READIFY: INTELLIGENT ASSISTANT TO IMPROVE READING AND COMPREHENSION SKILLS IN ENGLISH LANGUAGE

S. A. D. S. Kumarathunga

(IT21118340)

B.Sc. (Hons) Degree in Information Technology Specialization in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

READIFY: INTELLIGENT ASSISTANT TO IMPROVE READING AND COMPREHENSION SKILLS IN ENGLISH LANGUAGE

S. A. D. S. Kumarathunga

(IT21118340)

B.Sc. (Hons) Degree in Information Technology Specialization in Software Engineering

Department of Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

DECLARATION

I declare that this is my own work, and this proposal does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other university or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Name	Student ID	Signature
S. A. D. S. Kumarathunga	IT21118340	

ABSTRACT

This research explores the application of Artificial Intelligence (AI) in enhancing reading and comprehension skills in the English language, focusing on the development of AI-enabled platforms capable of dynamically generating adaptable content and providing automated assessments and feedback for essay-type questions. By integrating Large Language Models (LLMs) and Prompt Engineering techniques, this platform aims to replicate human-like evaluations, offering personalized learning experiences tailored to individual learners' needs. The study investigates the effectiveness of LLM in understanding and grading student answers and the utility of LLMs in creating adaptive content. Through a comprehensive review of recent research, this paper highlights the current state and future directions of AI applications in education, underscoring the transformative potential of AI in improving English language proficiency and fostering a more equitable educational landscape.

Keywords: Generative AI, Reading Comprehension, Large Language Models (LLMs), English Language, Prompt Engineering, Automatic Quiz Generation, Retrieval Augmented Generation (RAG), Descriptive Answer Evaluation, AI Workflow, CEFR

ACKNOWLEDGEMENT

I am deeply grateful to my dissertation supervisors, Professor Dasuni Nawainna and Mr. Jeewaka Perera from the Faculty of Computing, whose insightful guidance and unwavering support were instrumental throughout my research journey. Their astute observations and constructive critiques played a pivotal role in shaping my research and elevating its overall quality.

I extend my heartfelt thanks to the CDAP team for their invaluable assistance and encouragement throughout my academic pursuits. Their commitment to both teaching and research has been a constant source of inspiration for me.

I would also like to express my sincere appreciation to our esteemed panel members for their thoughtful comments and feedback throughout the research process. Their contributions have significantly enhanced the depth and validity of my work.

Lastly, I wish to acknowledge the invaluable support and collaborative spirit of my fellow team members. Our engaging discussions and idea-sharing sessions have greatly enriched my understanding of the subject matter and broadened my perspectives.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

- Gen AI - Generative AI
- LLM - Large Language Model
- RAG - Retrieval-Augmented Generation
- UI - User Interface
- UX - User Experience
- DB - Database
- API - Application Programming Interface
- UAT - User Acceptance Testing
- CEFR - Common European Framework of Reference for Languages
- CoT - Chain of Thought

1. INTRODUCTION

1.1 Background

Advanced comprehension skills are crucial for mastering the English language. These skills include the ability to understand the implied meaning of a text, evaluate and analyze the content, and make inferences and deductions based on the information provided. Essential components of these skills involve connecting different parts of the text, applying background knowledge, identifying the main idea, finding important facts and supporting details, summarizing content, and generating as well as asking relevant questions [1].

The importance of reading comprehension skills in English language learning cannot be overstated, especially in an era where information is readily available yet requires critical evaluation. English, as a global language, is a gateway to vast amounts of information, and proficiency in comprehension is key to accessing, processing, and applying this information effectively. However, despite its importance, many learners struggle with advanced comprehension tasks due to factors such as limited vocabulary, lack of engagement, and insufficient practice in higher-order thinking skills.

This challenge has sparked significant interest in leveraging Artificial Intelligence (AI) and Large Language Models (LLM) to improve reading and comprehension skills. These advanced platforms offer personalized learning experiences, adapting to individual learner needs and providing real-time feedback. These platforms can analyze learner behavior, identify weaknesses, and offer targeted exercises to strengthen comprehension skills. The effectiveness of AI and LLM in educational contexts has been widely studied, but its application to enhance reading comprehension in English, particularly in non-native speakers, remains an area ripe for exploration.

1.2 Literature Review

The integration of AI in education, particularly for enhancing reading and comprehension skills in English, has gained considerable traction in recent years. This review explores the current state of research in this area, focusing on AI-enabled platforms designed to dynamically generate adaptable content, automatically assess essay-type responses, and provide feedback. By analyzing key studies in this domain, we aim to contextualize the advancements and challenges in utilizing AI for educational purposes, with a specific emphasis on improving English language proficiency.

1.2.1 Personalized Content Generation based on User Interest and Skill Level

Several studies have explored AI's potential in personalizing educational experiences. Laban et al. (2022) [2], has investigated automated question generation to support educators in quiz design. Their tools assist teachers in creating customized assessments. However, as noted in their work and exemplified by protocols like the Quiz Design Task, their methodology often relies on teacher-defined topics. This approach, while beneficial for pedagogical support, does not directly address personalization driven by individual student curiosity or specific topics of personal interest to the learner.

Thotad, Kallur, and Amminabhavi (2022) [3] presented an Automatic Question Generator designed to create semantically accurate and syntactically cohesive questions, primarily focusing on generating multiple-choice questions (MCQs) with correct answers and distractors from input text provided by teachers. Their system aims to help educators quickly assess student comprehension and allows students to evaluate their own understanding. While effective for generating factual questions from teacher-provided content, this approach is primarily designed as a tool for educators and focuses on factual question types (who, when, where, why, what), rather than being driven by the specific, potentially diverse, interests of individual students or generating more descriptive or open-ended question formats.

Lu et al. (2023) [4] introduced ReadingQuizMaker, a human-NLP collaborative system aimed at assisting instructors in designing high-quality reading quiz questions. Their system integrates NLP support into the instructor's workflow, allowing teachers to decide when and how to use AI models and edit the generated outcomes. This collaborative approach was praised by instructors for its ease of use and helpful suggestions, highlighting the importance of user control in AI-assisted creative tasks. However, ReadingQuizMaker requires significant human involvement (the instructor) in the question design process. In contrast, a system focused on automated quiz generation based directly on user interest aims for a higher degree of automation from the learner's perspective, minimizing the need for intermediary human editing or selection.

More recent work, such as that by Contreras-Arguello et al. (2025) [5], has explored integrating NLP techniques to automatically generate both summaries and questions from text provided by the teacher. Their system aims to save teachers time in creating educational resources, allowing them to focus on supporting students with greater reading comprehension needs. Similar to the aforementioned studies, this system's input text is provided by the teacher, and while it supports the reading comprehension process, the content generation is not directly initiated or guided by the specific, varied interests of the individual student.

1.2.2 Automated Assessment and Feedback Generation

The second major challenge is the absence of reliable tools for the automated assessment of essay-type questions, which traditionally require human evaluation due to the complexity and subjectivity involved in grading. This area has seen considerable research, yet existing solutions remain inadequate for high-stakes educational environments.

Moholkar et al. (2024) [6] provide a comprehensive survey of machine learning techniques for evaluating descriptive answers. Their survey covers various approaches, including supervised learning models and ensemble methods, highlighting their effectiveness in grading to some extent. However, the authors acknowledge the limitations in these models' ability to handle the nuances of human language, such as

context, tone, and rhetorical devices, which are crucial for evaluating essay-type questions. The models surveyed often struggle with maintaining consistency and fairness in grading, especially in cases requiring deep contextual understanding.

More recent studies have begun to explore the use of LLMs themselves for automated assessment. Xia et al. (2024) [7] examined the effectiveness of LLMs for automated essay scoring, specifically using the TOEFL Independent Writing Task. While their findings indicated promise, they also identified drawbacks, including the models' susceptibility to being overly influenced by superficial features like essay length or vocabulary richness, and a general lack of transparency regarding the rationale behind their scores. Furthermore, their work primarily focused on the assessment of writing tasks (essays), which presents different challenges compared to evaluating responses to reading comprehension questions, which requires assessing understanding of an external text rather than original composition.

Similarly, Hussein et al. (2019) [8] reviewed the evolution of automated essay scoring systems, from earlier rule-based methods to more advanced ML approaches. They concluded that despite considerable progress, consistently replicating human-like grading, particularly for complex tasks like evaluating creative or argumentative essays, remains an elusive goal. This suggests that traditional ML paradigms may inherently struggle with the subjective and complex aspects of grading free-text responses, pointing towards the need for more advanced computational strategies. Their work underscores the limitations of methods that do not fully harness the deep linguistic understanding offered by modern LLMs.

This research intends to overcome these limitations by developing a novel assessment tool that combines the strengths of machine learning with advanced linguistic analysis to better emulate human evaluators. This tool will focus on understanding the content and structure of essays more deeply, including context, argumentation, and rhetorical devices, thereby improving grading accuracy and fairness. Moreover, by integrating this tool with the dynamically adaptable content system, the platform would not only assess but also provide immediate, actionable feedback, thereby enhancing the learning process in real-time.

1.2.3 Conclusion

The research landscape for AI-enabled platforms aimed at improving reading and comprehension skills in English is marked by significant advancements but also by critical gaps, particularly in the areas of dynamically adaptable content and automated essay-type question evaluation. Existing studies have laid important groundwork but have not fully addressed these challenges. The proposed research seeks to fill these gaps by developing a more responsive and nuanced AI-enabled platform that provides both dynamic content adaptation and robust assessment tools for essay-type questions. By doing so, it aims to offer a more personalized and effective learning experience, bridging the current shortcomings in educational technology.

1.3 Research Gap

The existing research on AI-enabled platforms designed to enhance English language reading and comprehension skills reveals significant technological limitations, particularly concerning the dynamic adaptability of content and the robust assessment and feedback mechanisms for essay-type quizzes. Current educational technologies generally struggle to dynamically adjust content based on individual learner interests. This is especially problematic in English language education, where learners often require personalized content that evolves with their progress and caters to their specific learning interests to foster deep engagement and effective skill development.

While deep learning models have demonstrated impressive capabilities in generating human-like text, they remain constrained by their inability to continuously modify content to create tailored quizzes based on their unique interests. Existing research predominantly highlights platforms relying on static content, which proves ineffective in engaging learners or addressing their individual learning interests [9]. This inherent lack of adaptability impedes the cultivation of nuanced reading and comprehension skills, as learners aren't consistently provided with content that is both appropriately challenging and accessible.

Moreover, a significant barrier to developing critical reading and writing skills lies in the absence of robust tools capable of evaluating user-generated, descriptive, and open-ended answers to essay-type quizzes and providing constructive, meaningful feedback without human intervention. Essay-type questions demand not only a thorough understanding of the subject matter but also the ability to articulate thoughts coherently and persuasively. Despite advancements in Natural Language Processing (NLP), current AI systems struggle to provide accurate, context-sensitive feedback on such responses, which is vital for nurturing higher-order cognitive skills [10]. The complexity of evaluating argumentation, creativity, and coherence in open-ended answers often surpasses the capabilities of most automated systems, typically necessitating human intervention and consequently limiting scalability. In English language learning, where essay-type quizzes are paramount for assessing comprehension and language proficiency, this gap is particularly problematic.

Therefore, there is a pressing need for platforms that can leverage LLM-based multi-agentic RAG workflows to not only dynamically generate essay-type quizzes tailored to individual user interests and skill level but also accurately evaluate descriptive, open-ended user answers and provide constructive, meaningful feedback without human intervention. This approach aims to bridge the gap between personalized learning and scalable, high-quality education in English language comprehension.

1.3.1 Comparison with Similar Products

Table 1.1 Comparison of Similar Products

Apps	Dynamic Content	Evaluate answers without human involvement	Provide feedback without human involvement
Generic Reading Apps	✗	✗	✗
Duolingo	✓	Only offer MCQ based questions	Limited feedback
AceReader Pro	✗	✗	✗
ReadyRead	✗	✗	✗
Smart AI Reading Assistant for Reading Comprehension	✗	✗	✗
Proposed System	✓	✓	✓

When compared to existing products, the limitations of current reading and comprehension apps become more apparent. Generic reading apps, for instance, typically offer a limited amount of static content that does not adapt to the learner's progress or interests. This static nature fails to engage users over time and does not cater to the individual learning paths necessary for effective reading comprehension.

AceReader Pro [11], while aiming to improve reading comprehension through fast reading techniques, focuses predominantly on speed, which may not be suitable for all students, especially those still developing basic reading skills. Furthermore, AceReader Pro's [9] reliance on pre-existing content limits its adaptability and fails to align with the user's personal interests or learning needs.

Similarly, ReadyRead [12], a mobile application designed as supplementary material for reading comprehension, primarily addresses the issue of material insufficiency in the classroom but suffers from similar limitations. The app relies heavily on static, pre-created content and lacks personalized learning paths, thereby failing to provide dynamic content needed for effective skill development.

Most critically, none of these products possess the capability to generate dynamic, interest-tailored essay-type quizzes or accurately assess and provide comprehensive, automated feedback on descriptive, open-ended user answers, a crucial aspect of language learning that typically requires human evaluation. This gap underscores the need for a more sophisticated, AI-driven platform that can deliver both dynamic content generation and nuanced automated assessment tools, addressing the comprehensive needs of English language learners.

1.4 Research Problem

1.4.1 Lack of Tools for Generating Personalized Content Based on User Interest and Skill Level

In contemporary English education, one of the most pressing challenges is the lack of dynamically adaptable content that can adjust to the varying skill level and interests of learners. This contributes to student dissatisfaction, as they are presented with limited options for reading materials and content. As a result, students often find themselves bored and less engaged, which significantly hampers their ability to achieve high-level comprehension skills.

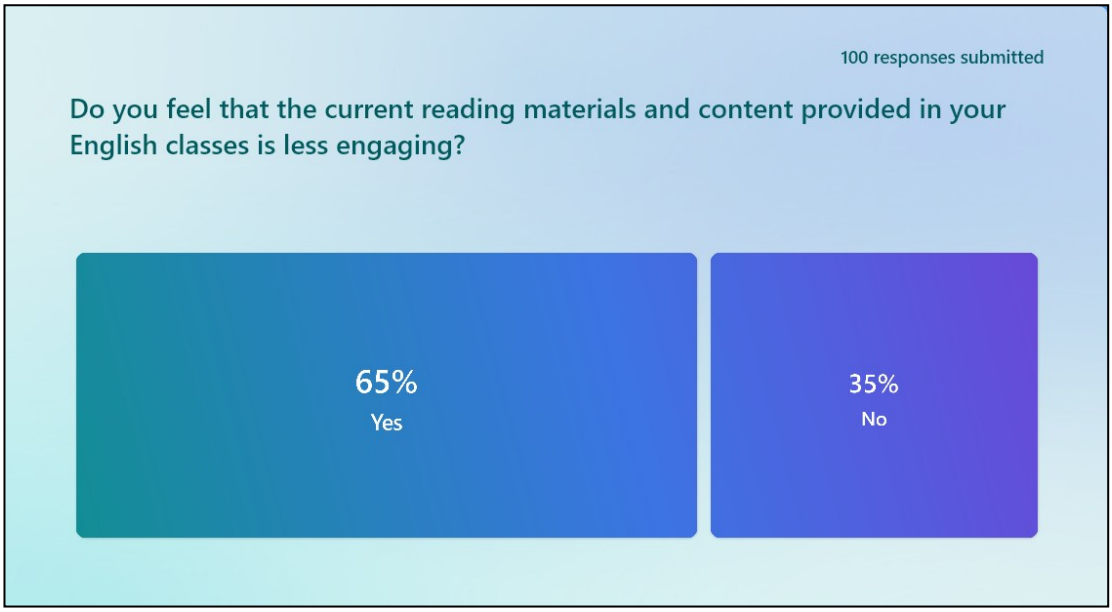


Figure 1.1 Survey report on student opinion regarding static reading materials

Introducing reading materials and exercises that dynamically adapt to student needs and interests can markedly improve student engagement with the materials, thereby enhancing comprehension skills. Such adaptability ensures that the content remains relevant and challenging, keeping students motivated and invested in their learning journey.

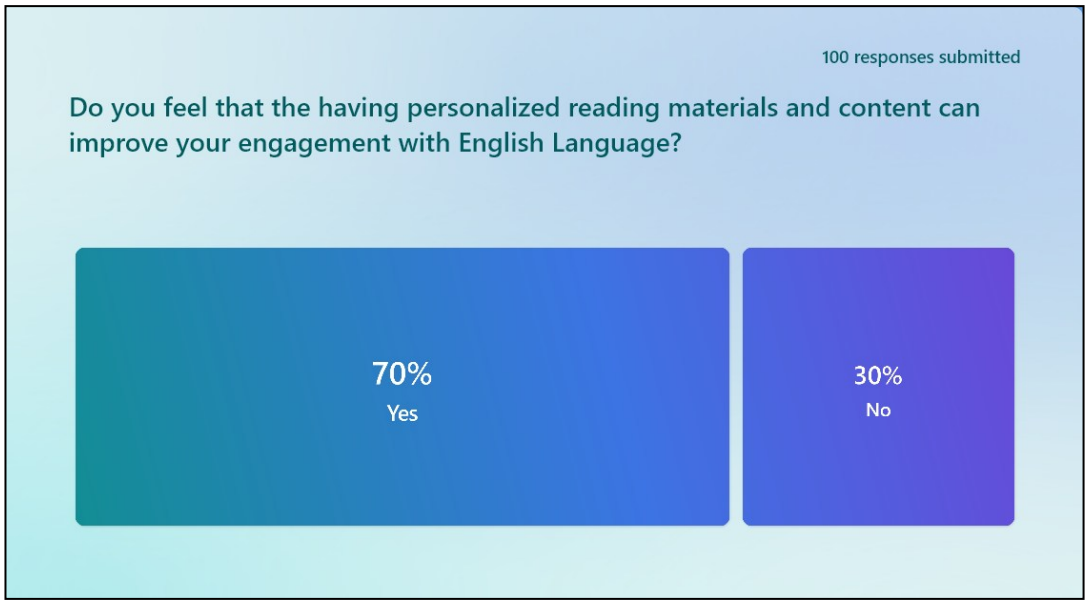


Figure 1.2 Survey report on student opinion regarding personalized reading material

Most critically, none of these products possess the capability to generate dynamic, interest-tailored essay-type quizzes or accurately assess and provide comprehensive, automated feedback on descriptive, open-ended user answers, a crucial aspect of language learning that typically requires human evaluation. This gap underscores the need for a more sophisticated, AI-driven platform that can deliver both dynamic content generation and nuanced automated assessment tools, addressing the comprehensive needs of English language learners.

1.4.2 Lack of Tools to Provide Assessments for Essay-Type Questions that Typically Require Human Evaluation

Another significant challenge in English education is the absence of tools capable of autonomously evaluating answers for essay-type questions. This limitation forces students to rely on their English teachers for feedback, a process that is not only time-consuming but also inefficient. Students often have to wait for extended periods to receive feedback on their work, which not only wastes valuable time but also limits the opportunities for immediate correction and skill enhancement. The delay in receiving feedback can hinder the development of critical thinking and writing skills, which are essential for success in English education and beyond.

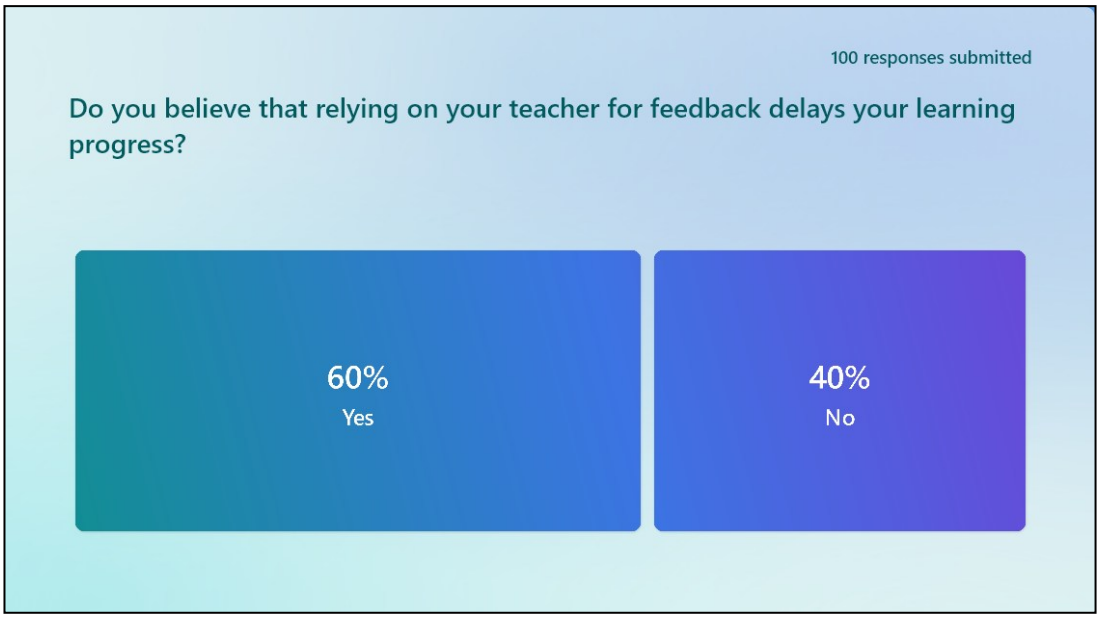


Figure 1.3 Survey report on student opinion regarding manual answer evaluation process

Having tools that automatically provide assessments and feedback for essay-type questions can significantly improve the student learning process. Such tools can offer immediate, personalized feedback, allowing students to identify and rectify errors promptly. This immediate feedback is crucial for facilitating rapid learning cycles and encouraging students to actively engage with their work, thereby enhancing their comprehension skills and overall academic performance.

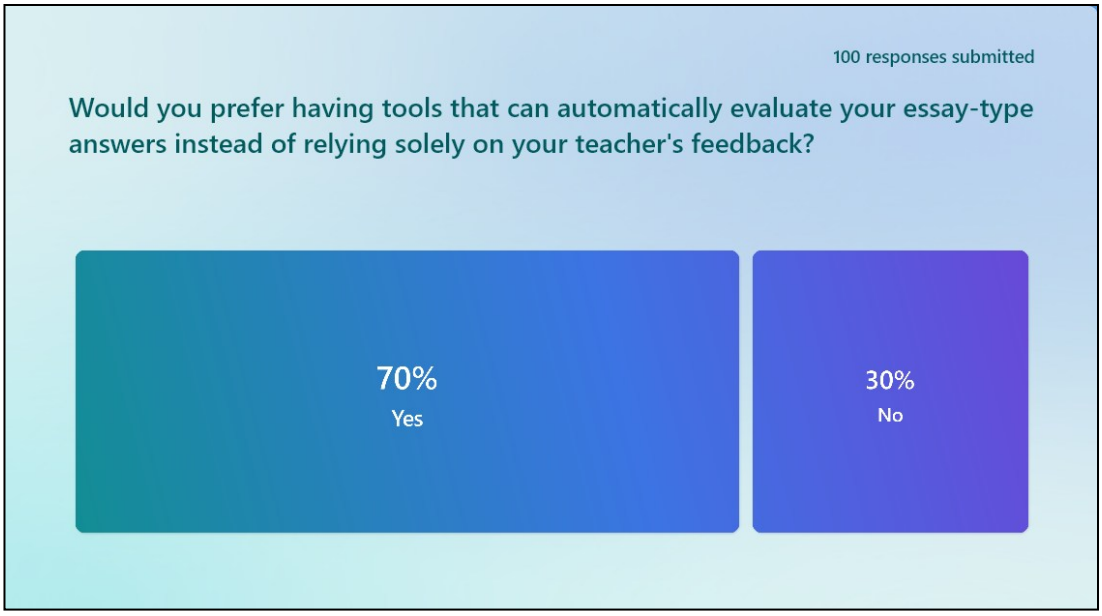


Figure 1.4 Survey report on student opinion regarding automated answer evaluation process

By addressing these challenges, educators can create a more dynamic, efficient, and personalized learning environment that better serves the needs of English language learners, paving the way for improved educational outcomes.

1.5 Research Objectives

1.5.1 Main Objective

The main objective of this study is to develop a web-based intelligent assistant that enhances English language learners' advanced comprehension skills—specifically sequencing, summarizing, and self-questioning—through personalized content delivery and automated answer evaluation using large language models.

1.5.2 Specific Objectives

These are specific objectives that must be reached in order to achieve the overall objective described above.

1. To develop a CEFR-based user assessment mechanism for initial placement and rule-based progression or regression based on ongoing performance in comprehension tasks.
2. To integrate a Retrieval-Augmented Generation (RAG) and Corrective-RAG (CRAG) framework that enhances the factual accuracy and contextual relevance of LLM-generated educational content.
3. To design and implement a dynamic content generation module using a large language model (LLM), capable of producing reading materials and comprehension exercises tailored to individual learners' interests and CEFR-based proficiency levels.
4. To engineer a multi-agent question generation workflow that autonomously creates descriptive, CEFR-aligned quizzes from user-provided topics using web-scraped research data and prompt-driven LLM agents.
5. To construct an automated descriptive answer evaluation system that uses multiple LLMs and confidence-based fallback mechanisms to assess user responses, assign scores, and generate constructive feedback without human intervention.
6. To deploy the web application with a modular microservices architecture and ensure cross-platform accessibility, scalability, and data security using cloud-native DevOps practices.

