FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Conversational Agents for Enhancing Education Through Scenario-Based Learning

José Pedro Teixeira Ramos

Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Gil Manuel Magalhães de Andrade Gonçalves

July 16, 2025

Conversational Agents for Enhancing Education Through Scenario-Based Learning

José Pedro Teixeira Ramos

Mestrado em Engenharia Informática e Computação

July 16, 2025

Resumo

A Aprendizagem Baseada em Cenários (Scenario-Based Learning – SBL) coloca os estudantes em situações reais ou simuladas para desenvolver competências através da resolução de problemas, tomada de decisões e reflexão. Baseia-se em etapas sequenciais com feedback em tempo real.

Apesar do seu potencial, a SBL exige uma forte intervenção do professor, especialmente na análise das interações e no fornecimento de feedback personalizado, o que limita a sua escalabilidade. A Inteligência Artificial (IA) pode automatizar estas tarefas, permitindo feedback contínuo e adaptado, mesmo em turmas numerosas, sem depender da presença constante de um professor.

Esta dissertação apresenta o design, desenvolvimento, testes e avaliação de um agente de SBL baseado em IA, capaz de fornecer feedback dinâmico e contextual em tempo real. O sistema foi construído com componentes-chave como um ficheiro de cenário estruturado em JSON, engenharia de prompts em camadas, Retrieval-Augmented Generation (RAG), rastreamento do histórico de interações e uma interface intuitiva. Estes elementos permitiram que a IA apoiasse os estudantes de forma adaptativa em diferentes pontos de decisão e trajetórias de aprendizagem. Na revisão da literatura, mais de 1.000 estudos foram inicialmente identificados, mas por meio de filtragem sistemática e avaliação de relevância, o número foi reduzido para 20 estudos principais que informaram o design e a implementação.

Foram testadas várias configurações para equilibrar desempenho, flexibilidade e custo. Técnicas como previsão do próximo passo, prompts encadeados e autorefinamento melhoraram o raciocínio da IA. A configuração padrão foi a mais robusta, e a sem auto-refinamento destacou-se pela eficiência, com mínima perda de qualidade. A troca de modelos reduziu os custos operacionais em até 57%.

A eficácia da solução foi avaliada com estudos junto de 10 utilizadores, no qual resolveram 27 cenários. Os resultados mostraram que 60% preferiram o agente de IA face aos métodos tradicionais e 70% consideraram-no altamente útil (4 em 5 ou mais). A maioria (70%) sentiu que o feedback se ajustava ao seu nível de compreensão, e 80% ficaram satisfeitos com o tempo de resposta e a qualidade da interação. O envolvimento também aumentou, com os utilizadores a relatarem maior imersão, motivação e interesse nos cenários.

A avaliação pedagógica e de usabilidade revelou uma melhoria nas taxas de conclusão à medida que os estudantes se adaptavam à interface e ao estilo do agente. Embora os primeiros cenários tivessem menor sucesso, as tarefas intermédias atingiram até 78%, indicando uma curva de aprendizagem positiva. O agente mostrou-se eficaz quando ajustado ao nível dos utilizadores, sendo valorizado como apoio ao ensino, embora sem substituir o professor.

Abstract

Scenario-Based Learning (SBL) places students in real or simulated situations to develop skills through problem-solving, decision-making, and reflection. It is based on sequential steps with real-time feedback.

Despite its potential, SBL requires strong teacher intervention, especially in analyzing interactions and providing personalized feedback, which limits its scalability. Artificial Intelligence (AI) can automate these tasks, enabling continuous and tailored feedback, even in large classes, without relying on the constant presence of a teacher.

This dissertation presents the design, development, testing, and evaluation of an AI-based SBL agent capable of providing dynamic and contextual feedback in real time. The system was built with key components such as a JSON-structured scenario file, layered prompt engineering, Retrieval-Augmented Generation (RAG), interaction history tracking, and an intuitive interface. These elements enabled the AI to adaptively support learners at different decision points and learning trajectories.

Various configurations were tested to balance performance, flexibility, and cost. Techniques such as next-step prediction, chained prompts, and self-refinement improved the AI's reasoning. The default configuration was the most robust, and the one without self-refinement stood out for its efficiency, with minimal loss of quality. Switching models reduced operational costs by up to 57%. In reviewing the literature, more than 1,000 studies were initially identified, but through systematic filtering and relevance assessment, this was narrowed down to 20 key studies that informed the design and implementation.

The effectiveness of the solution was assessed through user studies with a total of 10 participants, of which 27 scenarios where solved. The results showed that 60% preferred the AI agent to traditional methods and 70% found it highly useful (4 out of 5 or more). The majority (70%) felt that the feedback was tailored to their level of understanding, and 80% were satisfied with the response time and quality of the interaction. Engagement also increased, with users reporting greater immersion, motivation, and interest in the scenarios.

The pedagogical and usability evaluation revealed an improvement in completion rates as students adapted to the interface and style of the agent. Although the first scenarios were less successful, intermediate tasks reached up to 78%, indicating a positive learning curve. The agent proved to be effective when adjusted to the level of the users, being valued as a teaching support, although not replacing the teacher.

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) provide a global framework to achieve a better and more sustainable future for all. It includes 17 goals to address the world's most pressing challenges, including poverty, inequality, climate change, environmental degradation, peace, and justice.

The specific Sustainable Development Goals mentioned have the following names:

SDG 4 Ensure inclusive and equitable quality education and promote lifelong learning opportunities for all.

SDG	Target	Contribution	Performance Indicators and Metrics
4	4.1	Promotes interactive, engaging, and personalized learning through AI-driven Scenario-Based Learning (SBL), supporting improved learning outcomes and knowledge application.	Student performance improvement (e.g., pre- and post-test scores), task completion rate, scenario progression time.
	4.4	Enables skills development in complex domains (e.g., engineering, medicine, law) by simulating realistic, decision-based scenarios using LLM-based agents.	Number of learners completing scenarios, self-assessed skills acquisition, scenario feedback loops.
	4.5	Supports equitable access by adapting content to diverse learner needs, including different learning speeds and styles.	Interaction success rate across varied learner profiles, feedback inclusion for diverse users, dropout reduction.
	4.a	Enhances the learning environment with responsive, inclusive, and adaptive AI tutors that provide real-time support.	User satisfaction surveys, teacher evaluations, usability scores, accessibility metrics.

Acknowledgements

First and foremost I would like to express my sincere gratitude to João Reis and Liliana Antão for helping me build this thesis from the beginning, and for believing in me even when times were rough. Their contribution to the creation of this thesis was subpar and without their help and motivation it would be impossible to make this thesis.

I am also deeply thankful to my supervisor, Gil Manuel Magalhães de Andrade Gonçalves, for providing guidance and the opportunity to research this topic.

This work was supported by Medtiles, whose financial assistance is gratefully acknowledged.

I would also like to give acknowledgment to the Faculty of Engineering of the University of Porto for all the resources and research material provided.

On a personal note, I must express my gratitude to my family and friends, without them none of what I made in the last five years would be possible, they were with me since the beginning of my journey, they were there in the lows and highs and for that I am deeply thankful.

I would like also give special thanks to Gonçalo Rodrigues, for his presence and motivation throughout all this years.

José Pedro Teixeira Ramos

"Every new beginning comes from some other beginning's end."

Seneca

Contents

List of Figures

List of Tables

Listings

Abbreviations and Symbols

AI	Artificial Intelligence
ALP	Adaptive Learning Platform
API	Application Programming Interface
ATI	Aptitude Treatment Interaction
CoT	Chain-of-Thought
LLM	Large Language Models
MCQ	Multiple-Choice Question
MVC	Model-View-Controller
NLP	Natural Language Processing
RAG	Retrieval-Augmented Generation
SBL	Scenario-Based Learning
SPECS	Specifications
STEM	Science, Technology, Engineering, and Mathematics
UI	User Interface
VM	Virtual Machine

Chapter 1

Introduction

1.1 Context

In recent years, the methods of education have gone through significant transformations that were driven by technological advancements. Although today's generation are accustomed to using the Internet and smartphones [3], there is a lack of teaching methods made specifically for them [27], it is often said that current teaching methods, were students are made to "sit and listen", aren't very good at transferring valuable learning skills, and rather non-traditional methods should be used [30].

The rapid evolution of AI has opened many opportunities to improve the learning experience. With AI tools like ChatGPT being able to gather one million users in just five days [13], it is safe to say to it was well received by the public. While traditional learning methods often fail to accommodate the learners needs, paces, and preferences of individual students, AI can learn from the learner and adapt itself for optimal learning potential. Although LLMs such as ChatGPT can be useful for educational purposes, they were not designed purely for education and may lack some important features for peak learning performance. So, what does LLM miss? SBL:

> "is the use of scenarios as a vehicle for the teaching and learning process, providing students with the opportunity to learn from and apply their learning to realistic experiences." [12]

Studies show that creating situations that mimic realistic experiences can help self-efficacy in students [8], and with the help of AI the potential to create dynamic, adaptive learning environments that respond in real-time to student input, offering personalized guidance and support. By integrating AI into education, we can provide learners with customized experiences that are not only engaging but also pedagogically sound, allowing for deeper understanding and more effective knowledge retention.

1.2 Motivation

This research stems from the need to bridge the gap between traditional instructional methods and the increasing demand for individualized learning solutions. As education continues to shift toward digital platforms, it is crucial to develop tools that can accommodate a wide range of learners, each with unique abilities, learning speeds, and preferences. Current educational frameworks often lack the capacity to deliver such personalized experiences, leaving some students disengaged or overwhelmed.

By leveraging the power of AI, this educational agent aims to guide students through carefully designed educational scenarios, acting as a tutor capable of providing real-time assistance, answering questions and adapting to each student's progress. The goal is to create a more engaging, effective, and interactive learning experience that goes beyond static, one-size-fits-all approaches.

This work is being developed in collaboration with Medtiles, a company that is actively working with LLMs within the context of SBL. Their ongoing efforts in developing such systems have contributed directly to the conception of this thesis. The collaboration is also able to provide valuable insight about AI driven SBL systems.

1.3 Problem and Research questions

Even though there are many reasons supporting the use of SBL in conjunction with LLM, this task presents several challenges in its implementation. One of the main problems is delivering real-time, personalized feedback and guidance in a classroom setting where the teachers attention is limited. It is also important to note that AI tools for education already exist, but lack pedagogical foundation, which leads to a tool that fail to align with effective learning methods.

This research seeks to overcome these challenges by developing an LLM-based educational agent specifically tailored to support SBL environments.

The defined research questions are:

1. How can AI be effectively integrated into educational scenarios to provide real-time, personalized feedback?

2. In what ways can the integration of AI in education enhance personalized learning and student engagement?

3. How can the performance and effectiveness of an LLM-based educational agent be evaluated in terms of student outcomes and learning engagement?

4. Which assessment methods (qualitative and quantitative) best describes the pedagogical value, usability, and efficacy of an AI-driven SBL agent in a real-world scenario?

5. Which reasoning mechanisms and prompt engineering techniques can improve the agent's flexibility in response to various learning paths and SBL decision points?

Validation Strategy: To ensure that these questions are properly evaluated, different metrics will be analyzed. These metrics will contain quantitative and qualitative methods.

Quantitative metrics may include response time, accuracy of feedback, student performance improvement, number of individualized interventions or feedback loops provided by the AI for each student, Time spent on tasks, task completion rates, and interaction rates with the AI agent, interaction success rate and time taken for students to progress through increasingly challenging scenarios with AI assistance.

Qualitative methods involve student satisfaction, teacher evaluations, student reflections and case studies, including detailed analysis of individual students learning journeys with the AI.

1.4 Thesis structure

This thesis is comprised of five chapters. We start with the Introduction, which is the current chapter. Here, we present a brief overview of the evolution of of educational methods driven by technology and AI, revealing the gaps and potential of SBL, highlight the growing need for intelligent educational tools that adapt to individual learners. We also introduce the main problems and research questions addressed in this work. This chapter is structured into four subsections: Context, Motivation, Problem Statement and Research Questions.

The second chapter, entitled Literature Review & State of the art, has the content of all the literature chosen, its methods and the filtering criteria applied. Additionally, the state of the art is introduced later in the chapter, containing the most relevant parts of the literature review for this thesis, demonstrating their significance and objectives.

In the third chapter, entitled Solution and Implementation, as the name implies we present a comprehensible approach, methodology for the practical implementation of the thesis and all the steps taken to develop the software in question.

In chapter four, named Results and Analysis, presents how we test and evaluate our solution. As the workings of the application have already been described, the focus here is on the testing methodology, the obtained results, and their analysis.

Finally we reach the final chapter, the Conclusions, where we summarize the key insights gained from the work carried out throughout this thesis and reflect on the main findings. This chapter also discusses in detail the limitations encountered during the development process and presents potential directions for future research.

Chapter 2

Literature Review & State of the art

2.1 Introduction

This chapter presents a systematic literature review to examine existing research on SBL, personalized learning with AI, the application of LLMs in education, and the integration of pedagogical principles with LLMs. The goal is to identify relevant studies and analyze the state-of-the-art research, thereby guiding the development of an intelligent educational LLM-based agent that enhances SBL. It is also important to understand that, although a systematic methodology was employed, some references included in this thesis may not have undergone the full systematic review process.

2.2 Methodology

2.2.1 Literature Search Strategy

A structured and systematic search was conducted in two major databases, the ACM Digital Library and IEEE Xplore. The primary aim was to ensure broad and comprehensive coverage of relevant articles, conference papers, and reviews. To achieve this search queries were carefully designed to target the intersection of SBL, AI, LLMs, and their pedagogical integration. The search queries were:

SBL in Education (Q1)

("scenario-based learning" OR "SBL") AND (education OR pedagogy) AND (effective OR impactful) ("scenario-based learning" OR "SBL") AND (student OR learner) AND (engagement OR participation OR "learning outcomes" OR "knowledge application")

Personalized Learning with AI (Q2)

("adaptive learning" OR "personalized learning" OR "customized learning") AND (AI OR "artificial intelligence") AND ("learning environments" OR "education systems") (AI OR "artificial intelligence") AND (tutoring OR "student guidance")

LLM in Education (Q3)

("large language models" OR LLM OR "generative AI") AND (education OR "learning systems") AND ("real-time feedback" OR "dynamic feedback")

("large language models" OR LLM OR "generative AI") AND ("educational systems" OR "learning platforms") AND (tutoring OR "student guidance")

Pedagogical Integration with LLM (Q4)

("pedagogical integration" OR "education theories") AND ("large language models" OR LLM OR "generative AI") AND (learning OR tutoring)

("pedagogy" OR "teaching strategies") AND ("large language models" OR LLM OR "generative AI") AND ("learning outcomes" OR "student engagement")

("pedagogical models" OR "educational strategies") AND ("large language models" OR LLM OR "generative AI") AND (learning OR "student support")

LLM and SBL Integration (Q5)

("scenario-based learning" OR "SBL") AND ("large language models" OR LLM OR "generative AI")

("large language models" OR LLM OR "generative AI") AND ("scenario-based learning" OR "learning scenarios") AND ("real-time guidance" OR "adaptive feedback")

2.2.2 Eligibility Criteria

The eligibility criteria were defined to ensure both relevance and academic rigor. Only high-quality reviews articles, excluding works in progress, and published in the past four years (2020-2024), were included. A study was considered eligible if it satisfied all of the following criteria:

- The study explicitly discusses SBL in the context of education or pedagogy.
- The study explores personalized or adaptive learning using AI, with particular emphasis on LLMs.
- The research incorporates pedagogical theories or educational strategies that are integrated with AI or LLMs.
- Study proposes frameworks, models, or presents empirical results that demonstrate effective, impactful, or engaging educational outcomes resulting from the integration of AI and LLMs.

2.2.3 Screening and Selection Process

The screening process followed a multi-stage approach:

1. **Initial Search and Title Screening:** A preliminary search was conducted using the defined queries on the ACM Digital Library and IEEE Xplore databases, yielding a total of 1501, shown in Table 2.1.

Table 2.1: Summary of Query Results across ACM and IEEE

Query	Q1	Q2	Q3	Q4	Q5	Total
ACM	204	200	200	184	40	828
IEEE	65	200	148	236	24	673
Total	269	400	348	420	64	1501

2. **Remove duplicates and restrict year of publication:** After the initial search all duplicates were removed and the year of publication was restricted to papers only from the past four year (2020-2024), resulting in a list of 810 papers.

3. **Abstract and Keyword Screening:** The remaining studies were screened based on their abstracts and keywords, ensuring alignment with the research objectives, and were labeled with "Maybe" or "Promising" tags depending on its relevance. Following this review, 45 studies were tagged as "Maybe" and 40 as "Promising".

4. **Full-Text Review:** In the final stage, full texts of the selected studies were reviewed in depth. Studies that did not contribute to understanding SBL, personalized learning with AI, or pedagogical integration with LLMs were excluded. Ultimately, 20 studies were retained for inclusion.

2.3 State of the Art

This chapter provides a comprehensive review of the current state of the art in AI, with a particular focus on educational technologies powered by LLMs, as well as the pedagogical approach of SBL. The goal is to build the current landscape of relevant research.

- In section 2.4 "SBL and Its Integration with Technology" an introductory overview of SBL is presented, highlighting its core features and practical applications in real-world educational contexts.

- Section 2.5 "Transforming Education Through AI", begins with a brief introduction to AI, followed by subsections that describe key advancements and defining features of AI-driven educational systems.

- Section 2.6 "Pedagogical Approaches to Adaptive E-learning Systems" explores four pedagogical approaches that can be effectively applied within adaptive e-learning systems.

- Section 2.7 "LLMs in Education" will be the main section of this state of the art. It explains how LLMs are reshaping education by detailing their applications, underlying mechanisms, associated software, and the challenges and limitations involved.

- Finally section 2.8 "Conclusions and Identified Research Gap", presents the concluding reflections and outlines the research gap identified through this review.

2.4 SBL and its Integration with Technology

SBL is a practice used to reduce the gap between theoretical knowledge and practical application. Although it is not sufficient on its own to teach students effectively, when combined with the guidance of a teacher or tutor, it offers a highly practical approach to exploring and applying real-world challenges [12].

This pedagogical approach relies on several core elements essential to its effectiveness. Scenarios have to be challenging with realistic and complex problems, the narrative needs to be relatable and engaging, the decision-making to encourage learner autonomy and critical thinking, role-playing such as taking roles of individuals or groups, and a certain level of authenticity must be upheld [12]. There have been many application that use SBL, but there is one in particular that is worth mention, Edmodo, an educational social network. It was employed in a scenario based learning group and reflective group for comparing the results of SBL vs non-SBL approach [20], the results indicated that there were no significant differences between the two groups in the before the test, demonstrating that both groups were equally matched. However, their test scores revealed that SBL was a step ahead of the reflective group, showing a distinct advantage to the learner who received SBL.

2.5 Transforming Education Through AI

The integration of AI into Education is transforming traditional teaching methods and established practices, opening new possibilities for personalized, efficient, and engaging learning environments. But what is AI? According to [29]:

> "Artificial intelligence (AI) is technology that enables computers and machines to simulate human learning, comprehension, problem solving, decision making, creativity and autonomy."

This means that AI systems are capable of adapting and learning from constantly evolving information, reshaping their behavior accordingly. As a result, they can personalize the classroom environment to meet the individual needs of each student, ultimately enhancing learning outcomes.

2.5.1 Personalized Learning

AI-powered systems are increasingly capable of delivering personalized educational experiences by dynamically adjusting content to the needs, learning styles, and pace of individual learners [2], [4], [14], [6], [19], [22], [23]. This level of personalization is virtually impossible in traditional classroom settings, were the teacher has to teach classes with twenty or maybe even hundreds of learners. Time constraints make it unfeasible to identify each student's unique learning needs or provide real-time feedback on an individual basis. In contrast, machine learning algorithms analyze vast quantities of data - such as quiz scores, interaction patterns, and engagement metrics - to generate personalized learning paths. This ensures that students receive materials suited to their

unique proficiency levels. Furthermore, through continuous assessment and adaptation, these AI systems are able to deliver real-time feedback and support tailored to a diverse range of learning profiles.

Such systems are known as ALP. They employ algorithms and data-driven techniques to customize educational content, pacing, and feedback to the individual needs and learning styles of each student. By continuously analyzing performance and adapting in real-time, ALP can create a personalized learning experience that maximizes engagement and effectiveness.

2.5.2 Enhanced Learning Outcomes

The main motivation for incorporating AI tools in education is for students to enhance learning outcomes across various disciplines. In a study conducted by [2], ChatGPT was employed as a support tool for a tutoring software to help STEM students in solving programming problems. The research methodology involves three key phases. First, a course curriculum is designed to integrate ChatGPT as a support tool for both students and instructors during class sessions. Second, a specialized platform is developed to serve as an intermediary between students and the AI instructor, delivering personalized educational materials tailored through pre-embedded prompts. Finally, an experiment is conducted to compare the learning outcomes and satisfaction levels of participants in the AI-integrated course with those enrolled in conventional programming courses. During the experiment phase, participants were divided into two groups of comparable academic ability. Group A followed traditional teaching methods, while group B used the AI-supported tutoring system. The findings revealed that Group B outperformed Group A after becoming familiar with the platform, demonstrating the potential of AI-integrated tools to enhance educational performance.

2.5.3 Role Transformation in Teaching

The integration of AI into education will require teachers to adapt their roles in response to the automation of several routine tasks. Many responsibilities traditionally handled by educators—such as grading, taking attendance, and lesson planning, can now be automated, prompting a shift in focus from the role of a traditional lecturer to that of a facilitator or mentor.

In this new context, teachers will be increasingly responsible for clarifying doubts, guiding students through complex subjects, and offering deeper pedagogical insights. Importantly, the goal of implementing AI in education is not to replace educators, but rather to enhance student engagement and make teachers more available and responsive to individual learning needs [4], [5], [7].

2.5.4 Gamification and Engagement Through AI

AI in education introduces elements of gamification that transform traditional learning into an interactive and engaging experience [1]. ALP, for example, dynamically adjust content difficulty in response to student performance, mimicking gaming mechanics such as level-ups and challenges.

This approach fosters a sense of achievement and motivates learners to persist in their studies. Features like real-time feedback and rewards - like badges or auditory applause - have been shown to help student engagement and promote self-directed learning [11] [9].

Gamification not only increases motivation, but also improves the overall learning experience by creating a positive feedback loop. For instance, when students retry and succeed in solving problems after multiple attempts, their sense of accomplishment is reinforced, thereby fostering resilience and confidence [11].

2.6 Pedagogical Approaches to Adaptive E-learning Systems

According to [9], adaptive e-learning systems usually use one of four types of pedagogical approaches, each one addressing diverse different educational needs.

2.6.1 Macro-Accommodative Approach

Macro-accommodation refers to the adaptation of learning experiences by providing multiple alternatives for curriculum content, learning objectives, and delivery methods, based on students' individual profiles, including cognitive styles and prior knowledge. Using this approach, students can be grouped through standardized assessments, allowing for a basic level of personalization. Overall, macro-accommodation is most effective in large-scale instructional settings, such as foundational courses offered at the university level.

2.6.2 ATI Approach

ATI leverages instructional methods based of each student personal aptitudes. In this case, learning materials are aligned with the specific skills, knowledge, and intellectual abilities of the learner. It accomplishes this by having a diagnostic to assess meta-cognitive and reasoning abilities. ATI highlights the importance of fostering talent development and adapting educational environments to students' unique characteristics.

2.6.3 Micro-Adaptive Approach

Micro-Adaptive focuses on identifying and addressing students' specific, real time needs by tailoring instructional designs and tactics to enhance their learning experience. This approach emphasizes task-based diagnostics rather than pre-task assessments, dynamically adjusting based on learners' immediate performance, such as response errors and feedback. Micro-adaptive systems often achieve greater diagnostic accuracy for on-task performance compared to pre-task measures, enabling impactful instructional interventions. However, they face challenges in addressing the complexities of combined skill variables and contextual factors, as some student characteristics are temporary while others remain stable.

2.6.4 Constructivist Collaborative Approach

Constructivist Collaborative learning emphasizes collaboration and constructivist principles, where students actively construct knowledge and work in groups. This system encourages active knowledge construction and critical reasoning while also promoting real-time interaction among learners to discuss and analyze educational activities. This is achieved through community platforms that provide forums for students discussions, mediated by instructors who guide and facilitate the discourse. This approach highlights the value of collaboration and participation from all members of a class.

This study identified three primary adaptation method:

Macro-Adaptation, Micro-Adaptation, and ATI, which are system-dependent for the adaptation of each student and Constructivist Collaborative, which relies on student collaboration. For the system we aim to develop, we want the system to adapt to the learner in real-time, as they progress through the scenarios. In this case using the Micro-Adaptive approach is most appropriate, as it can diagnose each students needs with greater accuracy for on-task performance in real-time.

2.7 LLMs in Education

LLMs, such as GPT-4, have emerged in recent years, and are considered to many groundbreaking by many. These tools have transformed various aspects of daily life, including education, by redefining how knowledge is assessed and assimilated into our minds [17]. These models leverage large datasets and advanced algorithms to generate human-like text in response to problems and questions, while also accepting human language as input. Such system use NLP, which enables computers to interpret, manipulate, and understand human language.

LLMs can be applied in various facets of education, including:

- **Automated Content Generation**: The creation of educational materials such as MCQs, essays, and lesson plans has been significantly enhanced through the use of LLMs, as demonstrated by [21]. For example, frameworks like MCQGen combine RAG and advanced prompt engineering to automate the production of high-quality, diverse MCQs that are tailored to meet learners' individual needs [17].

- **Personalized Learning Paths**: By analyzing student interactions and feedback, LLMs dynamically adjust content delivery, ensuring that students receive content spanning various difficulty levels, from basic recall to complex analytical questions, addressing their diverse cognitive abilities and learning styles [17] [4].

- **Interactive Learning Assistants**: Virtual assistants powered by LLMs can simulate the interaction between a learner and a mentor, providing instant answers, explanation and feedback in real time [17], [2], [21]. Although over reliance on LLMs can reduce students

ability to tackle complex problem solving tasks on their own, LLMs may struggle with context specific queries or tasks outside their training data, a growing mistrust may also emerge among students over time, as they perceive a gap between their expectations and the tool's capabilities [25].

The generative nature of LLMs allows for adaptive learning experiences, such as **CoT** [17] which is a reasoning framework used in LLMs that enhances their capacity to perform complex reasoning tasks by generating a sequence of intermediate steps. Instead of producing a direct answer to a query, CoT prompts the model to articulate the reasoning process step-by-step, improving both the accuracy and transparency of problem-solving. This is achieved by prompting the model to break down complex queries into smaller sub-problems, solving each individually. For example, when solving a math problem, the model first calculates all intermediate values before providing the final result. By doing so, the LLM can more effectively handle tasks such as math word problems, logical reasoning, and multi-hop question answering. By structuring the reasoning process, CoT minimizes errors caused by oversimplification or skipped steps. This approach not only benefits the model's performance but also enhances user comprehension, as a coherent "reasoning chain" is formed [24].

Generative AI also enables us to use **Self-Refine Techniques** [17], an approach enabling LLMs to iteratively generate feedback on their outputs and refine them based on that feedback. Unlike methods that require supervised training or additional data, Self-Refine operates entirely within a single LLM framework. The process involves three main steps. The first one is feedback generation where the LLM reviews its own output and generates feedback in natural language, identifying specific issues or areas for improvement. This feedback guides the refinement process, often highlighting inefficiencies or logical flaws. The second step is refinement, were the feedback is applied to revise the original output. The final step is to repeat the first two steps until the desired quality is achieved [26]. A practical example of this approach is **Code Optimization**. Initially, the model generates a brute force solution to a programming problem, using excessive loops that result in inefficient code. Self-Refine detects the issue ("The code is slow due to nested loops...") and suggests a dynamic programming approach. The model then refines the output based on the feedback, thereby enhancing computational efficiency.

Although this approach is good in theory, this approach heavily depends on the capabilities of the chosen LLM, making some models struggle with generating accurate feedback and refining outputs effectively.

