 102274, 2023.

[13] J. G. Meyer, R. J. Urbanowicz, P. C. Martin, K. O'Connor, R. Li, P.-C. Peng, T. J. Bright, N. Tatonetti, K. J. Won, G. Gonzalez-Hernandez, *et al.*, "Chatgpt and large language models in academia: opportunities and challenges," *BioData Mining*, vol. 16, no. 1, p. 20, 2023.

[14] W. Gan, Z. Qi, J. Wu, and J. C.-W. Lin, "Large language models in education: Vision and opportunities," in *2023 IEEE International Conference on Big Data (BigData)*, pp. 4776–4785, IEEE, 2023.

[15] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, "Challenges and applications of large language models," *arXiv preprint arXiv:2307.10169*, 2023.

[16] M. J. Timms, "Letting artificial intelligence in education out of the box: educational cobots and smart classrooms," *International Journal of Artificial Intelligence in Education*, vol. 26, no. 2, pp. 701–712, 2016.

[17] V. Devedžić, "Web intelligence and artificial intelligence in education," *Educational Technology & Society*, vol. 7, no. 4, pp. 29–39, 2004.

[18] K. Flamm, *Creating the Computer: Government, Industry, and High Technology*. Washington, DC, USA: Brookings Institution Press, 1988.

[19] M. Campbell-Kelly, *Computer: A History of the Information Machine*. Evanston, IL, USA: Routledge, 2018.

[20] M. M. L. Cairns, "Computers in education: The impact on schools and classrooms," in *Life Schools Classrooms*, (Singapore), pp. 603–617, Springer, 2017.

[21] B. Whitby, *Artificial Intelligence: A Beginner's Guide*. Oxford, U.K.: Oneworld, 2008.

[22] B. Coppin, *Artificial Intelligence Illuminated*. Boston, MA, USA: Jones and Bartlett, 2004.

[23] K. Ahmad, W. Iqbal, A. El-Hassan, J. Qadir, D. Benhaddou, M. Ayyash, and A. Al-Fuqaha, "Data-driven artificial intelligence in education: A comprehensive review," *IEEE Transactions on Learning Technologies*, 2023.

[24] M. Lievertz, "Artificial intelligence in education," in *Artificial Intelligence and Machine Learning for Business for Non-Engineers*, pp. 125–140, CRC Press, 2019.

[25] L. Chen, P. Chen, and Z. Lin, "Artificial intelligence in education: A review," *Ieee Access*, vol. 8, pp. 75264–75278, 2020.

[26] R. Hamal *et al.*, "Integrating ai technologies in higher education: a systematic review of the literature," *TechTrends*, vol. 64, no. 1, pp. 13–29, 2020.

[27] I. Vasiliev and A. Eremeeva, "Evaluating the risks of artificial intelligence in learning environments," *Journal of Education and Learning*, vol. 12, no. 3, pp. 45–58, 2023.

[28] W. Holmes, K. Porayska-Pomsta, K. Holstein, E. Sutherland, T. Baker, S. B. Shum, O. C. Santos, M. T. Rodrigo, M. Cukurova, I. I. Bittencourt, *et al.*, "Ethics of ai in education: Towards a community-wide framework," *International Journal of Artificial Intelligence in Education*, pp. 1–23, 2022.

[29] M. Webb, T. Bell, N. Davis, Y. J. Katz, N. Reynolds, D. P. Chambers, M. M. Sysło, A. Fluck, M. Cox, C. Angeli, *et al.*, "Computer science in the school curriculum: Issues and challenges," in *Tomorrow's Learning: Involving Everyone. Learning with and about Technologies and Computing: 11th IFIP TC 3 World Conference on Computers in Education, WCCE 2017, Dublin, Ireland, July 3-6, 2017, Revised Selected Papers 11*, pp. 421–431, Springer, 2017.

[30] J. Kay, M. Barg, A. Fekete, T. Greening, O. Hollands, J. H. Kingston, and K. Crawford, "Problem-based learning for foundation computer science courses," *Computer Science Education*, vol. 10, no. 2, pp. 109–128, 2000.

[31] R. Tamassia, "Implementing algorithms and data structures: An educational and research perspective: (invited presentation)," in *International Symposium on Algorithms and Computation*, pp. 4–8, Springer, 1998.

[32] W. Steingartner, J. Eged, D. Radaković, and V. Novitzká, "Some innovations of teaching the course on data structures and algorithms," in *2019 IEEE 15th International Scientific Conference on Informatics*, pp. 000389–000396, IEEE, 2019.