2. METHODOLOGY

2.1 Research Domain

2.1.1 Generative AI

Generative AI refers to a subset of artificial intelligence technologies that specialize in creating new and original content, ranging from text and images to complex synthetic data and even deepfakes. This technology leverages advanced neural network techniques such as transformers, generative adversarial networks (GANs), and variational autoencoders (VAEs) to autonomously generate a wide array of outputs. Unlike traditional AI algorithms that operate based on predefined rules, generative AI excels in tasks requiring the creation of new content, especially in natural language processing (NLP) domains. It typically begins with a prompt or dataset to guide the content generation process, allowing for iterative exploration of content variations. The advent of generative AI has opened up innovative possibilities across various sectors, from creative fields to problem-solving, promising to revolutionize how we interact with technology and create content. The advent of generative AI has opened up innovative possibilities across various sectors, from creative fields to problem-solving, promising to revolutionize how we interact with technology and create content. The advent of generative AI has opened up innovative possibilities across various sectors, from creative fields to problem-solving, promising to revolutionize how we interact with technology and create content. The advent of generative AI has opened up innovative possibilities across various sectors, from creative fields to problem-solving, promising to revolutionize how we interact with technology and create content. The advent of generative AI has opened up innovative possibilities across various sectors, from creative fields to problem-solving, promising to revolutionize how we interact with technology and create content.

2.1.2 Large language models (LLM)

Large Language Models (LLMs) are a class of artificial intelligence models specifically designed to understand, generate, and manipulate human language. Characterized by their immense size, often comprising billions or even trillions of parameters, LLMs are trained on colossal datasets of text and code, allowing them to learn intricate patterns, grammar, semantics, and common-sense reasoning. The architectural backbone of most modern LLMs is the Transformer network, which utilizes an "attention" mechanism to efficiently handle long-range dependencies in text. While powerful in generating highly coherent and contextually relevant text, LLMs operate based on statistical probabilities, which can lead to "hallucinations" (generating factually incorrect information) or perpetuating biases present in their training data, and their knowledge is limited to their training cut-off.

2.1.3 Prompt engineering

Prompt engineering is a critical process in the development and refinement of generative AI systems, including Large Language Models (LLMs) and other generative AI tools. It involves carefully crafting input prompts that influence the direction, quality, and relevance of the output produced by these models. By designing effective prompts, developers can steer the AI towards specific types of content, answer formats, or behaviors, optimizing the model's performance for particular tasks or applications. This technique requires a blend of linguistic expertise and creative thinking to fine-tune prompts, extracting the most desirable outcomes from the AI models

2.1.4 RAG and C-RAG

Retrieval-Augmented Generation (RAG) is an architectural pattern designed to enhance LLMs by integrating an information retrieval component, thereby addressing inherent limitations such as knowledge cut-off and the tendency to "hallucinate" or generate factually incorrect information. The RAG process typically involves two main stages: first, a retrieval mechanism identifies and fetches the most relevant documents or passages from a predefined knowledge base based on a user query; second, these retrieved documents are then passed along with the original query to the LLM, which uses this augmented context to generate a more accurate, factually grounded, and specific response. This approach ensures that the LLM's output is rooted in verifiable external information, improving factual accuracy and transparency.

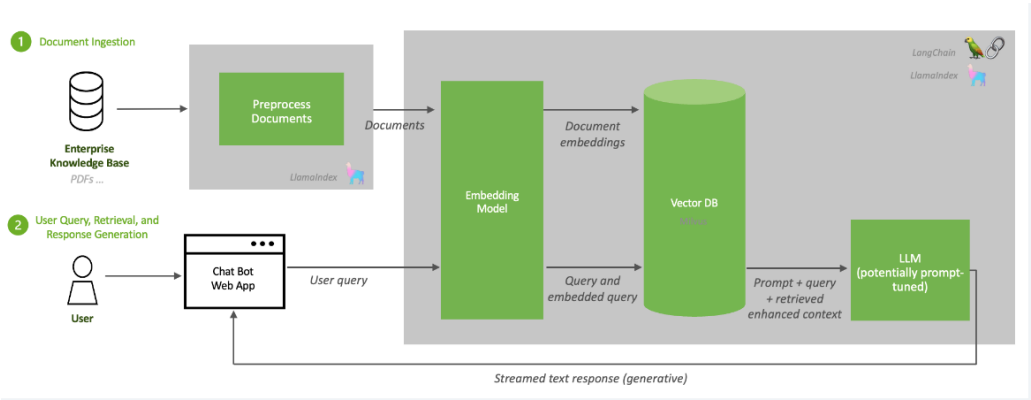


Figure 2.1 Survey report on student opinion regarding static reading materials

Building upon this, Corrective RAG (CRAG) further refines the RAG framework by adding a self-correcting mechanism to address the limitations of RAG models [13]. CRAG goes a step further by evaluating the quality of retrieved documents using a retrieval evaluator to determine if they are relevant, and if not, it performs a web search to find better information. This two-step process—assessing the retrieved content and then correcting it—ensures that the final output is more accurate and trustworthy.

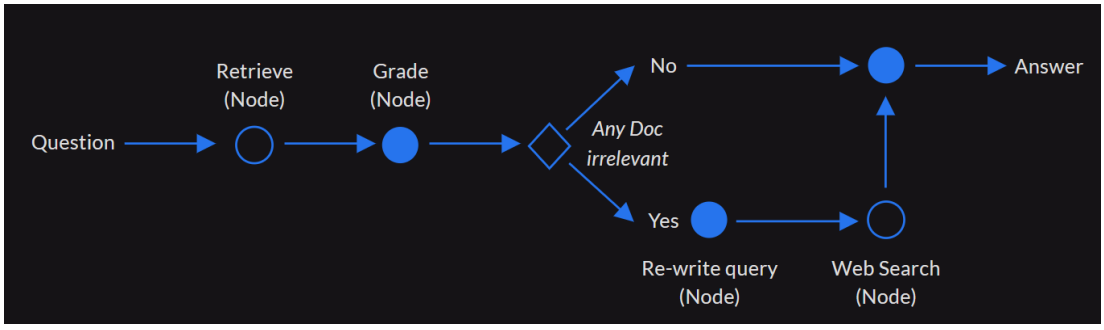


Figure 2.2 Corrective-RAG Process

2.2 Methodology

2.2.1 Project Requirements

This section outlines the essential requirements for the platform, derived from a comprehensive process of collecting and analyzing information from various sources to define the project's scope and objectives. This approach ensures that the platform effectively addresses the needs of all stakeholders involved in English language education in Sri Lanka.

A) Requirements Gathering and Analysis

The requirements gathering process for this platform includes:

Data Gathering:

- Collecting publications from the Sri Lankan Education Department to understand curriculum standards and learning objectives.
- Gathering fair-use materials from the internet to build a comprehensive content library.
- Ensuring compliance with copyright laws and obtaining necessary permissions.

Surveying Stakeholders:

- Parents: Understanding their expectations and concerns, and assessing their willingness to pay for premium services.
- Students: Identifying specific learning needs and preferences, along with preferred features and functionalities.
- Educators: Gaining insights into current teaching methods and desired outcomes.

B) Functional Requirements

- The system should be able to dynamically generate content based on the user's interest and skill level.
- The system should be able to accurately evaluate user answers based on given criteria without human involvement.
- The system should be able to provide extensive feedback about the user's answer.

C) Non-Functional Requirements

- User-Friendliness: The system should provide a cross-platform application for users while maintaining an attractive, responsive, and well-comprehensible interface.
- Reliability: The system should not fail or get stuck at any time throughout the process. Users should feel secure and confident while using the application. All sensitive information must be protected.
- Performance: The system should perform efficiently by providing fast and accurate results to the users.
- Availability: The application should be accessible to all users, regardless of language. It should be available whenever it is needed.

2.2.2

2.2.3 Tools and Technologies

This section details the key tools and technologies employed in the development of this platform, highlighting their specific roles and contributions to the system's architecture and functionality. The selection of these technologies is driven by the need for a robust, scalable, and intelligent system capable of delivering dynamic and accurate English language education content.

A) Frontend Application

The user-facing interface of the platform is developed using **React**. React is a declarative, component-based JavaScript library for building user interfaces. Its efficiency in managing dynamic content and providing a responsive user experience makes it an ideal choice for delivering an attractive and intuitive cross-platform application [14].

B) Backend Application

The backend services and APIs are built with **FastAPI**. FastAPI is a modern, fast (high-performance) web framework for building APIs with Python. Its asynchronous capabilities and automatic validation of data ensure efficient and reliable communication between the frontend and the underlying logic, supporting the system's performance requirements[15].

C) Large Language Model (LLM) Integration

For seamless integration and orchestration of Large Language Models (LLMs), the project leverages LangChain and LangGraph.

- **LangChain** is a framework designed to simplify the development of applications powered by LLMs. It provides modular components and tools for chaining together different LLM calls, managing prompts, and integrating with external data sources [16].
- **LangGraph** extends LangChain by enabling the creation of stateful, multi-actor applications with LLMs, allowing for more complex and dynamic interaction flows, which is crucial for advanced AI agent behaviors [17].

- **Tool Calling/Tool Use Pattern:** This pattern involves empowering LLMs to interact with external tools, APIs, or functions to perform specific actions or retrieve information beyond their internal knowledge. The LLM acts as an "agent" that decides when and how to use these tools to achieve a given task, enabling more complex and dynamic interactions with the real world.

D) Databases

The platform utilizes Supabase as its primary database solution. Supabase is an open-source Firebase alternative that provides a PostgreSQL database, authentication, instant APIs, and real-time subscriptions. Its comprehensive features simplify data management and ensure the protection of sensitive user information, contributing to the system's reliability [18].

For efficient semantic search and retrieval of relevant content, Pinecone is employed as the vector database. Pinecone serves as a specialized vector database designed for efficient semantic search and retrieval of relevant content [19].

E) Prompt Engineering Techniques

To optimize the performance and reasoning capabilities of the integrated LLMs, various advanced prompt engineering techniques are applied:

- **Zero-Shot Prompting:** This technique involves providing the LLM with a task or question without any prior examples, relying solely on its pre-existing knowledge to generate a response. It demonstrates the LLM's ability to generalize from its vast training data [20].
- **Few-Shot Prompting:** In contrast to zero-shot, few-shot prompting includes a small number of input-output examples within the prompt to guide the LLM's response, steering it towards the desired output format, tone, or style [21].
- **Chain of Thought (CoT):** This method enhances the reasoning capabilities of LLMs by encouraging them to break down complex problems into a series of intermediate reasoning steps. By explicitly prompting the model to "think step by step," CoT improves accuracy on tasks requiring multi-step reasoning, such as arithmetic or common sense problems [22].

D) Retrieval-Augmented Generation (RAG) Techniques

- **Corrective RAG (CRAG):** CRAG is an advanced RAG strategy that focuses on improving the accuracy and relevance of generated responses by incorporating mechanisms for self-reflection and self-grading of retrieved documents. It actively evaluates the relevance of retrieved information and can trigger corrective actions, such as web searches, if the initial retrieval is deemed insufficient, thereby reducing hallucinations and improving factual consistency[23].

2.2.4 System Design

2.2.4.1 System Diagram

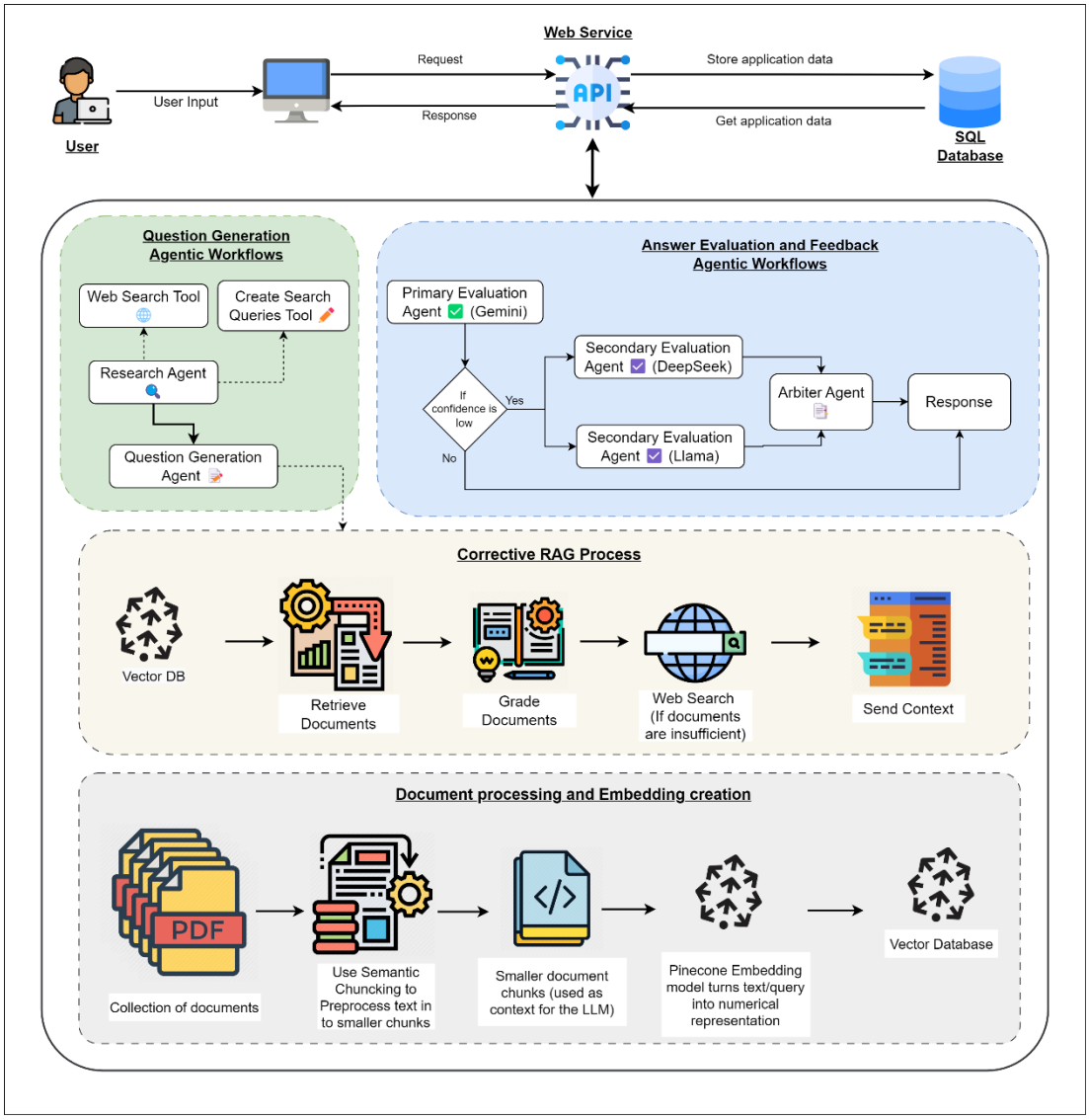


Figure 2.3 System Diagram

2.2.4.2 Use Case Diagram

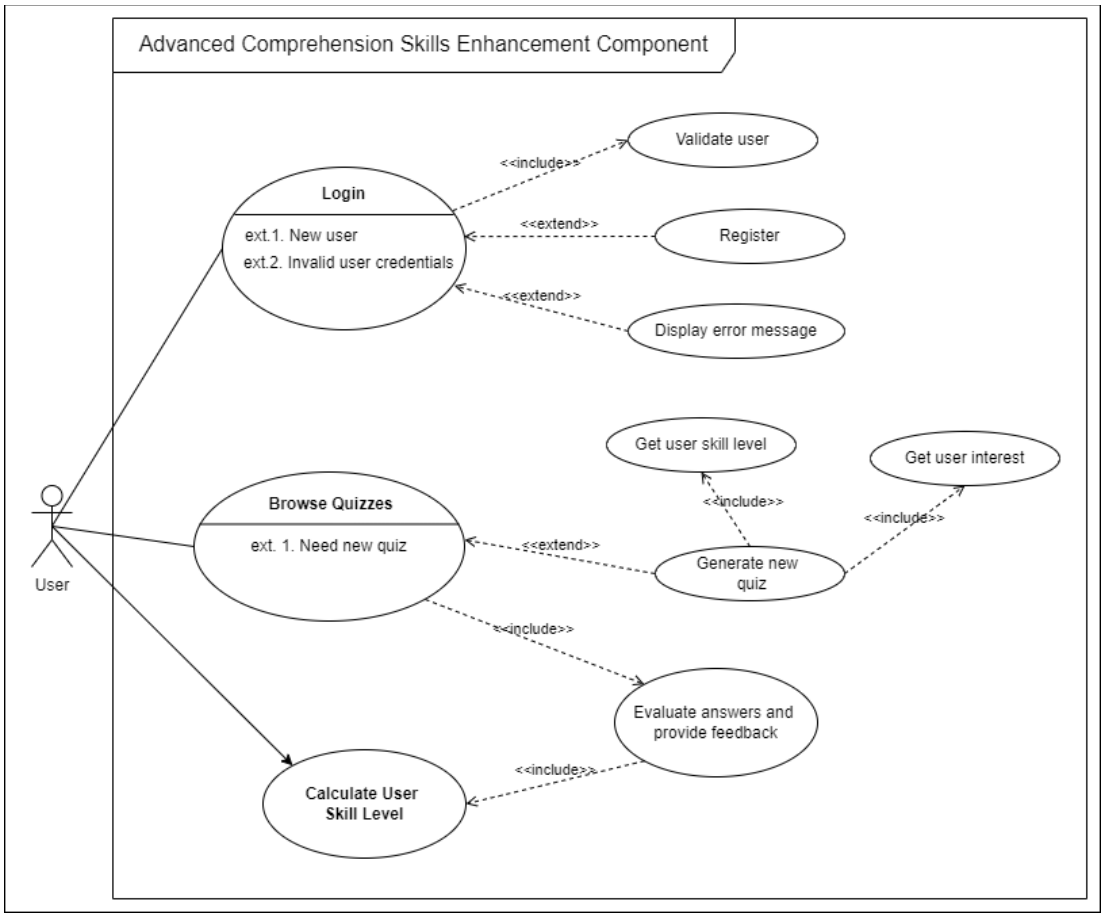


Figure 2.4 Use Case Diagram

2.2.5 RAG Embedding Creation Pipeline

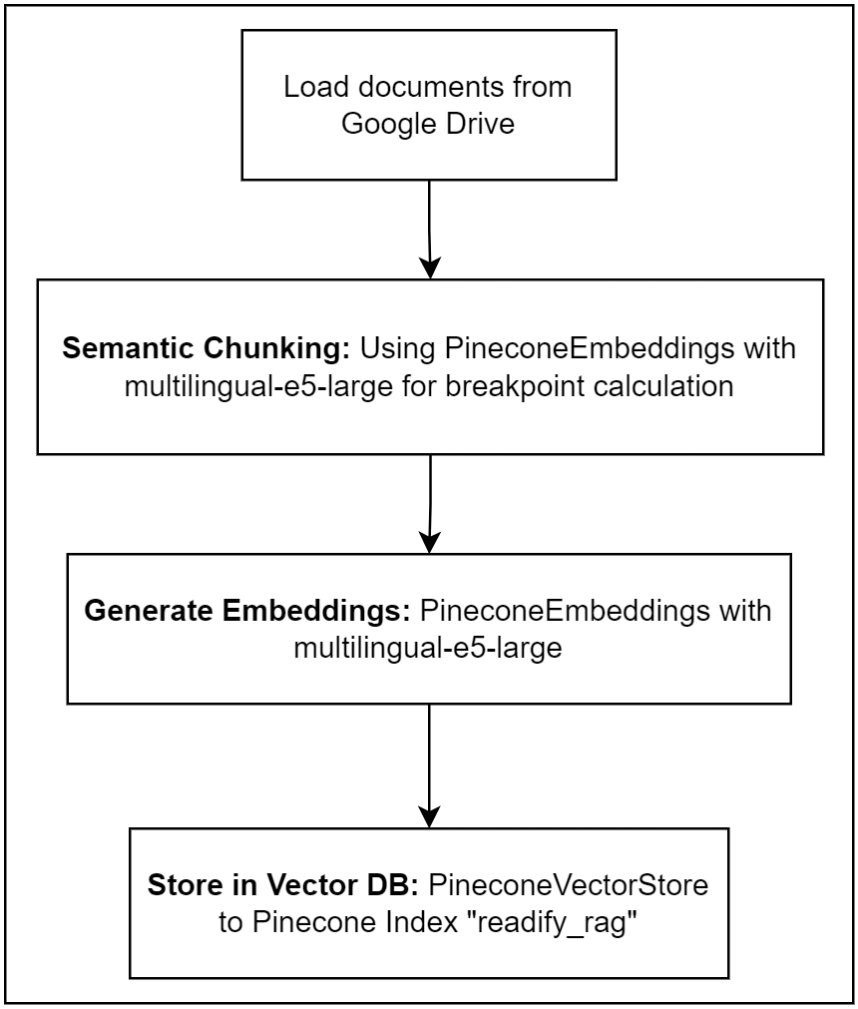


Figure 2.4 Embedding Generation and Storage

The RAG Embedding Creation Pipeline is designed to transform raw textual data into a searchable format that can be effectively utilized by a retrieval system. The process begins with the acquisition of source documents and culminates in their storage within a specialized vector database.

1. **Document Ingestion:** The initial phase involves loading source documents from a designated repository.
2. **Semantic Chunking:** Following ingestion, documents undergo semantic chunking, an advanced text splitting technique that goes beyond simple segmentation. It uses multilingual-e5-large embedding model provided by Pinecone to identify breakpoints where semantic meaning shifts, ensuring each "chunk" is a logically complete unit.

Semantic chunking offers key advantages for RAG applications:

a. Enhanced Retrieval Relevance: Preserves chunk integrity, increasing the likelihood of retrieving complete, contextually relevant information.
b. Reduced Contextual Noise: Creates focused chunks, minimizing irrelevant information for the LLM.
c. Improved LLM Performance: Provides semantically rich contexts, leading to more accurate and nuanced generations.
d. Optimized RAG Efficiency: Enhances retrieval precision, directly improving the quality of LLM output. Techniques like "context-aware chunking" are actively explored in frameworks like LangChain[20].

3. **Embedding Generation:** After semantic chunking, each text chunk is transformed into a high-dimensional numerical vector called an embedding. This process utilizing multilingual-e5-large model provided by Pinecone to capture the semantic meaning of the text. Chunks with similar meanings are mapped to numerically close points in the vector space, facilitating efficient similarity searches.

4. **Vector Database Storage:** The final step involves storing these generated embeddings in a specialized Pinecone Vector Database (e.g., Pinecone).

2.2.6 CRAG-based Multi-Agent Workflow for Question Generation

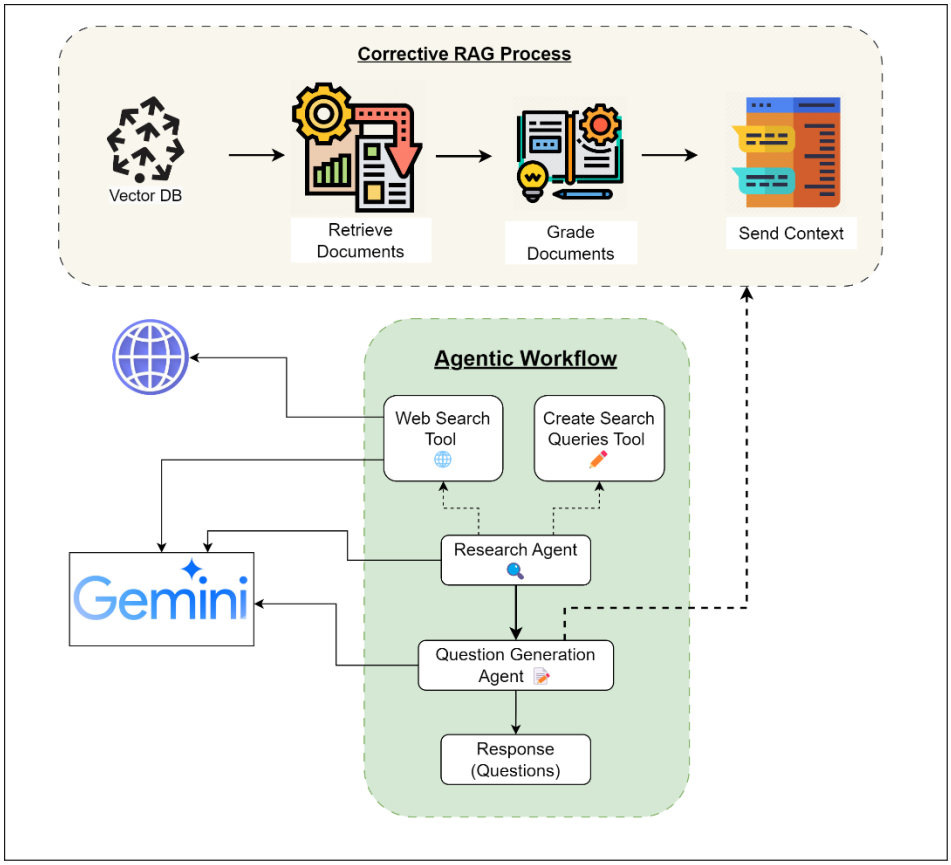


Figure 2.5 Multi-Agentic RAG workflow incorporating Corrective-RAG Process

The generation of reading paragraphs and corresponding comprehension questions is achieved through a Multi-Agentic RAG Workflow employing a Tool Use pattern. As illustrated in Fig. 2.5, this workflow automates the creation of essay-style questions and answers based on user-specified topics. The workflow is structured around a collaborative team of AI agents:

Research Agent: This agent is tasked with gathering relevant information about the user-provided topic. It utilizes a Tool-use pattern, interacting with external tools such as a 'Create Search Queries Tool' to formulate effective search requests and a 'Web Search Tool' to execute searches and retrieve data from the web.