2.7.1 Leveraging LLMs for Educational Technologies: ChatTutor, CodeTutor, and MCQGen

Has stated before, LLMs have become foundational tools in reshaping educational technologies in many aspects, such as interaction, adaptability and content generation. Innovation such as ChatTutor, CodeTutor, MCQGen and many others [15], [16], [18] exemplify the potential of LLMs in educational contexts, each offering unique functionalities and strengths

ChatTutor [10] is a dialog based tutoring system that uses NLP to deliver personalized learning experiences through systematic interactions. ChatTutor operates based on three core processes: interaction, reflection and reaction, which work together to dynamically adjust the learning experience:

- The system begins with **Interaction** were the learner engages in guided conversations. While other LLMs such as ChatGPT, can perform similar functions, they often struggle with long-term coherence due to text length limitations. Therefore, maintaining focus during interactions while delivering complete and relevant information is essential.
- The second stage, **Reflection**, involves the system evaluating the learner's understanding to generate insights into the learning process and adapt the tutoring approach accordingly.
- Finally, **Reaction**, which is the trigger for changing the behavior of the tutor, modifying course plans or responses based on the learner's progress. Unlike Reflection, which occurs continuously with each new response, Reaction is performed periodically in the background.

CodeTutor [25] is a tool that leverages LLMs to help student with code tasks, such as problem-solving, debugging, and conceptual understanding. CodeTutor is a web-based application that integrates OpenAI API, utilizing GPT-3.5. Like many web apps it has user authentication, local and server-side data storage and a dedicated UI. In this case the UI main workspace is a central chat window where users interact with CodeTutor. Learners can pose programming-related questions, request clarifications or seek assistance with code. The system then responds in a tutor-like manner, as the model is configured to simulate a teaching assistant, providing explanations, examples, and suggestions.

To maintain consistency in behavior, the OpenAI API allows limited customization. CodeTutor uses a "system role text" to define its behavior. This role ensures the model acts consistently as a Python tutor, even when the conversation exceeds token limits.

Additionally, the platform includes feedback mechanisms, such as conversational and message level feedback, allowing users to rate interactions and gauge engagement and satisfaction with the assistant.

A study involving CodeTutor in a semester-long computer science course revealed significant improvements in students' coding accuracy and final scores, compared to peers who relied solely on traditional resources. However, its dependence on accurate prompts and occasional issues with user trust highlight the importance of proper implementation and user training to unlock its full potential.

MCQGen [17] is a framework for MCQs that combines instructor inputs with student participation. Educators provide learning materials, like lectures notes, slides, textbooks, etc... into the app, then the content can be generated automatically. Alternatively, content can also be generated manually by by inputting specific questions and answer options. Each set of MCQs is then tagged with difficulty levels and channels for better organization.

The student workflow is similar to the instructors: students can create MCQs in the same way, but they can also solve existing questions and earn points. For example, one point is awarded for

easy questions and two points for harder ones. Upon submission, students receive feedback to enhance future contributions, and each submission is categorized by a pre-assigned channel (e.g., course modules).

The MCQ generation in this app uses some very interesting techniques, including CoT and Self-Refine, previously discussed. In combination with these it also also uses RAG [31], a method to enhance the accuracy and reliability of generative AI by linking it to external knowledge sources. Unlike traditional LLMs, which rely on pre-trained patterns, RAG enables models to retrieve up-to-date and specific information, addressing limitations in their static knowledge base. Here's how RAG works:

- The user query is transformed into a machine-readable numeric format.
- This numeric vector is compared with a vector database containing external knowledge.
- Relevant information is retrieved and converted back into natural language.
- Finally, the model combines retrieved data with internal knowledge to generate a comprehensive response—potentially citing the sources of retrieved content.

By employing RAG the application builds an external knowledge base that enhances the AI's understanding of the course materials, ensuring the generated MCQs are both contextually accurate and pedagogically relevant.

2.7.2 Challenges and Limitations

Although LLMs and AI are very prominent, there are some inherited challenges and limitations that persist across many of these systems. Some of the most significant issues include hallucinations, lack of domain-specific expertise, and dependency on high-quality prompts.

Hallucinations [28] occur when the model generates text that is inaccurate, nonsensical, or disconnected from reality. Essentially, the model "hallucinates" information that may appear plausible but is factually incorrect or entirely fabricated. These hallucinations can take several forms:

- Factual errors, where the model provides incorrect or invented information while presenting it as accurate.
- Contradictions, where the output may conflict with earlier statements or with the input prompt itself.
- Nonsensical content, in which the generated text lacks logical coherence or meaningful structure.
- Irrelevant information, where the response diverges from the user's input or context.

The cause for hallucinations include issues such as low-quality training data, divergence from source references, vague prompts that encourage guesswork, overfitting, and adversarial or "jailbreak" prompts.

Hallucinations represent a serious concern in contexts where accuracy and trustworthiness are critical, as they can contribute to misinformation, ethical concerns (such as biases or toxic), and undermine user trust in AI systems. To mitigate these issues, we can implement strategies such as:

- Fine-tuning models with domain-specific data.
- Adding moderation or verification layers to filter out inaccuracies.
- Incorporating external databases to cross-check factual content.
- Improving prompt engineering to guide the model more effectively.

Lack of Domain-Specific Expertise [32] is another limitation. While LLMs are trained on a wide range of datasets, they may lack the depth required for highly specialized domains. For example, in the context of SBL, it often requires deep contextual knowledge and accurate adaption to specific professional or real world scenarios, which generic LLMs may not fully possess.

Dependency on High-Quality Prompts [2] the performance of LLMs relies heavily on prompt quality and structure. Poorly constructed prompts can lead to irrelevant or suboptimal outputs.

2.8 Conclusions and Identified Research Gap

From this state of the art review, it is evident that LLMs and their applications in education have evolved significantly, yet a notable gap still remains between SBL and LLMs capabilities.

Although systems like ChatTutor, CodeTutor and MCQGen demonstrate the potential of LLMs for personalized tutoring, programing education and automated educational content creation, none of these platforms fully address the unique pedagogical requirements of SBL.

SBL demands a system capable of supporting dynamic, context-rich scenarios that can adapt to learners decisions and provide real-time feedback based on context. Despite the advancements in LLM-driven educational technologies, there is no dedicated system that fully integrates LLMs into SBL, especially when it comes to incorporating role-playing elements and delivering timely, decision-based feedback.

This gap represents a valuable opportunity for research and innovation. This is were intend to intervene, by developing an **LLM-based intelligent educational agent specifically designed for SBL**. The proposed system will allow learners to assume specific roles within scenarios to simulate real-world challenges while also providing real-time feedback and adaptive support based on role-specific actions and decisions.

By bridging the gap between LLM-driven technologies and SBL, this research aims to:

- Extend the capabilities of LLMs into an underexplored educational domain.

- Provide a scalable, flexible, and immersive tool for learners to engage with realistic educational experiences.
- Offer educators an innovative platform to to support experiential learning and promote the development of critical thinking and higher-order skills.
- Directly address existing challenges and limitations in integrating LLMs into SBL environments.

Chapter 3

The solution and Implementation

As previously outlined in Chapter 2, there exists a significant research gap in the integration of LLMs within SBL. While existing systems such as ChatTutor, CodeTutor, and MCQGen show promise in personalized tutoring, programming education, and content generation, they do not adequately address the unique pedagogical demands of SBL, particularly the need for dynamic scenario handling, role-specific adaptability, and real-time feedback grounded in learner decisions.

To address these limitations, the system is intended to simulate realistic learning environments where users can assume specific roles, make decisions, and receive contextual, adaptive feedback, thereby aligning with the principles of experiential and active learning.

3.1 Approach

To tackle these challenges and create an effective solution, the development process will be structured into three key phases.

We start by creating prompts, leveraging the knowledge of prompt engineering drawn from the most important papers [17], [2], [15] and [16]. This phase is particularly critical, as the prompts we select must be capable of accommodating a diverse range of scenarios. They cannot be overfitted, but must have strong generalization capabilities while also maintaining a high level of precision. To inform the prompts with the necessary context, we construct structured JSON files that comprehensively capture the details of each scenario. These files serve as a modular and scalable means of encoding scenario-specific data, enabling the prompts to dynamically access relevant information during inference without hardcoding any assumptions.

The second phase involves creating a platform through which a user can interact seamlessly with the AI by an intuitive and accessible UI. The UI should be user friendly, serving as the primary gateway for users to access and communicate with the AI. Additionally, the UI should ensure efficient, real-time interaction, facilitating a smooth and engaging user experience that aligns with the platform's goals.

The third and final phase will involve iteratively enhancing and refining the system based on user feedback and performance metrics. This process will focus on addressing any identified

limitations, optimizing functionality, and integrating new features to ensure the platform remains responsive to user needs. By adopting an agile, feedback-driven approach, the system can continuously improve its performance, usability and relevance over time, delivering an increasingly effective and engaging user experience.

3.2 Methodology

The methodology serves as the roadmap for the project, outlining the key tasks, milestones, and timelines necessary to achieve the project's objectives. The timeline illustrated in Figure 3.1 visually represents this plan, enabling effective project tracking and management. A more detailed version is provided in Appendix C.1.

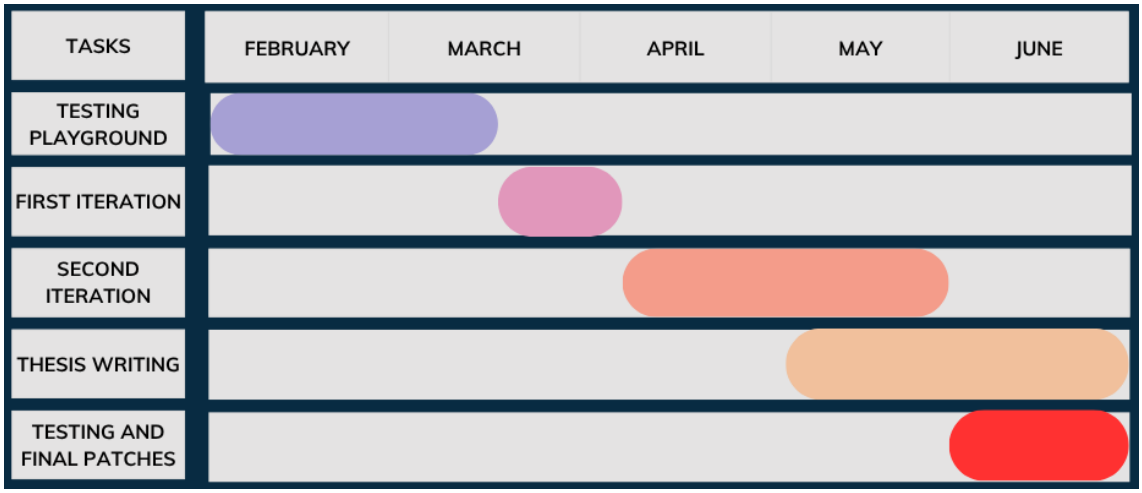

Figure 3.1: Timeline of the project.

3.2.1 Phase 1: Testing playground (Weeks 1–6)

The first phase focuses on testing prompts, leveraging insights from key research papers, including [17], [2], [15] and [16]. This phase is crucial, as the quality and generalization of these prompts will determine the system's ability to handle diverse scenarios effectively.

- **Key Activities:**
  - Analyze prompt engineering strategies presented in the referenced literature.
  - Design prompts capable of generalizing across various scenarios while maintaining precision.
  - Test initial prompts for overfitting and refine them to ensure optimal generalization capabilities.
- **Milestone:** Finalization of a robust set of scenario-based prompts by week 6.

3.2.2 Phase 2: First Iteration (Weeks 7–9)

The second phase involves building a full system that includes an intuitive and accessible UI, and an initial configuration of the AI agent.

- **Key Activities:**
  - Design a user-friendly UI layout based on user-centric design principles.
  - Develop a platform supporting real-time interaction between users and the AI.
  - Test the UI to ensure it aligns with the project's objectives, offering a smooth and engaging user experience.
  - Build the AI agent logic with the prompts created during Phase 1.
- **Milestone:** Deployment of the first functional prototype of the system by week 9.

3.2.3 Phase 3: Second Iteration (Weeks 10–17)

The second iteration focuses on evaluating and enhancing the system based on iterative testing and informed feedback.

- **Key Activities:**
  - Collect user feedback and performance data to identify limitations and improvement opportunities.
  - Implement iterative updates to optimize functionality and add new features.
- **Milestone:** Final deployment of the enhanced and refined platform in Week 17.

3.2.4 Phase 4: Testing and Final Patches (Weeks 18–20)

After the final iteration of the system has been completed, we will start testing it with quantitative and qualitative results using users and unittests.

- **Key Activities:**
  - Create and prepare comprehensive tests.
  - Run the tests and collect both usage data and system outputs..
  - Conduct an analysis of the results.
- **Milestone:** Completion of all testing activities and documentation of results by week 20.

3.3 Solution

After the preceding phases, we gain a clearer idea of what the solution might look like. Since we need a UI, backend and business logic working together in a structured way, it makes sense to use the MVC design pattern. In this case the UI elements would be under the View, the business logic would be in the Controller and the backend/LLM calls would be in the Model.

A visual representation of this architecture is shown in Figure 3.2. It illustrates the use of the MVC design pattern.

In the View, we have the UI class, responsible for rendering the chatbot's visual interface.

The Controller manages user input, directing it to the LogicHandler, which determines how the system should respond based on the current state and logic.

Finally, in the Model, the AI agent performs a loop of LLM calls. Based on the user input and system state, it determines when the scenario is complete and terminates the loop accordingly.

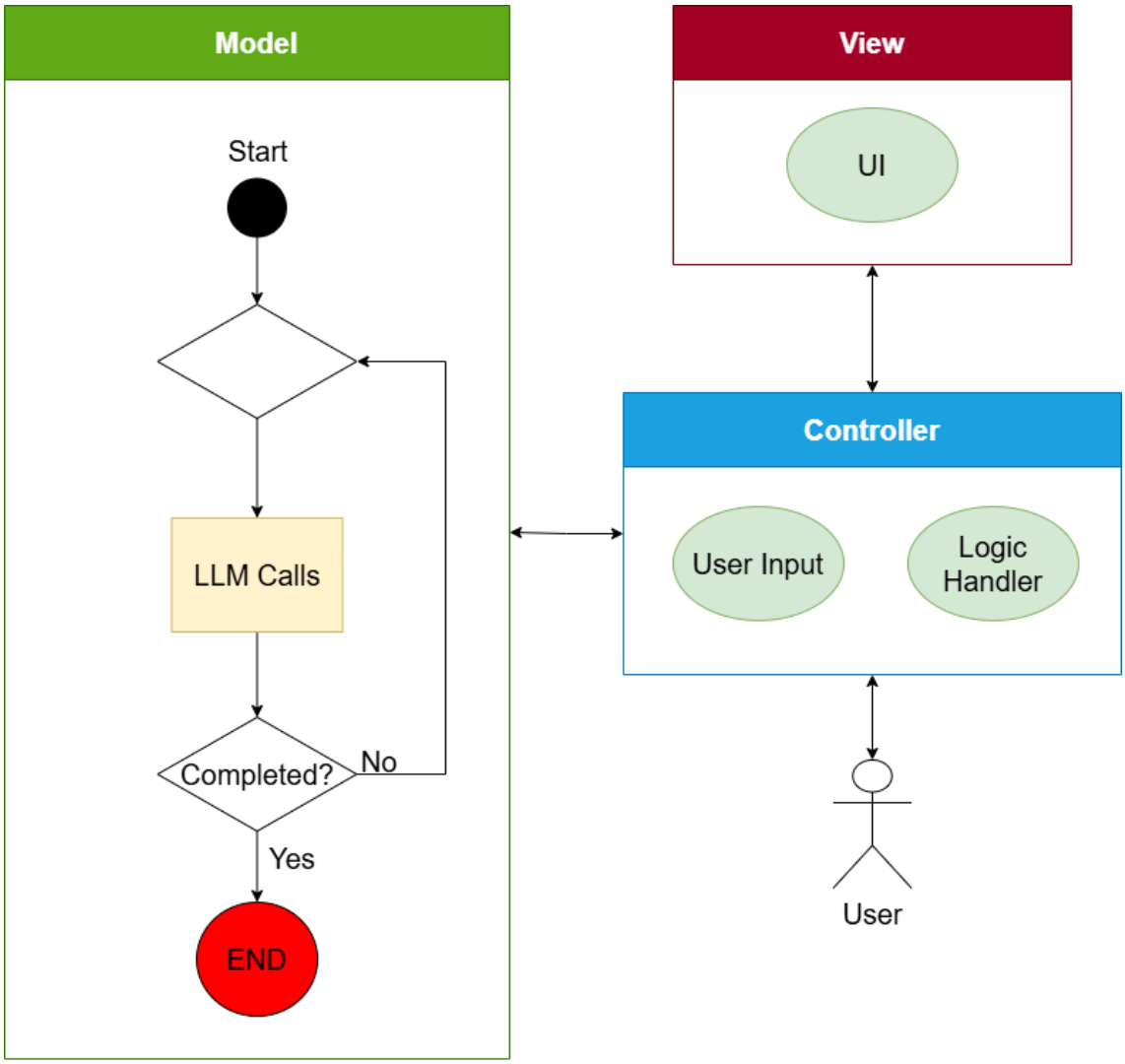

Figure 3.2: Flow chart showcasing the possible system solution.

3.4 Implementation

The reason for having two versions was due to my lack of prior experience with this type of project, this being my first attempt. The first version served as a testing playground, where ideas were tested, while the second version represented the stage where they were all implemented and put together. This is not to say that the second version did not undergo any changes, most of its development involved combining components from the first version and, in some cases, enhancing them.

Throughout the remainder of this thesis, I will primarily refer to the second version, but I will also document the full development process, including insights from the initial testing playground. Before doing so, I will first explain the steps taken to create the first version.

3.5 Initial Pipeline

To explain the pipeline of the first iteration, we examine the flowchart in Figure 3.3.

The first notable feature is the MVC pattern, which was implemented to separate data, UI and application logic into different files. This separation facilitates easier development, maintenance, and testing.

3.5.1 Controller

The controller is where data is handled, it serves as a pathway for communication between the user, the model and the view, and that is its main function. Its primary role includes processing user input and handling the logic for each input. Additionally, it is responsible for initializing the program, so that everything works in unison.

3.5.2 View

The View is corresponding to the UI, in this first iteration tkinter was chosen due to its simplicity and functionally, it looked and functioned similar to a chatbot, as we can see from Figure 3.4. Some of its key functions included creating the view and sending text.

3.5.3 Model

This is where the LLM Agent and reasoning mechanisms operate. The main components for this thesis rely on this module, since it stores all of the LLMs calls and reasoning.

To quickly break down the steps used for this part we will follow the reasoning steps show in Figure 3.3.

3.5.3.1 Initialization

The system begins by loading a predefined scenario from a JSON file into the LLM. This scenario provides domain-specific context and establishes a consistent narrative structure from the outset.

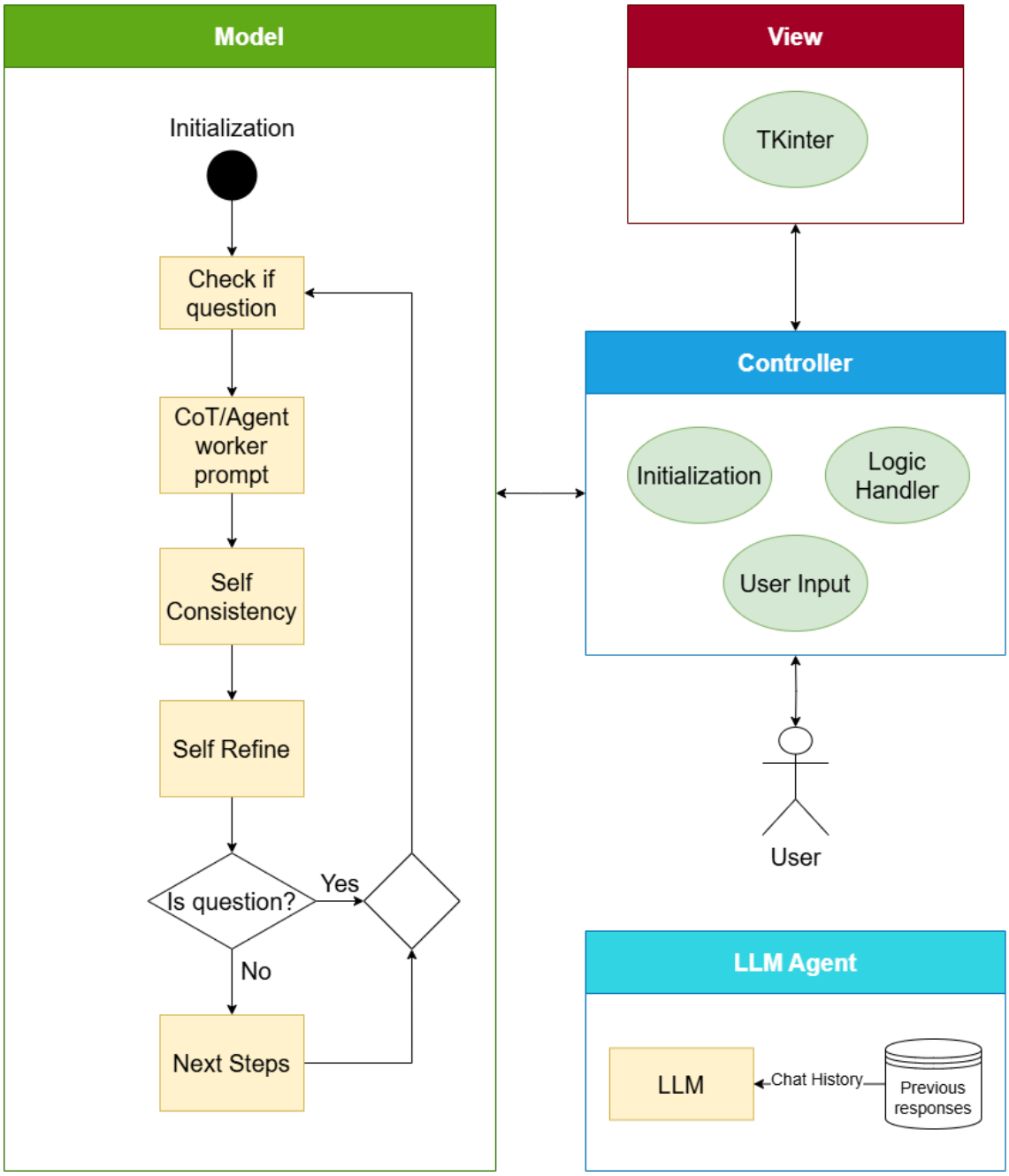

Figure 3.3: Flow chart showcasing the primitive system process.

By embedding a structured scenario upfront, the system reduces ambiguity and helps limit hallucinations that might occur from vague or open-ended prompts. Alternatives, such as generating scenarios dynamically, were considered but discarded due to the risk of lower factual grounding and coherence.

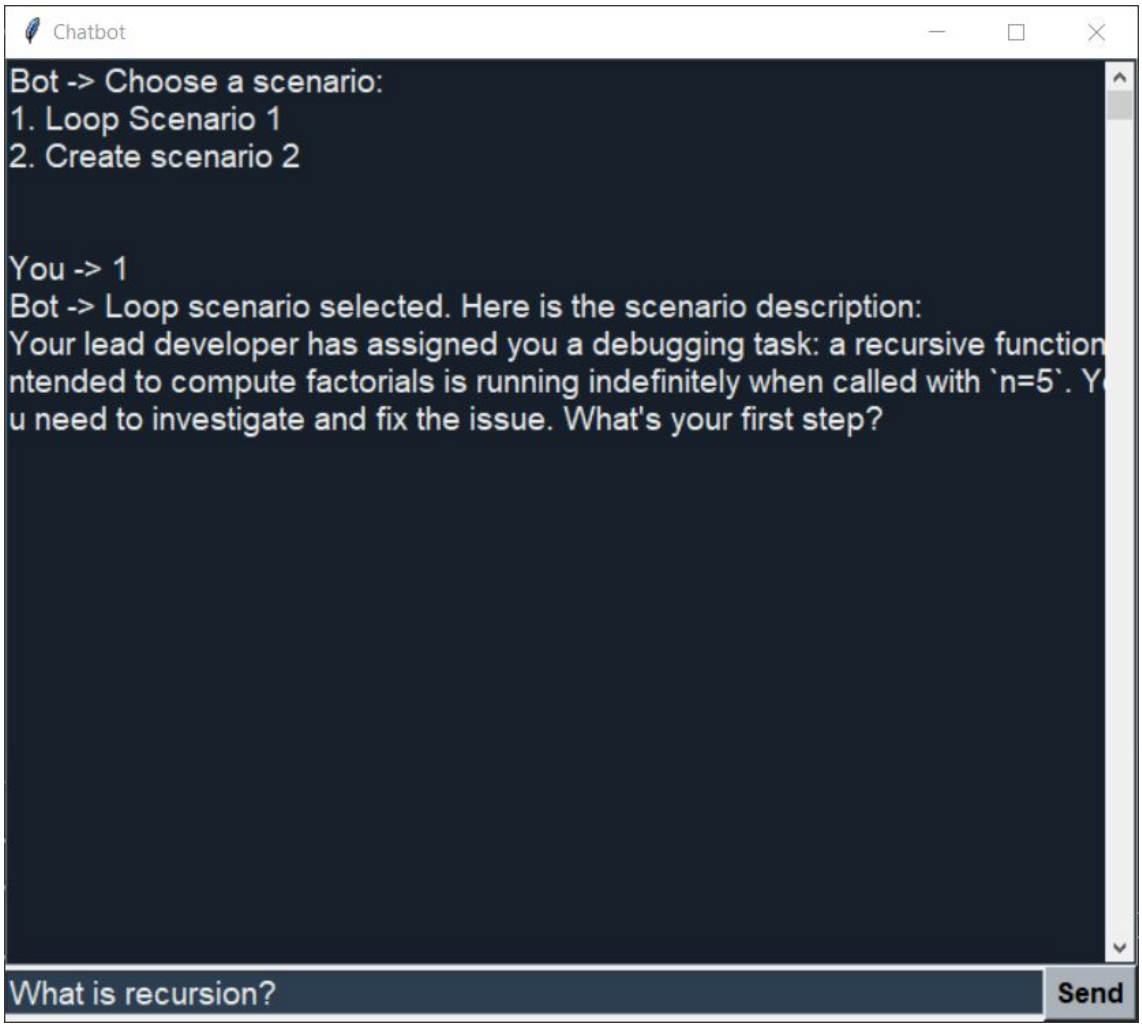

Figure 3.4: UI for tkinter.

3.5.3.2 Safe state

The safe state maintains the complete chat history and serves as the entry and exit point of the system loop. This persistent memory enables continuity, which is essential to maintain contextual integrity and avoid contradictions, one of the known causes of hallucinations. Additionally, preserving message history supports self-refinement later in the pipeline. Stateless alternatives were considered but offered limited capacity for sustained, consistent interaction.

3.5.3.3 Question Classification

At this stage, the system distinguishes between questions and action-based inputs. This routing improves prompt clarity, allowing the system to better structure its response logic. Because LLMs are highly sensitive to prompt structure (as per the challenge of prompt dependency), separating intent types helps optimize performance and prevents irrelevant or off-topic responses.

3.5.3.4 CoT/Agent Worker Prompt

The model is prompted using CoT prompting combined with task-specific instructions. CoT enhances reasoning by encouraging the LLM to decompose complex tasks into intermediate steps, which mitigates the risk of hallucinations and improves coherence. Compared to zero-shot or few-shot prompting, CoT has shown greater alignment with human reasoning, especially in structured environments like SBL.

3.5.3.5 Self-Consistency

To further enhance response reliability, the system uses a self-consistency mechanism, generating multiple answers and consolidating them into a final output. This approach helps reduce hallucinations by filtering out inconsistent or logically incoherent responses. While deterministic outputs were faster, they were more prone to single-point errors and hallucinations, making self-consistency a safer alternative.

3.5.3.6 Self-Refine

The self-refinement stage introduces a form of internal critique where the model generates feedback on its own response and attempts to improve it. This mirrors reflective human reasoning and compensates for the model's lack of deep domain expertise by encouraging iterative quality control. External human-in-the-loop refinement was an alternative, but self-refinement offers greater scalability while still mitigating low-confidence outputs.

3.5.3.7 Next Steps Generation

When the user provides an action instead of a question, the system generates suggested next steps within the scenario. This guides the interaction forward in a structured manner, helping maintain the relevance of the conversation and reducing prompt ambiguity, a key factor contributing to hallucinations. A passive system that waits for user initiative was considered, but it was more susceptible to drifting off-topic.

3.5.4 Limitations

Has it is apparent this initial pipeline is not optimal nor finished as there are a lot of things that need to be arranged.