[33] S. Su, E. Zhang, P. Denny, and N. Giacaman, "A game-based approach for teaching algorithms and data structures using visualizations," in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pp. 1128–1134, 2021.

[34] Y. Tang, Z. Xu, and Q. Li, "Probe into e-education of "algorithm and data structure","

in *Information Engineering and Applications: International Conference on Information Engineering and Applications (IEA 2011)*, pp. 1576–1581, Springer, 2012.

[35] S. MacNeil, A. Tran, J. Leinonen, P. Denny, J. Kim, A. Hellas, S. Bernstein, and S. Sarsa, "Automatically generating cs learning materials with large language models," *arXiv preprint arXiv:2212.05113*, 2022.

[36] T. Krüger and M. Gref, "Performance of large language models in a computer science degree program," in *European Conference on Artificial Intelligence*, pp. 409–424, Springer, 2023.

[37] M. Tran, "Prompt engineering for large language models to support k-8 computer science teachers in creating culturally responsive projects," in *Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 2*, pp. 110–112, 2023.

[38] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein, and J. Leinonen, "Experiences from using code explanations generated by large language models in a web software development e-book," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pp. 931–937, 2023.

[39] C. Glynn, E. Hed, A. Pexa, T. Pohlmann, I. Rahal, and R. Hesse, "Caet: Code analysis and education tutor," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 2*, pp. 1656–1657, 2024.

[40] V. A. Hamaniuk, "The potential of large language models in language education," *Educational Dimension*, vol. 5, pp. 208–210, 2021.

[41] S. MacNeil, J. Kim, J. Leinonen, P. Denny, S. Bernstein, B. A. Becker, M. Wermelinger,

A. Hellas, A. Tran, S. Sarsa, *et al.*, "The implications of large language models for cs teachers and students.," in *SIGCSE (2)*, p. 1255, 2023.

[42] S. Joseph, "Large language model-based tools in language teaching to develop critical thinking and sustainable cognitive structures.," *Rupkatha Journal on Interdisciplinary Studies in Humanities*, vol. 15, no. 4, 2023.

[43] Z. Jiang, F. F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig, "Active retrieval augmented generation," *ArXiv*, vol. abs/2305.06983, 2023.

[44] K. Wang, J. Ramos, and R. Lawrence, "Chated: A chatbot leveraging chatgpt for an enhanced learning experience in higher education," *arXiv preprint arXiv:2401.00052*, 2023.

[45] H. Li, Y. Su, D. Cai, Y. Wang, and L. Liu, "A survey on retrieval-augmented text generation," *ArXiv*, vol. abs/2202.01110, 2022.

[46] S. S. Manathunga and Y. A. Illangasekara, "Retrieval augmented generation and representative vector summarization for large unstructured textual data in medical education," *ArXiv*, vol. abs/2308.00479, 2023.

[47] M. Liffiton, B. E. Sheese, J. Savelka, and P. Denny, "Codehelp: Using large language models with guardrails for scalable support in programming classes," in *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, pp. 1–11, 2023.

[48] A. Y. Sun, V. Nair, E. Schumacher, and A. Kannan, "Conscendi: A contrastive and scenario-guided distillation approach to guardrail models for virtual assistants," *arXiv preprint arXiv:2304.14364*, 2023.

[49] T. Rebedea, R. Dinu, M. N. Sreedhar, C. Parisien, and J. Cohen, "Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails," pp. 431–445, 2023.

[50] C. A. Shaffer, "Opendsa: An interactive etextbook for computer science courses," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pp. 5–5, 2016.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[52] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[53] "Supabase: Real-time database querying, authentication, and messaging." https://supabase.com, 2023.

[54] "Next.js." https://nextjs.org, 2023.

[55] LeetCode, "Leetcode: The world's leading online programming learning platform," 2024. Accessed: 2024-08-18.

[56] https://4bqza6d30ed.typeform.com/report/o4PkW41Z/Hs3SBX3ZVxxhUrhZ, "Report: Gurukul user study survey results," 2024.

[57] B. Caldwell, M. Cooper, L. G. Reid, G. Vanderheiden, W. Chisholm, J. Slatin, and J. White, "Web content accessibility guidelines (wcag) 2.0," *WWW Consortium (W3C)*, vol. 290, no. 1-34, pp. 5–12, 2008.

[58] "Gurukul vt - forms." https://gurukulvt.vercel.app/forms, 2024.

[59] "Mendable.ai: Ai-powered mental health for the workplace." https://www.mendable.ai, 2023.