Question Generation Agent: Using the information retrieved by the Research Agent and incorporating data enhanced by the Corrective Retrieval Augmented Generation (C-RAG) process, this agent generates a set of comprehension questions along with detailed answers, taking into account the user's current CEFR Level. The output is formatted in a structured JSON format for subsequent processing.

Corrective Retrieval Augmented Generation (CRAG) Process: To ensure generated content is factually grounded, Retrieval Augmented Generation (RAG) combines Large Language Models with external knowledge retrieval, typically from a vector database. This grounds LLM responses. However, standard RAG's performance hinges on the quality of retrieved data; irrelevant or inaccurate retrieval yields suboptimal outputs.

Readify utilizes the Corrective Retrieval Augmented Generation (CRAG) Process to overcome this limitation. CRAG enhances standard RAG by evaluating retrieved information quality. The workflow retrieves data from the Vector Database (Pinecone), assesses its relevance and quality, and based on evaluation, either refines the retrieved set or triggers alternative retrieval strategies if data is insufficient. This self-correcting process ensures the knowledge supplied to the downstream Question Generation Agent is accurate and contextually relevant, fundamentally improving the quality of generated questions and answers.

2.2.7 Multi-Model Agentic Workflow with Confidence-Based Fallback and Arbitration Mechanism For Answer Evaluation and Feedback Generation

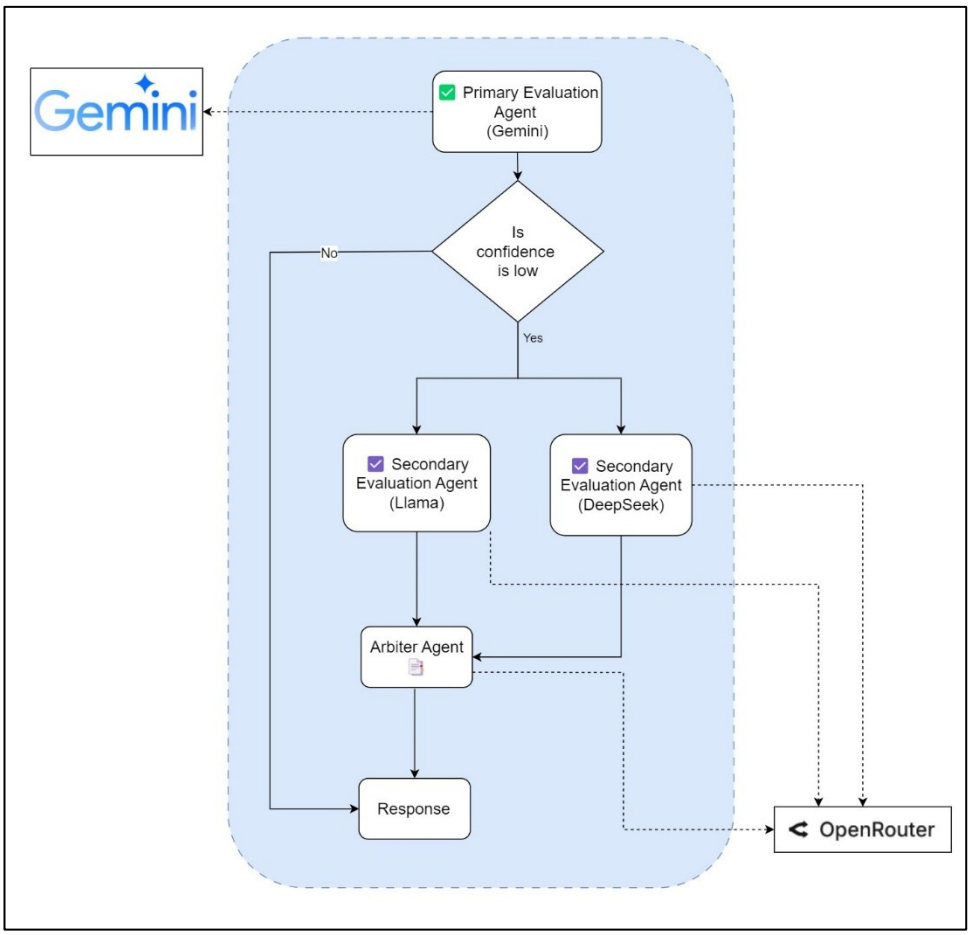


Figure. 2.6 Multi-Model Agentic Workflow incorporating a Confidence-Based Fallback and Arbitration Mechanism

As shown in Fig. 2.6, Evaluating student responses and providing constructive feedback is managed through a Multi-Model Agentic Workflow incorporating a Confidence-Based Fallback and Arbitration Mechanism. This framework enhances the robustness and accuracy of the evaluation process by leveraging multiple language models and a synthesis mechanism.

Primary Evaluation Agent (Gemini): The evaluation process begins with the application of Google's Gemini generative model. Gemini performs a detailed contextual analysis of the student's submitted response against the expected answer. It generates evaluative feedback and calculates a confidence metric indicating its certainty in the assessment.

Secondary Evaluation (DeepSeek and Llama via OpenRouter): If the confidence metric from the Primary Evaluation Agent falls below a predefined threshold, a fallback mechanism is activated. This triggers secondary evaluation agents utilizing DeepSeek and Llama language models, accessed via the OpenRouter API. These models provide independent, parallel assessments of the student's response.

Arbiter Agent: This final agent receives the feedback generated by the secondary evaluation models (DeepSeek and Llama). It performs a synthesis and evaluation of these multiple perspectives, resolving potential discrepancies and creating a consolidated, refined, and comprehensive feedback response for the student. This arbitration process enhances the reliability and fairness of the automated assessment.

2.2.8 CEFR-Based Level Assessment and Dynamic Progression

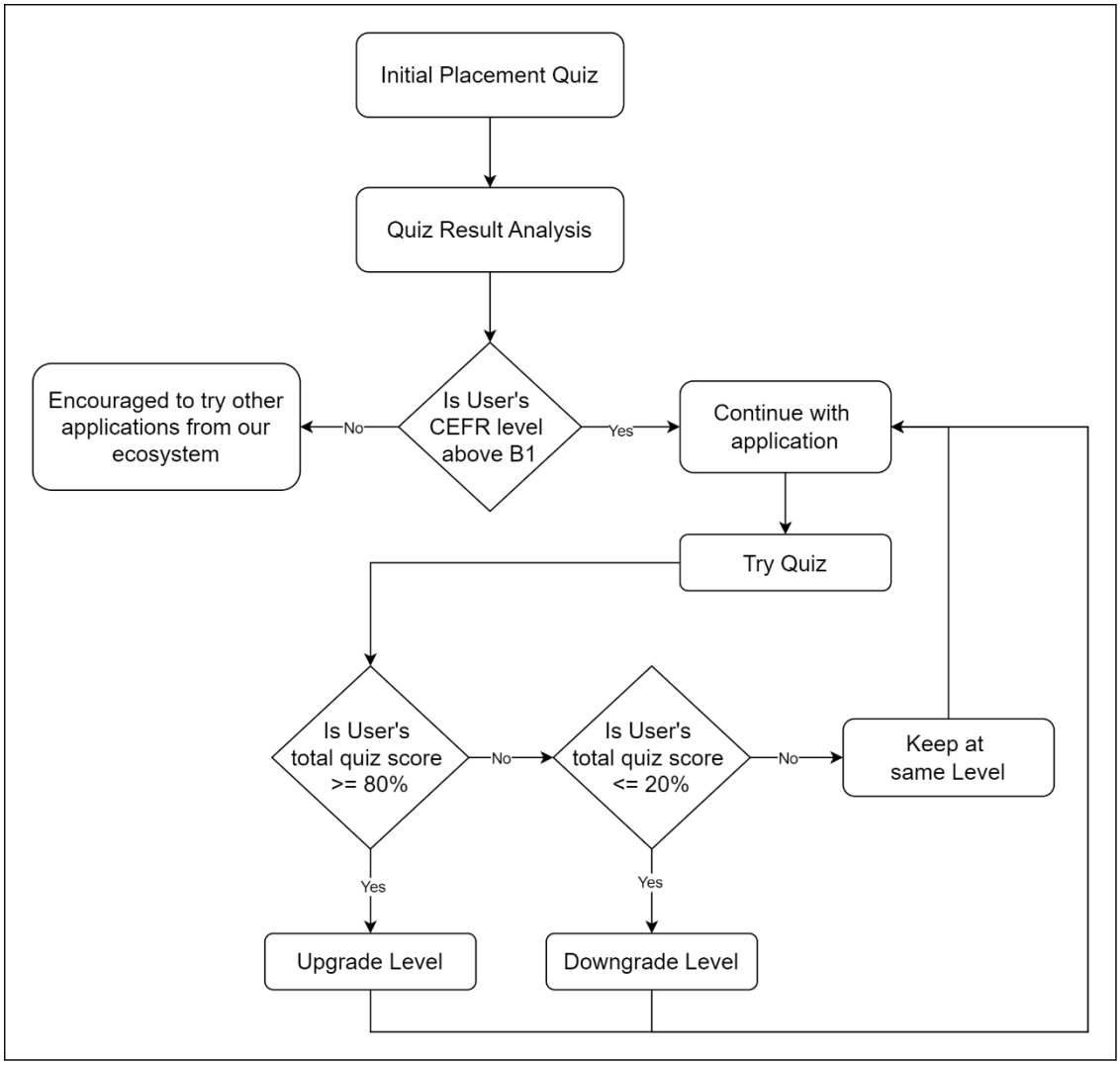


Figure. 2.7 CEFR-Based Level Assessment and Dynamic Progression

This system uses a CEFR-based assessment framework to accurately determine and dynamically adjust a user's language proficiency level. The process begins with an initial placement quiz specifically designed to evaluate a user's current standing within the Common European Framework of Reference for Languages (CEFR), focusing on levels B1, B2, C1, and C2.

Initial Placement and Redirection: Upon completion of the initial quiz, the system analyzes the user's performance. If a user scores below the B1 CEFR level, they are gently yet firmly encouraged to explore our other modules. This ensures that all users begin their learning journey at an appropriate and supportive entry point.

Rule-Based Level Progression and Regression: After the initial placement, the system incorporates a rule-based mechanism for continuous level adjustment. This dynamic system ensures that a user's assigned CEFR level accurately reflects their evolving proficiency.

- **Level Upgrades:** If a student achieves a score of 80% or more on any subsequent quiz, their CEFR level is automatically upgraded. This achievement indicates a strong grasp of the material at their current level and readiness to tackle more complex questions. This ensures that high-achieving users are consistently challenged and engaged with content that matches their accelerating progress.
- **Level Downgrades:** Conversely, if a student scores 20% or less on a current quiz, their CEFR level is downgraded. A score this low suggests that the user may be struggling with the material at their assigned level and could benefit from practicing at a slightly less advanced stage.

This dual-directional adjustment mechanism ensures that each user is consistently presented with content that is optimally challenging yet attainable, fostering effective and personalized language learning.

2.3 Implementation and Testing

2.3.1 Implementation

2.3.1.1 RAG Embedding Creation Pipeline

A) Environment Setup and Dependencies

```
# Import Dependecies
import logging
import sys
import os
from google.colab import drive
from google.colab import userdata
from langchain_core.documents import Document
from langchain_community.document_loaders import SimpleDirectoryReader
from uuid import uuid4
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_experimental.text_splitter import SemanticChunker
from langchain_pinecone import PineconeEmbeddings
from langchain_pinecone import PineconeVectorStore
from pinecone import Pinecone, ServerlessSpec

# Mount Google Drive
DRIVE_PATH = '/content/drive'

try:
    drive.mount(DRIVE_PATH)
    print("Google Drive mounted successfully!")
except Exception as e:
    print(f"Error mounting Google Drive: {e}")
    exit()

DATA_PATH = '/content/drive/MyDrive/ReadifyRAG/'
```

Figure. 2.8 Environment Setup and Dependencies

The code begins by setting up the environment and importing essential tools. These include standard Python functionalities for system operations and logging, alongside specialized modules for interacting with Google Colab (like mounting Drive and accessing secure user data). It also imports components from the LangChain framework for document handling, text splitting—including advanced semantic chunking—and seamless integration with Pinecone for generating and managing embeddings within a vector store. The core Pinecone client library is also imported for direct service interaction.

This code snippet also ensures that Google Drive is mounted, allowing the script to access documents stored in a specified folder (/content/drive/MyDrive/ReadifyRAG/). This is crucial for environments like Google Colab where data often resides in cloud storage. The try-except block handles potential errors during the mounting process, providing robust error handling.

B) Pinecone Configuration and Pinecone Index Management

```
# Configure Pinecone
pinecone_api_key = userdata.get('pinecone_api_key')
os.environ["PINECONE_API_KEY"] = pinecone_api_key
pc = Pinecone(api_key=pinecone_api_key)

# Create Index
index_name = "readify"

if pc.has_index(index_name):
  pc.delete_index(name=index_name)

if not pc.has_index(index_name):
  pc.create_index_for_model(
      name=index_name,
      cloud="aws",
      region="us-east-1",
      embed={
        "model":"multilingual-e5-large",
        "field_map":{"text": "chunk_text"}
    }
  )
```

Figure. 2.9 Environment Setup and Dependencies

The Pinecone API key is retrieved securely using *userdata.get()*, which is a best practice for handling sensitive information in Colab. The API key is then set as an environment variable and used to initialize the Pinecone client.

This code also efficiently manages the Pinecone index. It first sets the index name to "*readify*" and checks for its existence. For development, any existing index with the same name is deleted to ensure a clean slate, though a more robust update strategy might be preferred in production. If the index doesn't exist, it's created as a serverless instance on AWS (us-east-1), configured to use the "multilingual-e5-large" embedding model, with text content mapped to a "*chunk_text*" field. Finally, it initializes an object to interact with this specific Pinecone index.

C) Embedding and Vector Store Initialization

```
index = pc.Index(index_name)

# Initialize embeddings with a specific model
embeddings = PineconeEmbeddings(model="multilingual-e5-large")

vector_store = PineconeVectorStore(
    index=index,
    embedding=embeddings
)
```

Figure. 2.10 Embedding and Vector Store Initialization

Here, the *PineconeEmbeddings* class is initialized with the "*multilingual-e5-large*" model. This object will be responsible for converting text into vector embeddings. Subsequently, a *PineconeVectorStore* is created, linking the Pinecone index with the chosen embedding model. This *vector_store* object provides a high-level interface for adding and querying documents.

D) Document Loading and Semantic Chunking

```
# Loading documents
loader = SimpleDirectoryReader(DATA_PATH)
raw_documents = loader.load()

text_splitter = SemanticChunker(embeddings, breakpoint_threshold_type="gradient")
chunks = text_splitter.split_documents(raw_documents)
print(len(chunks))

# Adding chunks to vector store
vector_store.add_documents(documents=chunks)
```

Figure. 2.11 Document Loading and Chunking

This section focuses on loading and preparing documents. It initializes a loader to read documents from the specified path, then loads them into memory. An important step here involves using semantic chunking, an advanced method that leverages the embedding model to identify natural breaks in the text based on semantic similarity, creating more coherent and meaningful chunks than traditional splitting. The loaded documents are then split into these semantically organized chunks, with the total count printed for monitoring.

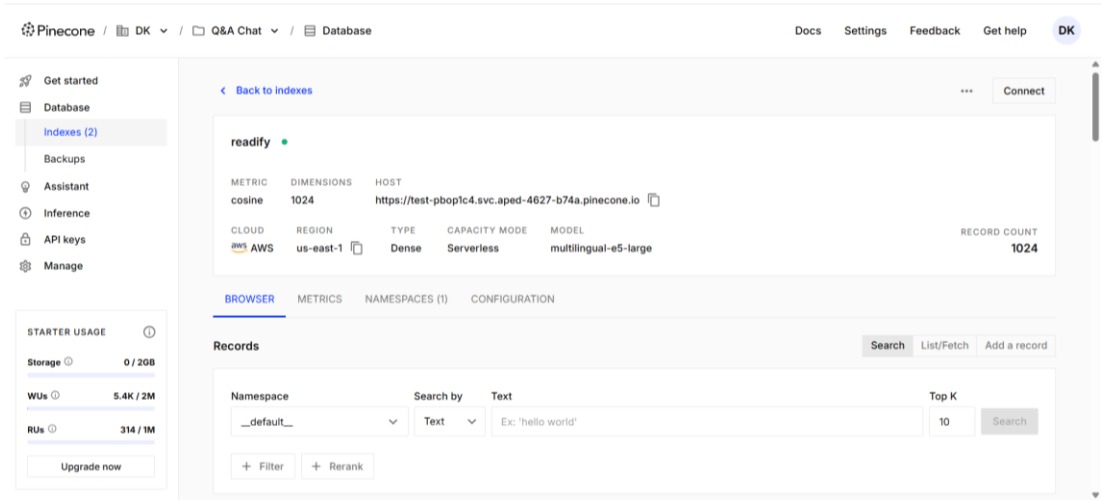


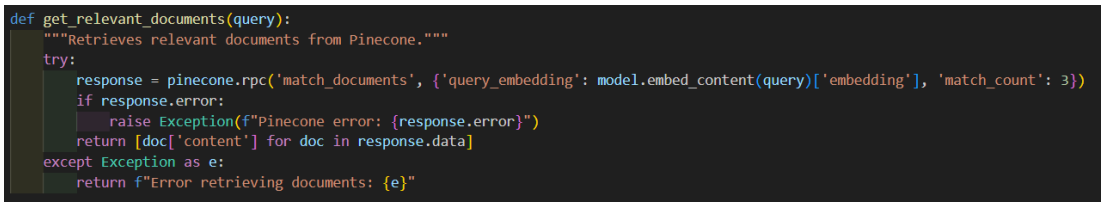
Figure. 2.12 Vector DB Dashboard

2.3.1.2 Corrective RAG Process

This system implements a Corrective Retrieval-Augmented Generation (C-RAG) system designed to provide relevant contextual information for generating questions. This is achieved through a multi-stage process involving document retrieval, relevance grading, and context consolidation.

A) Document Retrieval

The initial step involves retrieving potentially relevant documents from a Pinecone vector database. Upon receiving a user query, the *get_relevant_documents* function first embeds the query using the Gemini-Flash model's *embed_content* capability. This generates a numerical vector representation of the query's semantic meaning.



```
def get_relevant_documents(query):
    """Retrieves relevant documents from Pinecone."""
    try:
        response = pinecone.rpc('match_documents', {'query_embedding': model.embed_content(query)['embedding'], 'match_count': 3})
        if response.error:
            raise Exception(f"Pinecone error: {response.error}")
        return [doc['content'] for doc in response.data]
    except Exception as e:
        return f"Error retrieving documents: {e}"
```

Figure. 2.13 Document Retrieval

This query embedding is then utilized to perform a similarity search within the Pinecone database via an RPC call to *match_documents*. The system is configured to retrieve the top 3 most semantically similar documents, identified by their content. Error handling is included to catch and report issues during the Pinecone RPC call.

B) Relevance Grading

Following retrieval, the gathered documents undergo a relevance grading process to filter out irrelevant information. The *grade_documents* function iterates through each retrieved document and assesses its relevance to the original user query using the Gemini-Flash model.

A prompt is constructed that explicitly instructs Gemini to act as an "*expert grader*," evaluating if the document contains keywords or semantic meaning related to the question. The model is constrained to provide a binary "*yes*" or "*no*" response, indicating relevance. This programmatic grading ensures that only highly pertinent documents are considered for context generation, reducing noise and improving the quality of the subsequent output. A *GradeDocuments* Pydantic model is used to parse and validate the model's output, ensuring a standardized "*yes*" or "*no*" score.

```python
def grade_documents(query, documents):      You, 7 hours ago • Progress Presentation 02 …
    """Grades documents for relevance using Gemini."""
    graded_docs = []
    for document in documents:
        prompt = f"""
        You are an expert grader assessing relevance of a retrieved document to a user question.
        Follow these instructions for grading:
            - If the document contains keyword(s) or semantic meaning related to the question, grade it as relevant.
            - Your grade should be either 'yes' or 'no' to indicate whether the document is relevant to the question or not.

        Retrieved document:
        {document}

        User question:
        {query}
        """
        response = model.generate_content(prompt)
        try:
            grade = GradeDocuments(binary_score=response.text.strip().lower())
            if grade.binary_score == "yes":
                graded_docs.append(document)
        except Exception as e:
            print(f"Error grading document: {e}")
    return graded_docs
```

Figure. 2.14 Relevance Grading

The prompt's strength lies in its simplicity, clarity, and constraints. It avoids open-ended questions that could lead to verbose or irrelevant responses. By defining the role, providing explicit grading criteria, and structuring the input, the prompt effectively guides the Gemini model to perform a focused and accurate relevance assessment. This design is important for the downstream processing in the RAG pipeline, ensuring that only contextually appropriate documents are passed for question generation, ultimately contributing to higher quality and more relevant outputs.

C) Context Generation

The final stage, implemented in *generate_context_for_questions*, consolidates the graded, relevant documents into a comprehensive context string.

```python
def generate_context_for_questions(query):
    """Generates context for question generation using RAG."""
    graded_docs = get_graded_documents(query)
    if isinstance(graded_docs, str): #check if an error occured.
        return graded_docs
    return " ".join(graded_docs)
```

Figure. 2.15 Context Generation

2.3.1.3 CRAG-based Multi-Agent Workflow for Question Generation

This system employs a multi-agent workflow orchestrated to generate quiz content. This implementation leverages LLMs and web scraping techniques to achieve its goal.

A) Create Search Queries Tool

```python
def create_search_queries(topic: str, description: str) -> list[str]:
    """
    Creates a list of search queries based on a given topic and description using a CoT (Chain of Thought) approach.
    Maximum of 3 queries.

    Args:
        topic (str): The main topic for which to generate search queries.
        description (str): A brief description or context about the topic.

    Returns:
        list[str]: A list of generated search queries.
    """

    prompt = f""" ···

    response = model.generate_content(prompt)
    queries = [q.strip() for q in response.text.split('\n') if q.strip()]
    return queries[:3]
```

Figure. 2.16 Create Search Queries Tool Code

The *create_search_queries* function uses the gemini-2.0-flash model to generate a list of search queries. Its primary role is to interpret the user's need (expressed through topic and description) and formulate queries that are most likely to yield relevant information from a search engine.

```
agent > QuestionGenerationWorkflow.py > perform_web_search
32  def create_search_queries(topic: str, description: str) -> list[str]:
45      prompt = f"""
46      You are an expert at generating effective search queries.
47      Your goal is to create a maximum of 3 highly relevant search queries for a given topic and its description.
48      Think step-by-step to construct these queries.
49
50      Topic: {topic}
51      Description: {description}
52
53      Step 1: Understand the core need.
54      What is the central information being sought about the topic and description?
55      Identify key entities, actions, and desired outcomes.
56
57      Step 2: Brainstorm initial keywords.
58      Based on the core need, list potential keywords and phrases that directly relate to the topic and description.
59      Consider synonyms and related concepts.
60
61      Step 3: Formulate diverse query angles.
62      Think about different ways someone might search for this information. Consider:
63      - Broad overview queries
64      - Specific detail queries
65      - Problem/solution queries
66      - Comparative queries (if applicable)
67      - "How to" or "What is" type queries
68
69      Step 4: Refine and combine keywords.
70      Combine the brainstormed keywords into concise and effective search queries.
71      Use operators like "AND", quotes for exact phrases, etc., if beneficial (though for general search, natural language often works well
72      Ensure each query is distinct and targets a slightly different aspect if possible.
73
74      Step 5: Select the top 3 queries.
75      From your formulated queries, select the three most effective and diverse queries that are most likely to yield comprehensive results
76      If fewer than 3 strong queries are generated, that is acceptable.
77
78      Now, generate the search queries for the provided Topic and Description.
79      Provide only the list of queries, one per line, without any additional text or numbering.
80      """
```

Figure. 2.17 Prompt used in Create Search Queries Tool

The prompt employs several advanced prompting techniques to guide the language model's behavior effectively. It begins with **Role-based prompting** by assigning a clear persona*: "You are an expert at generating effective search queries."* This sets the stage for the model to adopt a specialized perspective, encouraging it to apply its internal knowledge as a query formulation expert.