The most important one is how the system is supposed to know that the scenario has been completed, since there is nothing checking if the user has completed the scenario the system would never end, making it impossible to go through a scenario.

Another crucial detail that is missing is RAG, has it was previously mentioned in the state of the art. With RAG the LLM would have up to date information and would be able to

3.6 The Current Pipeline

The updated version of the flowchart in Figure 3.5 adds a few important steps to make the system smarter and more reliable. It now uses a RAG system to pull in extra information when needed, and it checks if a question is relevant or if a stage is already completed before responding. There's also a new "redirect" step for off-topic questions and a clear end point if the task is done. On the front end, the old Tkinter interface has been replaced with Streamlit to make the user experience smoother and more web/user-friendly.

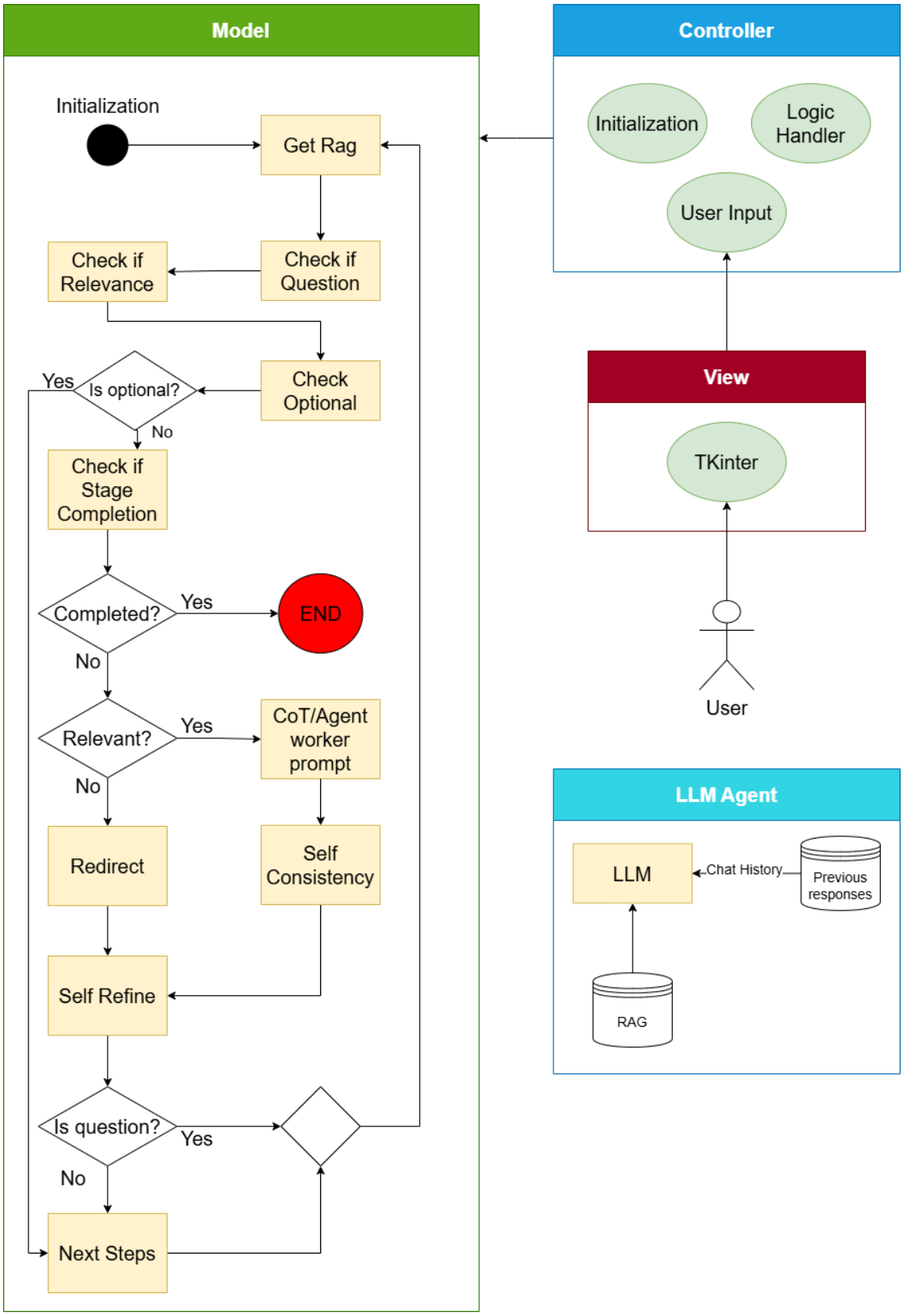

Figure 3.5: Flow chart showcasing the final system process.

3.6.1 View

The View was replaced with Streamlit, so that the user experience could be improved. In this little change we are now able to use the chatbot in the web browser, making it possible to implemented this chatbot in a website, something that was not possible before with tkinter. Further more the UI is now more appealing to the user because of the better graphics and progress bars, as shown in Figure 3.6.

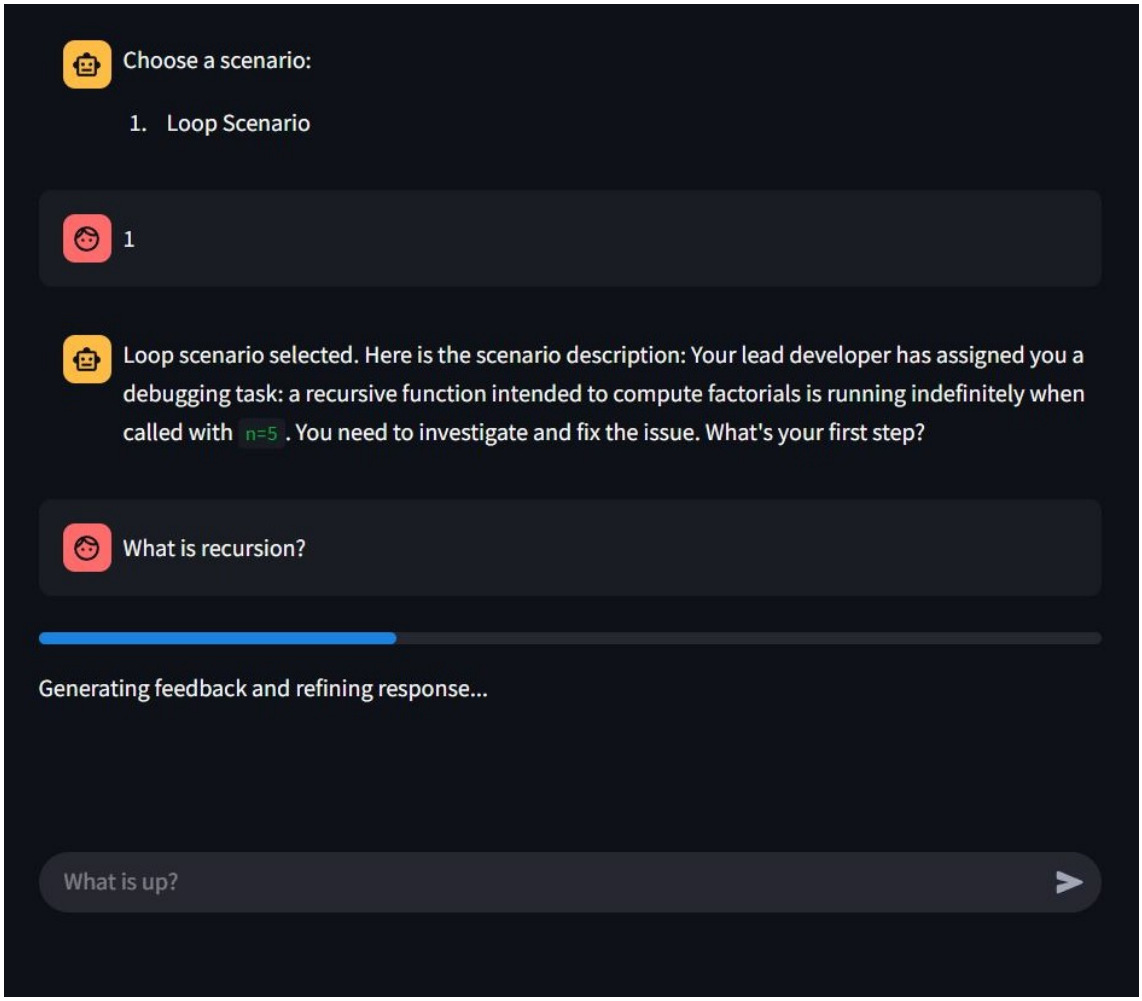

Figure 3.6: UI for Streamlit.

3.6.2 Model Revised

This section outlines the main differences in the revised model compared to the previous version, highlighting design choices that aim to improve coherence, trustworthiness, and domain alignment, particularly in mitigating hallucinations, managing prompt dependency, and reinforcing relevance in user interactions.

3.6.2.1 Initialization

As with the previous version, the system starts by loading the scenario from a structured JSON file into the chat history. However, unlike before, an unnecessary initial LLM call was removed to streamline the process, its outputs did not add value and only delayed scenario loading times. This decision optimizes system responsiveness without sacrificing functionality.

Additionally, the RAG component is now initialized at this stage. By preloading a knowledge base, the system ensures that relevant factual content can be retrieved and referenced during user interactions, directly addressing the hallucination problem by grounding answers in external, verifiable data sources. Without RAG, responses risk being entirely model-generated and therefore more prone to fabrications.

3.6.2.2 Check Relevance

This module evaluates whether the user's input is aligned with the current stage's objective. LLMs often struggle with vague or misaligned prompts (prompt dependency issue), which can derail the task flow or lead to irrelevant outputs. By explicitly assessing relevance early, the system ensures that downstream processing only occurs on meaningful input, reducing the likelihood of generating off-topic or hallucinated content.

Alternatives like free-form interpretation were considered but proved too error-prone in structured, goal-oriented scenarios.

3.6.2.3 Check Optional

This step consists of two parts. First, the system determines whether the user is asking for optional or supplemental information. If confirmed, the model provides contextually appropriate details.

This addresses the lack of domain-specific expertise challenge: optional info often requires domain knowledge or procedural nuance. By detecting and delivering optional content explicitly, the system helps close the expertise gap and improves user support in scenario navigation. A monolithic approach where optional and primary objectives were handled the same way led to confusion and less targeted outputs.

3.6.2.4 Check Stage Completion

In this stage, the model determines if the current task (or scenario stage) has been completed. If so, it updates the system state accordingly or ends the session.

This structured approach reduces dependency on user-formulated cues (prompt quality issue) by proactively guiding flow transitions. Without this, users might need to issue complex or specific prompts to signal progression, increasing the chance of misinterpretation or failure to detect task completion.

3.6.2.5 Redirect

If the user's input is judged irrelevant by the Check Relevance step, the redirect module activates to guide them back on track. This prevents the model from responding to off-topic or nonsensical inputs, both of which are common sources of hallucination. Instead of producing a plausible but incorrect response, the system intervenes and nudges the user toward a more relevant action.

Alternatives like permissive generation (responding to all inputs regardless of relevance) were tested but led to coherence breakdowns and undermined user trust in the system's instructional integrity.

3.7 LLMs & APIs

As is evident, the use of LLMs is crucial to the development and deployment of this conversational agent. Since developing an LLM from scratch is beyond the scope of this thesis, an API was used.

In our case groq API was chosen because it provides low-latency responses which are essential for maintaining a smooth and interactive user experience, supports multiple powerful models such as those from Meta (llama), OpenAI (whisper), Google (gemma) and even from DeepSeek. Additionally, it offers competitive pricing, including a free tier, and is highly scalable, making it suitable for both prototyping and potential real-world deployment.

Although the API sounds promising, there are a few associated with the free tier, such as rate limits shown in Figure 3.7. Because of this the development was not as straightforward as it could have been. Occasionally, rate limits were triggered during program execution, causing components to take longer than necessary or, in worse cases, halting testing entirely for the day. This had a obvious negative impact on development, sometimes leading to misleading outputs and delays with no immediate solution. However, as a result, token usage was significantly reduced in an effort to "fix" this issue. Ultimately, this led to a system that was both faster and more cost-efficient to operate.

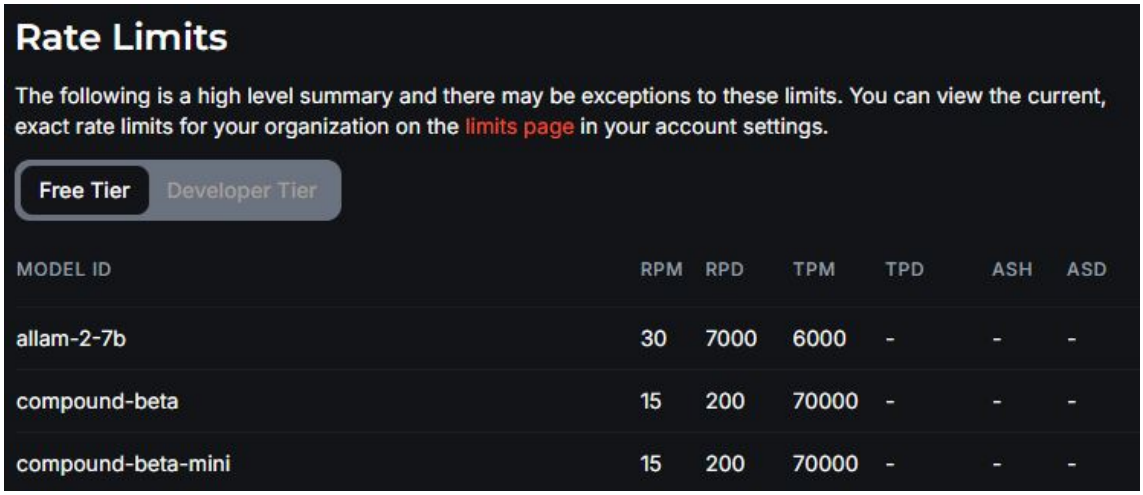

Figure 3.7: Groq API Rate Limits

3.8 Data Communication & Controller

As mentioned beforehand, the controller component of the system handles all of the data transfer between the UI and the Backend, having a similar behavior to the Figure 3.8.

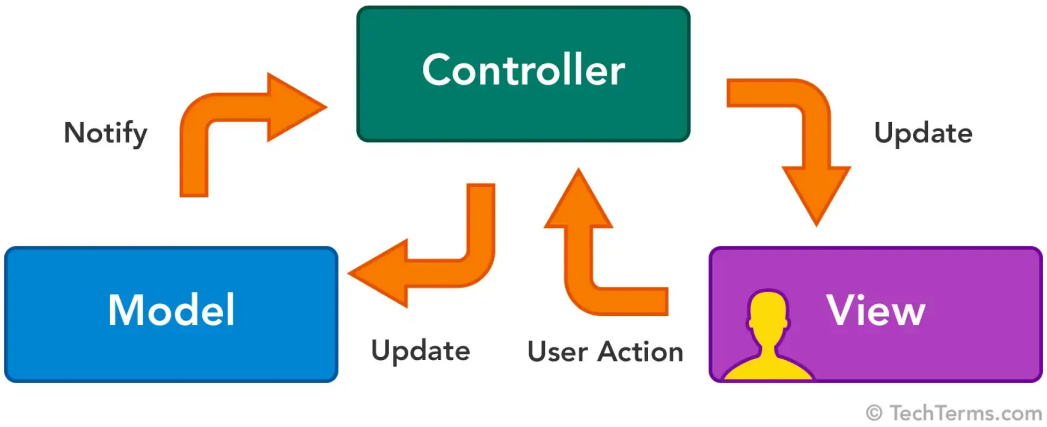

Figure 3.8: MVC.

Inside the controller resides the chatbot class which is initialized in main, along with the ChatbotGUI class that represents the UI. The Chatbot class serves as the core of the application, managing the chatbot's state, scenarios, and interactions with the user. It uses a state-based approach to handle different stages of the conversation, such as the initial greeting, scenario selection, and scenario-specific logic.

The Chatbot class has three main attributes: stage, llm, and view. The stage attribute tracks the current phase of the chatbot's interaction (e.g., "Initial", "Options", or "scenario"). The LLM attribute represents the logical reasoning engine (LLM), which is initialized using the set_llm method. The view attribute is set using the set_view method, linking the chatbot to its graphical interface.

The chatbot's main functionality is implemented in the get_response method, which processes user input and determines the appropriate response based on the current stage. In the "Initial" stage, the chatbot prompts the user to select a scenario. In the "Options" stage, it validates the user's choice and, if valid, transitions to the "scenario" stage, where it loads the selected scenario using the set_llm method. During this process, progress bar updates are displayed to provide feedback to the user. In the "scenario" stage, the chatbot processes user commands, interacts with the LLM logic, and resets the system once the user has completed the scenario.

3.9 UI

The UI is defined by the ChatbotGUI class, that creates a chatbot interface using the Streamlit library. This class is designed to manage the chatbot's UI, handle user input, and display chat history.

The initialize_session_state method sets up the session state to store chat history. Streamlit's session_state is used to persist data across user interactions. If no chat history exists, the method initializes it with an empty list and adds an initial message from the chatbot by calling the provided response_function.

The display_chat_history method iterates through the stored chat messages in the session state and displays them in the chat interface. Each message is rendered using Streamlit's st.chat_message and st.markdown components, which format the messages based on their role "user", shown in Figure 3.9, or "assistant", shown in Figure 3.10.

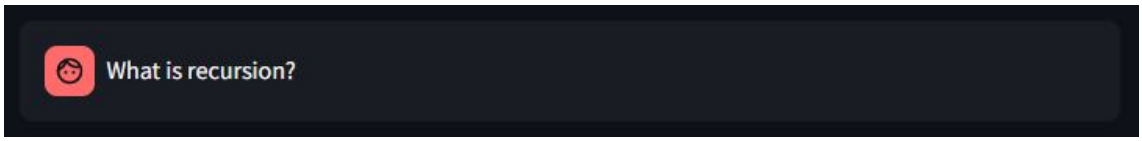

Figure 3.9: "user" message format.

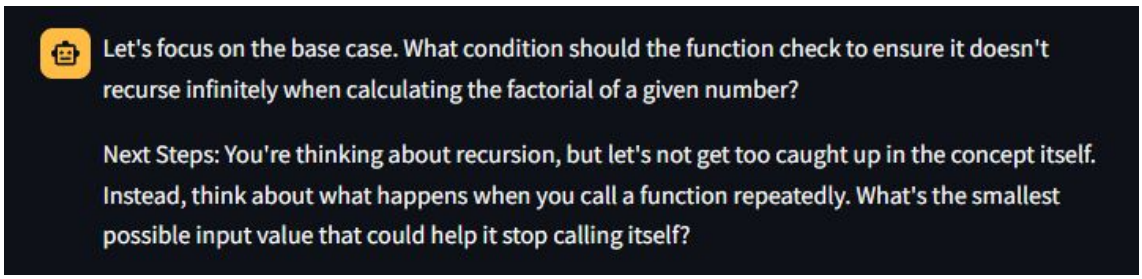

Figure 3.10: "assistant" message format.

The handle_user_input method manages user interactions. It captures input from the user via st.chat_input, adds the user's message to the chat history using the add_message method, and generates a response using the generate_response method. The chatbot's response is then added to the chat history and displayed.

The add_message method is responsible for appending a new message to the session state and rendering it in the chat interface. It ensures that both the content and the role (user or assistant) are stored and displayed correctly.

The generate_response method acts as a bridge to the chatbot's logic layer. It calls the get_response method of the logic object to generate a response based on the user's input.

Additionally, the class includes methods for managing a progress bar (progress_bar_create, progress_bar_delete, and progress_bar_percentage) to simulate the chatbot "thinking." These methods use Streamlit's progress bar and text components to provide visual feedback during response generation.

Finally, the run method orders the chatbot's functionality. It displays the chat history and handles user input, ensuring a seamless interaction loop. This method is intended to be called to start the chatbot application. Overall, the ChatbotGUI class encapsulates the logic for building an interactive chatbot interface with Streamlit.

3.10 RAG

One of the biggest challenges of this dissertation was implementing RAG. As previously mentioned, RAG is used to retrieve relevant external knowledge to generate more accurate, context-aware responses by combining information retrieval with generative models. For context, the following guidelines were used to support the implementation, unfortunately this reference required substantial modification to suit our specific use case.

3.10.1 Setup

The first step to having RAG is having a Vector Database. In our case we used Pinecone, as it was the database used in the reference implementation and is available for free. However, the free tier allows only a single index and has limited read/write capacity. This was relatively straightforward, we signed up for Pinecone, obtained an API key, connected to the database, and created our index. Following these steps, we obtained a fully functional vector database, ready for integration with our RAG pipeline.

3.10.2 Embedding

The second step is to convert questions into a vector representation using an embedding model. Embeddings are relatively easy to understand; they transform a sentence into a list of numbers, for example, "How do I get a replacement Medicare card?" turns into [-0.02388945; 0.05525852; -0.01165488;... 0.00577787; 0.03409787; -0.0068891]. This vector captures the semantic meaning of the sentence, not just its words. That way, similar sentences will have similar embeddings, even if they use different words.

To create an embedding, a machine learning model is used, in our case we utilized "sentence-transformers/all-MiniLM-L6-v2", a light weight machine learning model capable of converting text or images into high-dimensional embeddings. A more complex model could have been used for this purpose, but the time difference was too much for the dataset we chose (approximately 10000 parameters).

Now we are ready to start the embedding process. To accomplish this, we use the build_index function, which handles creating and managing a Pinecone index for storing and retrieving embeddings generated from a dataset. The process involves three main steps: loading and processing the dataset, ensuring the index exists and is correctly configured, and finally upserting the processed data and embeddings into the index.

The first step is loading and processing the dataset. The function begins by initializing a SentenceTransformer model to generate embeddings. It checks for cached files to avoid redundant computation. If these files exist, they are loaded in parallel using a ThreadPoolExecutor. Otherwise, the dataset is loaded from the huggingface website. Then metadata fields are processed using helper functions like ensure_string and truncate. These helpers ensure that metadata values are consistently formatted as strings and truncated to a maximum length of 1000 characters to

reduce size and tokens used in future use. The dataset is further processed to remove unnecessary columns, and text chunks are created for embedding generation. The embeddings are computed in batches to optimize memory usage, and both the processed data and embeddings are cached for future use.

Then we ensure the index exists and matches the required configuration. This is achieved by checking whether a Pinecone index with the specified name (self.index_name) already exists. If it does, the function verifies whether the index's embedding dimension matches the model's embedding dimension. If there is a mismatch, the existing index is deleted and recreated. If the index does not exist, it is created with the appropriate configuration, including the embedding dimension, a cosine similarity metric and a serverless specification for deployment. The function waits until the index is fully ready before proceeding.

Once the index is ready, we proceed with the final step, upsert data into the index. The function upserts the processed data and embeddings into it in batches of 256. Each batch consists of tuples containing an ID, the corresponding embedding vector, and metadata. This ensures that the data is efficiently stored in the index for later retrieval. The use of batching helps manage memory and network resources during the upsert process.

The build_index function is designed to prepare a Pinecone index for efficient retrieval of embeddings. By preprocessing the dataset, managing the index lifecycle, and upserting data in a structured manner, it lays the foundation for a system that can perform fast and accurate similarity searches, which are essential for tasks like RAG. The use of caching, batching, and helper functions ensures that the process is both efficient and robust.

3.10.3 Query the Database

After the embedding is done we can go straight to the final step, retrieving and formatting the most relevant documents from the index based on a given query. The get_docs function does just that, it takes two parameters: query, a string representing the user's search input, and top_k, an integer specifying the number of top results to return. The method performs three main tasks: encoding the query, querying the Pinecone index, and formatting the retrieved documents.

First, the method encodes the input query into a vector representation using the self.encoder function. This encoder is the same machine learning model used for generating the text into embeddings, converting text into numerical vectors. These embeddings are essential for performing similarity searches in the Pinecone index, as they allow the system to compare the query with stored document embeddings in a high-dimensional space.

Next, the method queries the Pinecone index using the encoded query vector (xq). The self.index.query function searches for the top_k most similar embeddings in the index and retrieves their associated metadata. The include_metadata=True parameter ensures that the metadata, such as the document context, is included in the results. The results are stored in the res variable, which contains a list of matches, each with its similarity score and metadata.

The method then extracts the relevant document text from the metadata of each match. Specifically, it accesses the context field within the metadata of each result and stores these in the docs list. This list contains the raw text of the top-matching documents.

Finally, the method formats the retrieved documents into a human-readable string. It iterates over the docs list, numbering each document and appending its content to a formatted string. The documents are separated by double newlines for clarity. The formatted string is returned as the output of the method, providing users with a clear and accessible summary of the retrieved content.

3.11 Scenarios

To represent the scenarios we chose to use a JSON file format. This way it is easier to access data and also to create the scenarios.

Each JSON file has the following parameters ai_role, user_role, scenario_name, ai_persona, place, task, format, exemplar, stage_description, hint, positive_feedback, constructive_feedback, next_stage_condition, all_optional, stages and tones.

Each of these parameters has a function serving a distinct purpose. Starting with the ai_role, user_role and ai_persona, has it name implies it describes the role of the AI and the user. An example input might be "Virtual Mentor" for the AI, "Junior Developer" for the user and "Patient Programming Instructor with a focus on beginners" for the ai_persona.

After this we have scenario_name, task, stage_description, place and format. These parameters serve to provide insight of the scenario for both the user and the LLM, as the first message of the chat is initialized with the text from the scenario_description.

Next we have the parameters for how the LLM should react, this is represented in exemplar, hint, positive_feedback, constructive_feedback and next_stage_condition.

In the next part we have the actual information and stage steps to complete the scenario, here we have the all_optional which will be a list of information like "Ask for blood pressure: 210/105 mmHg", "Ask for heart rate: 87 bpm"... Additionally some information is accessible only hrough specific stage steps. The stages consists of a list of stage_step where inside we have another list with information about the steps needed to complete the stage step, this information include the description, a hint and the correct_response to the step. This structure ensures that the LLM receives only the relevant information for the current stage, maintaining contextual accuracy and pedagogical alignment.

Finally, we have the tone specification, here we simply specify the tone in which we want the LLM to respond, for example "Clinical", "Encouraging".

This structure is used to build prompts that include all necessary details for our LLM agent.

Listing 3.1: JSON structure for defining an educational scenario

```
{
    "ai_role": "",
    "user_role": "",
    "scenario_name": "",
    "ai_persona": "",
    "place": "",
    "task": "",
    "format": "",
    "exemplar": "",

    "stage_description": "",
    "hint": "",
    "positive_feedback": "",
    "constructive_feedback": "",
    "next_stage_condition": "",
    "all_optional": []

    "stages": [
        {"stage_step":
            [
                [],
                {
                    "description": "",
                    "hint": "",
                    "correct_response": ""
                }
            ]
        }
    ],

    "tones": []
}
```

3.12 Chat History

The chat history serves to maintain context by preserving both the user's previous inputs and the LLM's responses. This enables the LLM to provide more insightful and coherent feedback on the user's progress. This is implemented by having a list variable with the initial value being the scenario introduction. In earlier versions, this initial value included an LLM call, but it was later removed after being deemed unnecessary.

Listing 3.2: Inital prompt for the chat history.

```
def Inital_prompt(self):
    prompt_template = f"""
        You are an AI agent acting as {self.ai_role}, assisting a user playing the
            role of {self.user_role} in the scenario "{self.scenario_name}".
        Your persona is {self.ai_persona}, and your goal is to guide the user
            through a scenario-based learning experience at {self.place}.

        ### Task:
        {self.task}
        The task must be completed step by step, progressing through multiple
            stages. Each stage presents a new challenge or decision point.

        ### Format:
        {self.format}

        ### Exemplar:
        {self.exemplar}

        ### Interaction Rules:
        - At each stage, provide an **initial prompt** describing the situation.
        - If the user struggles, provide **subtle hints** to **never reveal the
            correct answer** directly.
        - Respond dynamically to the user's input, offering **adaptive feedback**
            based on their choices.
        - Only advance the user when they make the correct or reasonable decision.

        ### Response Format:
        - **Initial Prompt**: {self.stage_description}
        - **Hints**: {self.hint}
        - **Feedback**:
        -  **Correct**: {self.positive_feedback}
        -  **Incorrect**: {self.constructive_feedback}
        - **Next Stage**: {self.next_stage_condition}
        ### Stage description:
        """

        # Iterate through the stages and print descriptions
        for i, stage in enumerate(self.stages):
            first = True
            for j, step in enumerate(stage["stage_step"]):

                if first:
                    prompt_template += f"""
                ### Stage {i+1}:"""
                    first = False

                prompt_template += f"""
```

```
                ## Step {i+1}.{j+1}:
                    - **Description**: {step["description"]}
                    - **Hint**: {step["hint"]}
                    - **Correct Response**: {step["correct_response"]}
                """

        prompt_template += f"""
        ### Tone & Style:
        - Use a **{self.tones[0]}** to match the urgency of the scenario.
        - Write in a **{self.tones[1]}** for clarity and engagement.
        - Maintain a **role-playing dynamic** to keep the user immersed in the
            experience.
        """
```

To use the chat history two helper functions were created, prepare_conversation_history and update_conversation_history. Its main points are to prepare the last four messages and to update them.

This design ensures that the LLM receives sufficient context from prior conversations, without flooding it with the entire history, which may exceed token limits. It keeps in mind the prompt and the latest interactions while continuing to follow the conversation to the end.

This modular approach improves the coherence of the LLMs responses and enhances the user's experience in progressing through the scenario.

3.13 Prompts and LLM calls

We now arrive to the main focus of the thesis, prompt design and LLM calls. This is where all of the LLM logic happens, and it's also the main choke point were improvements and token optimization can be done. As previously outlined in the pipeline shown in Figure 3.5, the Conversational Agent follows a specific sequence of LLM calls and reasoning techniques to function correctly.

Firstly we determine whether the user input is a question or not, then we check for relevance to assess if the input pertains to the current scenario or stage. Next, we evaluate whether the input includes optional information that may indicate a user request for additional content. After that we check for stage completion this step ensures that the user has met the conditions necessary to advance in the scenario or complete it.

Following this, the pipeline branches into three distinct paths:

- If the input is **relevant**, the system engages a *CoT/Agent Worker prompt*, followed by a *Self-Consistency* step to validate reasoning quality and robustness.
- If the input is **not relevant**, the system bypasses reasoning steps and instead *redirects the user* with contextual guidance to realign their response with the scenario goals.
- If the input is **requesting information**, the system ignores all reasoning steps and instead goes straight to the *next steps*.

After processing either of the first two paths, they merge so that we can have two rounds of Self Refine, enhancing the quality and coherence of the model's response through iterative refinement.

After this has been done we do a final conditional operation, this time checking if the user input is a action, question or information:

- If it's an **action** or **information**, the agent triggers a prompt that generates *Next Steps* to guide the user forward.
- If it's a **question**, the system doesn't do anything.

Once this logic loop is completed, the system is reset and ready to process the next user message.

3.13.1 Check if it is a Question

The is_question method is designed to determine whether a given user input is a question. It employs a combination of simple rule-based checks and a more advanced LLM to make this determination. This hybrid approach ensures that the method can handle both straightforward and nuanced cases effectively.

Listing 3.3: Partial code for defining if a user input is a question.

```
def is_question(self, user_input):
    input_str = user_input.strip().lower()
    question_words = ['what', 'when', 'where', 'who', 'why', 'how', 'can', 'could',
         'should', 'would', 'is', 'are', 'does', 'did', 'will']

    if input_str.endswith('?'):
        return True
    if any(input_str.startswith(word) for word in question_words):
        return True

    user_input_prompt = f"""Is the following input a question?
    Input: "{user_input}"
    Respond with True if it's a question, otherwise respond with False.
    """
```

The method begins by normalizing the input string. It trims any leading or trailing whitespace and converts the text to lowercase for consistent processing. It then checks two basic conditions to identify questions. First, it checks if the input ends with a question mark (?), which is a common indicator of a question. Second, it checks if the input starts with any of a predefined set of question words, such as "what," "why," "how," or "is." If either of these conditions is met, the method immediately returns True, indicating that the input is a question.

If the input does not meet these initial criteria, the method proceeds to use the LLM for further evaluation. It constructs a prompt that explicitly asks the model to determine whether the input is

a question. The prompt includes the user's input and instructs the model to respond with True if the input is a question or False otherwise.

The response from the LLM is then processed. The method extracts the content of the response, trims any whitespace, and converts it to lowercase. If the response is "true", the method returns True, confirming that the input is a question. Otherwise, it returns False. This fallback mechanism ensures that even inputs that are ambiguous or unconventional can be evaluated with the help of the LLM.

3.13.2 Relevance

The is_important method is designed to evaluate whether a given user input matches one of the predefined correct responses for the current stage of a process. It uses a LLM to perform this evaluation based on a structured prompt. The method also updates the state of the process if a match is found and the input is not a question.

The method begins by constructing a detailed prompt that includes instructions, rules, and examples to guide the LLM in its evaluation. The prompt specifies that the LLM should compare the user input to a numbered list of possible correct responses and return the index of the matching response if an exact or near-exact match is found. If no match exists, the LLM should return -1. The rules emphasize precision, instructing the LLM not to guess or infer based on vague similarities. Several examples are included in the prompt to illustrate how the LLM should handle different scenarios.

Listing 3.4: Prompt for checking if the user input is relevant to the current scenario stage.

```
prompt = ("You will be given a user input and a numbered list of possible correct
    responses."
"Compare the user input to each response. Only return the index of the response
    that is an exact or near-exact match. If none match, return -1.\n\n"
"Rules:\n"
"- Do not guess or infer based on vague similarity.\n"
"- Only match if the meaning is directly and clearly aligned.\n"
"- Return just the index (e.g., -1, 0, 1, 2...)\n\n"
"Possible responses:\n"
)

for index, correct_response in enumerate(self.stage_correct_response[temp]):
    prompt += f"{index}: {correct_response}\n"

prompt += f"\nUser Input:\n{user_input}\n\nReturn only the index or -1 (e.g., -1,
    0, 1, 2...)."
prompt += ( "Examples:\n"
"---\n"
"User Input:\nI would take their blood pressure.\n"
"Correct Responses:\n"
```

```
"0: Check the patients pulse\n"
"1: Administer CPR\n"
"2: Measure the patients blood pressure\n"
"3: Call for emergency support\n"
"Answer: 2\n"
"---\n"
"User Input:\nSee if they are alive.\n"
"Correct Responses:\n"
"0: Check the patients pulse\n"
"1: Administer CPR\n"
"2: Measure the patiens blood pressure\n"
"3: Call for emergency support\n"
"Answer: -1\n"
"---\n"
"User Input:\nUse a for loop to print numbers 1 to 10.\n"
"Correct Responses:\n"
"0: Define a function in Python\n"
"1: Use a for loop to print numbers 1 to 10\n"
"2: Create a list with 10 elements\n"
"3: Write a recursive factorial function\n"
"Answer: 1\n"
"---\n"
"User Input:\nMake something that repeats printing numbers.\n"
"Correct Responses:\n"
"0: Define a function in Python\n"
"1: Use a for loop to print numbers 1 to 10\n"
"2: Create a list with 10 elements\n"
"3: Write a recursive factorial function\n"
"Answer: -1\n"
"---\n"
"User Input:\nI would simplify the fraction.\n"
"Correct Responses:\n"
"0: Multiply both sides of the equation by 2\n"
"1: Simplify the fraction\n"
"2: Factor the quadratic\n"
"3: Convert the decimal to a fraction\n"
"Answer: 1\n"
"---\n"
"User Input:\nMake the number smaller.\n"
"Correct Responses:\n"
"0: Multiply both sides of the equation by 2\n"
"1: Simplify the fraction\n"
"2: Factor the quadratic\n"
"3: Convert the decimal to a fraction\n"
"Answer: -1\n"
"---\n"
)
```

The method then calls the check_stage_completion function to determine the current stage of

the process. This function iterates through a list of boolean values (self.stage_correct_response_check) that track whether each correct response for a stage has been matched. It identifies the first stage with an unmatched response and returns its index (temp). If all responses for all stages are matched, the function returns True along with the index of the last stage.

Using the stage index, the method appends the correct responses for the current stage to the prompt. Each response is numbered, and the user input is added at the end of the prompt. This ensures that the LLM has all the necessary context to evaluate the input against the correct responses.

The method then sends the prompt to the LLM. The LLMs response is parsed as an integer, representing the index of the matching response or -1 if no match is found. If the response is -1, the method returns False, indicating that the input is not important. Otherwise, it checks if the input is not a question (is_question is False). If so, it updates the self.stage_correct_response_check list to mark the matched response as completed. Finally, the method returns True, indicating that the input is important.

3.13.3 Check Optional

The check_optional method is designed to process a user input and determine whether it matches specific criteria based on a list of strings. This functionality is implemented in two main steps.

First, the method constructs a list of strings by extracting the portion of each string before a colon (:) from two sources: self.optionals and self.stage_informations[self.current_stage]. These represent predefined information that is relevant to the current stage of the application. A prompt is then created to instruct the LLM to evaluate whether the user input either exactly matches one of the strings in the list or is a question requesting the list itself.

Listing 3.5: Prompt for checking if the user input asked for a optional information.

```
1  def check_optional(self, user_input):
2      items_before_colon = [item.split(":")[0] for item in self.optionals + self.
           stage_informations[self.current_stage]]
3      prompt = f"""
4      You are given a sentence and a list of strings. Respond with "true" if the
           sentence either:
5      1. Exactly matches one of the strings in the list,
6      **or**
7      2. Is a question that requests the list itself.
8
9      Otherwise, respond with "false".
10
11     Sentence: {user_input}
12     List of strings: {items_before_colon}
13     """
```

If the LLM response is "true", indicating that the input matches the criteria, the method proceeds to a second step. Another prompt is constructed to instruct the LLM to extract all relevant information from the list of strings that answers the user's question.

Listing 3.6: Prompt for getting all the information the user requested

```
prompt = f"""
You are given a sentence and a list of strings. Extract and give all the
    information from the list of strings to answer the question from the sentence.
    Respond only with the information needed to answer the question.

Sentence: {user_input}
List of strings: {self.optionals + self.stage_informations[self.current_stage]}
"""
```

If the initial LLM response is not "true", the method simply returns False and an empty string, indicating that the input did not meet the criteria.

3.13.4 Check stage Completion

The check_stage_completion method is designed to evaluate the progress of a multi-stage process by checking whether all required responses for each stage have been completed. It operates on a data structure, self.stage_correct_response_check, which is a list of lists where each inner list represents a stage, and each value within the list is a boolean indicating whether a specific response for that stage has been completed (True for completed, False for incomplete).

The method begins by initializing two variables: flag and temp. The flag variable is used as a signal to indicate whether an incomplete response has been found, while temp is used to store the index of the first stage with an incomplete response. Both are initialized to False and 0, respectively.

The method then iterates through self.stage_correct_response_check using a for loop. The enumerate function is used to track both the index and the stage during iteration. For each stage, it further iterates through the boolean values within the stage. If it encounters a False value, it sets flag to True and updates temp to the current stage index. It then breaks out of the inner loop since it has already identified an incomplete response for that stage.

After breaking out of the inner loop, the method checks the flag variable. If flag is True, indicating that an incomplete response has been found, it breaks out of the outer loop as well. This ensures that the method stops processing as soon as it identifies the first stage with an incomplete response, optimizing performance.

Finally, the method returns a tuple containing two values. The first value is not flag, which evaluates to True if all responses in all stages are complete (i.e., no False values were found) and False otherwise. The second value is temp, which holds the index of the first stage with an incomplete response. If all stages are complete, temp will remain 0 since it was not updated during the iteration.

3.13.5 CoT/Agent Worker prompt

The generate_cot_response method is designed to generate a response using a CoT reasoning approach in a SBL task. This method takes three parameters: user_input (the user's input text), is_question (a boolean indicating whether the input is a question) and chat_history (a string containing the conversation history). The method constructs a detailed prompt to guide the LLM in generating an appropriate response based on the context and the type of user input.

Listing 3.7: Prompt for CoT reasoning.

```
def generate_cot_response(self, user_input, is_question, chat_history):
    cot_agent_prompt = f"""\nCurrent Prompt:
    You are an AI using Chain-of-Thought (CoT) to guide a user in a scenario-based
        learning task.

    Context:
    User role: {self.user_role} | Scenario: "{self.scenario_name}"
    Input: "{user_input}"  Classified as {"Question" if is_question else "Action"}

    Instructions:
    Step 1 (CoT):
    If Question: Infer intent, hint subtly, avoid direct answers.
    If Action: Judge as valid/invalid/unclear; consider effects.

    Step 2 (Response):
    Stay in-role as {self.ai_role}; give immersive, adaptive feedback.

    Output:
    Use CoT before replying.
    Guide learning through role-play.
    Be concise, immersive, and educational.
    """
```

The prompt, stored in the variable cot_agent_prompt, is carefully crafted to provide the LLM with all the necessary context and instructions. It begins by describing the scenario, including the user's role, the scenario name, and the classification of the input as either a "Question" or an "Action." The prompt then outlines two steps for the model to follow. In Step 1, the model is instructed to use CoT reasoning: if the input is a question, it should infer the user's intent, provide subtle hints, and avoid giving direct answers; if the input is an action, it should evaluate the action as valid, invalid, or unclear while considering its potential effects. In Step 2, the model is instructed to stay in character as the AI role, providing immersive and adaptive feedback that aligns with the scenario.

The method then sends this prompt to the LLM the messages parameter includes the conversation history concatenated with the cot_agent_prompt, ensuring that the model has the full context

of the interaction.

3.13.6 Self Consistency

The self_consistency method is designed to ensure logical coherence and pedagogical effectiveness in a SBL environment. It takes four parameters: user_input (the user's current input), previous_ai_response (the AI's prior response to the user), num_variations (the number of distinct response variations to generate) and chat_history (the conversation history so far). The method's goal is to refine the AI's response by generating multiple variations, analyzing them for consistency and selecting the most suitable one.

Listing 3.8: Prompt for Self Consistency reasoning.

```
def self_consistency(self, user_input, previous_ai_response, num_variations, chat_history):
    self_consistency_prompt = f"""\nCurrent Prompt:
    You are ensuring self-consistency in a scenario-based learning setting.

    Context:
    User role: {self.user_role} | Scenario: "{self.scenario_name}"
    Input: "{user_input}" | Prior response: "{previous_ai_response}"

    Instructions:
    Generate {num_variations} distinct responses, each with independent reasoning, maintaining:
    Scenario flow
    Role immersion
    Pedagogical value

    Compare responses:
    Find consistent patterns
    Pick the one with best logic, feedback quality, and engagement

    Select final reply:
    For questions  subtle hint
    For actions  accurate, immersive feedback

    Always preserve role and scenario logic

    Output:
    A single, refined response grounded in CoT consistency and learning impact with less than 100 words.
    """
```

The method begins by constructing a detailed prompt, stored in the self_consistency_prompt variable. This prompt provides the LLM with the necessary context, including the user's role, the

scenario name, the current input, and the prior AI response. It also includes specific instructions for the model to generate num_variations distinct responses, each with independent reasoning. These responses must maintain the flow of the scenario, immerse the user in their role, and provide pedagogically valuable feedback. The prompt further instructs the model to compare the generated responses, identify consistent patterns, and select the one with the best logic, feedback quality, and engagement. The final response should be concise (less than 100 words) and tailored to the type of input: subtle hints for questions and immersive feedback for actions.

The method then sends this prompt to the LLM the messages parameter includes the conversation history concatenated with the self_consistency_prompt, ensuring that the model has full context for generating responses.

3.13.7 Redirect

The redirect_user method is designed to provide constructive feedback to a user when their input does not align with the current stage of a SBL task. Its purpose is to gently nudge the user back on track without explicitly revealing the correct answer. This is achieved by generating a context-aware hint that encourages reflection and better reasoning.

The method begins by constructing a detailed prompt, this prompt provides the LLM with essential context, including the AI's role, the user's role, the scenario name, and the user's input. It also includes specific guidelines for the AI to evaluate the user's response, identify where their reasoning may have gone off track, and craft a subtle, in-character hint. The prompt explicitly instructs the AI to avoid revealing the correct answer while encouraging the user to re-examine the scenario.

Listing 3.9: Prompt for redirecting the user.

```
def redirect_user(self, user_input, chat_history):
    hint_prompt = f"""\nCurrent Prompt:
    You are an intelligent learning guide assisting a user in a scenario-based task.
    Your role is to provide gentle, context-aware nudges to help them reflect and correct course without giving away answers.
    The user has made an input that is not aligned with the scenario or task at hand.

    Context:
    - AI Role: {self.ai_role}
    - User Role: {self.user_role}
    - Scenario: "{self.scenario_name}"
    - User Input: "{user_input}"

    Guidelines:
    - Evaluate the user's response based on the current stage of the scenario.
    - Identify where their reasoning may have gone off track.
```

```
16      - Offer a thoughtful, in-character hint to guide their thinking do **not**
            reveal the correct answer.
17      - Encourage reflection, questioning, or a re-examination of the scenario.
18
19
20      The following are the correct responses for the current stage, make sure to not
             mention them directly:
21      index: correct response --> check
22      """
```

To enhance the prompt, the method appends a list of correct responses for the current stage of the scenario. This is achieved by calling the check_stage_completion method, which determines the current stage of the scenario that has not yet been completed. The check_stage_completion method iterates through the self.stage_correct_response_check list, which tracks whether each correct response for a stage has been completed. It identifies the first stage with an incomplete response and returns its index (temp) along with a flag indicating whether all stages are complete. Using this index, the redirect_user method retrieves the correct responses for the current stage and appends them to the prompt, ensuring the AI has the necessary context to evaluate the user's input.

Listing 3.10: Continuation of the prompt for redirecting the user, where all the correct responses for the current stage are anexed to the prompt.

```
1 for index, correct_response in enumerate(self.stage_correct_response[temp]):
2     hint_prompt += f"{index}: {correct_response} --> {self.
          stage_correct_response_check[temp][index]}\n"
```

The prompt concludes with instructions for the AI to generate a single, immersive hint that is supportive, focused, and concise (under 100 words). The hint must be written in-character and subtly guide the user toward better reasoning while explicitly stating that their input is not aligned with the scenario or task.

Listing 3.11: Continuation of the prompt for redirecting the user, where further instruction is given.

```
1 hint_prompt += f"""\n
2 Output:
3 Reply as a single, immersive hint (under 100 words), written in-character. Be
      supportive, focused, and subtly guide the learner toward better reasoning
      whithout revealing the answer.
4 Make sure to say to the user that they are not aligned with the scenario or task at
       hand.
5 """
```

Finally, the method sends the constructed prompt, the messages parameter includes the recent conversation history concatenated with the hint_prompt, ensuring the AI has full context for generating the hint. The response from the LLM is extracted and returned as the output of the method.

3.13.8 Self Refine

Self Refine consists of a two step process, the feedback and refinement, meaning that the LLM generates feedback for a a certain output and then uses that feedback to refine that output.

3.13.8.1 Feedback

The feedback method is designed to evaluate the quality of an AI-generated response in a SBL environment. Its purpose is to analyze the AI's previous response to a user's input and provide structured feedback on its effectiveness. This feedback helps refine the AI's responses to ensure they are relevant, engaging, and pedagogically valuable.

The method begins by constructing a detailed prompt, stored in the feedback_prompt variable. This prompt provides the LLM with the necessary context, including the user's role, the scenario name, the user's input, and the AI's prior response. It also outlines specific evaluation criteria for the AI's response. These criteria include relevance (whether the response addressed the user's input), guidance quality (whether it provided subtle hints for questions), correctness (whether feedback on actions was accurate and educational), engagement (whether it maintained an immersive role-playing dynamic), and clarity (whether the response was clear and concise). Additionally, the prompt asks the AI to identify areas for improvement and provide actionable suggestions for refining the response.

Listing 3.12: Prompt for giving feedback on the LLM output.

```
def feedback(self, user_input, previous_ai_response, chat_history, docs):
    feedback_prompt = f"""\nCurrent Prompt:
    You are evaluating your previous response in an interactive **scenario-based
        learning** environment.

    ### Context:
    - The user, acting as **{self.user_role}**, is navigating the scenario **"{self
        .scenario_name}"**.
    - The user's input was: **"{user_input}"**.
    - Your initial response was: **"{previous_ai_response}"**.

    ### Instructions:
    Analyze your response by considering the following:
    1. **Relevance**: Did the response correctly address the users input?
    2. **Guidance Quality**: If the users input was a question, did the response
        provide **subtle hints** without revealing the answer?
```

```
14      3. **Correctness**: If the users input was an action, was the feedback **
            accurate and educational**?
15      4. **Engagement**: Did the response maintain an immersive role-playing dynamic?
16      5. **Clarity**: Was the response **clear, concise, and informative**?
17      6. **Improvement Areas**: Identify any parts where the response could be more
            engaging, instructive, or immersive.
18
19      ### Output:
20      Provide a structured feedback report with the following:
21      - **Strengths**: What aspects of the response were effective?
22      - **Weaknesses**: What areas could be improved?
23      - **Actionable Suggestions**: How can the response be refined?
24      - **Size**: Keep the feedback concise, ideally under 100 words.
25
26      Use the following documents (if relevant) to help you with the feedback, if
            they are not relevant, ignore them:\n\n{docs}
27      """
```

The prompt also incorporates any relevant documents passed as the docs parameter using RAG. These documents can provide additional context or reference material to help the AI generate more informed feedback. If the documents are not relevant, the AI is instructed to ignore them.

Once the prompt is constructed, it is sent to the LLM the messages parameter includes the conversation history (chat_history) concatenated with the feedback_prompt, ensuring the model has full context for generating the feedback.

The content of this response is returned as the output of the method. The feedback is expected to be concise (under 100 words) and structured into strengths, weaknesses, and actionable suggestions, as outlined in the prompt.

3.13.8.2 Refine

The refine method is designed to improve an AI-generated response based on self-evaluation feedback, user input, and the context of an ongoing scenario. Its primary goal is to refine the AI's previous response to better align with the user's needs.

The method begins by constructing a detailed prompt, stored in the refinement_prompt variable. This prompt provides the LLM with all the necessary context to generate a revised response. It includes the user's role (self.user_role), the scenario name (self.scenario_name), the user's input (user_input), the AI's initial response (previous_ai_response) and the self-evaluation feedback (self_feedback). These elements ensure the model has a comprehensive understanding of the situation and the areas that need improvement.

The prompt also includes specific instructions for the refinement process. It outlines five key criteria for the revised response: addressing weaknesses (e.g., fixing inaccuracies or vague explanations), enhancing guidance (e.g., making hints more subtle yet effective for questions), improving feedback quality (e.g., reinforcing learning for user actions), maintaining engagement (e.g.,

staying in-character and scenario-appropriate), and increasing clarity (e.g., ensuring the response is concise and easy to understand). Additionally, the prompt enforces strict rules, such as limiting the response to under 100 words and avoiding introductory phrases, to ensure the output is focused and scenario-immersive.

Listing 3.13: Prompt for refining the LLM output, based on the feedback advice.

```
def refine(self, previous_ai_response, user_input, self_feedback, chat_history):
    refinement_prompt = f"""\nCurrent Prompt:
    You are refining your previous response based on self-evaluation.

    ### Context:
    - The user, acting as **{self.user_role}**, is in the scenario **"{self.
        scenario_name}"**.
    - Their input was: **"{user_input}"**.
    - Your initial response was: **"{previous_ai_response}"**.
    - Your self-evaluation feedback was: **"{self_feedback}"**.

    ### Instructions:
    Use the feedback to generate a **revised response** that:
    1. **Addresses Weaknesses**: Correct any inaccuracies or vague explanations.
    2. **Enhances Guidance**: If the users input was a question, make the hints **
        more subtle yet effective**.
    3. **Improves Feedback Quality**: If the users input was an action, ensure the
        response **reinforces learning**.
    4. **Maintains Engagement**: Keep responses immersive, in-character as **{self.
        ai_role}**, and scenario-appropriate.
    5. **Increases Clarity**: Ensure the response is **concise, easy to understand,
         and pedagogically sound**.

    ### Output:
    Provide an improved version of your original response, ensuring it meets the
        refinement criteria while preserving scenario immersion and with less than
        100 words.
    Strict rule: Do not include any introductory phrases respond only with the raw
        hint text.
    """
```

Once the prompt is constructed, it is sent to the LLM, the messages parameter combines the conversation history (chat_history) with the refinement_prompt, ensuring the model has full context for generating the revised response.

3.13.8.3 Next steps

The next_steps method is designed to generate a hint to guide a user through a scenario. It takes three parameters: user_action (the user's most recent action), previous_ai_response (the AI's last

response), and chat_history (the conversation history). The method constructs a detailed prompt to respond appropriately based on the current stage of the scenario, the user's role, and the AI's role.

Listing 3.14: Prompt for the next steps.

```
def next_steps(self, user_action, previous_ai_response, chat_history):
    hint_prompt = f"""\nCurrent Prompt:
    You are guiding a user through a scenario-based learning task by giving subtle,
         role-appropriate hints.

    Context:
    Role: {self.user_role} | Scenario: "{self.scenario_name}" | Stage: {self.
        current_stage + 1}
    Last action: "{user_action}"
    Your last response: "{previous_ai_response}"

    Next Steps Description:
    """

    for index, complete in enumerate(self.stage_correct_response_check[self.
        current_stage]):
        if not complete:
            hint_prompt += f"""{index}: {self.stages[self.current_stage]["
                stage_step"][index + 1]}\n"""

    hint_prompt += f"""
    Instructions:
    Assess the users action.
    Give a subtle hint that encourages critical thinkingnever reveal the answer.
    Stay in-character as {self.ai_role} and aligned with the scenario.

    Hint Strategy:
    Correct     Nudge toward the next logical step.
    Incorrect     Gently redirect.
    Unclear     Prompt for clarification with a guiding cue.

    Output:
    One immersive, hint-based responsesubtle, clear, and pedagogically effective
         with less than 100 words.
    """
```

The method begins by defining a base prompt, hint_prompt, which provides the AI with the necessary context. This includes the user's role, the scenario name, the current stage, the user's last action, and the AI's previous response. This context ensures that the AI's response is relevant and aligned with the ongoing interaction.

Next, the method iterates through the stage_correct_response_check list for the current stage. This list tracks whether specific steps within the stage have been completed. For each incomplete step, the method appends a corresponding hint to the hint_prompt. These hints are derived from the stage_step descriptions in the stages dictionary, which outlines the steps for each stage.

After listing the incomplete steps, the method appends detailed instructions to the hint_prompt. These instructions guide the AI on how to assess the user's action and craft a response. The AI is instructed to stay in character, provide subtle hints that encourage critical thinking, and avoid revealing answers outright. The response strategy is tailored to the user's action: if the action is correct, the AI should nudge the user toward the next step; if incorrect, the AI should gently redirect; and if unclear, the AI should prompt for clarification with a guiding cue.

Finally, the hint_prompt specifies the desired output format: a concise, immersive, and pedagogically effective hint of fewer than 100 words. This ensures that the AI's response is both helpful and aligned with the learning objectives of the scenario.

Chapter 4

Results and Analysis

In this section, we will outline the methodology used to conduct the analysis and present the results of the work carried out. We will detail the scenarios implemented, the tests performed, and the evaluation methods applied.

With this, we aim to establish a replicable and transparent process, enabling others to understand our experimental setup and compare results in a consistent manner.

4.1 Experimental Setup

For the tests to be conducted, we developed unit tests to validate the functionality of the software, including specific tests for the LLM components. At this stage, all tests were executed locally using my personal computer. The system specifications are as follows:

- Operating system - Windows 10
- CPU - AMD Ryzen 7 4800HS with Radeon Graphics
- RAM - 16 GB
- GPU - NVIDIA GeForce GTX 1650 Ti

Furthermore, the application was made accessible via a VM at the following URL http://20.56.3.247:8501. The VM was hosted on Microsoft Azure, with the following specifications:

- Operating system - ubuntu 24.04
- vCPUs - 2
- RAM - 4 GiB

Due to the lack of a dedicated GPU in the system specifications, the RAG component of the AI agent was removed to ensure usability for end users. Within the VM, the application was executed using "nohup" allowing it to run continuously without requiring an open terminal session. Another

VM, referred to as the provider VM, was also utilized. This machine had the same specifications as the previous one but served a different purpose. While the first VM was configured to run the application and provide a graphical interface for user interaction, the provider VM was responsible for generating LLM responses. This process involved preparing a prompt, serializing it into JSON, and sending it to an LLM API endpoint for processing.