The prompt then incorporates **Chain of Thought (CoT)** reasoning as its core technique, explicitly breaking down the task into a five-step process.

- Step 1 requires the model to understand the core need, promoting abstraction of the user's intent.
- Step 2 initiates broad brainstorming of initial keywords, fostering ideation.
- Step 3 prompts the creation of diverse query angles, helping to avoid redundancy and ensure comprehensive coverage of the topic.
- Step 4 focuses on refining and combining the keywords into effective queries.
- Step 5 involves selecting the top three queries, ensuring only the strongest are presented. This structured breakdown improves the logical flow and overall quality of the output by simplifying a complex task into manageable components.

Finally, a concise output instruction *"Provide only the list of queries, one per line, without any additional text or numbering"* ensures the output is structured in a clean, parsable format, facilitating seamless integration with downstream Python code without requiring further text processing.

B) Web Search Tool

The Web Search Tool, implemented as the `*perform_web_search_and_aggregate*` function, is designed to execute search queries generated by the `*create_search_queries*` tool and extract relevant textual content from the resulting web pages. This tool takes a list of search queries, performs web requests to Google Search, and systematically retrieves useful information from the top search results. It uses the *BeautifulSoup* for parsing HTML.

```python
def perform_web_search(queries: list[str]) -> str:
    """
    Performs web searches for a list of queries and aggregates the relevant text from the search results.

    Args:
        queries (list[str]): A list of search queries.

    Returns:
        str: A concatenated string of relevant text from the search results.
    """

    aggregated_results = []
    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36"
    }

    for query in queries:
        try:
            search_url = f"https://www.google.com/search?q={query}&num=5"
            response = requests.get(search_url, headers=headers)
            response.raise_for_status()

            soup = BeautifulSoup(response.content, "html.parser")
            search_results_divs = soup.find_all("div", class_="tF2Cxc")

            for result_div in search_results_divs:
                link = result_div.find("a")["href"]
                try:
                    page_response = requests.get(link, headers=headers)
                    page_response.raise_for_status()

                    page_soup = BeautifulSoup(page_response.content, "html.parser")
                    paragraphs = page_soup.find_all("p")
                    text_content = " ".join([p.get_text() for p in paragraphs])
                    if len(text_content) > 50: # Only add substantial content
                        aggregated_results.append(text_content)
                except requests.exceptions.RequestException as e:
                    print(f"Error fetching {link}: {e}")

        except requests.exceptions.RequestException as e:
            print(f"Web search failed for query '{query}': {e}")
        except Exception as e:
            print(f"An unexpected error occurred during web search for query '{query}': {e}")

    return " ".join(aggregated_results[:5])
```

Figure. 2.18 Web Search Tool Code

C) Research Agent

The Research Agent orchestrates the information gathering process. It acts as an intelligent coordinator, deciding when to generate new search queries and when to perform web searches, iteratively building a comprehensive understanding of the topic.

The *research_agent* function employs an iterative loop, leveraging the *create_search_queries* and *perform_web_search_and_aggregate* tools to gather information. It operates on a principle of self-correction and continuous improvement, assessing its current *research_data* and determining the next logical action. The agent can take up to *max_steps* (5) to refine its research.

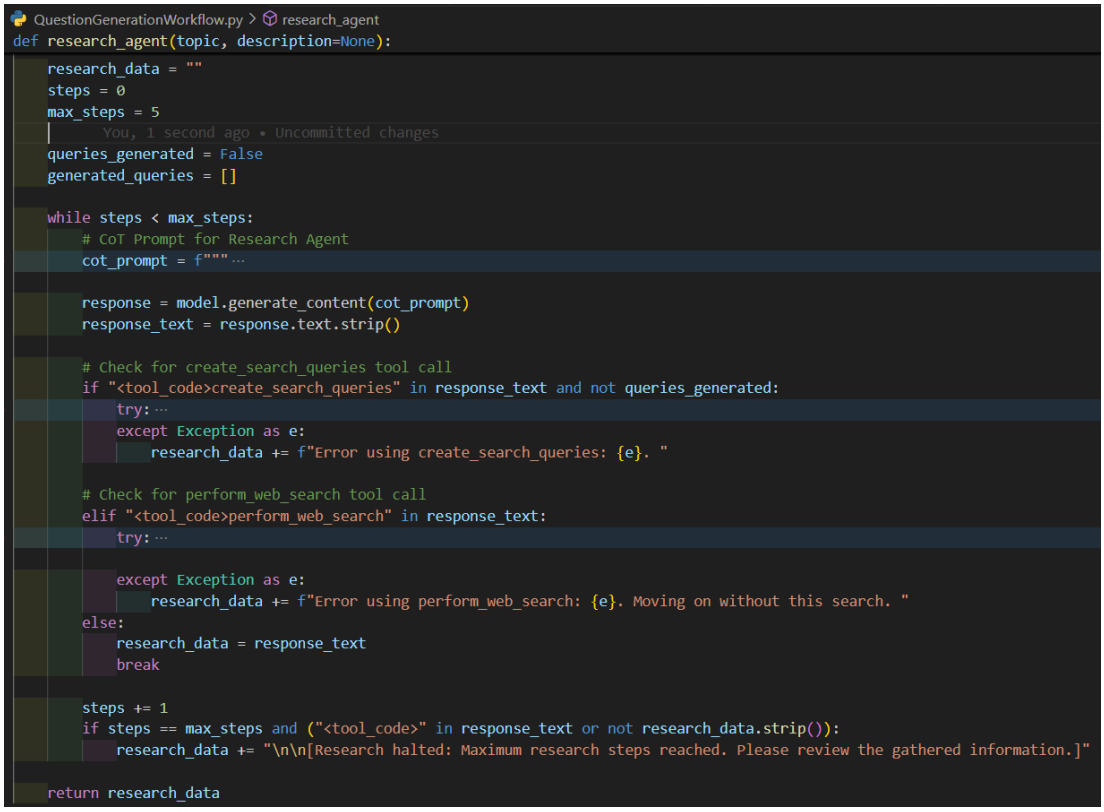
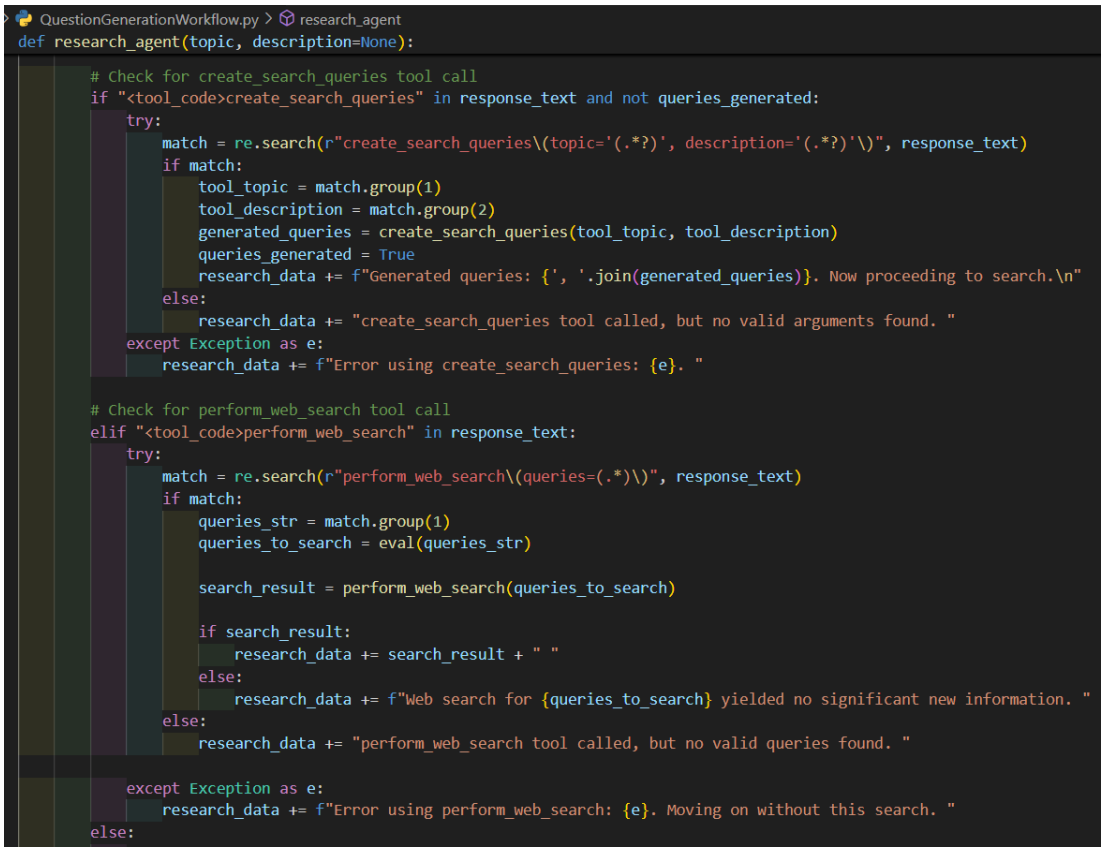


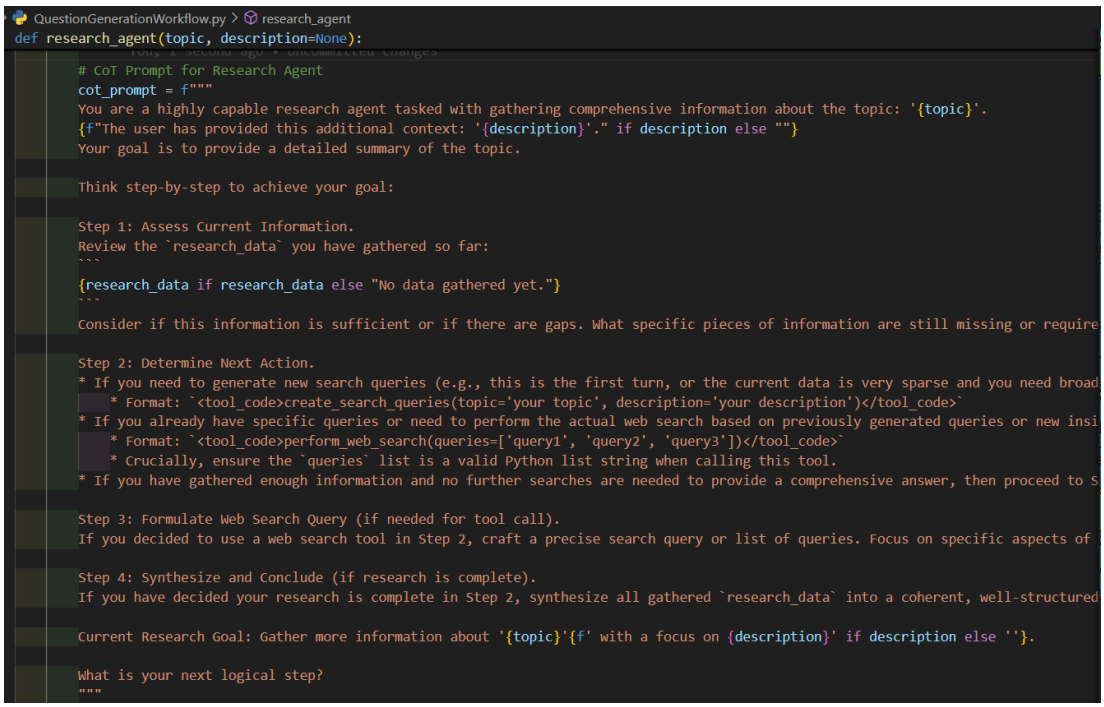
Figure. 2.19 Research Agent Code

Rather than performing actions directly, agent relies on the language model to decide when to invoke external tools through structured directives. These tool calls are detected and parsed using pattern recognition, after which the corresponding specialized functions are executed with the extracted parameters. The results are then incorporated back into the system's working context, allowing it to iteratively refine its understanding.



```
QuestionGenerationWorkflow.py > research_agent
def research_agent(topic, description=None):
    # Check for create_search_queries tool call
    if "<tool_code>create_search_queries" in response_text and not queries_generated:
        try:
            match = re.search(r"create_search_queries\(topic='(.*?)', description='(.*?)'\)", response_text)
            if match:
                tool_topic = match.group(1)
                tool_description = match.group(2)
                generated_queries = create_search_queries(tool_topic, tool_description)
                queries_generated = True
                research_data += f"Generated queries: {', '.join(generated_queries)}. Now proceeding to search.\n"
            else:
                research_data += "create_search_queries tool called, but no valid arguments found. "
        except Exception as e:
            research_data += f"Error using create_search_queries: {e}. "

    # Check for perform_web_search tool call
    elif "<tool_code>perform_web_search" in response_text:
        try:
            match = re.search(r"perform_web_search\(queries=(.*)\)", response_text)
            if match:
                queries_str = match.group(1)
                queries_to_search = eval(queries_str)

                search_result = perform_web_search(queries_to_search)

                if search_result:
                    research_data += search_result + " "
                else:
                    research_data += f"Web search for {queries_to_search} yielded no significant new information. "
            else:
                research_data += "perform_web_search tool called, but no valid queries found. "

        except Exception as e:
            research_data += f"Error using perform_web_search: {e}. Moving on without this search. "
    else:
```

Figure. 2.20 Tool Integration



```
QuestionGenerationWorkflow.py > research_agent
def research_agent(topic, description=None):
    # CoT Prompt for Research Agent
    cot_prompt = f"""
    You are a highly capable research agent tasked with gathering comprehensive information about the topic: '{topic}'.
    {f"The user has provided this additional context: '{description}'." if description else ""}
    Your goal is to provide a detailed summary of the topic.

    Think step-by-step to achieve your goal:

    Step 1: Assess Current Information.
    Review the `research_data` you have gathered so far:
    ```
    {research_data if research_data else "No data gathered yet."}
    ```
    Consider if this information is sufficient or if there are gaps. What specific pieces of information are still missing or require

    Step 2: Determine Next Action.
    * If you need to generate new search queries (e.g., this is the first turn, or the current data is very sparse and you need broad
        * Format: `<tool_code>create_search_queries(topic='your topic', description='your description')</tool_code>`
    * If you already have specific queries or need to perform the actual web search based on previously generated queries or new insi
        * Format: `<tool_code>perform_web_search(queries=['query1', 'query2', 'query3'])</tool_code>`
        * Crucially, ensure the `queries` list is a valid Python list string when calling this tool.
    * If you have gathered enough information and no further searches are needed to provide a comprehensive answer, then proceed to S

    Step 3: Formulate Web Search Query (if needed for tool call).
    If you decided to use a web search tool in Step 2, craft a precise search query or list of queries. Focus on specific aspects of

    Step 4: Synthesize and Conclude (if research is complete).
    If you have decided your research is complete in Step 2, synthesize all gathered `research_data` into a coherent, well-structured

    Current Research Goal: Gather more information about '{topic}'{f' with a focus on {description}' if description else ''}.

    What is your next logical step?
    """
```

Figure. 2.21 Research Agent Prompt

Research Agent prompt is an example of advanced prompting techniques, particularly focusing on control flow and dynamic decision-making.

- **Role-based Prompting:** "*You are a highly capable research agent tasked with gathering comprehensive information...*" clearly defines the LLM's responsibility and persona, setting the context for its actions.
- **Chain of Thought (CoT):** The prompt uses a sophisticated CoT structure with four distinct steps:
    - **Step 1. Assess Current Information:** The LLM is provided with the *research_data* accumulated so far. This step requires LLM to reflect on its current knowledge state, identify gaps, and determine what additional information is needed.
    - **Step 2. Determine Next Action:** Based on its assessment, the LLM is instructed to decide whether to call *create_search_queries, perform_web_search_and_aggregate*, or conclude the research. This step introduces conditional logic into the LLM's decision-making process.
    - **Step 3. Formulate Web Search Query (if needed for tool call):** This acts as an internal planning step, ensuring that if a search tool is to be called, the LLM first thinks about the precise queries required.
    - **Step 4. Synthesize and Conclude (if research is complete):** This step outlines the final desired behavior: if research is complete, synthesize the findings into a coherent summary, without any tool calls or "*Thinking Process*" output.
- **Tool Use Pattern:** This is a core prompting technique here. The prompt explicitly defines the syntax for calling external tools. This acts as a clear instruction to the LLM on how to interact with the environment. The LLM's output is parsed by the Python code to trigger the actual function calls. This allows the LLM to act as a planner, delegating execution to specialized external functions.
- **State Management / Contextual Awareness:** The *research_data* is explicitly passed back into the prompt in each. This provides the LLM with the current state of its knowledge, allowing it to make informed decisions about whether

to continue research, what to search for, or when to conclude. This mimics a form of short-term memory within the research process.

- **Zero-Shot Prompting:** While not providing explicit examples of the entire multi-turn research process (which would be complex), the prompt for tool usage is very direct, acting like a zero-shot instruction for how to format tool calls. The overall structure implicitly guides the LLM without requiring extensive prior examples for this specific iterative research loop.

D) Question Generation Agent

The Question Generation Agent is responsible for transforming the *research_data* into structured quiz questions, paragraphs, and model answers, tailored to a specified CEFR language proficiency level.

This agent receives the aggregated *research_data*, the original topic, an optional description, and a *cefr_level*. A key aspect is its integration with Corrective RAG to enhance the factual basis of the questions. It then uses the LLM to formulate questions and answers according to strict CEFR guidelines and outputs them in a predefined JSON format.

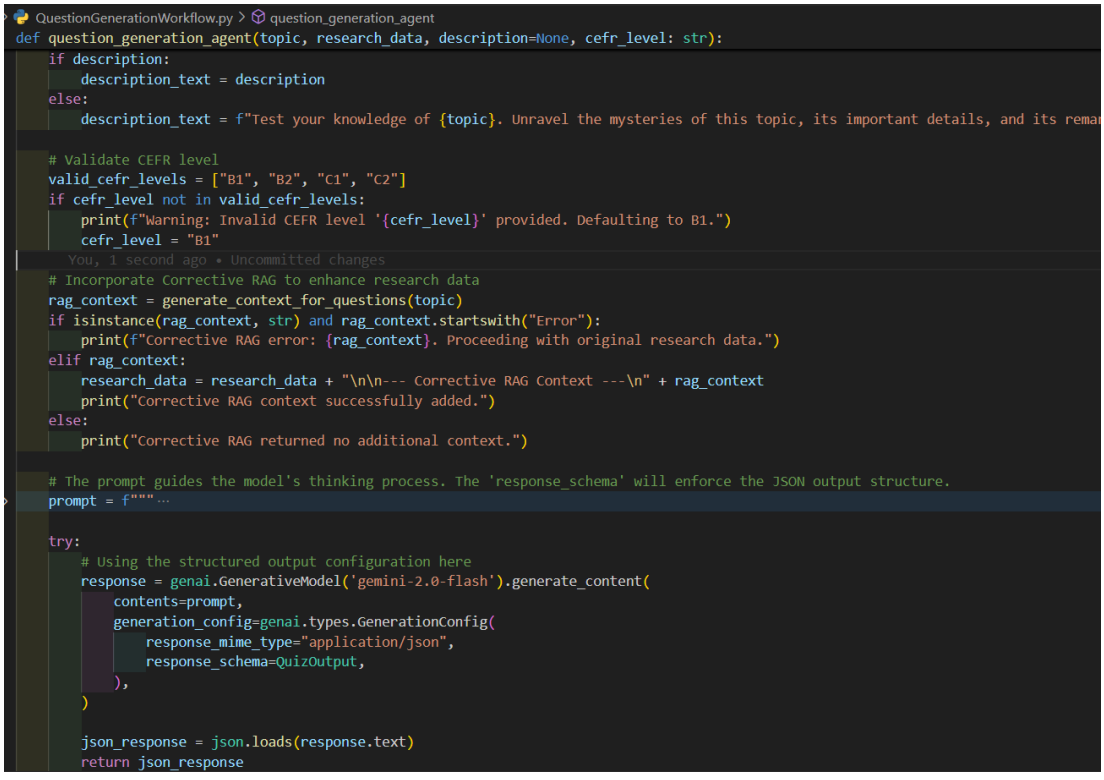


Figure. 2.22 Question Generation Agent Code

```
# The prompt guides the model's thinking process. The 'response_schema' will enforce the JSON output structure.
prompt = f"""
You are an expert question generation agent. Your task is to create high-quality, essay-type questions based on provided research data.
The questions should challenge the user's understanding and encourage detailed answers.

Crucially, tailor the language, vocabulary, complexity, and depth of the questions and model answers to a CEFR level of {cefr_level}.

CEFR Level Guidelines for Question Generation:
* B1 (Intermediate): Focus on common, practical topics.
                     Questions should be straightforward, requiring direct answers.
                     Vocabulary should be familiar.
                     Answers should be clear and concise, summarizing main points.
* B2 (Upper Intermediate): Introduce more complex topics and require opinions, justifications, and more detailed explanations.
                           Vocabulary should be broader, allowing for nuanced expression.
                           Answers should be well-developed, demonstrating good analytical skills.
* C1 (Advanced): Address abstract and complex topics. Questions should require synthesis, evaluation, and critical thinking.
                 Vocabulary should be extensive and idiomatic.
                 Answers should be articulate, coherent, and demonstrate sophisticated understanding.
* C2 (Proficiency): Cover highly complex and specialized topics.
                    Questions should demand nuanced understanding, sophisticated argumentation, and the ability to handle abstract and hypothetical situations.
                    Vocabulary should be precise and comprehensive, including academic and technical terms where appropriate.
                    Answers should be highly detailed, demonstrating mastery of the subject matter and language.

Topic: {topic}
Description for Questions: {description_text}
Desired CEFR Level: {cefr_level}

Research Data:
```
{research_data}
```

Think step-by-step to generate the questions and answers:

Step 1: Review and Understand the Research Data in Context of CEFR Level.
Carefully read through the `Research Data` provided. Identify the main themes, key concepts, important facts, and significant details related to `{topic}`.
As you read, consider how this information can be framed into questions appropriate for a {cefr_level} learner.

Step 2: Identify Potential Question Areas for the Target CEFR Level.
Based on your understanding, brainstorm several distinct areas within the research data that could form the basis of a question.
Ensure these areas allow for questions and answers that align with the linguistic and cognitive demands of a {cefr_level} learner. Aim for variety.

Step 3: Draft Questions and Supporting Paragraphs Tailored to CEFR Level.
For each potential question area, draft a concise, clear essay-type question.
* Word Choice: Use vocabulary and grammatical structures appropriate for {cefr_level}.
* Complexity: Adjust sentence length and complexity to match {cefr_level}.
* Inference/Analysis: For higher levels (C1, C2), questions can require more inference, synthesis, or critical analysis.
                      While for lower levels (B1, B2), they should be more direct.
Then, extract or synthesize a relevant `paragraph` from the `Research Data` that directly relates to the question and provides context.
The language of this paragraph should also be adjusted to roughly match the {cefr_level} if possible, while retaining factual accuracy.

Step 4: Formulate Detailed Model Answers Tailored to CEFR Level.
For each drafted question, write a comprehensive and accurate `model_answer` based *solely* on the provided `Research Data`.
* Vocabulary and Grammar: Use vocabulary and grammatical structures appropriate for a {cefr_level} response.
* Depth and Detail: Adjust the level of detail and argumentation to what is expected of a {cefr_level} learner's answer.
* Coherence: Ensure the answer is well-organized and coherent.

Step 5: Ensure Correct Quantity.
Make sure you generate exactly {num_questions} questions.

Step 6: Finalize for JSON Output.
Organize all generated questions, paragraphs, and answers according to the `QuizOutput` schema provided as a response schema.
Ensure all fields are correctly populated. Do not include any extra text, comments, or markdown outside of the JSON structure.
"""
```

Figure. 2.23 Question Generation Prompt

The prompt for the *question_generation_agent* is highly advanced, focusing on content generation constraints, quality, and structured output.

- **Role-based Prompting:** "*You are an expert question generation agent.*" defines the LLM's role, setting the expectation for high-quality questions.
- **Chain of Thought (CoT):** A six-step CoT guides the LLM through the entire question generation process:
    - Step 1: Review and Understand the Research Data in Context of CEFR Level. This initial step ensures the LLM processes the input carefully and understands the dual constraints of factual content and linguistic complexity.
    - Step 2: Identify Potential Question Areas for the Target CEFR Level. This encourages the model to brainstorm diverse angles within the research data, leading to varied questions.

- Step 3: Draft Questions and Supporting Paragraphs Tailored to CEFR Level. This step directly addresses the core task, emphasizing tailoring language, complexity, and requiring contextual paragraphs.
- Step 4: Formulate Detailed Model Answers Tailored to CEFR Level. This ensures the answers are comprehensive, accurate, and aligned with the specified CEFR level's expected depth and vocabulary. The instruction "*based solely on the provided Research Data*" is a critical grounding technique to prevent factual errors or hallucinations.
- Step 5: Ensure Correct Quantity. This is a direct instruction to meet a specific numerical requirement for questions.
- Step 6: Finalize for JSON Output. This step is crucial for reinforcing the desired output format. This detailed CoT ensures that the LLM systematically constructs each part of the quiz content according to all requirements.

- **Constraint-based Prompting:** The prompt explicitly and repeatedly emphasizes the cefr_level. This is a powerful form of constraint: "*Crucially, tailor the language, vocabulary, complexity, and depth of the questions and model answers to a CEFR level of {cefr_level}.*" This is a global constraint.
- **Zero-Shot Learning with Detailed Guidelines:** The "*CEFR Level Guidelines for Question Generation*" section acts as a very detailed instruction set. Instead of providing full input-output examples, it provides extensive descriptions of the characteristics expected for each CEFR level (B1, B2, C1, C2) in terms of topic focus, question requirements (direct vs. critical thinking), vocabulary, and answer quality. This enables the LLM to generalize CEFR-level appropriate language and reasoning across different topics without needing a specific example for each topic.
- **Structured Output:** This is a critical technical implementation detail. While the prompt includes "*Step 6: Finalize for JSON Output... according to the QuizOutput schema provided as a response schema,*" the actual enforcement is done via the *genai.types.GenerationConfig* in the API call:

E) Multi Agent Workflow

The *multi_agent_workflow* function serves as the central orchestrator, coordinating the execution of the Research Agent and the Question Generation Agent to produce a complete quiz.

```python
def multi_agent_workflow(topic, description=None, cefr_level: str):
    """
    Orchestrates the sequential multi-agent workflow for research and question generation.
    """

    research_data = research_agent(topic, description)

    if "failed" in research_data or "[Research halted" in research_data:
        print("Research agent encountered an issue or halted prematurely.")
        return {"error": "Research failed or halted prematurely", "details": research_data}

    questions_and_answers = question_generation_agent(topic, research_data, description, cefr_level)
    return questions_and_answers
```

Figure. 2.24 Multi Agent Workflow Code

This top-level function integrates the two main components of the system: research and question generation. It takes the topic, description, and *cefr_level* as initial inputs. It first calls the *research_agent* to gather all necessary information. Upon successful completion of the research phase, it then passes the collected *research_data* to the *question_generation_agent* to create the final quiz content. It also includes basic error handling for the research phase.

F) Frontend Implementation

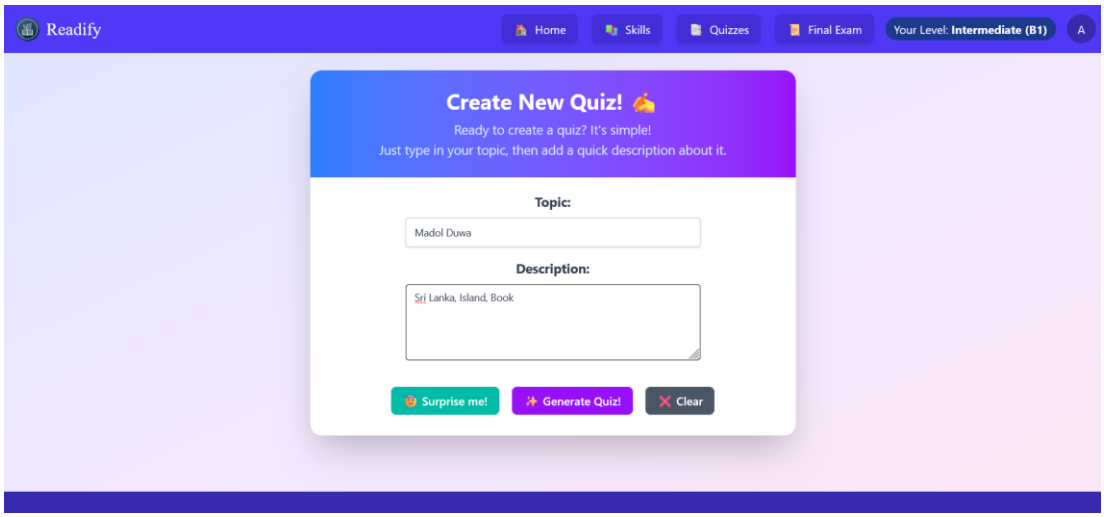


Figure. 2.25 Insert Details to Generate New Quiz

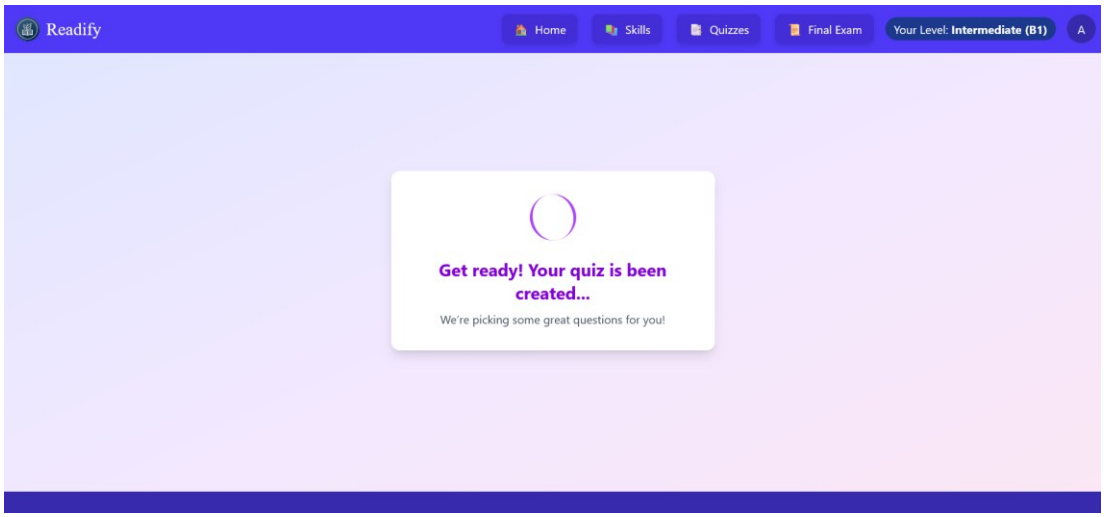


Figure. 2.26 Generating Quiz

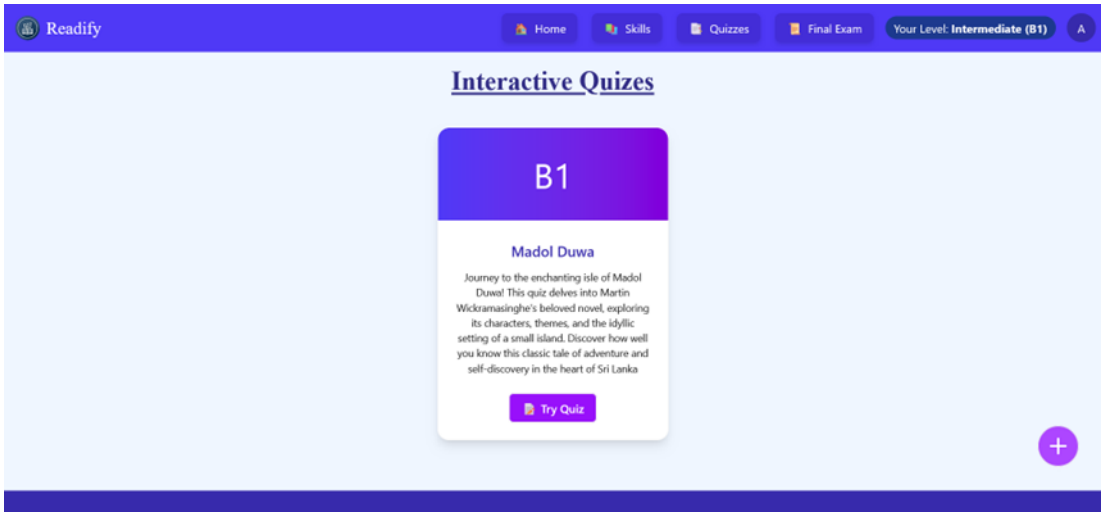


Figure. 2.27 Generated Quiz

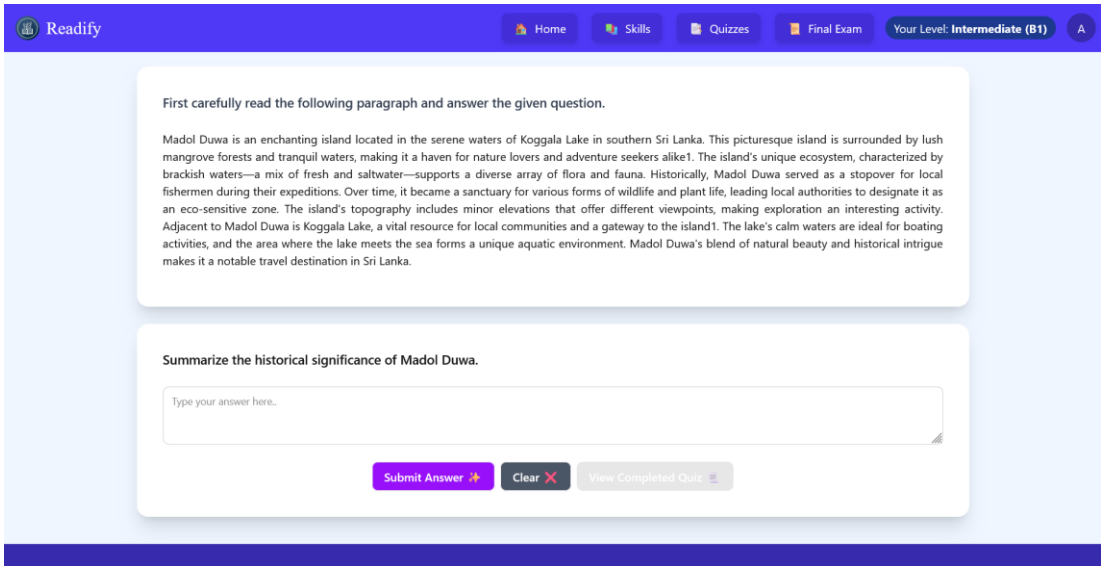


Figure. 2.29 Generated Question

G) Database

q... i.	paragraph text	question text	model_answer text
66	Madol Duwa holds a special place in Sri Lankan popular culture, largely due to the	What is the Madol Doova book about?	Adventures of Upali Giniwella and his
67	Madol Duwa holds a special place in Sri Lankan popular culture, largely due to the	How does Madol Doova book contribute to Madol Duv	The novel's depiction of adventures o
68	Madol Duwa holds a special place in Sri Lankan popular culture, largely due to the	Analyze the impact of Martin Wickramasinghe's novel	It made Madol Duwa a symbol of hop
69	Madol Duwa is an enchanting island located in the serene waters of Koggala Lake i	What unique ecosystem characteristics does Madol Du	It has brackish waters supporting dive
70	Madol Duwa is an enchanting island located in the serene waters of Koggala Lake i	Suggest a title for the first paragraph about Madol Duv	Madol Duwa: A Natural Haven in Kog
71	Madol Duwa is an enchanting island located in the serene waters of Koggala Lake i	Summarize the historical significance of Madol Duwa.	It was a stopover for fishermen and la

Figure. 2.30 Generated Questions in Database

H) CEFR level of generated paragraphs

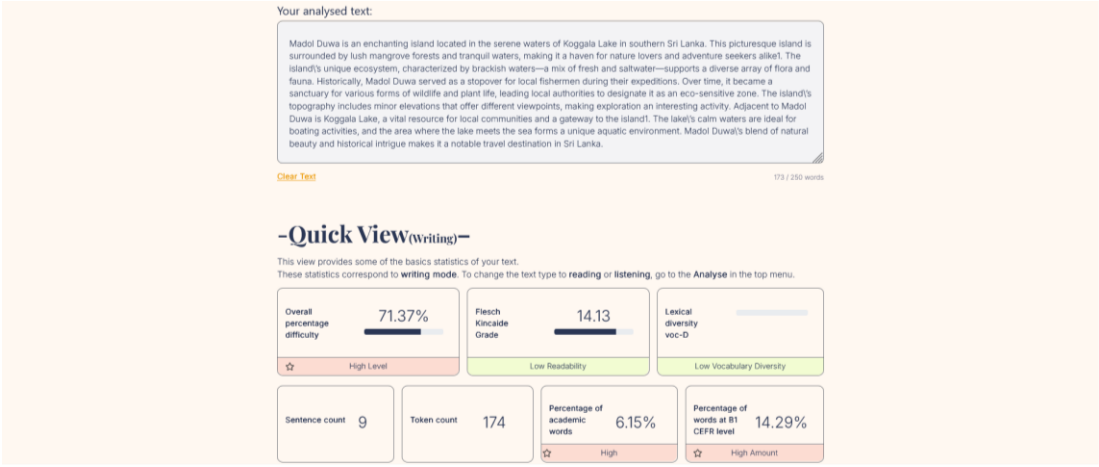


Figure. 2.31 Check CEFR level of the First Paragraph using Text Inspector [24]

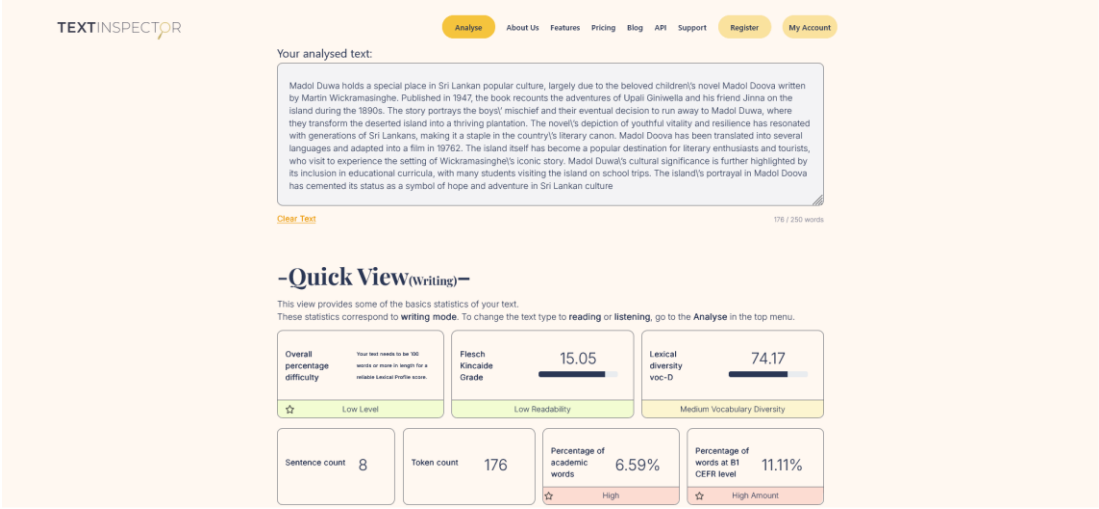


Figure. 2.32 Check CEFR level of the Second Paragraph using Text Inspector

2.3.1.4 Multi-Model Agentic Workflow with Confidence-Based Fallback and Arbitration Mechanism for Answer Evaluation and Feedback Generation

This workflow is a multi-agent system designed for evaluating student answers. The system employs a two-tiered evaluation approach, utilizing a primary evaluation agent and a secondary evaluation agent with an arbitration mechanism, to ensure reliable feedback.

A) Primary Evaluation Agent

The Primary Evaluation agent serves as the initial layer of student answer validation. Its core functionality revolves around providing a comprehensive assessment of a student's response to a given question, considering a provided paragraph and a model answer. It generates a score, determines correctness, offers detailed feedback, and rates its own confidence in the evaluation.

The Gemini-Flash model is chosen as the primary model due to its advanced capabilities in understanding complex instructions, generating nuanced feedback, and adhering to strict output formats, specifically JSON. Gemini's strong performance in natural language understanding and generation tasks makes it highly suitable for evaluating textual answers and providing human-like feedback.

```python
def primary_evaluation_agent(paragraph, question, user_answer, model_answer):
    """Validates student answer, provides feedback, and rates confidence using Gemini."""

    prompt = f""" ...

    try:
        response = client.models.generate_content(
            model="gemini-2.0-flash",
            contents=prompt,
            config={
                "response_mime_type": "application/json",
                "response_schema": Feedback,
            },
        )
        parsed_feedback: Feedback = response.parsed      You, 25 minutes ago • Final Vers

        return {
            "score": parsed_feedback.score,
            "correctness": parsed_feedback.correctness,
            "feedback": parsed_feedback.feedback_message,
            "confidence": parsed_feedback.confidence
        }
    except Exception as e:
        return {"error": f"Answer validation failed: {e}", "confidence": 0}
```

Figure. 2.33 Primary Evaluation Agent Code

```
prompt = f"""
You are a helpful and kind teacher.
1. Analyze the Question and Paragraph: First, fully understand what the question is asking and
                                       what information the provided paragraph offers to answer it.
2. Evaluate User Answer: Use the created rubric to evaluate the student's answer. Also consider the given model answer.
3. Assign Score: Assign a score out of 5 to the student's answer based on your evaluation.
4. Determine Correctness: Clearly state whether the student's answer is Correct or Incorrect based on the assigned score.
                          A score of 4 or higher is considered "Correct."
5. Compare User Answer to Model Answer: Next, carefully compare the user's response directly against the given model answer.
6. Identify Gaps: Determine precisely what key words, key facts, details, or concepts are present in the model answer and
                  the paragraph but are missing, unclear, or inaccurately represented in the student's answer.
7. Formulate Feedback:
    - If the student's answer is identical or highly similar to the model answer (and thus scores high), provide warm praise.
    - Otherwise, mention that the answer is not matching, offer gentle encouragement, and helpful constructive feedback.
    - Clearly state what specific information was lacking or could be improved.
    - Highlight any correct elements or good efforts made by the student.
    - Suggest concrete ways to enhance their understanding or answer next time.
8. Adhere to Constraints: Ensure the final feedback is under 150 words and contains NO symbols or markdown characters.
                          Feedback should be clear and easy to understand.
9. Rate Confidence: Rate your confidence in this evaluation and feedback on a scale of 1 to 10,
                    where 1 is very low confidence and 10 is very high confidence.
10. Final Feedback Output: Provide the assigned score as a number, your determination of correctness (Correct/Incorrect),
                           your formulated feedback message, and your confidence rating.

Paragraph: {paragraph}
Question: {question}
User Answer: {user_answer}
Model Answer: {model_answer}
"""
```

Figure. 2.34 Primary Evaluation Agent Prompt

The prompt for the primary_evaluation_agent is meticulously crafted to guide the Gemini model through a structured evaluation process. Several techniques are employed to enhance the quality and consistency of the feedback:

- **Role-Playing:** The prompt begins with "*You are a helpful and kind teacher.*" This sets a clear persona for the AI, encouraging it to adopt a supportive and constructive tone in its feedback.
- **Chain of Thought (CoT):** The prompt breaks down the evaluation process into 10 distinct, numbered steps. These steps guide the model through a logical sequence of operations:
    - **Analyze the Question and Paragraph:** Ensures the model grasps the context.
    - **Evaluate User Answer:** Directs the model to assess the student's response using implicit rubric and the model answer.
    - **Assign Score:** Requires a numerical score out of 5, providing a quantifiable measure.
    - **Determine Correctness:** Defines a clear threshold (4 or higher) for "*Correct*" standardizing the binary classification.
    - **Compare User Answer to Model Answer:** Emphasizes direct comparison for identifying discrepancies.
    - **Identify Gaps:** Directs the model to pinpoint specific missing or inaccurate information.

- **Formulate Feedback:** Provides detailed guidelines for crafting constructive feedback, including praise for good answers and specific suggestions for improvement. This section is critical for generating actionable feedback.
- **Adhere to Constraints:** Imposes a strict word limit (under 150 words) and formatting constraints (NO symbols or markdown characters). This ensures the feedback is concise, easy to read, and consistent.
- **Rate Confidence:** Asks the model to self-assess its confidence (1-10). This is a crucial technique for implementing the fallback mechanism to the Secondary Evaluation agent. Low confidence signals potential ambiguity or difficulty in evaluation, triggering a re-evaluation by other models.
- **Final Feedback Output:** Specifies the exact JSON structure for the output, reinforcing the *response_schema* and ensuring parseability.

B) Secondary Evaluation Agent

The Secondary Evaluation agent acts as an important fallback mechanism, engaged when the primary agent expresses low confidence in its assessment. It leverages a multi-model approach to provide a more validated and comprehensive evaluation.

The Secondary Evaluation agent operates by orchestrating evaluations from multiple independent large language models and then synthesizing their feedback through an "*arbiter*" model.

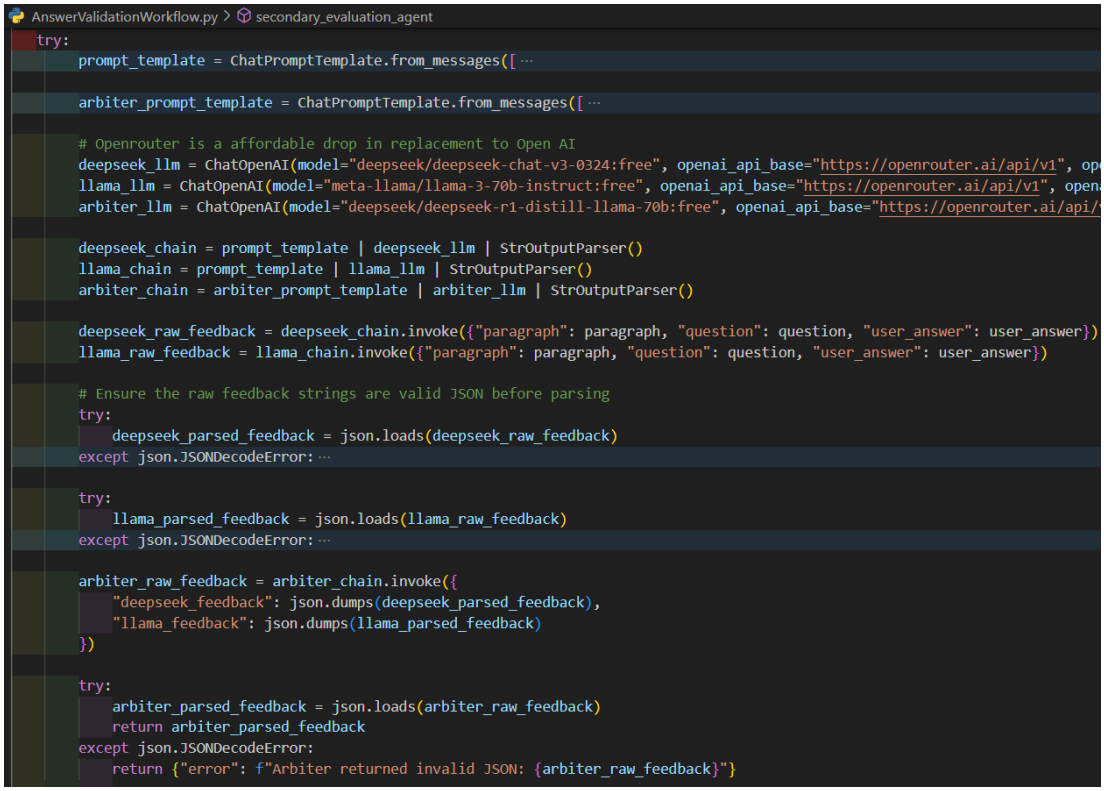


Figure. 2.35 Secondary Evaluation Agent Code

This sequence of operations involves parallel evaluation, where the same paragraph, question, user answer, and model answer are simultaneously sent to two distinct large language models, *Deepseek-V3* and *Llama-3-70b*, utilizing the OpenRouter API. Following this, independent feedback generation occurs as each model produces its own evaluation based on a shared prompt. Subsequently, an arbitration phase takes place; the individual feedback outputs from Deepseek and Llama are then fed into a third model, the *arbiter_llm (deepseek-r1-distill-llama-70b)*, which synthesizes these into a single, refined feedback. Finally, the system ensures robustness through built-in error handling for JSON parsing, which allows the process to still derive a sensible output even if one of the models returns malformed JSON.

The selection of specific models for the secondary evaluation agents is strategic, aiming for diversity in model architecture and training data to mitigate biases and enhance the robustness of the combined evaluation.

Deepseek-V3: Offers a balance of capability and efficiency. The DeepSeek-V3 model is a 671B total parameter Mixture-of-Experts (MoE) model with 37B activated for each token. Being a free tier model via OpenRouter suggests it's a cost-effective option for parallel processing while still providing competent linguistic understanding and generation.

Llama-3-70B: As a powerful model, Llama-3-70b provides a different perspective and potentially deeper reasoning capabilities. Its instruction-tuned nature makes it adept at following complex evaluation guidelines.