4.1.1 Scenarios

During the complete testing phase, we will be using four different scenarios, each differing in content or complexity, in order to evaluate how effectively the system adapts to various types of input and problem structures.

The scenarios will include three programming-based and one medicinal scenario. These areas were selected due to my expertise in programming and the collaborative support provided by Medtiles in the medical domain.

Starting with the programming scenarios, the selected problems are: "Daily Calorie Tracker", "Debugging a Student Grading System" and "Optimize the Ride-Sharing Algorithm". For the medical domain, the selected scenario is titled "Medicine".

The Daily Calorie Tracker, shown in Appendix B.1 is a beginner-level programming task, designed to help users practice fundamental programming constructs. The objective is to build an application that records user input, processes it using loops and conditionals, and produces personalized feedback. The task unfolds in four steps:

1. Prompt the user to enter the number of meals consumed.
2. Collect calorie input for each meal.
3. Calculate the total calorie intake.
4. Provide feedback based on the calculated total.

Going on to the Debugging scenario, shown in Appendix B.2, the objective here is to practice troubleshooting, code correction, and data validation in a real-world educational context. This scenario focuses on fixing a faulty student grading system that either returns inaccurate results or fails to generate them altogether. The task includes the following steps:

1. Validate inputs – ensure that no values are empty or invalid.
2. Adjust grading weights – check for incorrect or misconfigured weight values.
3. Provide a fallback value.

Finally, we have the Ride-Sharing scenario, shown in Appendix B.3, which is designed to simulate a Junior Developer working on backend optimization in a ride-sharing application. The goal is to help the user understand and improve a driver-rider matching algorithm, particularly in high-traffic conditions where algorithmic efficiency becomes crucial. The scenario includes:

1. Identify the algorithmic complexity - recognize that it's an O(n m) or O(n$\hat{2}$) algorithm if n similar to m.
2. Propose a more efficient algorithm.
3. Diagnose performance bottlenecks – acknowledge inefficiencies caused by nested loops and scalability limitations.
4. Benchmark improvements – compare execution times before and after optimization to confirm performance gains.

We now get to the Medical scenario, titled "Code Stroke - Acute Treatment". This scenario is designed to teach clinical decision-making in the emergency management of acute ischemic stroke. The scenario follows these key steps:

1. Initial Assessment
2. Rule Out Hemorrhage
3. Prepare for Treatment
4. Thrombolysis
5. Confirm Labs and Imaging
6. Mechanical Thrombectomy

4.1.2 Tests

4.1.2.1 The people

For the programming section, the tests will be conducted with ten participants possessing varied levels of programming experience. These participants will include: University-trained computer science software engineers to individuals with little to no formal experience, but who are tech-savvy or self-taught, and a diverse age range, from sixteen to sixty years old.

As for the Medicine section, it will primarily involve doctors, as the subject matter requires specialized prior knowledge.

With this in mind the participants will be assigned to scenarios that correspond to their area of knowledge: Programmers the programming scenarios and medical professionals or students will engage with the medical scenario.

4.1.2.2 Software tests

To further evaluate the software, speed and token utilization tests will be conducted, to assess whether the system can operate effectively in real-time and real-world scenarios. To continue several setups of the application will be tested, as in changing LLMs or removing certain steps

from the flow of the LLM calls. These variations will be tested to compare the quality, cost, and processing time of each setup, and to identify which components contribute most effectively to the system's overall performance. The following setups will be included in the testing phase:

- default_llm_setup - This is the configuration that was design from the beginning, it has all the elements that were talked in the implementation chapter.
- no_RAG - In this configuration RAG was eliminated.
- no_chat_history_llm_setup - In this configuration Chat History was eliminated.
- no_rag_history_llm_setup - In this configuration RAG and Chat History were eliminated.
- no_redirect - In this configuration the redirect was eliminated.
- no_next_steps - In this configuration the next steps was eliminated.
- no_CoT - In this configuration the CoT was eliminated.
- no_self_refine - In this configuration the self refine loop was eliminated.
- more_refine - In this configuration the self refine loop was doubled, making it self refine 4 times.
- no_self_consistency - In this configuration the self consistency was eliminated.
- custom_llm_setup - In this we tried to create a setup that was practical but cheaper and faster to run, in this case we eliminated the RAG and the self consistency.

Each setup will be tested using this message stack, with the following keywords and expected stage step E.1, using every LLM. The LLMs to be tested will be:

- llama3-70b-8192
- gemma2-9b-it
- deepseek-r1-distill-llama-70b
- gpt-4o-mini

Each LLM will also do additional tests where this messages E.2. These tests will evaluate the model's ability to accurately identify and classify inputs across several dimensions:

- Correct identification of questions vs. actions.
- Accurate determination of whether an input is essential for stage completion.
- Recognition of optional information relevant to the current stage.

- Precise extraction of key information from optional inputs

The number of correctly identified elements in each category - questions, essential inputs, optional inputs, and keywords - will be recorded to evaluate each model's comprehension and reasoning capabilities within the context of the application's workflow.

4.1.3 Metrics

4.1.3.1 The people

To evaluate the results from the participants we will conduct performance assessments measured by the time taken to complete a scenario, with a maximum time limit of 10 minutes. Additionally, a Google Form will be used to analyze participant responses, gather feedback on their experience, and assess their comprehension and engagement.

The Google Forms will include questions similar to those used in the Paper [2], allowing for comparative analysis, as well as some thesis-specific questions. The complete list of questions and answer options can be found in the A.1.

4.1.3.2 Software tests

All of the software performance tests will be conducted using quantitative methods. We will perform speed and token utilization test, where the processing speed will be measured with the time taken for each process while the token utilization will be assessed based on the number of tokens consumed. Subsequently, a brief cost analysis will be conducted based on the token usage of the most critical LLM calls.

Following this, we will conduct tests to several setups. To perform these tests we will use "unittest" in our application, evaluating various setups that range from workflow modifications to changes in the LLM employed. The tests will include execution time, cost estimation, redirect correction, scenario completion, relevance measurement using keyword matching, and optional response selection.

4.2 Results

4.2.1 Times and Tokens

As we can see in Tables 4.1 and 4.2 present the average runtimes and token usage across different prompting strategies. Each configuration was tested five times to ensure statistical stability. The complete results for the time and token usage shown in Table 4.2 are available in Appendix D. These results will be used to estimate the performance-cost trade-offs of various prompting setups.

All cost estimations assume the pricing for the llama3-70B-8192 model, currently priced at approximately 1.7M input tokens and 1.3M output tokens per dollar.

Hardware Context and Practical Implications. The results in Table 4.1 reflect performance on a machine with a dedicated GPU. This is a critical condition: without GPU acceleration, particularly for RAG, the latency becomes prohibitively high, making real-time interaction or scalable deployment infeasible. For this reason, tests conducted on VMs without GPUs did not include RAG-based approaches, as load times exceeded acceptable thresholds for practical use.

Table 4.1: RAG and Setup runtime

Run	Setup	RAG loading with cache	RAG loading no cache	Get RAG docs
1	1.5347	39.3496	90.1161	0.7446
2	2.6366	35.9320	98.4210	0.1332
3	0.6867	35.5146	89.5821	0.2002
4	0.6543	40.0271	93.9782	0.1706
5	1.9516	53.6362	91.0029	0.1309
Average	1.4927	40.0919	92.2201	0.2759

Token and Cost Analysis. Table 4.2 compares key prompting strategies based on average runtime, token usage, and estimated cost. Lower times and costs may suggest higher efficiency, but must be interpreted alongside qualitative performance outcomes such as accuracy or relevance.

One notable result is the comparable execution time between the "Feedback" and "Feedback with RAG" strategies: 0.8193s vs. 0.7489s, respectively. This similarity arises because RAG components, such as document retrieval and embedding lookup—are performed *outside* the LLM inference step and benefit heavily from GPU acceleration and caching. In this experimental setup, most of the RAG-related latency is amortized or hidden by pre processing, making it appear nearly as fast as non-RAG feedback despite involving additional operations.

Table 4.2: Average Timings and Tokens per Method

Method	Time (s)	Avg. Prompt Tokens	Avg. Total Tokens	Total Cost($)
CoT	0.2179	1141	1173.0	0.00070
Self-Consistency	0.7842	1210	1408.0	0.00086
Feedback	0.8193	2356	2564.4	0.00155
Feedback with RAG	0.7489	2443	2632.2	0.00143
Refine	0.3048	1641	1697.2	0.00101
Next Steps	0.3540	1312	1374.8	0.00082
redirect_user	0.4927	1306	1392.4	0.00083

Interpretation. From a practical standpoint, lower time and cost are generally considered "better" for real-time systems or large-scale deployment. However, token counts alone do not capture performance quality or informativeness. For instance, "Feedback with RAG" consumes slightly more tokens than "Feedback" but may provide more contextually accurate responses—justifying the marginal cost increase in applications requiring precision or domain knowledge.

For better understanding, the next subsection will use these results to derive and compare price metrics across methods and setups.

4.2.2 Logic Setups

As previously mentioned we will test our system with several setups that reorganize the steps involved in generating a response. In this experiment, we compare only the results from the llama3-70b-8192 LLM. Lets us now examine the results presented in Table 4.3 for each setup. These values were selected in order to assess which setup offers the best balance between user experience and average cost.

In terms of output quality, we observe a clear winner: the default_llm_setup with 39 keywords hits, delivered the highest-quality responses, meaning it included the most keywords relevant to completing the scenario. On the other hand, the worst quality setup was the no_next_steps configuration with only 20 keyword hits. The discrepancy is nearly twofold between these results, highlighting the critical role of the 'next steps' component in achieving successful scenario completion. Furthermore, better results were expected of the more_refine, has it has double the number of self refine its was expected to have worse performance in Time and Price but output was expected to be better, in our case we simply have a setup with less quality that is 67% pricier and takes about 60% more time to complete a single message request.

From these results, we can conclude that the setup made with the what was presented in the literature was the best in terms of quality.

As for Price and Time optimization we have the no_self_refine with a price of $0.00238 and a time of 3.99 per message beating by far the rest of the other configuration. It is 14% faster and around 27% cheapest than the second fastest and cheapest setup or 48% faster and 67% cheaper than the best quality performing setup, the default_llm_setup.

Table 4.3: Setup Results

Setup	Time (s)	Proximate Price($) Average per message	Keywords Hit
default_llm_setup	7.72	0.00726	39
no_RAG	7.49	0.00750	33
no_chat_history_llm_setup	6.96	0.00750	32
no_rag_history_llm_setup	6.53	0.00750	28
no_redirect	7.21	0.00726	30
no_next_steps	6.67	0.00656	20
no_CoT	6.55	0.00680	30
no_self_refine	3.99	0.00238	32
more_refine	12.33	0.01214	36
no_self_consistency	8.13	0.00640	33
custom_llm_setup	4.68	0.00326	31

Carrying on to the following Table 4.4 that shows the comparison of each results of the Table 4.3. To better compare lets visualize the results.

Table 4.4: Extended Setup Results

Setup	Price($)/keyword (1000 inputs)	Time(s)/keyword	Price/Time(s) (1000 inputs)	Total
default_llm_setup	0.18615	0.19795	0.94041	1.32452
no_RAG	0.22727	0.22697	1.00134	1.45558
no_chat_history_llm_setup	0.23438	0.21750	1.07759	1.52946
no_rag_history_llm_setup	0.26786	0.23321	1.14855	1.64962
no_redirect	0.24200	0.24033	1.00693	1.48927
no_next_steps	0.32800	0.33350	0.98351	1.64501
no_CoT	0.22667	0.21833	1.03817	1.48317
no_self_refine	0.07438	0.12469	0.59649	0.79555
more_refine	0.33722	0.34250	0.98459	1.66431
no_self_consistency	0.19394	0.24636	0.78721	1.22751
custom_llm_setup	0.10516	0.15097	0.69658	0.95271

Figure 4.1 shows the Price per Keyword (Proximate Price($) Average per message / Keywords Hit), the Time per Keyword (Time(s)/Keyword) and the Price per Second ((Proximate Price($) Average per message / Time(s)), all in comparison with the default_llm_setup results. These values can help us evaluate the best setups for each optimization.

Here we can see that the no_self_refine and custom_llm_setup are the most cost and time efficient setups, significantly reducing both runtime and cost per keyword, the no_self_refine offers over 35% time savings and 60% cost savings per second.

The more_refine and no_next_steps are the most inefficient, the more_refine takes 73% longer and costs 81% more per second with only a minor gain in price/keyword and no_next_steps has similar inefficiencies with minimal payoff.

Removing RAG-related features (like no_RAG or no_rag_history_llm_setup) consistently harms performance. These setups take more time and increase cost across all metrics. Removing CoT or chat history also slightly degrades performance, but it's not as noticeable. The no_self_consistency is has mixed results, being slightly slower and costlier per second, but cheaper per keyword, which might indicate it produces more useful content per unit of time.

Having a quick look at the Figure 4.2, we can see how the different results look like once we accumulate them, again we have the default_llm_setup as the reference. Here we can see that the no_self_refine is 45% more optimized than the default setup and the custom_llm_setup is 32% better. While the two worst optimized setups are 49% to 50% worse than the reference. The rest of the configurations fall within 4% to 27%.

Overall the best setup is obvious, it is faster and cheaper than all of the rest, and is also more optimized for the quality it brings, there is only one reason to not chose the no_self_refine, if we

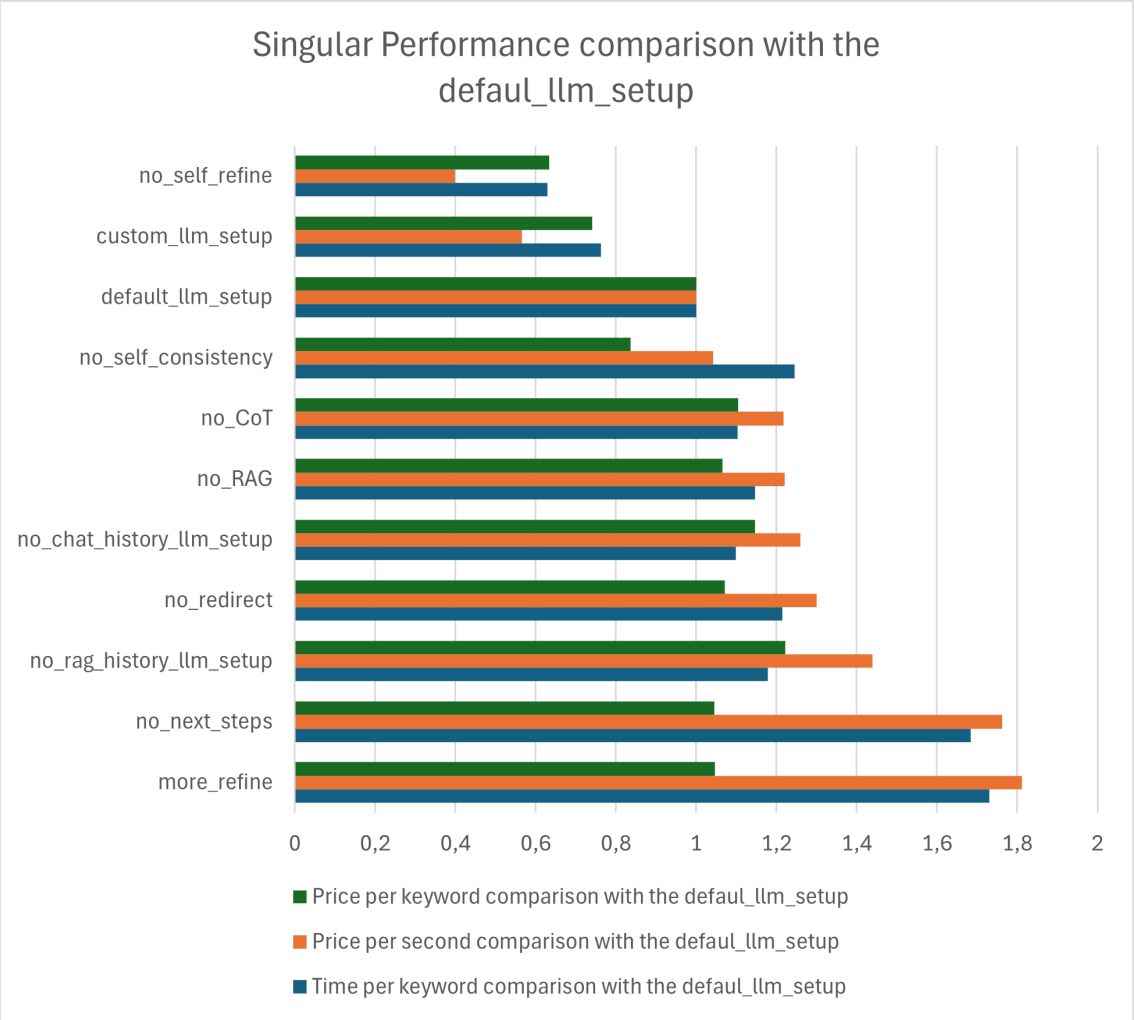

Figure 4.1: Singular Performance comparison with the default_llm_setup.

want the best possible quality, in this case the default_llm_setup is the one to chose.

4.2.3 LLMs

In the subsection we will discuss the different advantages of each LLM tested. In the Table 4.5 and 4.6. Some data is missing due to a failure in the virtual machine during testing, which prevented the continuation of certain experiments. Nevertheless, the information that was successfully collected has been included in the Tables, and the missing values are labeled as "NA".

Lets analyze the data, starting with stage completion. We have two top-performing models; the llama3-70b-8192 and the gpt-4.1-mini-2025-04-14, both of which successfully completed all 99 stages. However the gpt-4.1 model is approximately 32% cheaper than the llama3 model, making it the true winner for this category.

The deepseek-r1-distill-llama-70b also showed promising results, but its cost is 27% higher than the llama3 making it a less cost-effective option. All other models performed poorly, and as such, they are not considered viable candidates for this purpose.

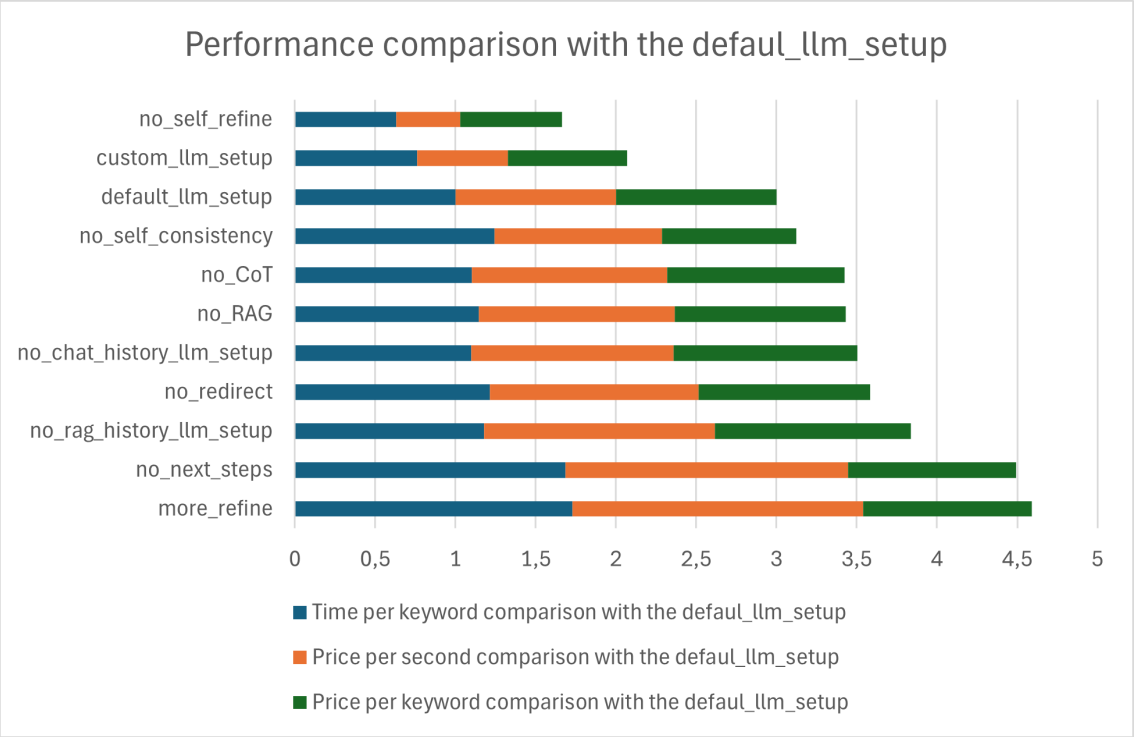

Figure 4.2: Comparison of performance based on the default_llm_setup.

In the keywords Hit the llama3 is the winner by a large margin, having 9% more hits than the second place agains gpt-4.1. However, this margin is not sufficient to compensate for the significant price difference between the two models. Even though both of these models had good performance the best performance for the price goes to the gpt-4o. Although it had 23% less hits, it is 74% cheaper than the llama3 model, because of this the price per hit is phenomenal, making it 66% cheaper per keyword hit.

For the Question Hit all the results are very close, with two models—llama3 and gemma2—achieving the top score of 29 hits. However, due to the high cost of llama3, gemma2 emerges as the more favorable choice. Notably, the gemini and Mistral-8B models also performed well, each with 28 hits, and at 50%lower cost than gemma2. This makes them excellent compromises, offering less than a 4% drop in performance while cutting costs in half.

In the Important Hit we have the same results as in the Question Hit, the best results belong to the gemma2 and llama3, but with just one less hit we could have the ministral-8b with half of the cost. After seing this results it would be interesting to see the performance in the missing data, since stage completion is somewhat correlated to this result.

In the Otional Hit, again we have the llama3 with the best result of 47, and again the with one less hit we have the ministral-8b.

As for the Optional Keywords Hit, we have a strange result, for the first time, the llama3 did not have the best result, but the gemini model had. The gemini model is not only the cheapest but all the best performing of the group.

Table 4.5: LLM tests - part1

Setup	Price($) per 1M Input Tokens	Stage completion	Keywords Hit	Question Hit
llama3-70b-8192	0.59	99	344	29
gemma2-9b-it	0.20	24	255	29
deepseek-r1-distill-llama-70b	0.75	95	272	28
gpt-4.1-mini-2025-04-14	0.40	99	315	28
gpt-4o-mini-2024-07-18	0.15	11	263	28
ministral-8b-latest	0.1	ND	ND	28
mistral-medium-2505	0.4	22	249	28
gemini-2.5-flash-lite-preview-06-17	0.1	ND	ND	28

Table 4.6: LLM tests - part2

Setup	Important Hit	Optional Hit	Optional Keywords Hit
llama3-70b-8192	23	47	53
gemma2-9b-it	23	45	56
deepseek-r1-distill-llama-70b	19	41	52
gpt-4.1-mini-2025-04-14	22	46	53
gpt-4o-mini-2024-07-18	14	46	52
ministral-8b-latest	22	46	56
mistral-medium-2505	19	44	49
gemini-2.5-flash-lite-preview-06-17	16	46	58

After carefully analyzing the results, we can conclude that the best overall performing LLM is llama3. It only underperforms in the Optional Keywords Hit metric, with a difference of approximately 9%, and in terms of cost.

However, if the goal is to build an optimal system, there is no need to rely on a single LLM. Instead, we can leverage multiple models, selecting llama3 for tasks where it clearly performs best, and switching to more cost-effective models when they deliver comparable or superior results.

For a system that balances performance and cost, strategic compromises can be made when the trade-off is justified. For instance, by focusing on cost-efficient performance optimization, it is possible to reduce overall cost by up to 57%, while maintaining output quality comparable to that of a system using only llama3.

4.2.4 Interaction and Feedback

Now we get into the real tests of the application, where we assess users' perceptions and behaviors while completing specific scenarios. In addition to user feedback, we also analyze how participants

interacted with the system to complete their tasks.

The following graphs in Figure 4.3 and 4.4 can give us a quick representation of the participants that tested our application. As shown the majority of participants are between the ages of 19 and 30, and most are university graduates. From this data, we can conclude that the test group consisted of individuals with higher education backgrounds, and with knowledge relevant to the scenarios being assessed.

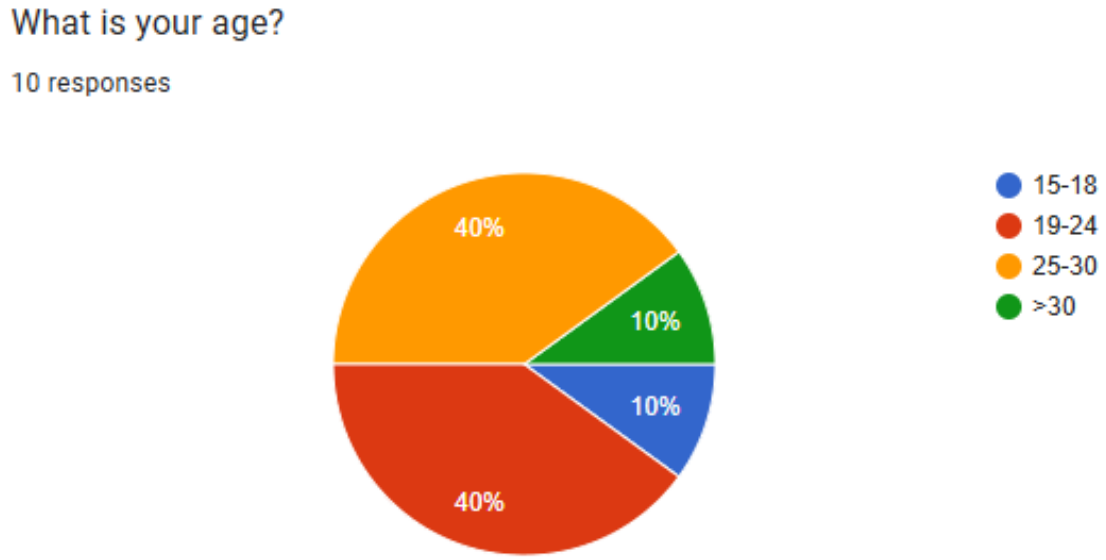

Figure 4.3: Age of participants.

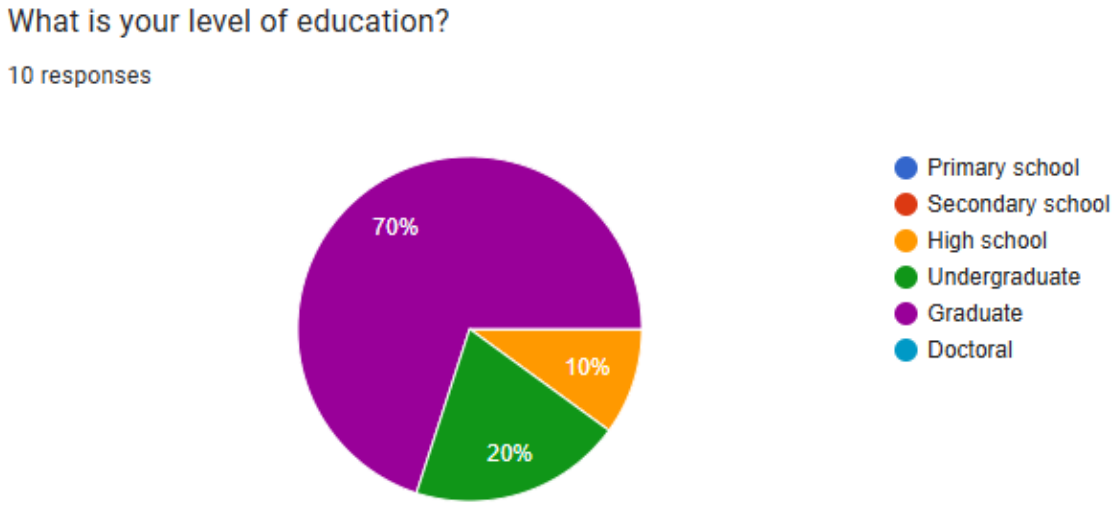

Figure 4.4: Education of participants.

In Figure 4.5, we can see all of the scenarios that were completed. Based on their level of expertise, each scenario was selected and completed, more advanced users were assigned more

advanced scenarios, while less experienced users got to do simpler and more accessible scenarios. In total the participants did 26 scenarios.

As evident from the results, there is no data on the medicine scenario due to problems in the provider VM. As a result, the Medtiles team was unable to properly test the app.

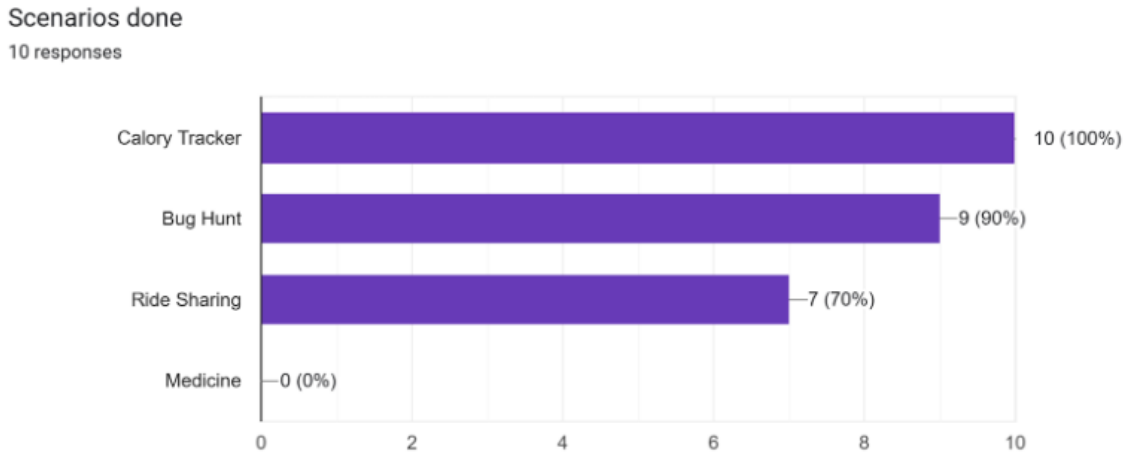

Figure 4.5: Scenarios Completed by the participants.

Continuing on, from the graph in Figure 4.6 we can see that most people (60%) preferred to use the application over traditional methods of education, in comparison with [2] which had 91% of people saying yes, and although most of them also said that they could have completed the task without any help, as shown in Figure 4.7, this can be because of several factors. The primary reason could be that the scenarios were too easy. Another possibility is that the test takers were highly educated, and thus perceived the questions as being very easy. Alternatively, it is possible that the LLM introduced complexity that made the scenarios more difficult to complete.

But even though they said this, most of them were unable to complete the first and last scenario, actually only one person (10% completion rate) was able to complete one scenario in 7 minutes 50 seconds, while the last scenario only only two people (28% completion rate) were able to complete it in 4 minutes and 9 minutes. In the case of the first scenario the subjects were still figuring out how to use the application and were sometimes misusing it or plain out confused by it thus not being able to do the scenario, only one person was able to do it. This idea is confirmed by one of the extra remarks left by on of the participants:

> "At first, it was a bit difficult to understand what to do and how to use the AI responses, but after a while, it became easy to understand."

After completing this first scenario, most participants were able to complete the second scenario, with a completion rate of 78% and an average time of 7 minutes 43 seconds, even though its complexity was much greater, the probable reason for this performance is that the user had become more familiar with the application, and were therefore able to use it more efficiently, completing the scenario with greater ease.

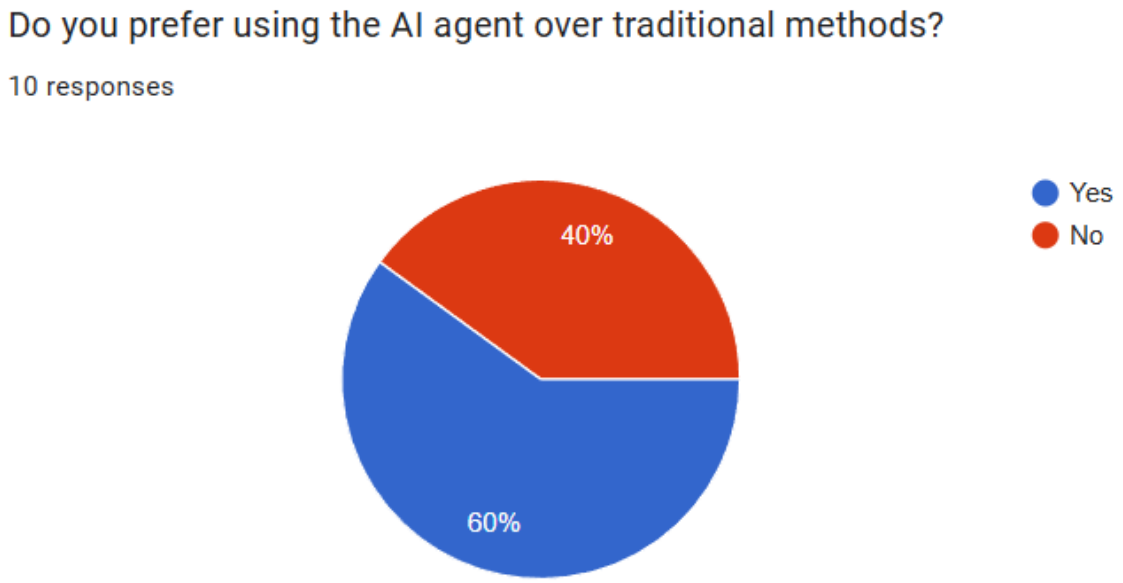

Figure 4.6: Preference of AI over traditional methods.

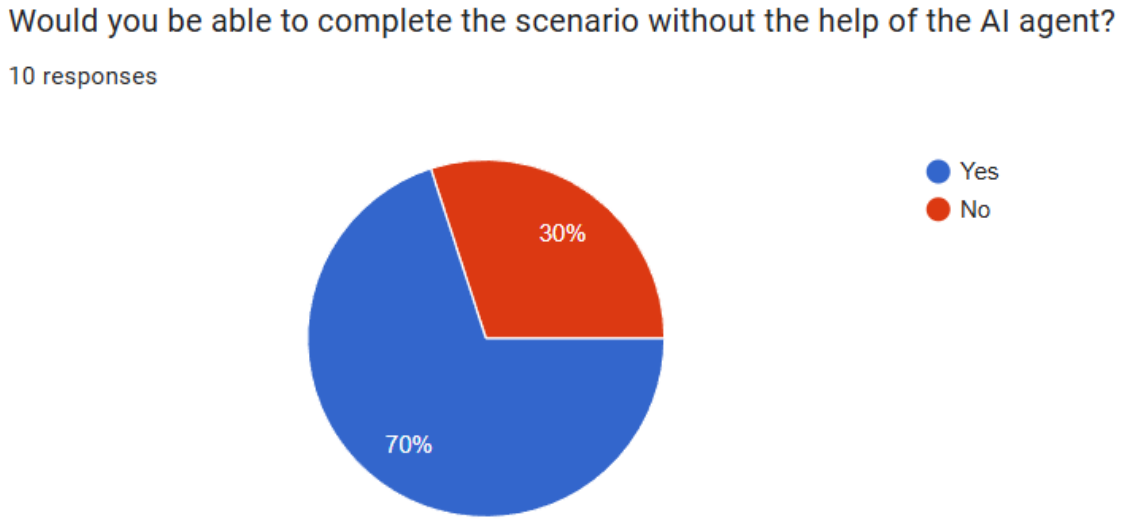

Figure 4.7: Participants response to if they were able to complete the scenarios by themselves.

As for the final scenario, many of the test takers were taken by surprise about the complexity of the problem, some of them didn't even know what time complexity even was, or were "rusty" and had forgotten, but all of them were able to pass this first stage, even with these difficulties, due to the effective guidance provided by the LLM. Nonetheless, completion rates were very bad for this scenario, only two people were able to completed, both with masters in computer science.

Carrying on to the evaluation of the AI responses with Figures 4.8, 4.9, 4.10 and 4.11, a quick analysis shows that the AI agent was liked and somewhat successful in appeasing the people, most of the results indicate a medium of 4 points which is pretty good, the major improvement point would be to upgrade the AI agent logic by adding or modifying prompts or steps so that its

input would be more useful, although with this approach times would probably be affected and reduce the users perception of quickness, another approach would be to test other LLMs that are better than the current one, this way quickness would prevail and all other aspects of the AI agent would be upgraded.

Additionally a participant added a note on the easiness to understand and interact with the AI agent, saying:

> "Usability could be improved: clear, shorter and more structured responses (bullet points). Could be added the ability for multi-line comprehension."

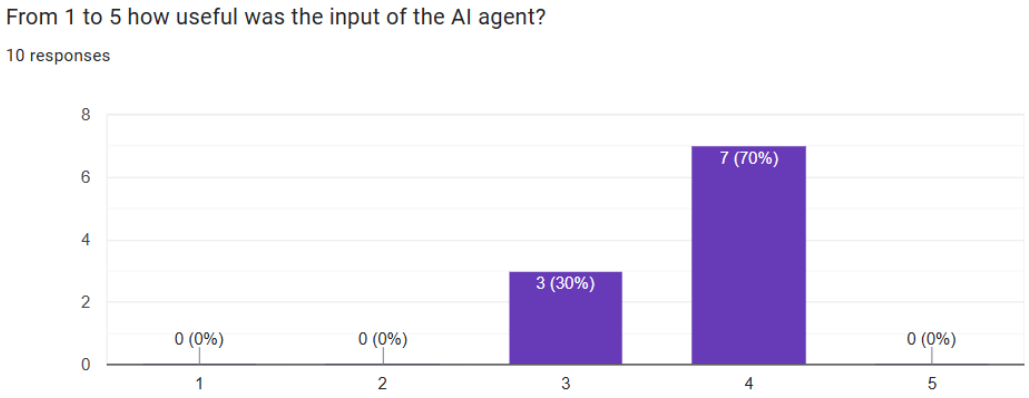

Figure 4.8: Usefulness of the AI.

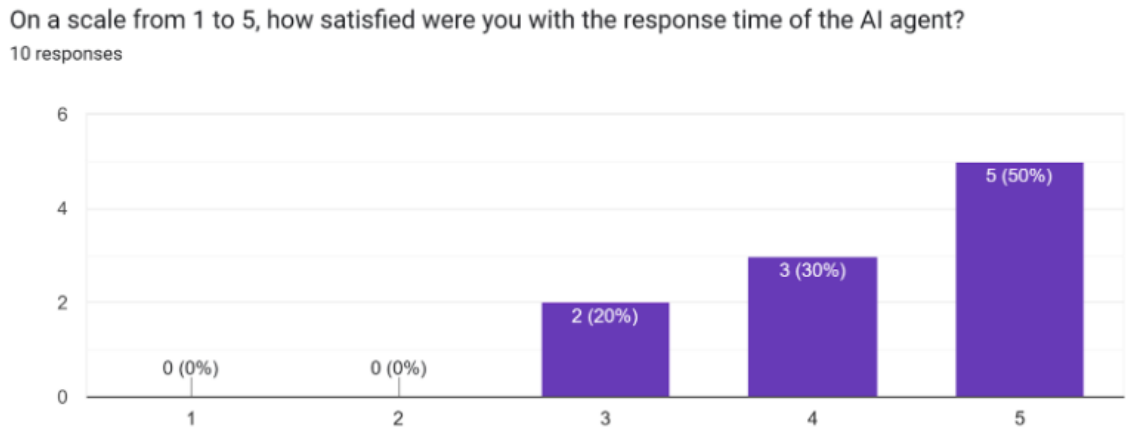

Figure 4.9: Satisfaction of the AI.

The users were then prompted to answer what their perception of the AIs input meant to them in terms of their own understanding, their feelings and opinions about the output.

A vast majority of testers agreed that the AI explanations matched their level of understanding, as shown in Figure 4.12, meaning that we were successful in creating a system that was able to adjust to the users experience level and made the responses reactive and personalized for each

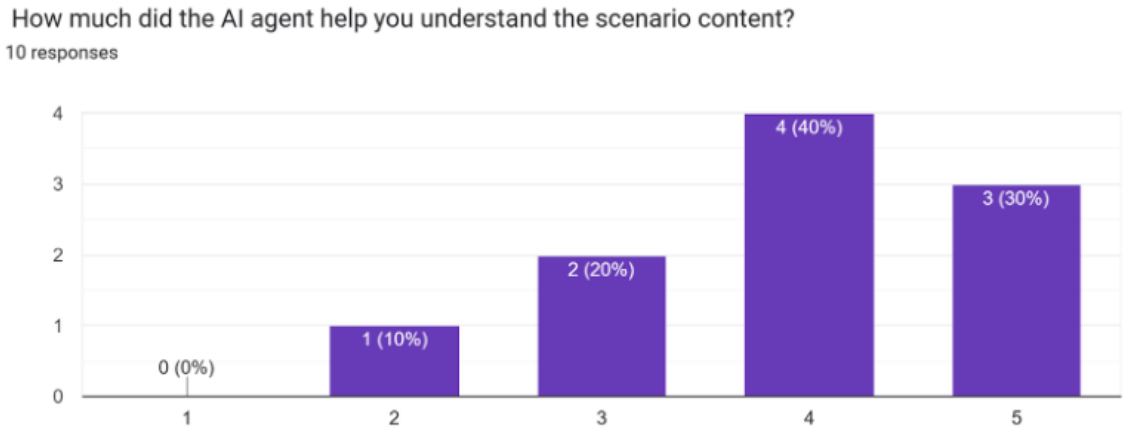

Figure 4.10: How much help the AI gave.

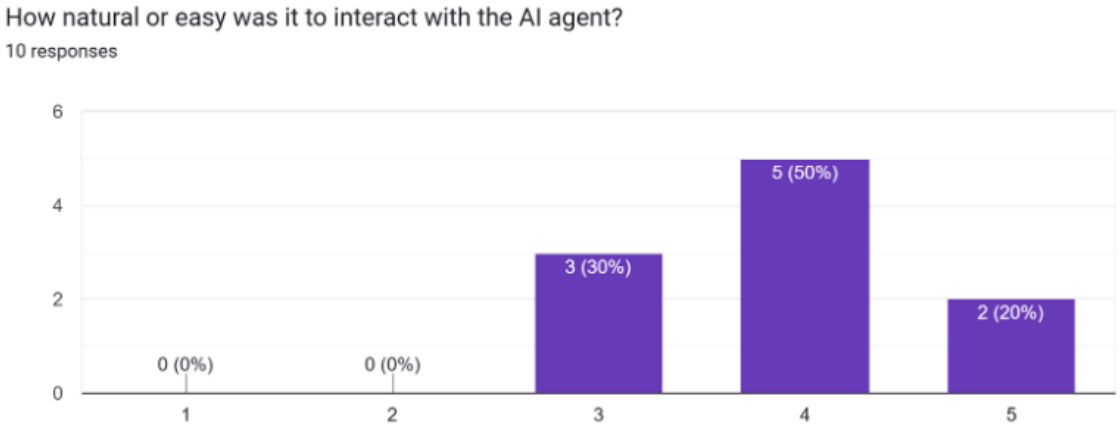

Figure 4.11: How natural was the AI.

person. Compared to [2] which had 70% of people say that their experience was personalized, we had a huge upside in the regard.

This graph in Figure 4.13 might tell us why, most said that the AI was able to understand their question and input most of the time, and a small percentage thought it was always. The reason for this might be due to wrong interpretation of the user input by the AI. Overall this result show us that the AI did a good job.

Even though the AI agent designed to encourage users to reflect before answering, we observe a split among users, some think it did a somewhat good job at making them think while the other half think it definitely made them think independently, as shown in Figure 4.14. This variation could be attributed to differences in perceived scenario difficulty. Some participants were able to pass some scenarios very easily with a lot of time to spare, while others were struggling to complete them. As a result, those who found the tasks more challenging may have been compelled to engage in deeper cognitive processing, whereas others encountered questions that appeared more straightforward.

We then asked their opinion on their preference of tutor for SBL, the results in Figure 4.15

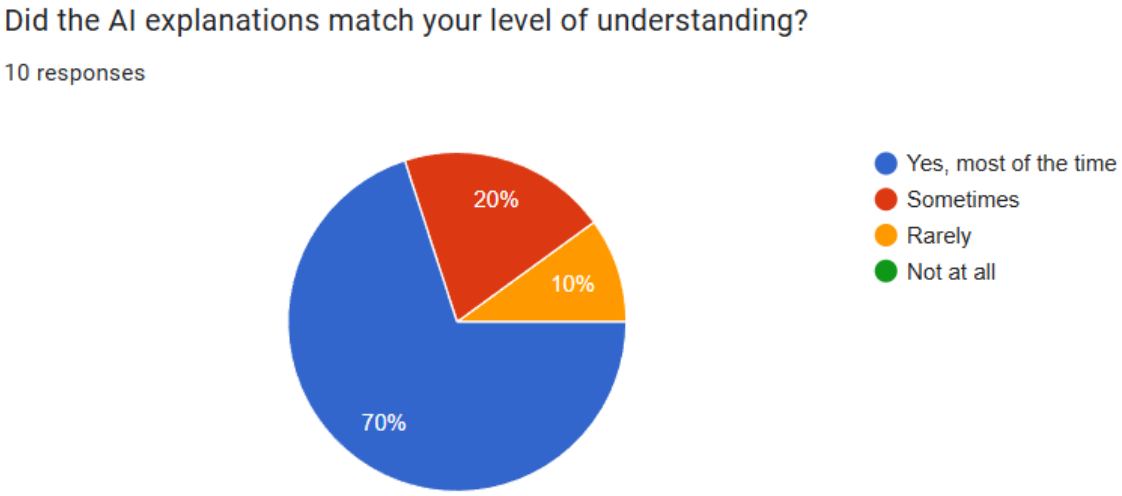

Figure 4.12: Results for how the AI match the users level of understanding.

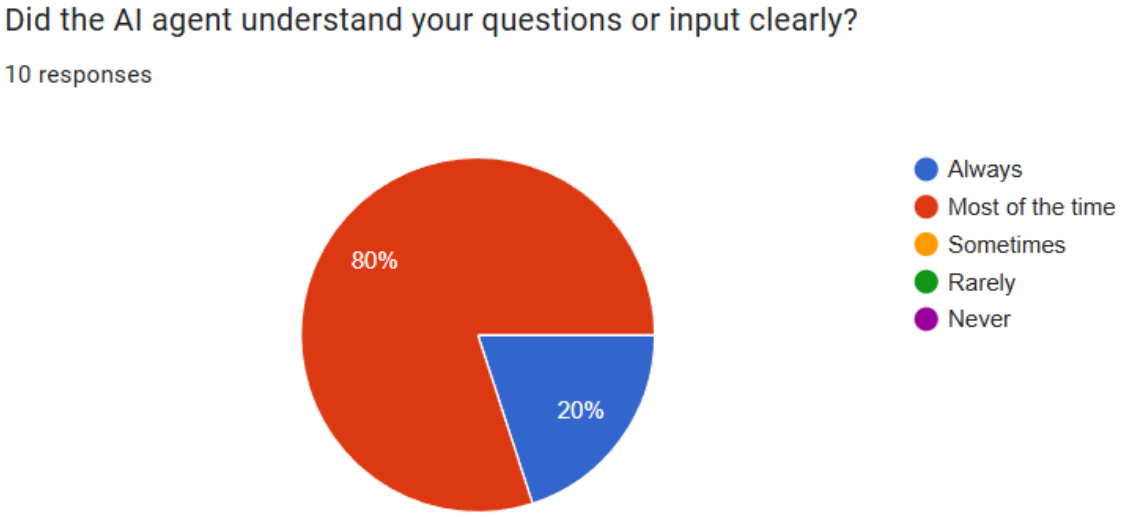

Figure 4.13: Results for the question if the AI understood user input.

show that a vast majority answered that they would prefer a mix between working with a human instructor and the AI agent. Although this result did not fully align with our expectations, it is still considered a positive outcome.

There can also be a interpretation that because we are living in a era of change and most people have not yet fully accepted AI into their lives, as per the Innovation Adoption Life-cycle. When innovation emerges, individuals adopt it at different stages: from early adopters to late adopters, and various groups in between. Although LLMs have been going for a while they are still relatively new to the masses and some people are still rejecting to use them simply because they have a preference for traditional methods over it regardless of drawbacks. In contrast, others adopted this technology immediately upon learning about it.

As all of the participants that used the tool had a conventional traditional education, they may be more likely to prefer traditional methods over newer ones.

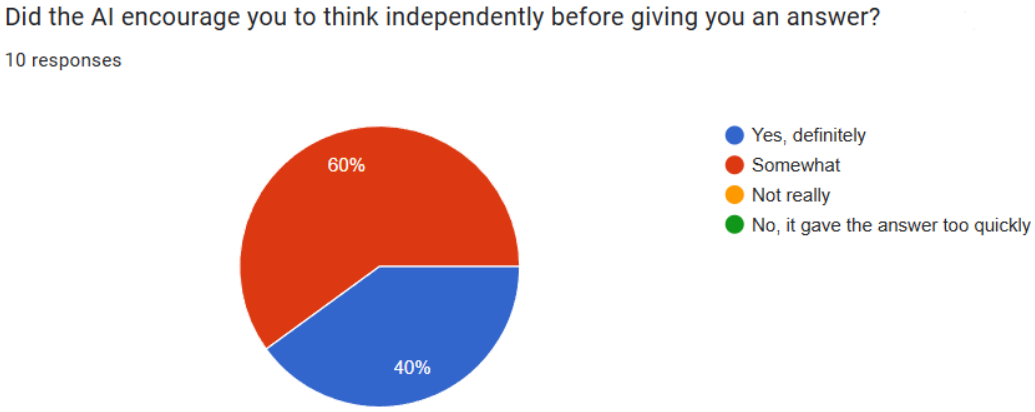

Figure 4.14: Results for if the AI made you think independently.

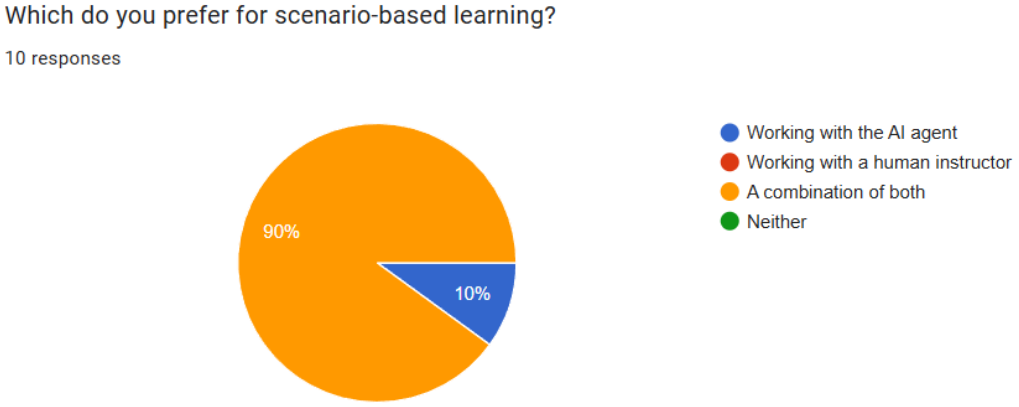

Figure 4.15: Results for the preferred teaching method.

Finally they were asked what they though were the benefits and drawbacks of using the AI agent, the results can be found in Figures 4.16 and 4.17.

In terms of benefits we can see that there is a trend for increased accessibility, responsiveness, and support. All the users considered the main advantage of having an AI agent to be its ability to remain constantly available to take questions and give real time responses while being helpful to break down complicated topics and remaining personalized for each user.

The main drawbacks were that using the AI may discourage independent thinking while being unsure if the AI's feedback is reliable, correct or if it didn't understand student questions accurately.

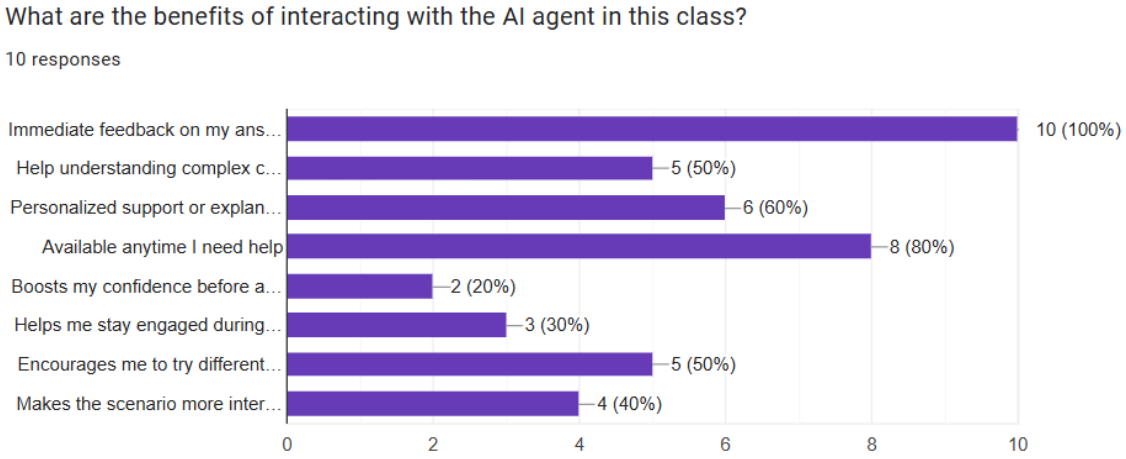

Figure 4.16: Results for the benefits of interacting with the AI agent.

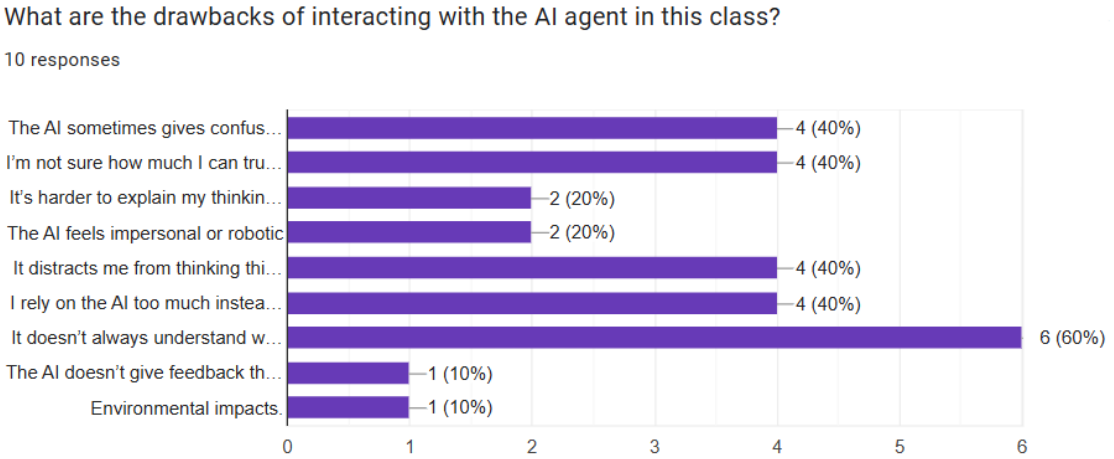

Figure 4.17: Results for the drawbacks of interacting with the AI agent.

4.3 Research questions

4.3.1 Q1:How can AI be effectively integrated into educational scenarios to provide real-time, personalized feedback?

According to section 3.4 the effective integration of AI into SBL hinges on leveraging LLMs using NLP. However, achieving real-time, personalized feedback requires a multi-layered approach that extends beyond the use of LLMs alone. The following components were essential to accomplishing this integration:

- JSON File: A well-defined JSON file encapsulates all necessary information about the scenario, including context, stages, instructional goals, and expected learner actions.
- Prompt engineering: Carefully designed prompts, grounded in instructional design principles and literature on effective feedback, guide the LLM to deliver context-aware and

pedagogically aligned responses.

- Prompt layering: Sequential and hierarchical organization of prompts ensures coherent, logical, and efficient interaction flows, enhancing both the accuracy and response time of the system.
- RAG: A retrieval system supplements the LLM by quickly sourcing relevant scenario-specific or learner-specific information, thus improving the specificity and relevance of generated feedback.
- Chat History: Continuously incorporating the users interaction history enables the system to track learner progression and tailor feedback dynamically, reinforcing personalization over time.
- UI: A user-friendly interface facilitates smooth interaction, enabling learners to engage intuitively with the AI agent during scenario resolution.

When combined, these components enable a robust, responsive, and pedagogically meaningful integration of AI into educational scenarios. The resulting system not only supports real-time feedback but also adapts to the learner's actions, fostering a more personalized and engaging learning experience.

With the results presented in the pie chart of Figure 4.6, 60% of the participants said that they prefer the AI agent over traditional methods and 70% of participants voted 4 out of 5 for the AI usefulness in the input, according to the data shown in Figure 4.8. Further validating the potential of this approach.