Deepseek-r1-distill-llama-70b for the Arbiter: This specific model is selected as the arbiter for its likely ability to perform complex reasoning and synthesis. The "distill-llama-70b" part suggests it is a distilled version of a Llama-70b model, potentially offering a good balance of performance and inference speed for the arbitration task. Its role is not just to combine but to intelligently synthesize, identify the most accurate elements, and resolve discrepancies between the two initial evaluations.

```
prompt_template = ChatPromptTemplate.from_messages([
            ("human", """
            You are a helpful and kind teacher tasked with evaluating a student's answer based on a given paragraph and question.
            Task: Provide a comprehensive evaluation of the student's answer.
            Step 1. Understand the Core Question: First, analyze the question to identify the main topic and what specific information is being sought.
            Step 2. Extract Relevant Information from Paragraph: Carefully read the provided paragraph and identify all facts, concepts,
                    and details that are directly relevant to answering the question.
            Step 3. Evaluate User Answer against Question and Paragraph: Compare the user's answer against both the question's requirements and
                    the information found in the paragraph.
            Step 4. Identify Strengths and Weaknesses: Determine what parts of the user's answer are accurate, complete, or well-articulated (strengths).
                    Also, identify what information is missing, incorrect, vague, or misinterpretated (weaknesses).
            Step 5. Assign a Score (1-5): Based on the completeness and accuracy, assign a numerical score out of 5.
                    A score of 4 or higher indicates a "Correct" answer.
            Step 6. Determine Correctness: Explicitly state if the answer is "Correct" or "Incorrect" based on the assigned score.
            Step 7. Construct Detailed Feedback:
                - If the score is high (4 or 5), provide positive reinforcement, briefly mentioning why the answer is good.
                - If the score is lower, provide constructive criticism. Clearly state what specific information was lacking or incorrect.
                  Offer gentle suggestions for improvement.
                - Ensure the feedback is concise, under 150 words, and free of symbols or markdown characters.
            Step 8. Format Output as JSON: Present your final evaluation in a JSON object with the following keys:
                    score (integer), correctness (string: "Correct" or "Incorrect"), and feedback_message (string).

            Paragraph: {paragraph}
            Question: {question}
            User Answer: {user_answer}
            Model Answer: {model_answer}
            """)
        ])
```

Figure. 2.36 Secondary Evaluation Agent Prompt

This prompt shares many objectives with the primary agent's prompt, but it's specifically tailored to guide the secondary evaluation models—Deepseek-V3 and Llama-3-70B in performing a comprehensive evaluation.

- **Role-Playing:** The instruction "*You are a helpful and kind teacher...*" establishes a consistent persona, ensuring that the feedback generated by these models maintains a supportive and constructive tone.
- **Step-by-Step Instructions:** The 8 steps are designed to guide the model through a structured evaluation process. It's similar to the primary agent, but with a slight variation in emphasis:
    - **Understand the Core Question:** Focuses on the question's main topic and specific information sought.
    - **Extract Relevant Information from Paragraph:** Emphasizes careful reading and identifying relevant facts.
    - **Evaluate User Answer against Question and Paragraph:** This step explicitly instructs the model to compare the student's answer against both the question's requirements and the information derived from the paragraph.
    - **Assign a Score (1-5):** A numerical score provides a quantifiable measure of performance, with a clear threshold of 4 or higher defined for a "Correct" answer, standardizing the grading.
    - Determine Correctness: This mandates an explicit binary classification ("*Correct*" or "*Incorrect*") based on the assigned score, ensuring clarity in the overall assessment.
    - **Construct Detailed Feedback:** This provides specific guidelines for crafting constructive criticism. It emphasizes offering positive reinforcement for good answers, clearly stating what was lacking for lower scores, and offering gentle suggestions for improvement.
    - **Constraints:** Enforces brevity (under 150 words) and prohibits symbols or markdown, ensuring the feedback is easily digestible and consistently formatted.
    - **Format Output as JSON:** Explicitly requests the final output in a JSON object with predefined keys (*score*, *correctness*, *feedback_message*). This structured output is vital for the automated parsing and subsequent processing by the arbiter agent.

C) Frontend Implementation

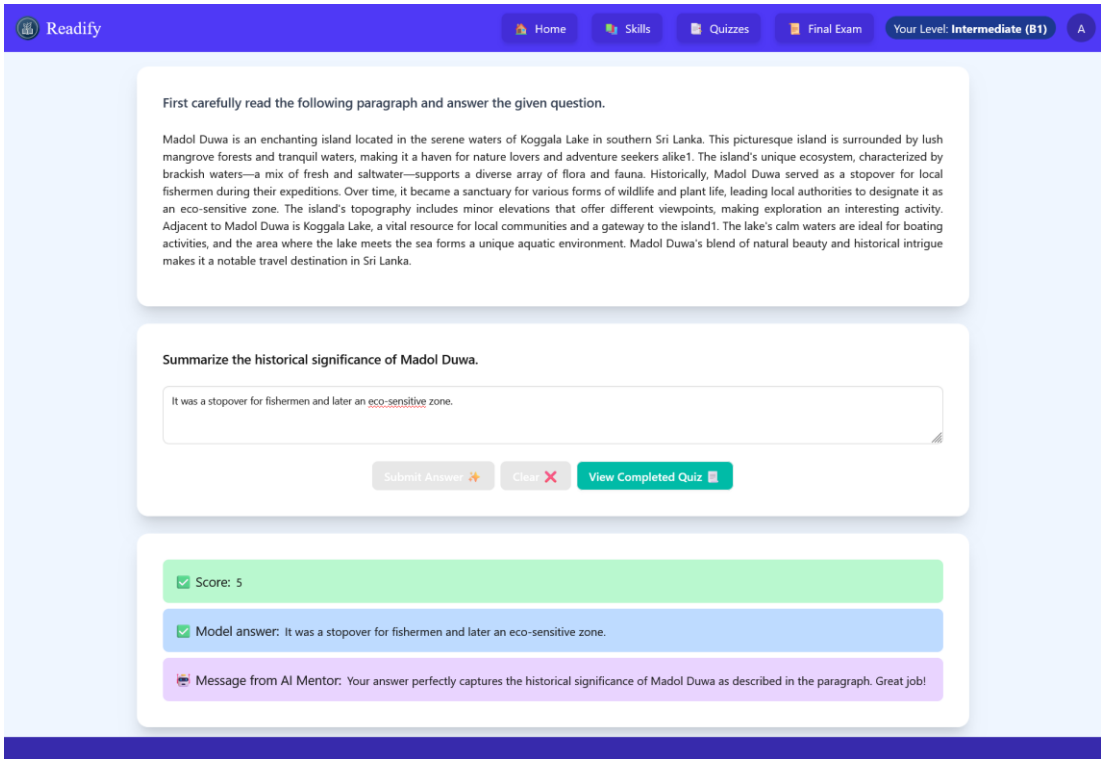


Figure. 2.37 Frontend Implementation

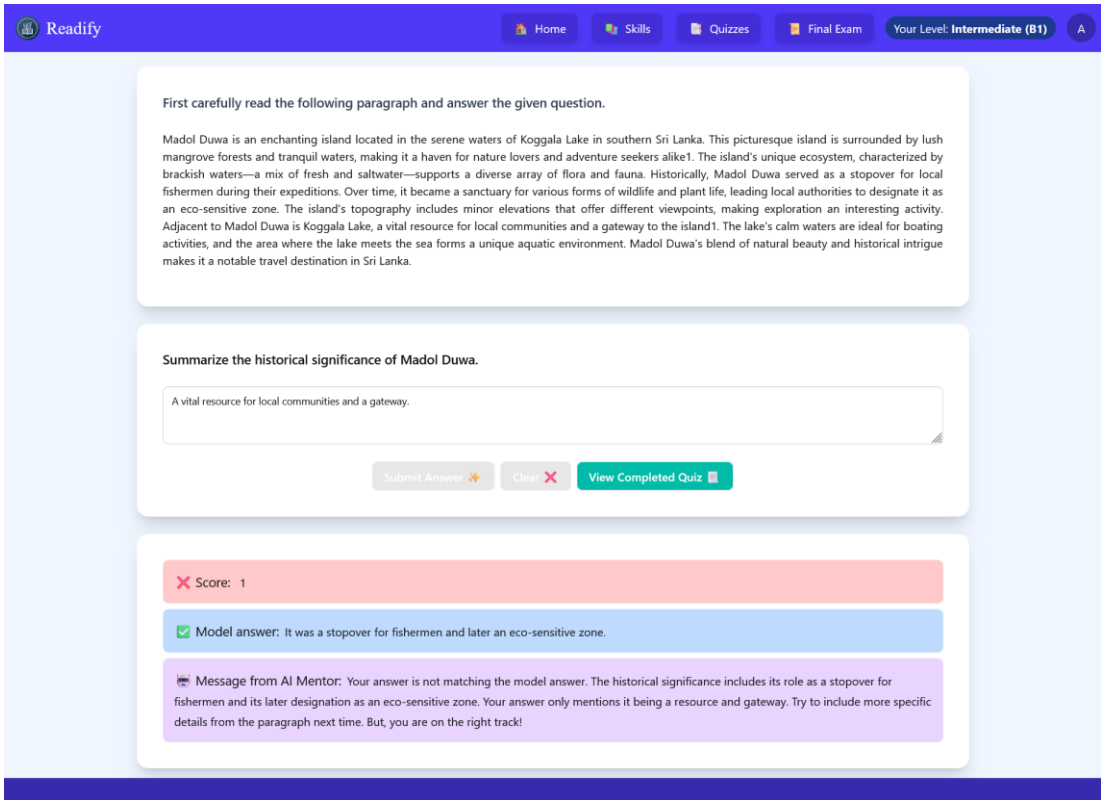


Figure. 2.38 Frontend Implementation

2.3.1.5 CEFR-Based Level Assessment and Dynamic Progression

Upon registration, new users are required to complete an initial placement quiz designed to evaluate their existing level of language proficiency. This assessment includes a range of questions aligned with various levels of the Common European Framework of Reference for Languages (CEFR).

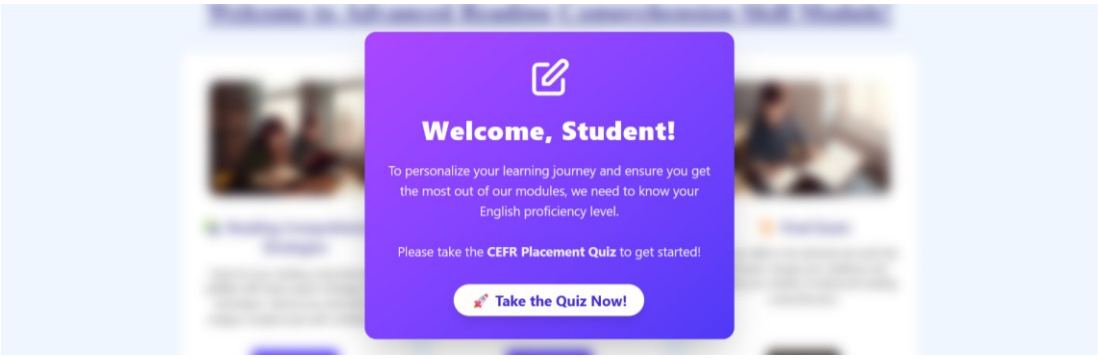


Figure. 2.39 Initial Welcome Screen for Users

The primary objective of this quiz is to establish a reliable baseline of the user's competencies, ensuring that subsequent content delivery is appropriately tailored to their current abilities.

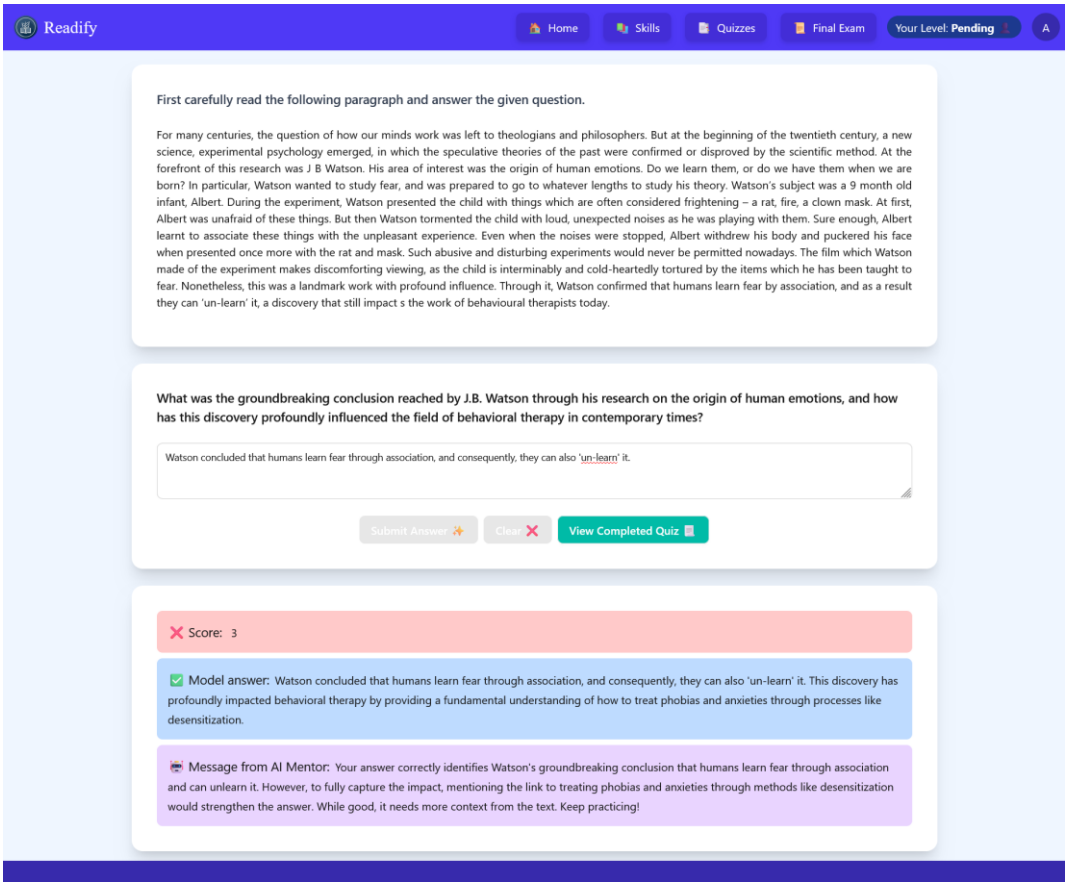


Figure. 2.40 CEFR Placement Quiz

Following the completion of the placement quiz, the system conducts an automated analysis of the user's responses. The resulting score is systematically mapped to the CEFR scale, ranging from A1 (beginner) to C2 (proficient). This process enables precise categorization of the user's language proficiency level, thereby ensuring that learners are placed in a manner that corresponds accurately to their demonstrated skills and knowledge.

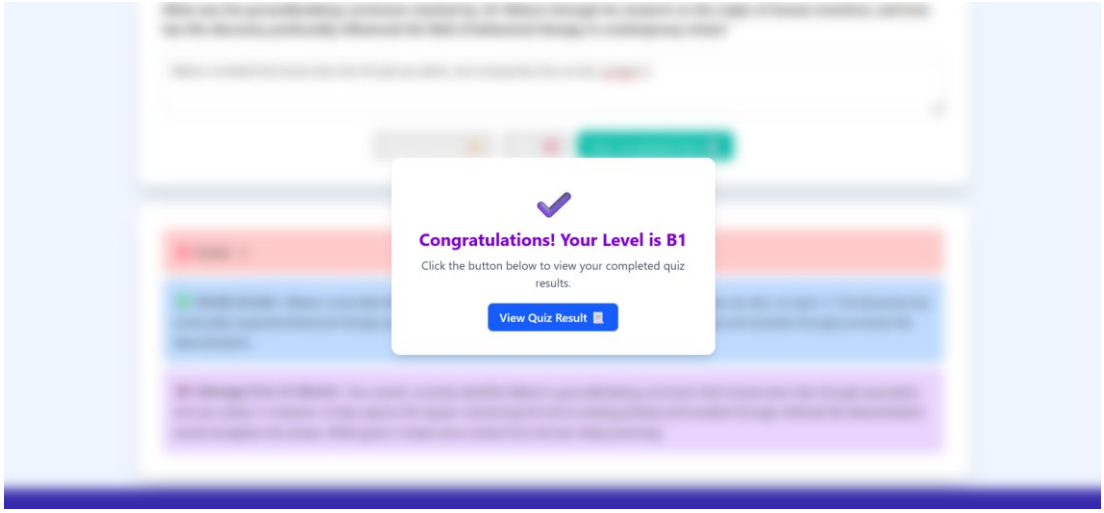


Figure. 2.41 Display Users CEFR Level

Users who achieve a CEFR level of B1 or above are permitted to proceed with the full functionality of the application. This threshold has been established to ensure users possess the foundational linguistic competence necessary for engaging with the platform's learning modules.

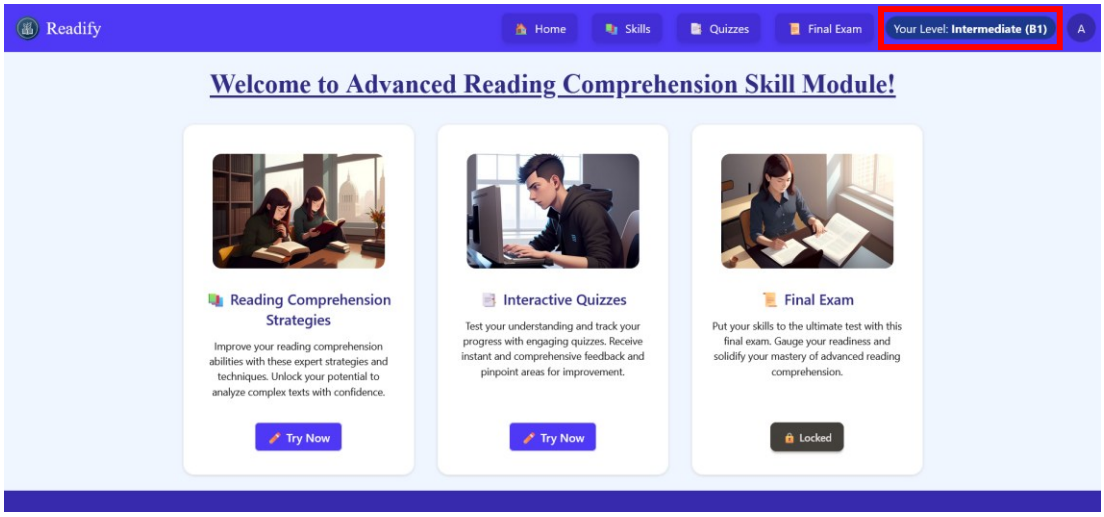


Figure. 2.42 Users In CEFR Level B1 or Above Can Continue to Application

When a user creates a new quiz for the first time, the application with references their assigned CEFR level to generate appropriately leveled content. After each quiz undertaken within the application, performance data is analyzed to monitor user progress and inform subsequent content adaptation.

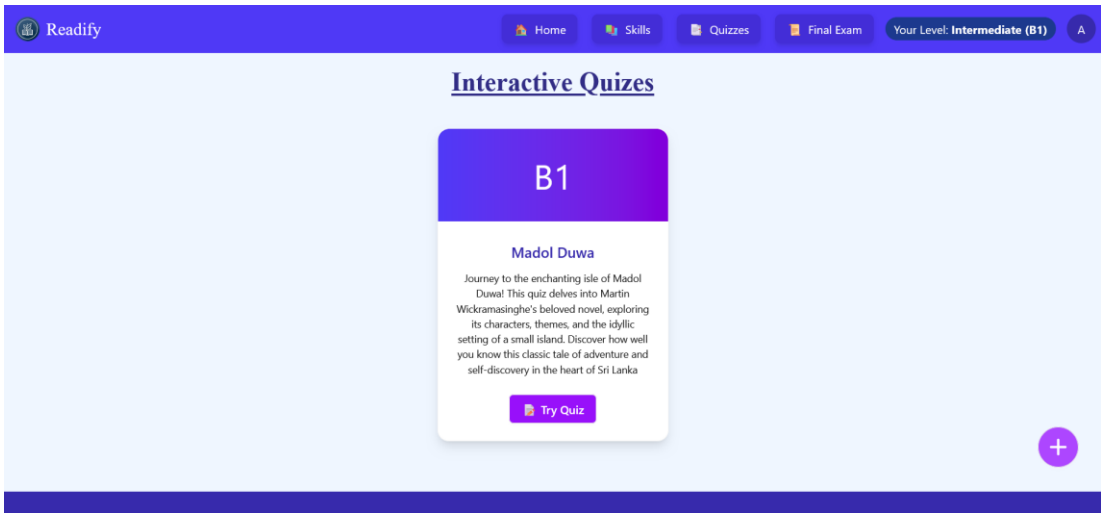


Figure. 2.43 Quiz Created According to User's CEFR Level

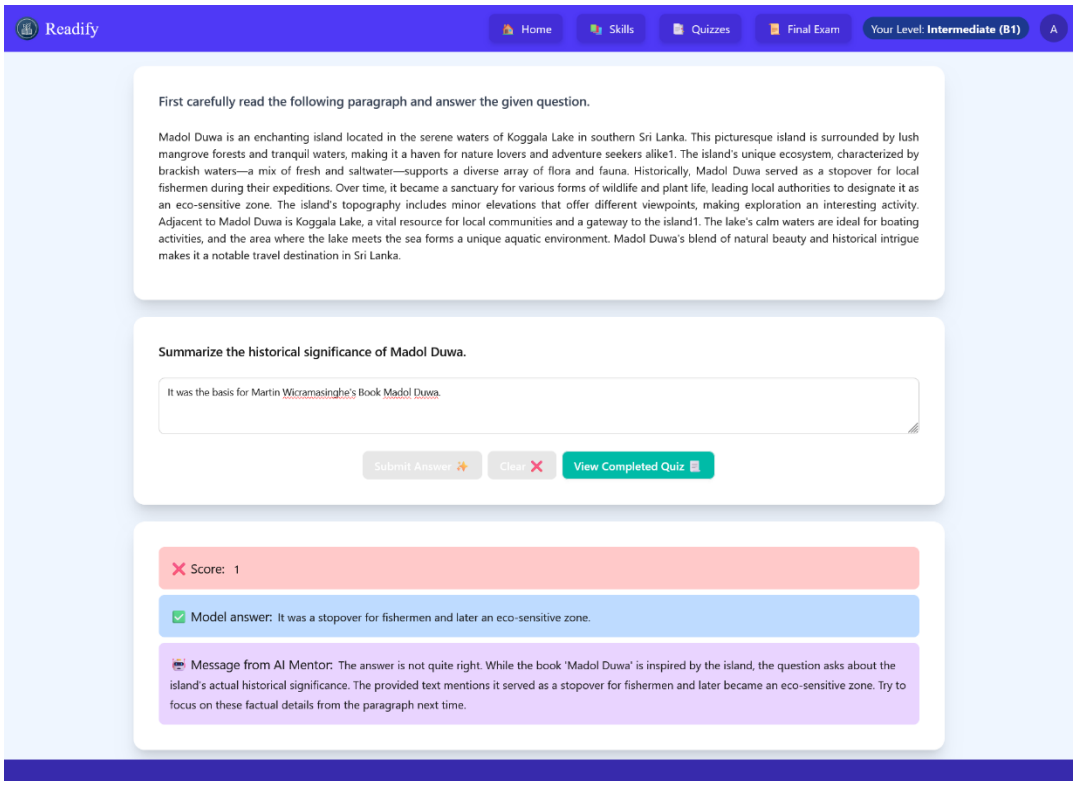


Figure. 2.44 User Answers the Generated Questions

The system incorporates a rule-based adjustment mechanism to maintain alignment between user skill level and quiz difficulty. If a user attains a score exceeding 80% on a given quiz, their CEFR level is incrementally advanced, reflecting demonstrated

mastery of the current level. Conversely, a score below 20% results in a level reduction, indicating the need for reinforcement of foundational concepts. Scores falling within the 20%–80% range prompt no level change, maintaining the user's current proficiency classification.

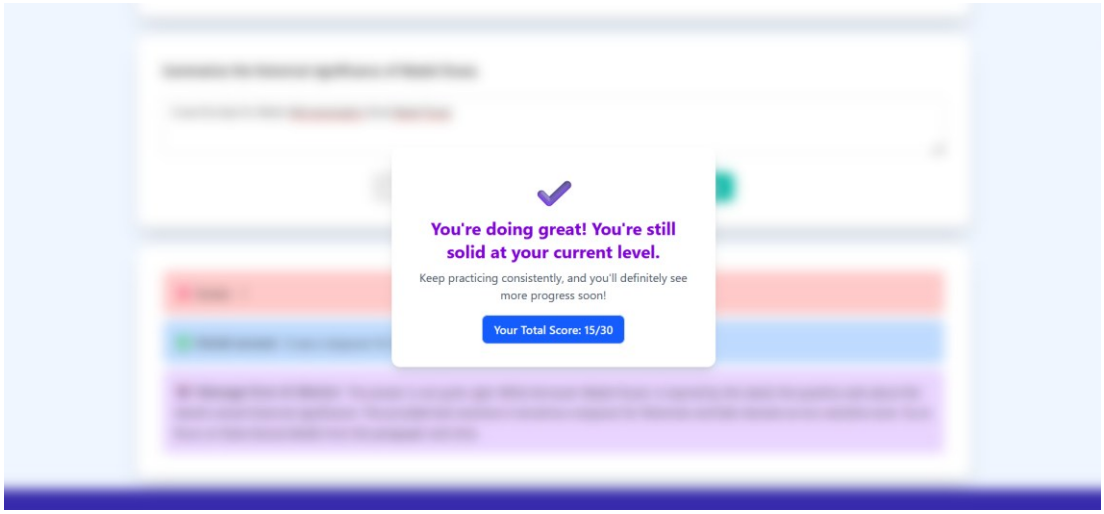


Figure. 2.45 User CEFR Level is Calculated

Each time a user initiates a new quiz, the application dynamically references their most recently assigned CEFR level to generate appropriately leveled content. This ensures that the material presented remains both relevant and suitably challenging, thereby supporting a personalized learning progression. The adaptive nature of this mechanism is important to sustain learner engagement and promoting measurable skill development over time.