4.3.2 Q2:In what ways can the integration of AI in education enhance personalized learning and student engagement?

The AI system is capable of adapting its response to the individual learner, ensuring that the user receives guidance, feedback, and hints according to their performance and needs. This is backed by the user responses in 4.2.4. By the data presented in Figures 4.12 and 4.10 we can see that 70% of all users said that the AI matched their level of understanding and 20% said that it matched sometimes.

Also we can see that 70% of the participant voted 4 or more out of 5, that the AI helped them understand the scenario, again this result can help us understand that the AI agent is capable of delivering personalized insight for most types of users. This assessment lets us conclude that the AI was able to adapt to the users individual needs.

The system is also capable of immersing students in realistic scenarios, making it possible for student engagement to be greater. According to the results in Figures 4.11, 4.13, 4.9 and 4.16, we can see the opinion of the participants about the AI immersion. 70% of participants voted 4 or more out of 5 in how natural it was to interact with the AI, 100% of participants said that the AI was able to understand their questions and inputs most of the time or always, 80% voted 4 or more

out of 5 for the satisfaction related to response time and finally participants were asked to what were the main benefits of the AI agent, 30% answered that the AI helped them stay engaged during the activity, 40% said it makes the scenario more interesting and 50% said that it encourages them to try different solutions.

The results strongly indicate that the AI system effectively adapts to individual learner needs, providing tailored guidance, feedback, and hints based on user performance. As evidenced by the data in Section 4.2.4, and Figures 4.12 and 4.10, demonstrates the system's ability to deliver personalized insight to a majority of users.

In addition to personalization, the AI agent proved effective in fostering engagement through realistic and immersive scenarios. According to Figures 4.11,4.13,4.9, and 4.16.

Overall, these findings support the conclusion that the AI agent not only adapts effectively to individual learning needs but also enhances SBL by promoting immersion, engagement, and exploratory learning behavior.

4.3.3 Q3:How can the performance and effectiveness of an LLM-based educational agent be evaluated in terms of student outcomes and learning engagement?

The performance and effectiveness of the LLM-based educational agent were assessed through user interaction, completion data, feedback forms, and preference surveys, as outlined in Section 4.2.4. These metrics allowed us to evaluate both learning outcomes (scenario completion, progression, and adaptation) and learner engagement (perceptions, usability, and AI usefulness).

The completion rate and time taken per scenario offered direct insight into the users learning performance. While 26 total scenarios were completed, data showed varying levels of difficulty and success. For example, although the initial and final scenarios had low completion rates (10% and 28%, respectively), the second scenario showed a significant improvement (78% completion), likely due to users becoming more familiar with the system. This trend reflects a positive learning curve facilitated by the AI agent. The fact that the more difficult final scenario was completed only by users with advanced degrees further supports the alignment between task complexity and user expertise.

According to our data, over 60% of users preferred the AI-driven application over traditional learning methods (Figure 4.6). Despite many claiming they could complete tasks alone (Figure 4.7), actual results showed that many relied on the AI's assistance, particularly in more complex scenarios. Users also reported that the AI contributed to independent thinking (Figure 4.14) and generally matched their level of understanding (Figure 4.12).

User satisfaction and perceived usefulness were high across the board (Figures 4.8 to 4.11), with several comments highlighting areas for improvement such as response clarity. Importantly, participants favored a hybrid learning model (Figure 4.15), suggesting that while LLMs offer effective support, the human element remains essential for deeper engagement and trust in feedback.

From the participants reflections (Figures 4.16 and 4.17), key benefits of the LLM-based agent included real-time responsiveness, personalized help, and constant availability. Main drawbacks

focused on concerns about over-reliance on the AI and uncertainty regarding the accuracy of its responses—highlighting the importance of AI transparency and reliability in educational contexts.

In summary, evaluation through scenario-based testing, user perception, and interaction data revealed that the LLM-based educational agent can effectively support learning and engagement when tasks are aligned with users knowledge levels and the system is intuitive to use. However, improvements in scenario complexity balancing, response clarity, and trust-building mechanisms are necessary to further enhance both learning outcomes and student confidence.

4.3.4 Q4:Which assessment method (qualitative + quantitative) best describes the pedagogical value, usability, and efficacy of an AI-driven SBL agent in a real world scenario?

To evaluate the pedagogical value, usability, and efficacy of the AI agent in real-world conditions, a mixed-methods assessment approach was used. This combined both quantitative metrics—such as scenario completion rates, task times, preference data—and qualitative feedback—such as participant reflections and open-ended comments—enabling a more comprehensive understanding of the agent's educational impact.

The quantitative data provided objective measurements of the system's usability and learning efficacy. The key metric included was scenario completion rates and times. These were critical in determining how well participants navigated each scenario and whether the agent could effectively scaffold learning. For instance, while the initial and final scenarios had low completion rates (10% and 28%, respectively), the second scenario showed a significantly higher success rate (78%), demonstrating learning progression and usability improvement over time.

Qualitative data offered deeper insight into learner perception, engagement, and critical usability issues.

User Remarks and Open-Ended Feedback, like comments, such as "At first, it was a bit difficult to understand what to do…" and suggestions for clearer, structured responses provided valuable guidance on improving onboarding and interaction design.

Perceived Pedagogical Role responses about independent thinking (Figure 4.14) and preference for blended instruction (Figure 4.15) reflected how the AI system is currently perceived—as a strong support tool rather than a complete substitute for human instructors.

Benefits and Drawbacks such as thematic analysis of responses (Figures 4.16 and 4.17) identified accessibility, responsiveness, and personalized guidance as major strengths, while concerns about over-reliance and trust in feedback highlighted limitations that should be addressed in future iterations.

4.3.5 Q5:Which reasoning mechanisms and prompt engineering techniques can improve the agent's flexibility in response to various learning paths and SBL decision points?

Improving the agent flexibility in navigating diverse learning paths and SBL decision points requires a deliberate combination of reasoning mechanisms and prompt engineering techniques. Our experiments tested a wide range of system setups and LLMs to identify which combinations best support adaptable, high-quality responses while balancing cost and performance. The results can be found in the Tables 4.3 and 4.3.

The evaluation of different system configurations highlights key reasoning components that substantially affect output quality and flexibility:

- Next-Step Prediction: The no_next_steps setup had the lowest keyword hit rate (20) out of all setups, performing nearly 50% worse than the default_llm_setup (39 hits). This confirms that explicit forward reasoning—guiding the agent to anticipate upcoming steps—plays a crucial role in maintaining continuity and supporting branching decision points in SBL.
- Self-Refinement: Surprisingly, adding more iterations of self-refinement (more_refine) led to higher latency (12.33s) and price per message ($0.01214) without a proportional improvement in quality (36 keyword hits). Conversely, completely removing self-refinement (no_self_refine) drastically improved efficiency (3.99s, $0.00238) while retaining solid performance (32 hits). This suggests that lightweight or minimal self-reflection is optimal for maintaining responsiveness without sacrificing adaptability.
- CoT Reasoning: Removing CoT (no_CoT) resulted in lower output quality (30 hits vs. 39), showing that explicit step-by-step reasoning remains important for navigating complex SBL stages. While not the most critical feature, it supports better interpretation and path tracking.
- RAG: Removing retrieval (no_RAG) or its integration with history (no_rag_history_llm_setup) caused consistent degradation in keyword coverage and efficiency, confirming that grounding the LLM in relevant contextual knowledge is important for decision-point flexibility.

The findings also indicate that the design of the prompt flow and structural elements strongly influence the agent's capacity to adapt across learning trajectories:

- Default Prompt Setup (default_llm_setup): This baseline prompt—built from best practices in the literature—proved to be the most pedagogically effective, achieving the highest keyword coverage (39 hits), albeit with increased time and cost. Its strength lies in its structured integration of instructions, next steps, historical context, and reflection mechanisms.
- No Self Refine Setup (no_self_refine): While not the top in quality (32 hits), this setup achieved excellent cost-efficiency (67% cheaper and 48% faster than the default). It offers a pragmatic tradeoff by simplifying prompts without severely compromising adaptability, making it ideal for scalable deployment.

- Dynamic Model Switching: Results from the LLM comparison (Tables 4.5 and 4.6) show that combining models—e.g., using llama3 for high-importance stages and switching to gpt-4o or gemini for less critical moments—can optimize performance and reduce cost by up to 57%, while preserving learning path flexibility.

Chapter 5

Conclusions

In this research we successfully developed a SBL tool using primarily LLMs. The project aimed to understand both the technical performance of several LLMs in analyzing user inputs for educational tasks, and the pedagogical impact of the system from the perspective of user experience.

From the setup evaluation tests, we confirmed that the setup created with the reviewed literature achieved the highest performance scores, with the exception of cost and execution time. This also showed us how certain steps can make a big difference in score, like how adding more steps doesn't necessarily mean better results, or that one crucial step can make the performance drop immensely, like removing the next steps. It further revealed that some setups can outperform the originally created setup, not in quality but in price, time, cost per quality, and time per quality.

From the LLM evaluation tests, we observed that the llama3-70b-8192 consistently achieved top scores across almost all categories of assessment, including stage completion, keyword identification, and scenario understanding. However, its high cost made it suboptimal for an optimized deployment. Conversely, gpt-4.1-mini-2025-04-14 and gpt-4o provided highly competitive performance at significantly lower costs, with gpt-4o in particular offering exceptional cost-efficiency for keyword detection and satisfactory performance on core hits. Furthermore, gemma2 and mistral-8b also showed potential, especially in hitting core educational content markers at very low prices, making them excellent candidates for future cost-effective systems.

Overall, the ideal solution may not rely on a single model, but rather a hybrid approach, leveraging the best aspects of multiple models: using higher-performing models when quality is essential and switching to cheaper models when performance differences are marginal. Our analysis suggests that this mixed strategy could reduce operational costs by up to 57% while maintaining comparable performance.

On the human interaction side, user feedback revealed strong acceptance of the system. Most participants expressed a preference for the AI-driven system over traditional methods, despite initial usability challenges. The AI agent was perceived as helpful, adaptive to users' levels of understanding, and capable of delivering personalized support. Most importantly it was able to guide users, especially in more complex scenarios.

However, some limitations were also noted. A few users struggled with understanding how to

initially interact with the AI agent, indicating that the onboarding process requires improvement. Moreover, while the system was praised for its responsiveness and accessibility, users voiced concerns about potential over-reliance on AI and uncertainty regarding the correctness and clarity of their interpretation of its responses. Lastly, the data also highlighted that the level of scenario difficulty and the user's educational background played a key role in shaping the perceived effectiveness of the tool.

5.1 Future work

Based on the work completed, there are a lot of steps remain unaddressed in this thesis.

For the users a structured tutorial or step-by-step onboarding process would help clarify the interaction flow and how to interpret AI feedback would be helpful to reduce user error and understanding of the tool. Additionally features that explain the AI's reasoning, such as visual indicators, bullet points, or summaries of logic should be introduced to make responses easier to understand.

In terms of tests, more advanced and newer LLMs should be tested and compared to the ones used in the making of this thesis to confirm the advance of LLM knowledge and technology, furthermore different setups should be tested to confirm that newer literature is evolving. To better understand the tool's generalization and usability at scale, a wider user testing should be conducted, including students from different educational backgrounds and age groups, as well as scenarios concerning other areas of study.

One interesting idea for this system would be to have a backend system that automatically selects the optimal LLM per task based on historical performance and cost metrics. This dynamic strategy would ensure quality where needed and cost savings elsewhere. In this way, the system would ensure that tasks requiring deeper reasoning use models like llama3 or gpt-4.1, while simpler or repetitive tasks use more lightweight models like mistral or gemini.

References

[1] J. A. J and R. Rajakumari. "Harnessing AI: Enhancing English Language Teaching through Innovative Tools". In: *2024 Third International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT)*. 2024, pp. 1–7. DOI: 10.1109/ICEEICT61591.2024.10718399.

[2] Mohammad Abolnejadian, Sharareh Alipour, and Kamyar Taeb. *Leveraging ChatGPT for Adaptive Learning through Personalized Prompt-based Instruction: A CS1 Education Case Study*. Conference Paper. 2024. DOI: 10.1145/3613905.3637148. URL: https://doi.org/10.1145/3613905.3637148.

[3] Juyeon Ahn and Yoonhyuk Jung. "The common sense of dependence on smartphone: A comparison between digital natives and digital immigrants". In: *New Media & Society* 18.7 (2016), pp. 1236–1256. DOI: 10.1177/1461444814554902.

[4] Y. J. J. Alawneh et al. "Adaptive Learning Systems: Revolutionizing Higher Education through AI-Driven Curricula". In: *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*. Vol. 1. 2024, pp. 1–5. DOI: 10.1109/ICKECS61492.2024.10616675.

[5] F. AlShaikh and N. Hewahi. "AI and Machine Learning Techniques in the Development of Intelligent Tutoring System: A Review". In: *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. 2021, pp. 403–410. DOI: 10.1109/3ICT53449.2021.9582029.

[6] H. Altaleb, S. Mouti, and S. Beegom. "Enhancing College Education: An AI-driven Adaptive Learning Platform (ALP) for Customized Course Experiences". In: *2023 9th International Conference on Optimization and Applications (ICOA)*. 2023, pp. 1–5. DOI: 10.1109/ICOA58279.2023.10308834.

[7] O. Azzi et al. "Examining the role of Artificial Intelligence on Educational Enhancement: A Comprehensive Literature Review". In: *2024 Mediterranean Smart Cities Conference (MSCC)*. 2024, pp. 1–5. DOI: 10.1109/MSCC62288.2024.10697019.

[8] Lisa Bardach et al. "The power of feedback and reflection: Testing an online scenario-based learning intervention for student teachers". In: *Computers & Education* 169 (2021), p. 104194. DOI: 10.1016/j.compedu.2021.104194.

[9] H. K. M. Al-Chalabi and A. M. Ali Hussein. "Pedagogical Approaches in Adaptive E-learning Systems". In: *2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*. 2020, pp. 1–4. DOI: 10.1109/ECAI50035.2020.9223194.

[10] Yulin Chen et al. *Empowering Private Tutoring by Chaining Large Language Models*. Conference Paper. 2024. DOI: 10.1145/3627673.3679665. URL: https://doi.org/10.1145/3627673.3679665.

[11] Eric C. K. Cheng and Tianchong Wang. *Exploring Pedagogies & Strategies for Integrating Adaptive Learning Platforms: A Case Study of a High School in Hong Kong*. Conference Paper. 2023. DOI: 10.1145/3599640.3599648. URL: https://doi.org/10.1145/3599640.3599648.

[12] Jason P. Davies and Norbert Pachler. *Teaching and Learning in Higher Education: Perspectives from UCL*. UCL Institute of Education Press, University College London, 2018. URL: https://discovery.ucl.ac.uk/id/eprint/10059996/.

[13] Fabio Duarte. *Number of ChatGPT Users (Oct 2024)*. Last accessed in October 2024. 2024. URL: https://explodingtopics.com/blog/chatgpt-users#top.

[14] S. Dutta et al. "Enhancing Educational Adaptability: A Review and Analysis of AI-Driven Adaptive Learning Platforms". In: *2024 4th International Conference on Innovative Practices in Technology and Management (ICIPTM)*. 2024, pp. 1–5. DOI: 10.1109/ICIPTM59628.2024.10563448.

[15] L. Frank et al. "Leveraging GenAI for an Intelligent Tutoring System for R: A Quantitative Evaluation of Large Language Models". In: *2024 IEEE Global Engineering Education Conference (EDUCON)*. 2024, pp. 1–9. DOI: 10.1109/EDUCON60312.2024.10578933.

[16] Eduard Frankford et al. *AI-Tutoring in Software Engineering Education*. Conference Paper. 2024. DOI: 10.1145/3639474.3640061. URL: https://doi.org/10.1145/3639474.3640061.

[17] C. N. Hang, C. Wei Tan, and P. D. Yu. "MCQGen: A Large Language Model-Driven MCQ Generator for Personalized Learning". In: *IEEE Access* 12 (2024), pp. 102261–102273. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3420709.

[18] J. X. Huang, Y. Lee, and O. W. Kwon. "DIRECT: Toward Dialogue-Based Reading Comprehension Tutoring". In: *IEEE Access* 11 (2023), pp. 8978–8987. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3233224.

[19] Y. Huang et al. "The Research on Dynamical Model of Adaptive Learning System". In: *2023 5th International Conference on Computer Science and Technologies in Education (CSTE)*. 2023, pp. 56–60. DOI: 10.1109/CSTE59648.2023.00017.

[20] Funda Gezer Hursen Cigdem; Fasli. *Investigating the Efficiency of Scenario Based Learning and Reflective Learning Approaches in Teacher Education*. 2017. DOI: 10.13187/ejced.2017.2.264. URL: https://eric.ed.gov/?id=EJ1146153.

[21] C. D. Kloos et al. "How can Generative AI Support Education?" In: *2024 IEEE Global Engineering Education Conference (EDUCON)*. 2024, pp. 1–7. DOI: 10.1109/EDUCON60312.2024.10578716.

[22] X. Kong et al. "Research of Educational Robot Application Scenarios Based on Learning System Elements". In: *2023 3rd International Conference on Educational Technology (ICET)*. 2023, pp. 61–65. DOI: 10.1109/ICET59358.2023.10424140.

[23] J. C. Lawrance et al. "Developing an AI-Assisted Multilingual Adaptive Learning System for Personalized English Language Teaching". In: *2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. 2024, pp. 428–434. DOI: 10.1109/ICACCS60874.2024.10716887.

[24] Zhan Ling et al. “Deductive Verification of Chain-of-Thought Reasoning”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 36407–36433. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/72393bd47a35f5b3bee4c609e7bba733-Paper-Conference.pdf.

[25] Wenhan Lyu et al. *Evaluating the Effectiveness of LLMs in Introductory Computer Science Education: A Semester-Long Field Study*. Conference Paper. 2024. DOI: 10.1145/3657604.3662036. URL: https://doi.org/10.1145/3657604.3662036.

[26] Aman Madaan et al. “Self-Refine: Iterative Refinement with Self-Feedback”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 46534–46594. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

[27] Michael Maloni, Mark S. Hiatt, and Stacy Campbell. “Understanding the work values of Gen Z business students”. In: *The International Journal of Management Education* 17.3 (2019), p. 100320. ISSN: 1472-8117. DOI: https://doi.org/10.1016/j.ijme.2019.100320. URL: https://www.sciencedirect.com/science/article/pii/S1472811719300126.

[28] Gabrijela Perković, Antun Drobnjak, and Ivica Botički. “Hallucinations in LLMs: Understanding and Addressing Challenges”. In: *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*. 2024, pp. 2084–2088. DOI: 10.1109/MIPRO60963.2024.10569238.

[29] Cole Stryker and Eda Kavlakoglu. *What is artificial intelligence (AI)?* Last accessed in December 2024. 2024. URL: https://www.ibm.com/topics/artificial-intelligence.

[30] Gurudeo A. Tularam and Patrick Machisella. “Traditional vs Non-traditional Teaching and Learning Strategies – the case of E-learning!” In: *International Journal for Mathematics Teaching and Learning* 19.1 (2018), pp. 129–158.

[31] Sonia Vakayil et al. “RAG-Based LLM Chatbot Using Llama-2”. In: *2024 7th International Conference on Devices, Circuits and Systems (ICDCS)*. 2024, pp. 1–5. DOI: 10.1109/ICDCS59278.2024.10561020.

[32] Huan Wang and Yan-Fu Li. “Large Language Model Empowered by Domain-Specific Knowledge Base for Industrial Equipment Operation and Maintenance”. In: *2023 5th International Conference on System Reliability and Safety Engineering (SRSE)*. 2023, pp. 474–479. DOI: 10.1109/SRSE59585.2023.10336112.

Appendix A

Google forms

A.1 Google form for the candidates

1. What is your age?

    - 15–18
    - 19–24
    - 25–30
    - >30

2. What is your level of education?

    - Primary school
    - Secondary school
    - High school
    - Undergraduate
    - Graduate
    - Doctoral

3. Do you prefer using the AI agent over traditional methods?

    - Yes
    - No

4. Would you be able to complete the scenario without the help of the AI agent?

    - Yes
    - No

5. From 1 to 5, how useful was the input of the AI agent?
    - 1 – Not helpful at all
    - 2
    - 3
    - 4
    - 5 – Extremely helpful
6. On a scale from 1 to 5, how satisfied were you with the response time of the AI agent?
    - 1 – Very dissatisfied
    - 2
    - 3
    - 4
    - 5 – Very satisfied
7. How much did the AI agent help you understand the scenario content?
    - 1 – Not at all
    - 2
    - 3
    - 4
    - 5 – Completely
8. How natural or easy was it to interact with the AI agent?
    - 1 – Very difficult
    - 2
    - 3
    - 4
    - 5 – Very natural
9. Did the AI explanations match your level of understanding?
    - Yes, most of the time
    - Sometimes
    - Rarely
    - Not at all
10. Did the AI encourage you to think independently before giving you an answer?

- Yes, definitely
- Somewhat
- Not really
- No, it gave the answer too quickly

11. Did the AI agent understand your questions or input clearly?

    - Always
    - Most of the time
    - Sometimes
    - Rarely
    - Never

12. Which do you prefer for scenario-based learning?

    - Working with the AI agent
    - Working with a human instructor
    - A combination of both
    - Neither

13. What are the benefits of interacting with the AI agent in this class? (Select all that apply)

    - Immediate feedback on my answers or ideas
    - Help understanding complex concepts
    - Personalized support or explanations
    - Available anytime I need help
    - Boosts my confidence before asking a question
    - Helps me stay engaged during the activity
    - Encourages me to try different solutions
    - Makes the scenario more interactive and fun
    - Other: _____

14. What are the drawbacks of interacting with the AI agent in this class? (Select all that apply)

    - The AI sometimes gives confusing or incorrect answers
    - I'm not sure how much I can trust the AI's feedback
    - It's harder to explain my thinking to an AI than to a teacher
    - The AI feels impersonal or robotic
    - It distracts me from thinking things through on my own

- I rely on the AI too much instead of solving problems myself
- It doesn't always understand what I'm asking
- The AI doesn't give feedback that matches my learning style
- Other: _____

15. Write your criticisms and suggestions for improving scenario completion:
    _____

Appendix B

Scenarios

B.1 Daily Calorie Tracker

```
{
    "ai_role": "Virtual Mentor",
    "user_role": "Junior Developer",
    "scenario_name": "Daily Calorie Tracker",
    "ai_persona": "Patient Programming Instructor with a focus on beginners",
    "place": "A simple coding workshop where you're learning basic programming",
    "task": "Create a program that takes calorie input from the user for each meal
        and summarizes daily intake with a health message.",
    "format": "Step-by-step text-based scenario where you build a basic loop-driven
         input program using conditionals and arithmetic.",
    "exemplar": "User: 'Ill use a loop to gather meal data'  AI: 'Great thinking!
        That keeps your code efficient and avoids repetition.'",
    "stage_description": "You're building a calorie tracker. It should ask how many
         meals were eaten, take calorie inputs, then give feedback based on total
        calories. What's the first part you'll implement?\n\nFor this scenario you
        don't need to write code, you can just state what you would do to complete
        the task.",
    "hint": "Start with asking the user how many meals they had.",
    "positive_feedback": "Nice! Asking for the number of meals gives structure to
        the loop.",
    "constructive_feedback": "Think about what you need to loop through before
        collecting input repeatedly.",
    "next_stage_condition": "Proceed once the user decides to prompt for meal count
         and plans to loop through input.",
    "all_optional": [
        "What do I need to do: Ask how many meals were eaten, take calorie inputs
            and then give feedback based on total calories."
    ],
    "stages": [
        {"stage_step": [
            [
```

```
                "What does the code look like: No code has been provided yet."
            ],
            {
                "description": "Ask the user how many meals they had today and
                    store that number in a variable.",
                "hint": "How do you get user input in Python?",
                "correct_response": "Prompt for the number of meals and save the
                    input as an integer."
            },
            {
                "description": "Initialize the total variable.",
                "hint": "How do store the total value of calories?",
                "correct_response": "Initialize a total variable to store the sum
                    of calories."
            }
        ]},
        {"stage_step": [
            [
                "What does the code look like:\nnumberOfMeals = input()\
                    ntotalCalories = 0"
            ],
            {
                "description": "Now loop through each meal in a loop.",
                "hint": "How do you get all the calorie inputs in a single run?",
                "correct_response": "Use a for loop to iterate through the number
                    of meals or use for i in range(number_of_meals)."
            }
        ]},
        {"stage_step": [
            [
                "What does the code look like:\nnumberOfMeals = input()\
                    ntotalCalories = 0\nfor i in range(numberOfMeals):"
            ],
            {
                "description": "Inside the loop, ask for the calories of each meal.
                    ",
                "hint": "How do you collect input for each meal's calories?",
                "correct_response": "Use input() to get calorie data for each meal.
                    "
            },
            {
                "description": "Sum the calories.",
                "hint": "How do you add the total calories?",
                "correct_response": "Add each meal's calories to the totalCalories
                    variable."
            }
        ]},
        {"stage_step": [
            [
```

```
                "What does the code look like:\nnumberOfMeals = input()\
                    ntotalCalories = 0\nfor i in range(numberOfMeals):\n
                    mealCalories = input()\n    totalCalories += mealCalories"
            ],
            {
                "description": "Output the total calories and give a message based
                    on the total.",
                "hint": "Now that you have the total, how will you provide the
                    feedback?",
                "correct_response": "Display total and use if/elif/else to give
                    personalized feedback."
            }
        ]},
        {"stage_step": [
            [
                "What does the code look like:\nnumberOfMeals = input()\
                    ntotalCalories = 0\nfor i in range(numberOfMeals):\n
                    mealCalories = input()\n    totalCalories += mealCalories\nif
                    totalCalories < 2000:\n    print('Great job! You stayed under
                    the recommended daily calories.')\nelif totalCalories < 2500:\n
                        print('Good job! You are close to the recommended daily
                    calories.')\nelse:\n    print('You might want to watch your
                    calorie intake.')"
            ],
            {
                "description": "Make user input robust.",
                "hint": "Everything is done, now how do you prevent from the user
                    writing invalid input like '-2'?",
                "correct_response": "Use a loop with a if inside that prevents
                    input from being less than 0."
            }
        ]}
    ],
    "tones": [
        "Friendly and supportive, reinforcing foundational skills.",
        "Clear and encouraging, building confidence with simple logic."
    ]
}
```

B.2 Debugging a Student Grading System

```
{
    "ai_role": "Virtual Mentor",
    "user_role": "Junior Developer",
    "scenario_name": "Debugging a Student Grading System",
```

```
    "ai_persona": "Experienced Software Engineer with a focus on educational tools and data integrity",
    "place": "A university IT department where you're assisting in maintaining internal software systems",
    "task": "Fix a Python script that calculates final student grades from assignment and test scores. Several students are receiving incorrect or missing grades.",
    "format": "Step-by-step text-based scenario where you diagnose and correct logical and structural issues in a data processing script.",
    "exemplar": "User: 'I will check the return statement in the grade calculation function.'  AI: 'Good idea! An incorrect return might explain the missing grades. What do you find?'",

    "stage_description": "Your supervisor reports that students are receiving inaccurate final grades, some students are getting grades of 0 or 'None' while others are getting scores that are not adding up correctly. The grading system was quickly written by an intern and now must be debugged. You are given access to the script and a sample dataset. Your job is to identify the bugs, correct them, and ensure the output is reliable. What is your first step?",
    "hint": "Start by analyzing how grades are calculated and look for signs of faulty logic or improper data handling.",
    "positive_feedback": "Great thinking! Starting with the grade calculation helps identify whether the problem is logic-based or data-related.",
    "constructive_feedback": "Try to narrow down whether the issue is with the data being read correctly or the way it's processed in the functions.",
    "next_stage_condition": "Proceed once the user identifies a suspicious calculation or return statement that might result in incorrect grades.",
    "all_optional": [
        "What does the code look like: import csv\n\ndef calculate_final_grade(assignments, tests):\n    if not assignments or not tests:\n        return None\n    assignment_avg = sum(assignments) / len(assignments)\n    test_avg = sum(tests) / len(tests)\n    final_grade = assignment_avg * 0.3 + test_avg * 0.8\n    return final_grade\n\ndef read_student_data(file_path):\n    with open(file_path, 'r') as f:\n        reader = csv.DictReader(f)\n        results = {}\n        for row in reader:\n            name = row['name']\n            try:\n                assignments = [int(row[f'assignment{i}']) for i in range(1, 4)]\n                tests = [int(row[f'test{i}']) for i in range(1, 3)]\n                grade = calculate_final_grade(assignments, tests)\n                results[name] = grade\n            except Exception as e:\n                print(f\"Error processing {name}: {e}\")\n                continue\n        return results\n\ngrades = read_student_data('student_grades.csv')\nfor student, grade in grades.items():\n    print(f\"{student}: {grade}\")",
        "What does the CSV look like: name,assignment1,assignment2,assignment3,test1,test2\nAlice,85,90,88,92,87\nBob,75,80,70,78,82\nCharlie,90,,95,88,"
    ],
```

```
20
21    "stages": [
22        {
23            "stage_step": [
24                [
25                    "What's the error: Error processing Charlie: invalid literal
                          for int() with base 10: ''\nAlice: 97.9\nBob: 86.5"
26                ],
27                {
28                    "description": "An error occurs when processing a student with
                          missing data, and their grade is skipped.",
29                    "hint": "Check whether the script is validating input before
                          converting it to integers.",
30                    "correct_response": "Check if the value is empty or invalid in
                          the 'assignments' and 'tests' variables before converting
                          it to an integer."
31                }
32            ]
33        },
34        {
35            "stage_step": [
36                [
37                    "whats the error: final_grade is not correct, Alice is getting
                          97.9 but it should be 88.95\nBob is getting 86.5 but it
                          should be 78.5\nCharlie is getting 53.7 but it should be
                          49.3"
38                ],
39                {
40                    "description": "The grade calculations are consistently too
                          high compared to expected values.",
41                    "hint": "Are the weightings for assignments and tests adding up
                           correctly?",
42                    "correct_response": "Adjust the weights in
                          calculate_final_grade so that they sum to 1."
43                }
44            ]
45        },
46        {
47            "stage_step": [
48                [
49                    "What's the issue: Some students are still being skipped, and
                          no grade is displayed even though their data seems mostly
                          complete."
50                ],
51                {
52                    "description": "Some students still have missing grades, even
                          though they have partial data available.",
53                    "hint": "How should the system handle missing or incomplete
                          values?",
```

```
                    "correct_response": "Provide a fallback value (e.g., 0) for
                        missing or invalid data instead of skipping the student
                        entirely."
                }
            ]
        }


    ],

    "tones": [
        "Professional and supportive, providing logical guidance without giving
            away direct answers.",
        "Clear and engaging, encouraging problem-solving through thoughtful
            questioning."
    ]
}
```

B.3 Optimize the Ride-Sharing Algorithm

```
{
    "ai_role": "Virtual Mentor",
    "user_role": "Junior Developer",
    "scenario_name": "Optimize the Ride-Sharing Algorithm",
    "ai_persona": "Systems Architect with deep knowledge in algorithms and
        performance tuning",
    "place": "A tech startup developing a ride-sharing backend during a traffic-
        heavy event",
    "task": "Analyze and improve an inefficient driver-rider matching algorithm to
        reduce assignment time and system lag.",
    "format": "Step-by-step optimization scenario involving analysis, algorithm
        design, and implementation tuning.",
    "exemplar": "User: 'The current method checks every driver for each rider.'  AI
        : 'Yes, thats an O(n^2) approach. Can we improve the lookup time?'",
    "stage_description": "You're assigned to optimize and test the matching system
        for peak-hour ride requests. The current solution loops over all drivers
        for every rider.\n\n What would be the time complexity of this approach,
        why is it inefficient during high traffic, and how can you improve it?",
    "hint": "Think about data structures that make spatial or distance queries more
         efficient.",
    "positive_feedback": "Exactly! Using better data structures can drastically
        reduce lookup time.",
    "constructive_feedback": "Looping through every option is costly. Is there a
        way to index drivers by proximity?",
    "next_stage_condition": "Proceed if the user suggests a spatial or priority-
        based structure like heaps or maps.",
```

```
    "all_optional": [
        "what does the code look like: import time\nimport random\ndrivers = [{'id': i, 'location': (random.randint(0, 100), random.randint(0, 100))} for i in range(1000)]\nriders = [{'id': j, 'location': (random.randint(0, 100), random.randint(0, 100))} for j in range(1000)]\ndef distance(a, b): return ((a[0] - b[0])**2 + (a[1] - b[1])**2)**0.5\ndef match_riders_to_drivers(riders, drivers): matches = []; used_drivers = set();\n for rider in riders:\n  closest_driver = None; min_dist = float('inf')\n  for driver in drivers:\n   if driver['id'] in used_drivers: continue\n   dist = distance(rider['location'], driver['location'])\n   if dist < min_dist: min_dist = dist; closest_driver = driver\n  if closest_driver: matches.append((rider['id'], closest_driver['id'])); used_drivers.add(closest_driver['id'])\n  else: matches.append((rider['id'], None))\n return matches\nstart = time.time()\nmatches = match_riders_to_drivers(riders, drivers)\nend = time.time()\nprint(f\"Matched {len(matches)} riders in {end - start:.2f} seconds.\")"
    ],
    "stages": [
        {
            "stage_step":
            [
                [],
                {
                    "description": "Review the current matching method: it loops through every driver for each rider. What is the time complexity?",
                    "hint": "How many operations if there are n drivers and m riders?",
                    "correct_response": "Recognize that it's an O(n * m) or O(n^2) algorithm if n similar to m."
                }
            ]
        },
        {
            "stage_step":
            [
                [],
                {
                    "description": "The current code loops through all drivers for every rider. Propose a more efficient structure.",
                    "hint": "Is there a way to not loop through all drives for every rider?",
                    "correct_response": "Suggest using spatial hashing, a k-d tree, or a heap for nearby driver selection."
                },
                {
                    "description": "Why is this inefficient during high traffic times?",
```

```
                    "hint": "Think in terms of scale: what happens when thousands
                        of requests come in?",
                    "correct_response": "Acknowledge the performance bottleneck due
                         to nested loops and scaling issues."
                }
            ]
        },
        {
            "stage_step":
            [
                [],
                {
                    "description": "Now that you've optimized the structure, check
                        how this affects performance under 1000 concurrent requests
                        .",
                    "hint": "How can you measure the performance impact of your
                        changes?",
                    "correct_response": "Compare execution time and confirm speedup
                        ."
                }
            ]
        }
    ],
    "tones": [
        "Analytical and strategic, encouraging performance-oriented thinking.",
        "Inquisitive and challenging, promoting deep understanding of optimization
            strategies."
    ]
}
```

B.4 Code Stroke - Acute Treatment

```
{
    "ai_role": "Neurologist",
    "user_role": "Medical Trainee",
    "scenario_name": "Code Stroke - Acute Treatment",
    "ai_persona": "Supportive and clinically rigorous neurologist guiding emergency
         stroke care decisions.",
    "place": "Emergency Room",
    "task": "Manage an acute ischemic stroke patient using available diagnostics
        and treatment options.",
    "format": "Interactive scenario-based learning with sequential decision-making.
        ",
    "exemplar": "Ask for vital signs, perform CT scan, assess eligibility for
        thrombolysis, and decide on thrombectomy.",
```

```
    "stage_description": "A 70-year-old man presents in the emergency room after being found by his spouse unable to speak and unable to move his right arm. At admission he had right hemianopsia, his right arm was paretic, his leg has diminished strength against resistance, he was unable to speak or follow commands.\n\nAs a Neurologist, how would you approach such a situation? You can ask for clinical information, such as asking for exams and give medication to the patient. E.g. What are the patient vital signs?",
    "hint": "Start with basic assessment and stabilize vital signs before proceeding to advanced imaging or treatments.",
    "positive_feedback": "Excellent decision! You prioritized time-sensitive interventions and considered stroke guidelines.",
    "constructive_feedback": "Consider the timing of symptom onset and contraindications before administering thrombolysis.",
    "next_stage_condition": "Once the patient is stabilized and all imaging/lab results are reviewed, proceed to definitive therapy.",
    "all_optional": [
      "Ask for modified Rankin scale (mRankin): In terms of autonomy, the patient has a modified Rankin score of 0.",
      "Make an ECG: His ECG shows atrial fibrillation.",
      "Measure gasometry: gasometry / arterial blood gas (ABG) was unremarkable.",
      "Ask for NIHSS: NIHSS score is 7.",
      "Ask for allergies: The patient has has no know allergies",
      "Ask for previous surgeries or trauma: he had no recent major trauma, no recent traumatic brain injury, no recent major surgery, no recent major bleeding, no recent gastrointestinal bleeding.  The patient has no other medical conditions or relevant antecedents.",
      "Ask for previous stroke, myocardial infarction or vascular events: no recent stroke, no recent myocardial infarction.",
      "Ask for troponine: troponine is negative."
    ],
    "stages": [
        {
            "stage_step": [
                [
                    "Ask for blood pressure: 210/105 mmHg",
                    "Ask for heart rate: 87 bpm",
                    "Ask for respiratory rate: 16 breaths/min",
                    "Ask for patient temperature: 36.6C",
                    "Ask for oxygen saturation: 99% on room air.",
                    "ECG monitoring: the patient has a regular cardiac rhythm with 85bpm.",
                    "Measure glycemia: at presentation 105mg/dL.",
                    "Ask for blood samples: Take blood samples and send for analysis.",
                    "Ask for patient medication: The patient is not taking any medication, including anticoagulants.",
                    "When was the patient last seen well: The patient was last seen well 3 hours before admission.",
```

```
                    "Ask for medical history: The patient has no known
                        cardiovascular risk factors including hypertension,
                        diabetes, dyslipidemia, cigarette smoking or alcohol
                        consumption.",
                    "Ask for weight: the weight is 80Kg."
                ],
                {
                  "description": "Check the patient's blood pressure to assess
                      cardiovascular status.",
                  "hint": "Consider a vital sign that indicates circulatory
                      function.",
                  "correct_response": "Ask for blood pressure"
                },
                {
                  "description": "Assess the patient's heart rate to determine
                      pulse strength and rhythm.",
                  "hint": "Think about how you would evaluate the pulse.",
                  "correct_response": "Ask for heart rate"
                },
                {
                  "description": "Evaluate the patient's respiratory rate to check
                      for breathing abnormalities.",
                  "hint": "Focus on how frequently the patient breathes.",
                  "correct_response": "Ask for respiratory rate"
                },
                {
                  "description": "Take the patient's body temperature to check for
                      fever or hypothermia.",
                  "hint": "This measure can indicate infection or thermal
                      regulation issues.",
                  "correct_response": "Ask for patient temperature"
                },
                {
                  "description": "Determine the oxygen saturation to evaluate the
                      patient's respiratory efficiency.",
                  "hint": "Think about how to assess oxygen levels non-invasively."
                      ,
                  "correct_response": "Ask for oxygen saturation"
                },
                {
                  "description": "Monitor the patient's cardiac activity with an
                      ECG to detect arrhythmias or ischemia.",
                  "hint": "Use a tool that gives a visual representation of the
                      heart's electrical activity.",
                  "correct_response": "ECG monitoring"
                },
                {
                  "description": "Measure blood glucose levels to assess for
                      hypoglycemia or hyperglycemia.",
```

```
                "hint": "This involves checking a key metabolic parameter with a drop of blood.",
                "correct_response": "Measure glycemia"
            }
        ]
    },
    {
        "stage_step": [
            [
                "Ask for a CT scan: non-contrast CT scan should be always performed before angiography to exclude haemorrhage and stroke mimics. Non-contrast CT scan showed no signs of acute ischemic stroke or haemorrhage.",
                "Ask for blood samples: Take blood samples and send for analysis.",
                "Ask for patient medication: The patient is not taking any medication, including anticoagulants.",
                "When was the patient last seen well: The patient was last seen well 3 hours before admission.",
                "Ask for medical history: The patient has no known cardiovascular risk factors including hypertension, diabetes, dyslipidemia, cigarette smoking or alcohol consumption.",
                "Ask for weight: the weight is 80Kg."
            ],
            {
                "description": "Order an imaging study to rule out hemorrhage or stroke mimics before proceeding with angiography.",
                "hint": "Think about the first-line brain imaging often done without contrast in suspected stroke cases.",
                "correct_response": "Ask for a CT scan"
            }
        ]
    },
    {
        "stage_step": [
            [
                "When was the patient last seen well: The patient was last seen well 3 hours before admission.",
                "Lower blood pressure: Lower blood pressure to a target of less or equal to 185/110mmHg. The drugs labetalol, nitrates, urapidil can be used.",
                "Ask for blood samples: Take blood samples and send for analysis.",
                "Ask for medical history: The patient has no known cardiovascular risk factors including hypertension, diabetes, dyslipidemia, cigarette smoking or alcohol consumption.",
```

```
                    "Ask for patient medication: The patient is not taking any medication, including anticoagulants.",
                    "Ask for a CT angiogram: showed a proximal occlusion of the M1 segment of left medial cerebral artery.",
                    "Ask for weight: the weight is 80Kg."
                ],
                {
                  "description": "Determine the time the patient was last in their normal state before the current condition began.",
                  "hint": "This helps establish the timeline for symptom onset.",
                  "correct_response": "When was the patient last seen well"
                },
                {
                  "description": "Initiate measures to reduce elevated blood pressure to a specific target to enable safe treatment.",
                  "hint": "Focus on managing hypertension to meet clinical thresholds.",
                  "correct_response": "Lower blood pressure"
                },
                {
                  "description": "Collect laboratory samples to support diagnostic and treatment decisions.",
                  "hint": "These are often taken using a syringe or vacutainer.",
                  "correct_response": "Ask for blood samples"
                },
                {
                  "description": "Gather information about the patient's prior health conditions or risk factors.",
                  "hint": "Useful for understanding predispositions or chronic issues.",
                  "correct_response": "Ask for medical history"
                },
                {
                  "description": "Check if the patient is currently taking any prescribed or over-the-counter medications.",
                  "hint": "This includes regular drugs or recent treatments.",
                  "correct_response": "Ask for patient medication"
                }
            ]
        },
        {
            "stage_step": [
                [
                    "Perform thrombolysis: Thrombolysis should be done up to 4.5h after the patient was last seen well. The recommended dose of alteplase is 0.9mg/Kg, for a maximum total dose of 90mg. The IV bolus dose is 10% of the 0.9 mg/kg treatment dose over 1 minute, followed by an intravenous infusion of the remaining 90% over 60min. The recommended dose of
```

```
                        tenecteplase is 0.25mg/Kg given as a single bolus (maximum
                        dose of 25mg).",
138                     "Ask for a CT angiogram: showed a proximal occlusion of the M1
                        segment of left medial cerebral artery."
139                 ],
140                 {
141                     "description": "Initiate thrombolytic therapy within the
                        therapeutic window for ischemic stroke using alteplase or
                        tenecteplase, following dosage protocols based on patient
                        weight.",
142                     "hint": "Consider time since symptom onset and weight-based
                        medication dosing for clot dissolution.",
143                     "correct_response": "Perform thrombolysis"
144                   }
145             ]
146         },
147         {
148             "stage_step": [
149                 [
150                     "Check blood analysis: Blood analysis were available 1h after
                        admission and were unremarkable, coagulation was normal and
                         platelets were 250.000.",
151                     "Ask for a CT angiogram: showed a proximal occlusion of the M1
                        segment of left medial cerebral artery."
152                 ],
153                 {
154                   "description": "Review the patient's blood analysis to check for
                       any abnormalities or signs of coagulopathy.",
155                   "hint": "Look into lab results that reflect hematological and
                       coagulation status.",
156                   "correct_response": "Check blood analysis"
157                 },
158                 {
159                   "description": "Order imaging to evaluate for vascular occlusion
                       in the brain.",
160                   "hint": "Consider advanced imaging that visualizes cerebral blood
                        vessels.",
161                   "correct_response": "Ask for a CT angiogram"
162                 }
163             ]
164         },
165         {
166             "stage_step": [
167                 [
168                     "Perform thrombectomy: Contact neuroradiology to perform
                        thrombectomy."
169                 ],
170                 {
```

```
                    "description": "Initiate appropriate intervention for a patient
                         with an ischemic stroke who is a candidate for clot
                        retrieval.",
                    "hint": "Think about involving a specialist to remove a
                        cerebral clot.",
                    "correct_response": "Perform thrombectomy"
                }
            ]
        }
    ],
    "tones": [
        "Clinical",
        "Encouraging"
    ]
}
```

Appendix C

Gantt Chart

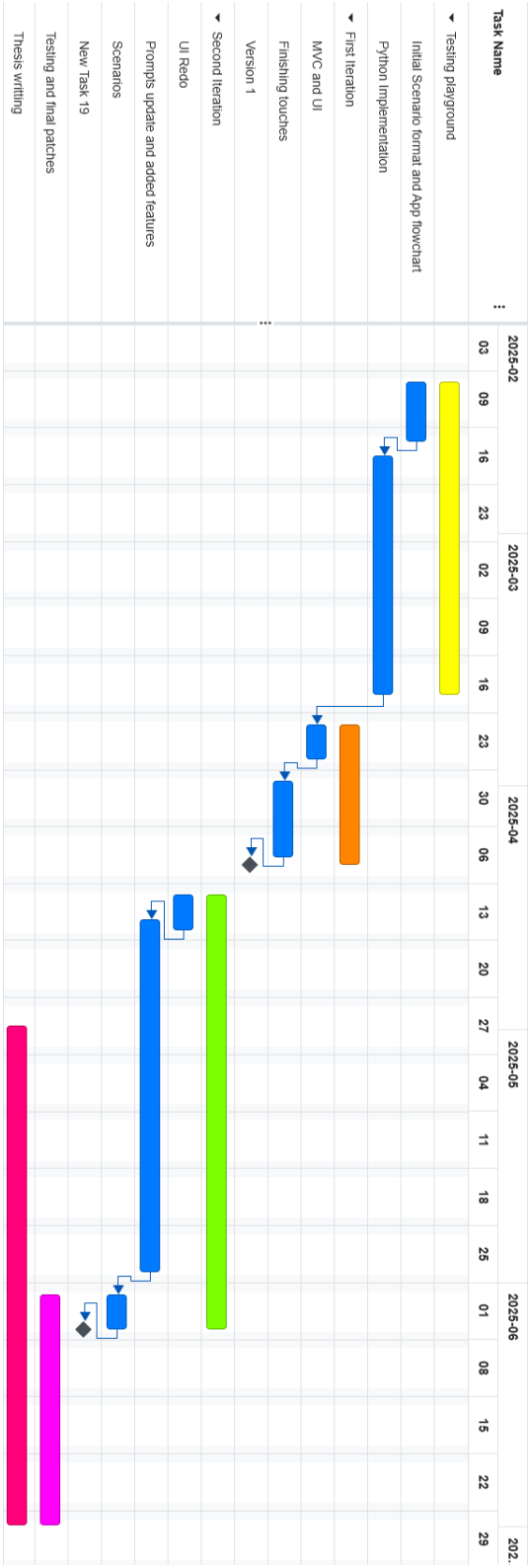

Figure C.1: Detailed Gantt chart showcasing the timeline of the project.

Appendix D

Times and Tokens

Table D.1: CoT timings and tokens

Run	Time (s)	Prompt Tokens	Total Tokens
1	0.1817	1141	1185
2	0.1650	1141	1183
3	0.2063	1141	1195
4	0.3110	1141	1199
5	0.2257	1141	1203

Table D.2: Self Consistency timings and tokens

Run	Time (s)	Prompt Tokens	Total Tokens
1	0.9068	1210	1464
2	0.7732	1210	1376
3	0.6923	1210	1417
4	0.7075	1210	1375
5	0.8410	1210	1408

Table D.3: Feedback timings and tokens

Run	Time (s)	Prompt Tokens	Total Tokens
1	0.9463	2356	2604
2	0.7625	2356	2542
3	0.7575	2356	2592
4	0.8888	2356	2553
5	0.7415	2356	2531

Table D.4: Refine timings and tokens

Run	Time (s)	Prompt Tokens	Total Tokens
1	0.3661	1641	1698
2	0.2728	1641	1696
3	0.3107	1641	1706
4	0.2742	1641	1690
5	0.3002	1641	1696

Table D.5: Next Steps timings and tokens

Run	Time (s)	Prompt Tokens	Total Tokens
1	0.4314	1312	1386
2	0.3789	1312	1371
3	0.2945	1312	1369
4	0.2943	1312	1364
5	0.3711	1312	1384

Redirect User #	**Time Taken (s)**	**Prompt Tokens**	**Completion Tokens**	**Total Tokens**
1	0.482464372	1306	78	1384
2	0.382513723	1306	68	1374
3	0.580152073	1306	103	1409
4	0.495865551	1306	93	1399
5	0.522907901	1306	90	1396

Table D.6: Redirect User Request Statistics

Feedback #	**Time Taken (s)**	**Prompt Tokens**	**Total Tokens**	
1	0.776161245	2443	205	2648
2	0.837258054	2443	201	2644
3	0.817106094	2443	183	2626
4	0.713589501	2443	195	2638
5	0.600417503	2443	162	2605

Table D.7: Feedback with RAG Request Statistics

Appendix E

Test Parameters

E.1 Message Stack for Setup tests part1

```
messages = [
    ["What is the circumference of the Earth?", ["focus", "how", "many", "meals", "today", "back", "track"], [[False, False], [False], [False, False], [False], [False]]],
    ["I ask the user the amount of meals eaten using input()", ["int(input())", "user", "input", "meals", "count", "input()", "variable", "assign", "total", "calories"], [[True, False], [False], [False, False], [False], [False]]],
    ["I use a integer to store the value of the user input and use another integer to store the value of the total calories", ["loop", "total", "calories", "num\_meals", "number", "meals"], [[True, True], [False], [False, False], [False], [False]]],
    ["I don't know what loops are can you explain them to me?", ["loop", "for", "while", "range", "num\_meals", "iterate", "control", "flow"], [[True, True], [False], [False, False], [False], [False]]],
    ["I would use a for loop with the number of meals", ["for", "loop", "iterate", "range", "total", "variable", "calories", "meal", "input"], [[True, True], [True], [False, False], [False], [False]]],
    ["I ask the user for input of the calories", ["total", "sum", "calories", "input", "+="], [[True, True], [True], [True, False], [False], [False]]],
    ["I add up all of the calories into a total calories variable", ["total", "+=", "accumulate", "calories", "loop", "sum"], [[True, True], [True], [True, True], [False], [False]]],
    ["I can give a message based off the amount of calories, like if the amount is over 2000 i will print something like \"to many calories\"", ["if", "conditional", "total", "calories", "feedback", "print", "message"], [[True, True], [True], [True, True], [True], [False]]],
    ["I would create a for loop inside the input and do a if that doesn't let values bellow 0", ["validation", "if", "loop", "input", "negative", "bigger", ">=", "0", "check", "while", "continue"], [[True, True], [True], [True, True], [True], [True]]]
]
```

E.2 Message Stack for Setup tests part2

```
messages = [
    ["What is the circumference of the Earth?", [True, False]],
    ["What is the circumference of the Earth", [True, False]],
    ["What happens if I ask the user eats 8000 calories", [True, False]],
    ["I ask the user the amount of meals eaten using input()", [False, True]],
    ["I use a integer to store the value of the user input and use another integer
        to store the value of the total calories", [False, True]],
    ["I don't know what loops are can you explain them to me?", [True, False]],
    ["I don't know what variables are can you explain them to me?", [True, True]],
    ["I would use a for loop with the number of meals", [False, False]],
    ["I ask the user for input of the calories", [False, True]],
    ["I add up all of the calories into a total calories variable", [False, False
        ]],
    ["I can give a message based of the amount of calories, like if the amount is
        over 2000 i will print something like \"to many calories\"", [False, False
        ]],
    ["Can I use a variable for everything", [True, True]],
    ["Should I use the variables for the message at the end", [True, False]],
    ["What does the code look like?", [True, False]],
    ["I would create a for loop inside the input and do a if that doesn't let
        values bellow 0", [False, False]],#
    ["I ask the user how many meals they had today using input()", [False, True]],
    ["I used `input()` to get how many meals the user had and stored it in a
        variable called `num\_meals`", [False, True]],
    ["How do I get user input in Python?", [True, True]],
    ["How do I store the total calories?", [True, True]],
    ["I initialized a variable called `total\_calories` to 0", [False, True]],
    ["Is it okay if I use `total = 0` to start counting the calories?", [True, True
        ]],
    ["Whats the best way to store how many meals someone had?", [True, True]],
    ["I assigned the users input to a variable using `int(input())`", [False, True
        ]],
    ["I think I should use a float for storing the number of meals", [True, True]],
    ["Should the total variable be inside the loop or outside?", [True, False]],
    ["Is this the right way? `meals = input('How many meals?')`", [True, True]],
    ["I declared `total = 0` before the loop", [False, True]],
    ["Why can't I just add all inputs directly without using a total variable?", [
        True, True]],
    ["I think I should ask the user about their meals and create a sum of calories"
        , [False, True]],
    ["I dont know how to get user input", [True, True]]
]
```

```
messages1 = [
    ["Make an ECG", [True, ["atrial", "fibrillation"]]],
    ["Make an electrocardiogram?", [True, ["atrial", "fibrillation"]]],
    ["Ask for modified Rankin scale", [True, ["autonomy", "modified", "Rankin", "
        score","0"]]],
    ["Should I use the Rankin scale?", [False, [""]]],
    ["Measure gasometry", [True, ["gasometry", "arterial", "blood", "gas", "
        unremarkable"]]],
    ["I don't know what gasometry is?", [False, [""]]],
    ["Ask for NIHSS", [True, ["NIHSS", "score", "7"]]],
    ["What does NIHSS stand for?", [False, [""]]],
    ["Perform the National Institutes of Health Stroke Scale", [True, ["NIHSS", "
        score", "7"]]],
    ["Ask for allergies", [True, ["no", "known", "allergies"]]],
    ["What are his allergies", [True, ["no", "known", "allergies"]]],
    ["Ask for previous surgeries or trauma", [True, ["no", "recent", "major", "
        trauma", "surgery", "bleeding"]]],
    ["Has the patient had any previous surgeries", [True, ["no", "recent", "major",
         "trauma", "surgery", "bleeding"]]],
    ["Ask for previous stroke, myocardial infarction or vascular events", [True, ["
        no", "recent", "stroke", "myocardial", "infarction"]]],
    ["Ask for troponine", [True, ["troponine", "negative"]]],
    ["What is the troponine", [True, ["troponine", "negative"]]],
    ["Is the troponine negative?", [True, ["troponine", "negative"]]],
    ["Ask for blood pressure", [True, ["210/105", "mmHg"]]],
    ["Can you give me the blood pressure", [True, ["210/105", "mmHg"]]],
    ["Ask for heart rate", [True, ["87", "bpm"]]],
    ["whats the hear rat in bpm", [True, ["87", "bpm"]]],
    ["Whats the patient bpm", [True, ["87", "bpm"]]],
    ["I measure his heart rate", [True, ["87", "bpm"]]],
    ["Ask for respiratory rate", [True, ["16", "breaths","min"]]],
    ["Ask for patient temperature", [True, ["36.6","C"]]],
    ["I take the temperature", [True, ["36.6","C"]]],
    ["Ask for oxygen saturation", [True, ["99%", "room", "air"]]],
    ["Is it possible to know the oxygen saturation", [True, ["99%", "room", "air"
        ]]],
    ["ECG monitoring", [True, ["regular", "cardiac", "rhythm", "85bpm"]]],
    ["Measure glycemia", [True, ["105mg/dL"]]],
    ["Ask for blood samples", [True, ["blood", "samples", "analysis"]]],
    ["Ask for patient medication", [True, ["not", "taking", "medication", "
        anticoagulants"]]],
    ["Did the patien take any medication?", [True, ["not", "taking", "medication",
        "anticoagulants"]]],
    ["When was the patient last seen well", [True, ["last", "seen", "well", "3", "
        hours"]]],
    ["Ask for medical history", [True, ["no", "known", "cardiovascular", "risk", "
        factors"]]],
    ["Ask for weight", [True, ["80","kg"]]],
    ["How many kg is the patient", [True, ["80","kg"]]],
```

```
    ["Ask for a CT scan", [False, [""]]],
    ["Lower blood pressure", [False, [""]]],
    ["Ask for a CT angiogram", [False, [""]]],
    ["Perform thrombolysis", [False, [""]]],
    ["Check blood analysis", [False, [""]]],
    ["Perform thrombectomy", [False, [""]]],
]
```