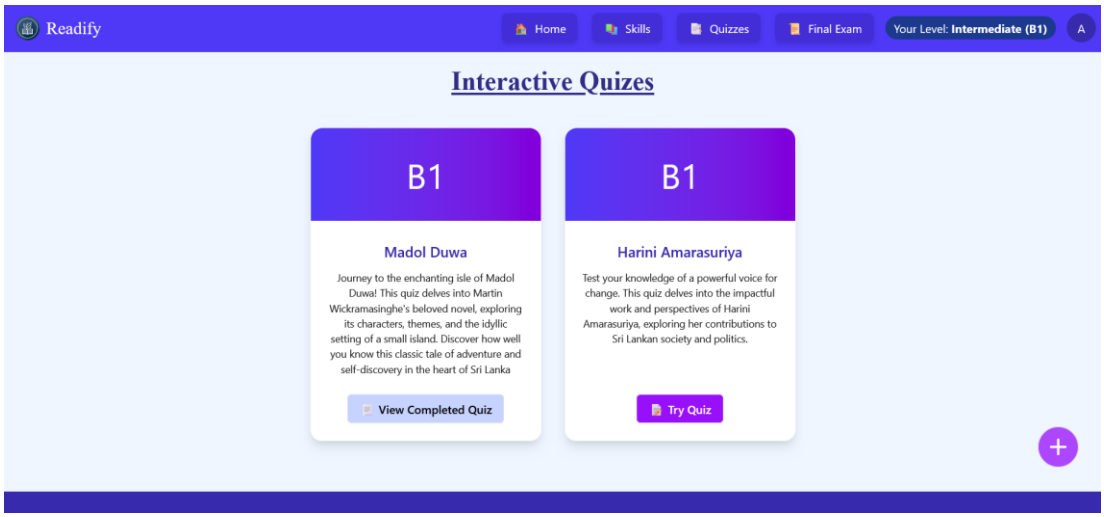


Figure. 2.45 Next Quiz is Generated According to CEFR Level

2.3.2 Testing

2.3.2.1 AI Workflow Evaluation

A) CRAG-based Multi-Agent Workflow for Question Generation

To ensure the quality of AI-generated educational content, this workflow evaluates questions and answers using LLM-as-a-Judge paradigm[25]. It uses the RAGAS framework to standardize evaluation while incorporating custom-defined metrics tailored for academic question generation tasks.

The system operates asynchronously and leverages Google's Gemini 2.5 model (via Langchain) to assess different qualitative aspects of generated question-answer pairs. Each evaluation is guided by a carefully designed prompt with a chain-of-thought structure to enable detailed and reliable scoring.

```python
# Adjust DATA_PATH if your file is in a different location in your Drive
DATA_PATH = '/content/drive/MyDrive/ReadifyEval/outputs.csv'
OUTPUT_PATH = '/content/drive/MyDrive/ReadifyEval/evaluation_results.csv' # Define an output path in Drive

# --- Set your Gemini API key
os.environ["GOOGLE_API_KEY"] = userdata.get('GOOGLE_API_KEY')

# --- Initialize Gemini LLM
llm = LangchainLLMWrapper(ChatGoogleGenerativeAI(model="gemini-2.5-flash-preview-04-17"))

@dataclass
class EvaluationMetric(Metric):
    name: str
    question_template: str = ""

    def init(self, *args: Any, **kwargs: Any) -> None:
        """Initializes the metric."""
        super().init(*args, **kwargs)

    # Make the score method asynchronous
    async def score(self, inputs: pd.DataFrame, references: pd.DataFrame) -> List[float]:
        prompts = [
            self.question_template.format(
                topic=t, desc=d, cefr=c, paragraph=p, question=q, answer=a
            )
            for t, d, c, p, q, a in zip(
                inputs["given_topic"],
                inputs["given_description"],
                inputs["cefr_level"],
                inputs["paragraph"],
                inputs["question"],
                inputs["model_answer"],
            )
        ]
        scores = []
        for prompt in tqdm(prompts, desc=self.name):
            try:
                # Await the asynchronous generate call
                response = await llm.generate([prompt])
                # Access the text attribute of the first generation chunk
                score = float(response[0].text.strip().split()[0])
                scores.append(min(max(score, 0.0), 5.0)) # Ensure score is between 0 and 5
            except Exception as e:
                print(f"Error processing prompt for {self.name}: {e}. Assigning 0.0.")
                scores.append(0.0) # Assign a default score in case of an error
        return scores
```

Figure. 2.46 Question Generation Workflow Evaluation

Custom Evaluation Metrics

The evaluation framework includes five custom metrics, each implemented using a subclass of *ragas.Metric* and evaluated through structured prompts. All metrics produce a score between 1 and 5.

Relevance to Topic metric assesses how well the generated question and its corresponding answer align with the given topic and its description. The evaluation prompt guides the model through identifying key ideas from the topic and checking whether these are clearly reflected in both the question and the answer. A high score indicates that the question is highly relevant and meaningfully connected to the topic, while a low score suggests that it deviates significantly or lacks topical focus.

```
EvaluationMetric(
    name="Relevance to Topic",
    question_template=(
        "You are evaluating how relevant a question is to a given topic.\n"
        "Step 1: Identify the key idea(s) in the topic: '{topic}' and its description: '{desc}'.\n"
        "Step 2: Analyze the question: '{question}' to see if it addresses those key ideas.\n"
        "Step 3: Consider whether the answer: '{answer}' also aligns with the topic.\n"
        "Step 4: Justify your reasoning.\n"
        "Finally, give a score from 1 to 5, where 1 = completely irrelevant, 5 = perfectly relevant.\n"
        "Output only the score at the end."
    ),
),
```

Figure. 2.47 Relevance to Topic Metric

Accuracy of Information is evaluated by comparing the answer to the factual content provided in a reference paragraph. The model first analyzes the paragraph to extract factual statements and then verifies whether the answer is consistent with these facts. Any factual errors, exaggerations, or unsupported claims are flagged during this process. The score reflects the degree to which the answer adheres to the source material, ensuring the output is trustworthy and evidence-based.

```
EvaluationMetric(
    name="Accuracy of Information",
    question_template=(
        "You are evaluating how factually accurate an answer is given a paragraph.\n"
        "Step 1: Read the paragraph: '{paragraph}' and understand the facts it conveys.\n"
        "Step 2: Analyze the question: '{question}' and the answer: '{answer}'.\n"
        "Step 3: Check whether the answer is consistent with and supported by the paragraph.\n"
        "Step 4: Mention any factual errors or unsupported claims.\n"
        "Finally, rate the accuracy from 1 to 5 (1 = inaccurate, 5 = factually correct).\n"
        "Output only the score at the end."
    ),
),
```

Figure. 2.48 Accuracy of Information

Accuracy of Information is evaluated by comparing the answer to the factual content provided in a reference paragraph. The model first analyzes the paragraph to extract factual statements and then verifies whether the answer is consistent with these facts. Any factual errors, exaggerations, or unsupported claims are flagged during this process. The score reflects the degree to which the answer adheres to the source material, ensuring the output is trustworthy and evidence based.

```
EvaluationMetric(
    name="Coverage of Topic",
    question_template=(
        "You are evaluating how well a question and its answer cover the key ideas in a paragraph and the assigned topic.\n"
        "Step 1: Identify key content in the paragraph: '{paragraph}' and topic: '{topic}'.\n"
        "Step 2: Determine if the question: '{question}' and answer: '{answer}' reflect these ideas.\n"
        "Step 3: Note if important ideas are missed or addressed.\n"
        "Step 4: Justify your conclusion.\n"
        "Give a score from 1 to 5 where 1 = minimal coverage, 5 = excellent and comprehensive coverage.\n"
        "Output only the score at the end."
    ),
),
```

Figure. 2.49 Coverage of Topic

Clarity and Understandability metric measures how easily a user can comprehend the question and answer. The prompt instructs the model to examine language complexity, sentence structure, and the presence of ambiguous or advanced vocabulary. The model then judges whether the language used is appropriate for the specified proficiency level. A high clarity score indicates that the question and answer are well-phrased and accessible to the learner.

```
EvaluationMetric(
    name="Clarity and Understandability",
    question_template=(
        "You are evaluating how clear and understandable a question and answer are for a learner.\n"
        "The learner is at CEFR level {cefr}.\n"
        "Step 1: Read the question: '{question}' and the answer: '{answer}'.\n"
        "Step 2: Identify any complex vocabulary, long sentences, or ambiguous expressions.\n"
        "Step 3: Assess if the pair would be understandable to a CEFR {cefr} student.\n"
        "Step 4: Justify your judgment.\n"
        "Then, give a score from 1 to 5, where 1 = very hard to understand, and 5 = very clear and simple.\n"
        "Output only the score at the end."
    ),
),
```

Figure. 2.50 Clarity and Understandability

Quality of Questions metric focuses on the overall construction and educational value of the generated question. The evaluation considers grammatical accuracy, specificity, and the potential of the question to encourage comprehension or critical thinking. The prompt leads the model to examine whether the question is relevant, well-formed, and

pedagogically sound. Higher scores reflect questions that are not only technically correct but also valuable in a learning context.

```
EvaluationMetric(
    name="Quality of Questions",
    question_template=(
        "You are evaluating the quality of the question.\n"
        "Step 1: Read the paragraph: '{paragraph}' and topic: '{topic}'.\n"
        "Step 2: Analyze the question: '{question}' for grammar, specificity, relevance, and potential to elicit meaningful answers.\n"
        "Step 3: Consider whether the question promotes comprehension or critical thinking.\n"
        "Step 4: Justify your evaluation.\n"
        "Finally, rate the question from 1 to 5 (1 = low quality, 5 = high quality).\n"
        "Output only the score at the end."
    ),
),
```

Figure. 2.51 Quality of Questions

B) Multi-Model Agentic Workflow with Confidence-Based Fallback and Arbitration Mechanism for Answer Evaluation and Feedback Generation

This evaluation workflow is built upon the same foundational structure as the previous evaluation process.

Custom Evaluation Metrics

Accuracy of Agent's Assessment metric evaluates how accurately the AI agent has scored a user's answer compared to a model (ideal) answer and the reference paragraph. The prompt guides Gemini through understanding the context, comparing answers, and assessing whether the agent's score and feedback are justified. A score from 1 (inaccurate) to 5 (perfectly accurate) is assigned.

```
EvaluationMetric(
    name="Accuracy of Agent's Assessment",
    question_template=(
        "You are an expert evaluator assessing the accuracy of an AI agent's assessment of a user's answer.\n"
        "Given the following:\n"
        "Paragraph: '{paragraph}'\n"
        "Original Question: '{question}'\n"
        "Ideal/Model Answer (by a good model, not necessarily the agent's): '{model_answer}'\n"
        "User's Answer: '{user_answer}'\n"
        "Agent's Feedback on User's Answer: '{agent_feedback}'\n"
        "Agent's Numeric Score for User's Answer: '{agent_score}'\n\n"
        "Step 1: Read the 'Paragraph' and 'Original Question' to understand the context.\n"
        "Step 2: Compare the 'User's Answer' to the 'Ideal/Model Answer' and the 'Paragraph' to determine its actual correctness.\n"
        "Step 3: Analyze the 'Agent's Feedback' and 'Agent's Numeric Score'. Does the agent's assessment (both feedback and score) accurately
        "Step 4: Justify your reasoning, highlighting any discrepancies or strong alignments.\n"
        "Finally, give a score from 1 to 5 for the **Accuracy of the Agent's Assessment**, where 1 = highly inaccurate assessment, 5 = perfec
        "Output only the score at the end."
    ),
```

Figure. 2.52 Accuracy of Agent's Assessment

Relevance of Feedback assesses whether the agent's feedback focuses on pertinent aspects of the user's answer and the original question. The model checks for tangents, irrelevant commentary, or generic phrasing. Feedback that is highly targeted and focused on the user's response scores higher.

```
EvaluationMetric(
    name="Relevance of Feedback",
    question_template=(
        "You are evaluating the relevance of an AI agent's feedback on a user's answer.\n"
        "Given the following:\n"
        "Paragraph: '{paragraph}'\n"
        "Original Question: '{question}'\n"
        "User's Answer: '{user_answer}'\n"
        "Agent's Feedback on User's Answer: '{agent_feedback}'\n\n"
        "Step 1: Understand the 'Original Question' and the 'User's Answer'.\n"
        "Step 2: Read the 'Agent's Feedback'. Is the feedback directly related to the 'User's Answer' and the 'Original Question'? Does it focu
        "Step 3: Identify any irrelevant information or tangents in the feedback.\n"
        "Step 4: Justify your reasoning.\n"
        "Finally, give a score from 1 to 5 for the **Relevance of Feedback**, where 1 = completely irrelevant, 5 = highly relevant and focused.
        "Output only the score at the end."
    ),
),
```

Figure. 2.53 Relevance of Feedback

Helpfulness and Constructiveness metric evaluates if the feedback offers specific, actionable advice that can help the learner improve. The prompt encourages Gemini to look for clarity in guidance, identification of mistakes, and educational value. Vague or generic comments receive lower scores.

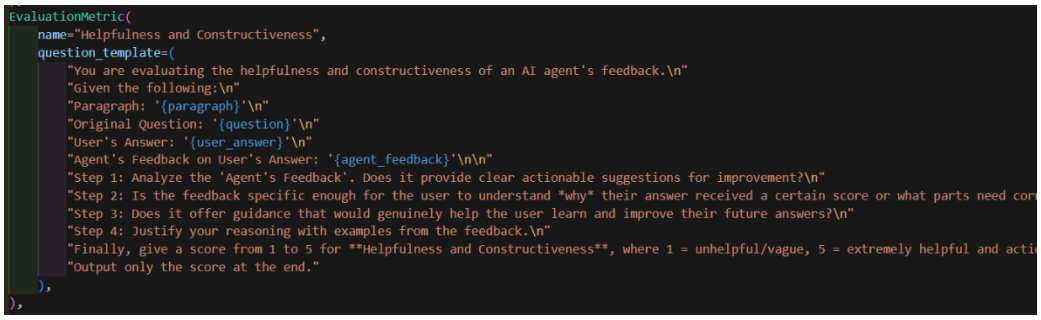


Figure. 2.54 Helpfulness and Constructiveness

Tone of Feedback metric analyzes whether the tone of the agent's response is supportive, respectful, and encouraging. Feedback should avoid sounding harsh, condescending, or overly critical. A warm and constructive tone earns a higher score.

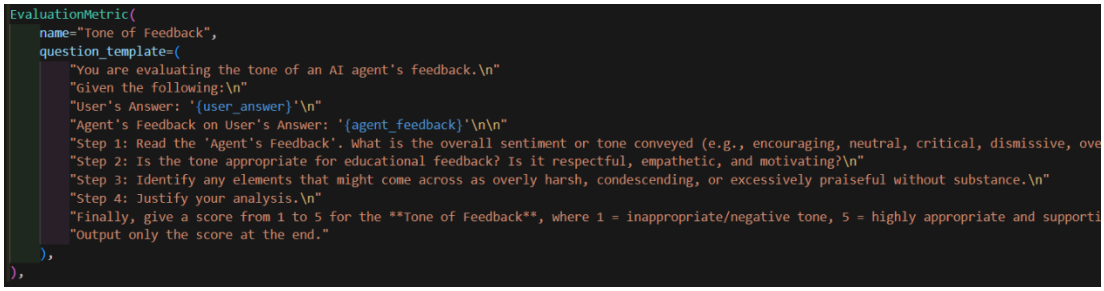


Figure. 2.55 Tone of Feedback

2.3.2.2 Integration Testing

Table 2.1 Test Case TC001

Test Case ID	TC001
Scenario	Show Initial Placement Quiz for new user
Input	A new user signs up with no quiz history
Expected Output	Initial placement quiz is displayed to the user immediately after registration
Actual Output	Initial placement quiz is displayed to the user immediately after registration
Status	**Pass**

Result:

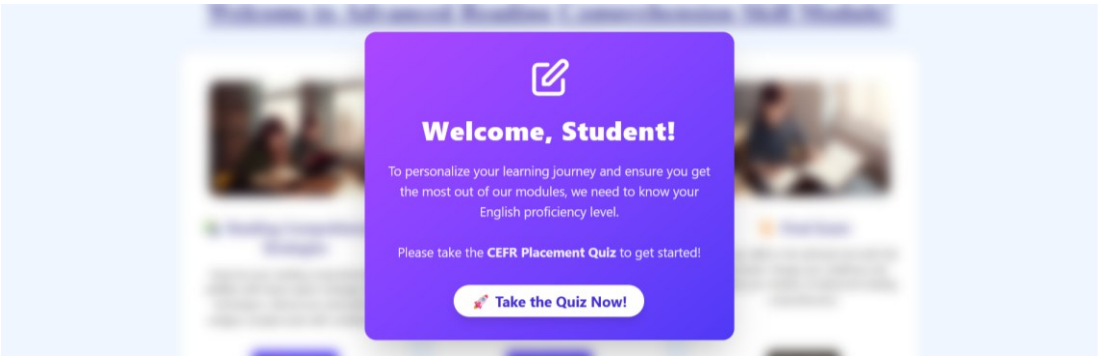
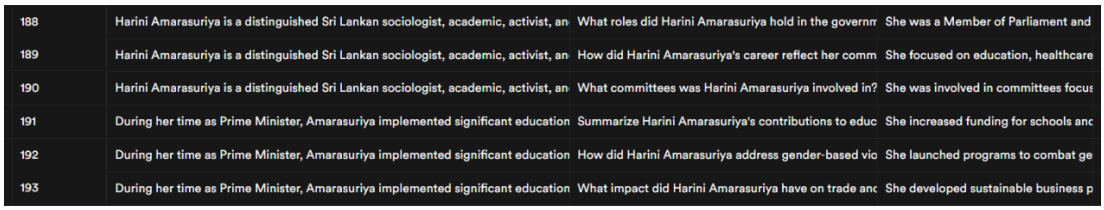


Figure. 2.56 Initial Welcome Screen for Users

Table 2.2 Test Case TC002

Test Case ID	TC002
Scenario	Question Generation based on given topic and user level
Input	"*Harini Amarasuriya*", User Level: B1
Expected Output	A set of quiz questions related to "*Harini Amarasuriya*" with B1-level difficulty is generated
Actual Output	A set of quiz questions related to "*Harini Amarasuriya*" with B1-level difficulty is generated
Status	**Pass**

Result:



188	Harini Amarasuriya is a distinguished Sri Lankan sociologist, academic, activist, an	What roles did Harini Amarasuriya hold in the governr	She was a Member of Parliament and
189	Harini Amarasuriya is a distinguished Sri Lankan sociologist, academic, activist, an	How did Harini Amarasuriya's career reflect her comm	She focused on education, healthcare
190	Harini Amarasuriya is a distinguished Sri Lankan sociologist, academic, activist, an	What committees was Harini Amarasuriya involved in?	She was involved in committees focus
191	During her time as Prime Minister, Amarasuriya implemented significant education	Summarize Harini Amarasuriya's contributions to educ	She increased funding for schools anc
192	During her time as Prime Minister, Amarasuriya implemented significant education	How did Harini Amarasuriya address gender-based vio	She launched programs to combat ge
193	During her time as Prime Minister, Amarasuriya implemented significant education	What impact did Harini Amarasuriya have on trade anc	She developed sustainable business p

Figure. 2.57 Generated Questions in Database

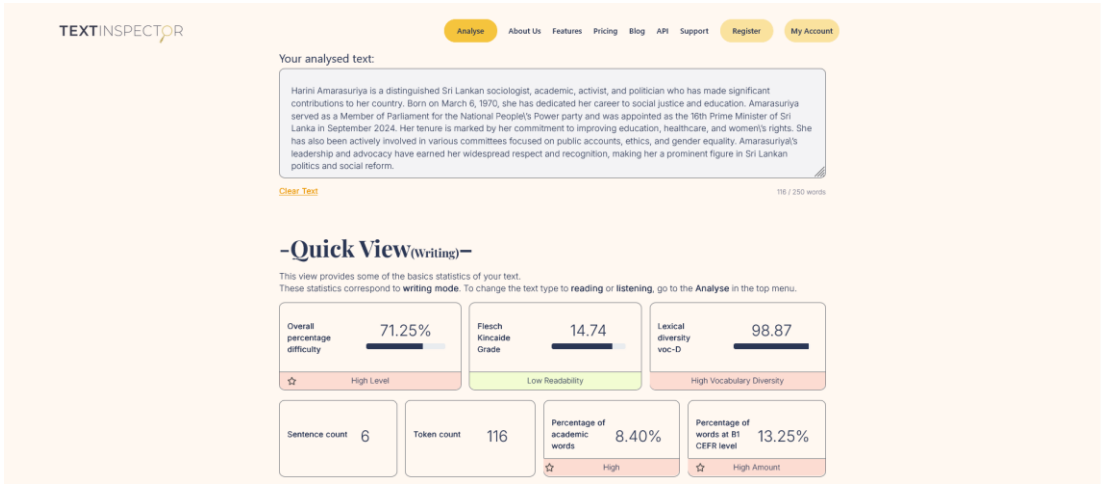


Figure. 2.58 Check CEFR Level of the First Paragraph Using Text Inspector

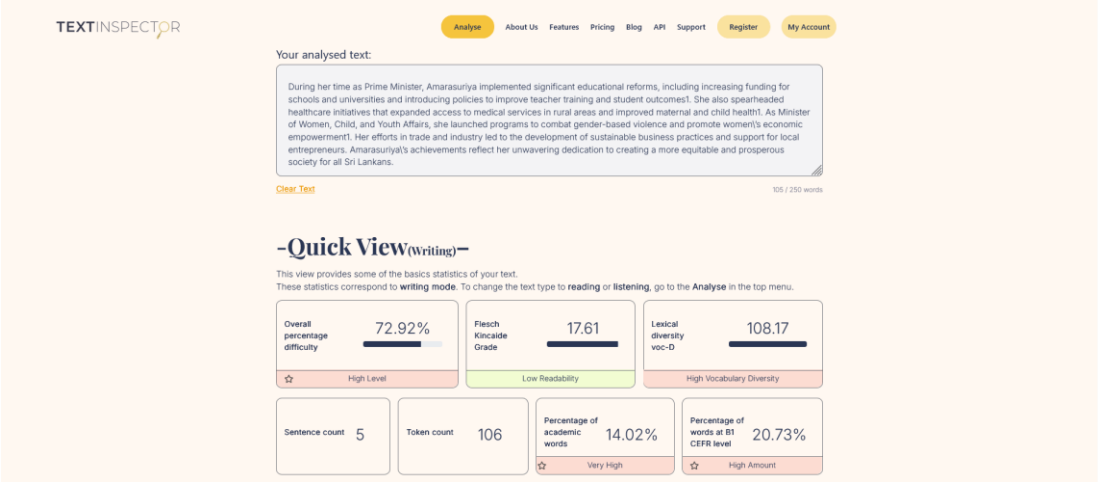


Figure. 2.59 Check CEFR Level of the First Paragraph Using Text Inspector

Table 2.3 Test Case TC003

Test Case ID	TC003
Scenario	Quizzes render correctly on UI
Input	All quizzes related to the user are loaded
Expected Output	Display all quizzes related to the user
Actual Output	Display all quizzes related to the user
Status	Pass

Result:

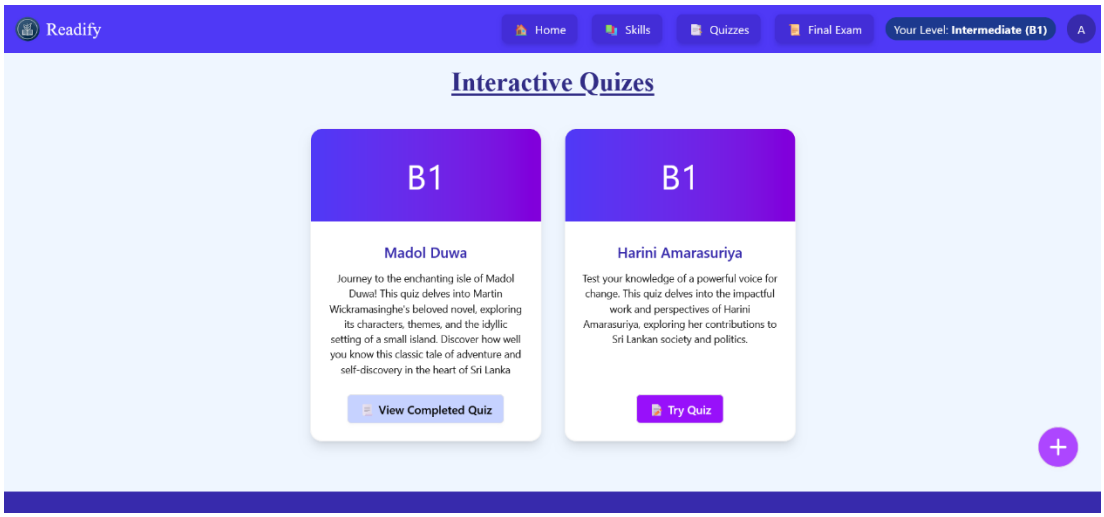


Figure. 2.60 Generated Quizzes

Table 2.4 Test Case TC004

Test Case ID	TC004
Scenario	All Questions render correctly on UI
Input	A quiz with 6 questions is loaded
Expected Output	Displays all questions, input fields/options, and navigation buttons properly
Actual Output	Displays all questions, input fields/options, and navigation buttons properly
Status	Pass

Result:

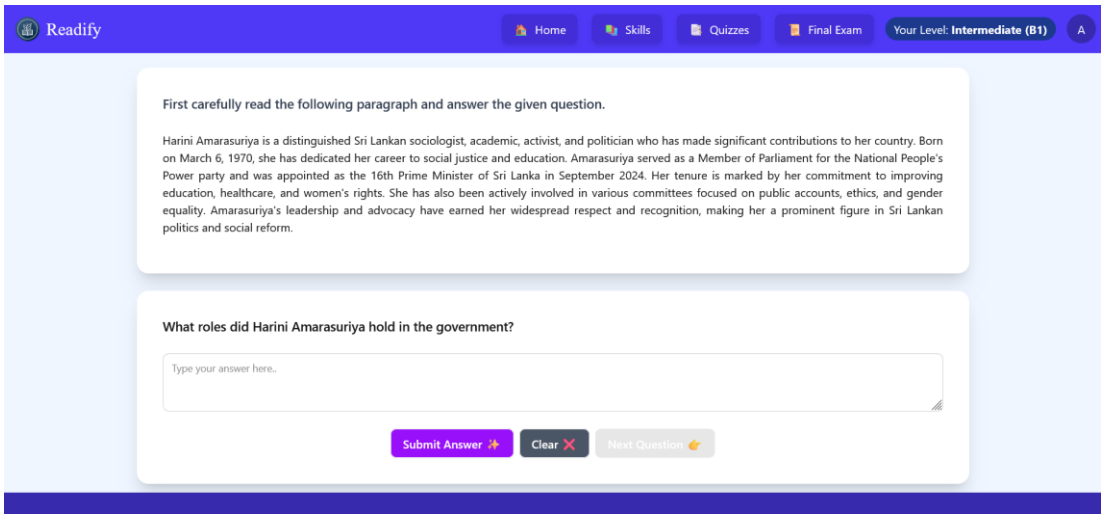


Figure. 2.61 Generated Questions

Table 2.5 Test Case TC005

Test Case ID	TC005
Scenario	Evaluate user answer
Input	User submits answers for a question
Expected Output	System evaluates answers, marks correct/incorrect responses, and provides feedback
Actual Output	System evaluates answers, marks correct/incorrect responses, and provides feedback
Status	Pass

Result:

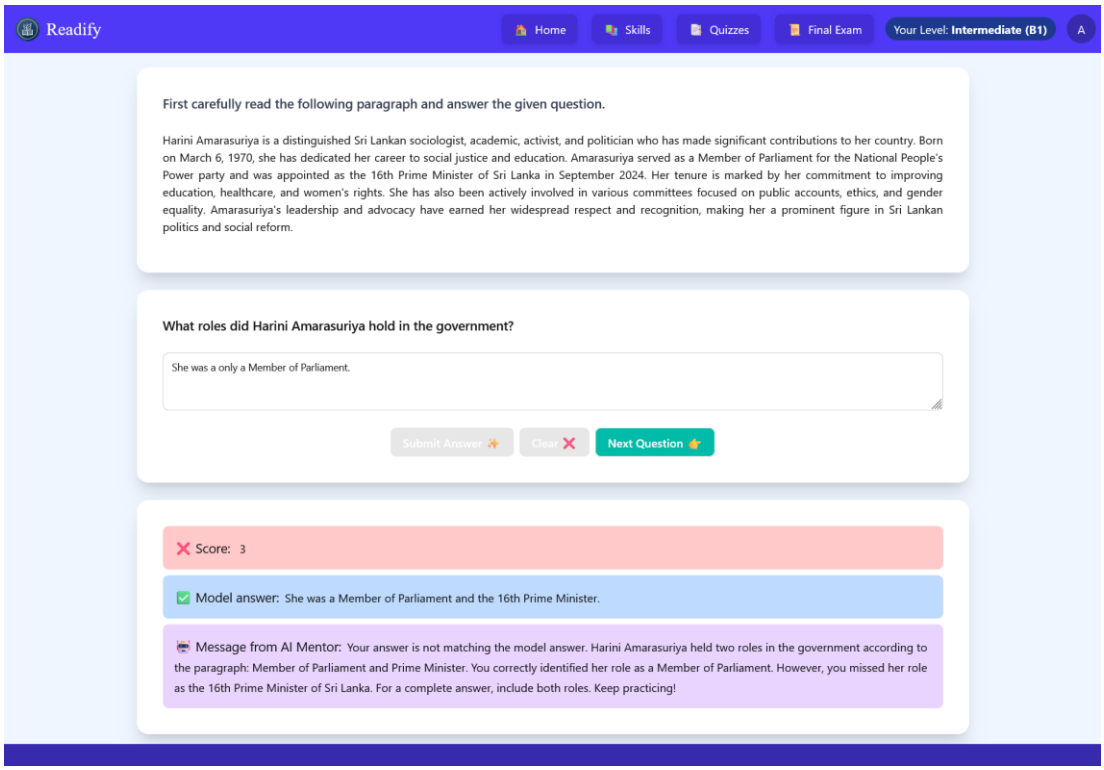


Figure. 2.62 Feedback for User Answer with Score

Table 2.6 Test Case TC006

Test Case ID	TC006
Scenario	View results for completed quiz
Input	User finishes quiz and clicks “View Completed Quiz”
Expected Output	Summary of score, correct/incorrect answers, and level feedback is displayed
Actual Output	Summary of score, correct/incorrect answers, and level feedback is displayed
Status	Pass

Result:

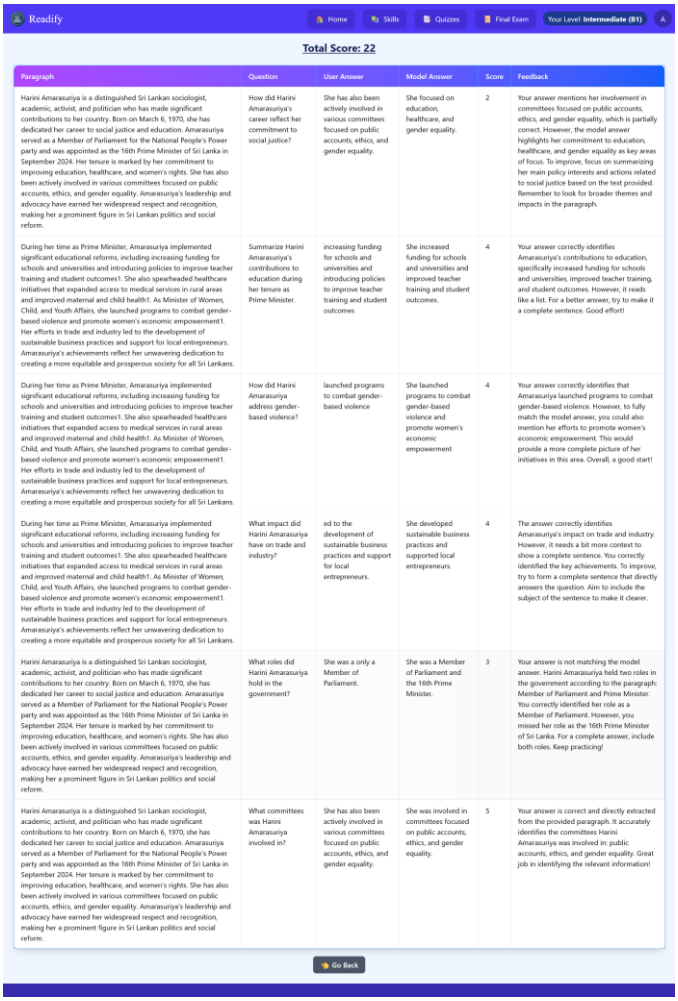


Figure. 2.63 View Quiz Results with Score

2.4 Commercialization

2.4.1 Demand for English Education Applications in Sri Lanka

The demand for English language proficiency in Sri Lanka remains critically high, driven by its necessity for higher education, professional employment, and global connectivity. Digital platforms, particularly mobile applications, are increasingly viewed as accessible and flexible tools to meet this educational gap. While specific market data for English learning apps in Sri Lanka is evolving, the broader surge in internet and smartphone adoption underscores a significant untapped potential for digital learning solutions. This trend is further supported by the government's push towards digital education transformation, making accessible and engaging English learning applications highly relevant to the local context.

2.4.2 Target Audience and Market

Our primary market segments for this application are:

- High School Students: Requiring English proficiency for examinations (e.g., G.C.E. O/L and A/L) and future academic pursuits.
- English Language Learners: A broad category encompassing individuals seeking to improve their English for various personal and professional reasons, from beginners to advanced learners.
- English Educators: Teachers looking for supplementary resources, assessment tools, and interactive content to enhance their classroom instruction.

2.4.3 Business Strategy: Dual-Tiered SaaS Model

The commercialization strategy for this application is built upon a flexible Software as a Service (SaaS) model, incorporating two distinct service tiers to maximize market reach and revenue potential.

1. **Free Tier** supported by advertisements primarily from educational institutions, ensures broad accessibility, encouraging widespread adoption among learners. This tier serves as an entry point, introducing users to the app's features and benefits.
2. **Premium Version** is available through a subscription-based model, offering an ad-free environment with additional features and resources.

By offering these two tiers, the app can attract a diverse user base, from casual users using the free version to dedicated learners and educators opting for the premium services. This dual-tiered approach not only diversifies revenue streams but also enhances user engagement and retention, positioning the app as a valuable tool in the educational landscape.

2.4.4 Marketing Strategies

Our marketing efforts will focus on strategic partnerships and direct engagement. We aim to collaborate with Non-Governmental Organizations (NGOs) and charities dedicated to education in Sri Lanka, offering subsidized or free premium access to underserved communities to promote digital literacy and English proficiency. This corporate social responsibility initiative will also enhance brand reputation. Additionally, we will conduct interactive seminars and workshops in high schools across the island. These sessions will introduce the app's features directly to our target student audience and English educators, demonstrating its practical benefits and encouraging immediate adoption through hands-on experience and targeted promotions. This multi-pronged approach ensures broad reach and community integration.

3. RESULTS AND DISCUSSION

3.1 Results

This section presents the evaluation results for the primary components of the Readify project: the AI agentic workflows for Automated Quiz Generation and Automated Assessment and Feedback Generation. To provide a robust and objective evaluation, an LLM-as-a-Judge methodology was employed for both workflows, leveraging a Chain-of-Thought (CoT) prompting strategy to guide a the language model acting as an expert evaluator through a detailed assessment process based on predefined criteria.

3.1.1 Automated Quiz Generation Workflow Related Results

As mentioned in the previous section, The Automated Quiz Generation Workflow, designed to create reading comprehension quizzes from user-specified topics, was evaluated using an LLM-as-a-Judge. The evaluation of the generated quizzes was based on several key metrics designed to capture the quality and utility of the educational content. These metrics, scored on a 5-point Likert scale (1 being the lowest, 5 the highest), included

- Relevance to Topic,
- Accuracy of Information,
- Clarity and Understandability
- Coverage of Topic
- Quality of Questions.

Additionally, a binary check for the Correctness of Individual Answers provided by the workflow was performed for each question to assess factual accuracy of the proposed solutions.

Across a set of evaluated quizzes covering various topics and containing multiple questions each, the Automated Quiz Generation Workflow demonstrated consistently strong performance. The aggregated results, representing the average scores across all evaluated quizzes for each scaled criterion, are presented in Table 1. As shown, average evaluation scores were high across all dimensions. Furthermore, the binary check conducted for each individual question revealed a high overall correctness rate for the agent's provided answers.

Table 3.1 Average Evaluation Scores for Automated Quiz Generation Workflow

Evaluation Criterion	**Average Score (1-5)**
Relevance to Topic	4.7
Accuracy of Information	4.8
Clarity and Understandability	4.6
Coverage of Topic	4.5
Quality of Questions	4.5

The aggregated results in Table 1 indicate that the quiz generation workflow is highly effective in producing relevant, accurate, and clear reading comprehension questions directly derived from provided textual content. The high scores in Relevance and Accuracy confirm the agent's ability to generate factually correct and on-topic content, while strong scores in Clarity and Quality of Questions suggest the output is well-phrased and suitable for educational use within this specific format. The solid Coverage score reflects that the questions effectively utilize the information present in the provided paragraphs.

3.1.2 Automated Assessment and Feedback Generation Related Results

The Automated Assessment and Feedback Generation Workflow, responsible for evaluating student open-ended answers to paragraph-based questions and providing feedback, was also rigorously evaluated using the LLM-as-a-Judge methodology. Evaluation metrics for this workflow specifically targeted the agent's assessment and feedback capabilities. Scaled metrics (1-5) included

- Accuracy of Agent's Assessment
- Relevance of Feedback
- Helpfulness and Constructiveness of Feedback
- Clarity and Understandability of Feedback
- Tone of Feedback.

The evaluation was conducted on a dataset of 25 distinct student response scenarios, capturing variation in student answers and question types derived from paragraphs. The aggregated results from these 25 evaluations, presenting the average scores for the scaled feedback quality criteria, are summarized in Table 2. As illustrated, the workflow demonstrated strong performance across these dimensions, with average scores ranging from 4.1/5 to 4.8/5. The accuracy of the agent's core judgment regarding the correctness of the student answer was also assessed binarily for each scenario, showing that the agent's judgment matched the LLM judge's independent determination in over 90% of the evaluated cases, demonstrating a high level of reliability in basic correctness classification.

Table 3.2 Average Evaluation Scores for Automated Assessment and Feedback Generation Workflow

Evaluation Criterion	**Average Score (1-5)**
Accuracy of Agent's Assessment	4.8
Relevance of Feedback	4.5
Helpfulness and Constructiveness	4.7
Clarity and Understandability	4.6
Tone of Feedback	4.4

These results indicate that the Automated Assessment and Feedback Generation Workflow is largely effective in its purpose. The high accuracy in core judgment demonstrates reliability in automatically determining the correctness of student responses based on the source material. The strong scores for feedback quality, particularly in Clarity and Tone, suggest that the feedback provided is easily comprehensible and delivered in an appropriate educational manner.

3.2 Research Findings

The Readify project was rigorously evaluated across its two primary AI agentic workflows: Automated Quiz Generation and Automated Assessment and Feedback Generation. Both workflows employed an LLM-as-a-Judge methodology, utilizing a Chain-of-Thought (CoT) prompting strategy to ensure objective and detailed assessments against predefined criteria.

3.2.1 Automated Quiz Generation Workflow Findings

The Automated Quiz Generation Workflow, designed to create reading comprehension quizzes from user-specified topics, demonstrated consistently strong performance across key metrics. The evaluation was conducted using a language model configured as an experienced AI Evaluation Specialist, which reviewed generated quizzes based on the user-provided topic, paragraph text, and correct answers.

The aggregated average scores (on a 1-5 scale) for the evaluated quizzes were notably high:

- Relevance to Topic: 4.7/5
- Accuracy of Information: 4.8/5
- Clarity and Understandability: 4.6/5
- Coverage of Topic: 4.5/5
- Quality of Questions: 4.5/5

These results confirm the agent's strong ability to produce factually correct and on-topic content that is well-phrased and suitable for educational use. Furthermore, a binary check for the correctness of individual answers revealed an overall correctness rate exceeding 95% across all evaluated questions in the dataset, reinforcing the factual accuracy of the generated solutions. While highly successful in generating comprehension questions from given text, the current evaluation primarily assesses this specific capability, and the generated questions were predominantly open-ended.

3.2.2 Automated Assessment and Feedback Generation Workflow Findings

The Automated Assessment and Feedback Generation Workflow, responsible for evaluating open-ended student answers and providing feedback, also exhibited robust performance. This workflow was assessed using the LLM-as-a-Judge methodology, with the LLM judge adopting the persona of an experienced Educational AI Evaluation Specialist. The evaluation dataset comprised 25 distinct student response scenarios, capturing variations in student answers and question types.

The aggregated average scores (on a 1-5 scale) for feedback quality criteria were consistently high:

- Accuracy of Agent's Assessment: 4.8/5
- Relevance of Feedback: 4.5/5
- Helpfulness and Constructiveness: 4.7/5
- Clarity and Understandability: 4.6/5
- Tone of Feedback: 4.4/5

The high accuracy in core judgment (4.8/5) demonstrates the reliability of automatically determining the correctness of student responses based on the source material. Critically, the agent's basic classification of student answers (correct/incorrect) matched the LLM judge's independent determination in over 90% of the evaluated cases, showcasing a high level of reliability in basic correctness classification. The strong scores for feedback quality, particularly in clarity and tone, indicate that the feedback provided is comprehensible and delivered in an appropriate educational manner. However, the evaluation dataset size of 25 scenarios, while valuable, may not fully capture the complete spectrum of complex or nuanced student answers.

3.3 Discussion

The Readify project aimed to address critical gaps in English language education by developing an intelligent assistant that provides personalized content generation and automated assessment with feedback for reading comprehension skills. The core methodology leveraged Large Language Models (LLMs) within advanced agentic frameworks, specifically employing a Multi-Agentic Retrieval-Augmented Generation (RAG) workflow for quiz generation and a Multi-Model Agentic Workflow for answer evaluation and feedback. The evaluation of Readify's components—Automated Quiz Generation and Automated Assessment and Feedback Generation—yielded promising results, validating the efficacy of AI-driven agentic approaches in enhancing reading comprehension.

The Automated Quiz Generation workflow, evaluated using an LLM-as-a-Judge methodology, demonstrated consistently strong performance across key metrics. The average scores for "Relevance to Topic" (4.7/5), "Accuracy of Information" (4.8/5), "Clarity and Understandability" (4.6/5), "Coverage of Topic" (4.5/5), and "Quality of Questions" (4.5/5) highlight the system's ability to produce highly pertinent, factually correct, and well-phrased questions directly derived from provided textual content. Furthermore, a correctness rate exceeding 95% for individual answers reinforced the factual accuracy of the generated solutions. This performance confirms the effectiveness of the Multi-Agentic RAG workflow, particularly its Corrective RAG (C-RAG) process, in grounding LLM responses with verifiable external information and ensuring contextual relevance. The system's ability to tailor quizzes to user interests, a stated research objective, was implicitly supported by the high relevance scores, as the underlying C-RAG mechanism is designed to gather information based on user-specified topics.

Similarly, the Automated Assessment and Feedback Generation workflow showed robust performance in evaluating open-ended student responses. Evaluated across 25 distinct student response scenarios using the LLM-as-a-Judge methodology, the workflow achieved high average scores for "Accuracy of Agent's Assessment" (4.8/5), "Relevance of Feedback" (4.5/5), "Helpfulness and Constructiveness" (4.7/5), "Clarity and Understandability" (4.6/5), and "Tone of Feedback" (4.4/5). Critically, the agent's core judgment regarding the correctness of student answers matched the LLM judge's

independent determination in over 90% of cases, demonstrating a high level of reliability. This success can be attributed to the Multi-Model Agentic Workflow, which incorporates a Confidence-Based Fallback and Arbitration Mechanism, leveraging multiple LLMs (Gemini, DeepSeek, Llama via OpenRouter) to enhance evaluation robustness and provide nuanced, context-aware feedback. This addresses the research problem of limited tools for automated, descriptive evaluations for essay-type questions, traditionally requiring human assessment.

Despite these positive outcomes, certain limitations were observed that warrant further consideration. For the automated assessment and feedback process, the evaluation dataset size of 25 scenarios, while providing a valuable initial assessment, may not fully capture the complete spectrum of complex or highly nuanced student answers. This limited scope could potentially affect the generalizability of the accuracy metrics, particularly when dealing with responses that deviate significantly from expected patterns or require more abstract reasoning for evaluation. Additionally, the reliance on an LLM as the sole judge, despite its efficiency and structured prompting, inherently carries the limitations and potential biases of the judging model itself. While the CoT prompting strategy aims to guide a sophisticated language model as an expert evaluator, human expert evaluations would provide a crucial baseline for further validation and refinement.

In conclusion, Readify demonstrates significant promise as an intelligent assistant for enhancing advanced English reading comprehension. The successful integration of LLMs within agentic frameworks for both automated quiz generation and sophisticated answer evaluation showcases the potential of AI to revolutionize personalized learning experiences. While the current implementation has proven effective, acknowledging its limitations provides a clear roadmap for future enhancements, ultimately paving the way for a more robust and comprehensive educational tool.

3.4 Future Works

Future work should focus on several key areas. Empirical studies with actual learners are crucial to quantify the tangible impact of Readify on their reading comprehension skills and to guide further development based on user interaction data. While the current quiz generation workflow is intentionally designed to generate questions from specific text snippets and to primarily produce open-ended questions, future enhancements could explore upgrading this workflow to synthesize information from a broader knowledge base for a more comprehensive topic, thus increasing content diversity and learning objectives. For the assessment component, comparisons with human expert evaluations are a crucial next step for rigorously validating the LLM-as-a-Judge methodology and identifying and mitigating any inherent biases. Finally, optimizing the performance of the computationally intensive AI workflows to reduce loading times is essential for delivering a seamless and responsive user experience. Addressing these areas will enable Readify to evolve into an even more comprehensive and impactful intelligent assistant for English language learners.

4. CONCLUSION

Developing advanced English reading comprehension skills is vital but challenging due to limited personalized practice and feedback. To address this, we introduced Readify, an intelligent assistant leveraging agentic LLMs for automated, tailored learning experiences. Readify features two core workflows: Automated Quiz Generation from text and Automated Assessment and Feedback Generation for student answers.

Using an LLM-as-a-Judge evaluation, the quiz generation workflow demonstrated high performance in creating relevant, accurate, and clear reading comprehension questions directly from provided text, with excellent correctness in generated answers. The assessment and feedback workflow, evaluated on 25 student scenarios, also showed strong results with high accuracy in assessing answers and generating feedback perceived as relevant, helpful, clear, and well-toned.

This research highlights the potential of agentic LLM frameworks to effectively automate content creation and provide reliable assessment and feedback for reading comprehension tasks, addressing key educational technology gaps. While promising, limitations exist, including the scope of text-based quiz generation and challenges in evaluating complex student responses. Future work should focus on empirical studies with learners to quantify impact and expand workflow capabilities for broader topics and nuanced interactions

5. REFERENCES

[1] Colorín Colorado, “Reading Comprehension Skills for English Language Learners,” Colorín Colorado, Mar. 12, 2007. https://www.colorincolorado.org/article/reading-comprehension-skills-english-language-learners

[2] P. Laban, C. Wu, L. Murakhovs'ka, W. Liu, & C. Xiong, "Quiz Design Task: Helping Teachers Create Quizzes with Automated Question Generation," in NAACL-HLT, 2022.

[3] Puneeth Thotad, Shanta Kallur and Sukanya Amminabhavi, "Automatic Question Generator Using Natural Language Processing", In Journal of Pharmaceutical Negative Results, pp. 2759-2764, Dec. 2022.

[4] X. Lu, S. Fan, J. Houghton, L. Wang, and X. Wang, ReadingQuizMaker: A Human-NLP collaborative system that supports instructors to design High-Quality Reading Quiz questions. 2023, pp. 1–18. doi: 10.1145/3544548.3580957.

[5] M. L. Contreras-Arguello, M. A. Paredes-Valverde, A. M. T. Vásquez, and M. Del Pilar Salas-Zárate, “Automatic generation of summaries and questions to support the reading comprehension process,” in Communications In Computer And Information Science, 2024, pp. 81–92. doi: 10.1007/978-3-031-75702-0_7.

[6] K. Moholkar, M. Chaturvedi, A. Jain, A. Parkhe and K. Singh, "Machine Learning Techniques for Descriptive Answer Evaluation: A Comprehensive Survey," 2024 International Conference on Inventive Computation Technologies (ICICT), Lalitpur, Nepal, 2024, pp. 1412-1419, doi: 10.1109/ICICT60155.2024.10544714.

[7] W. Xia, S. Mao, & C. Zheng, "Empirical Study of Large Language Models as Automated Essay Scoring Tools in English Composition Taking TOEFL Independent Writing Task for Example," ArXiv, vol. abs/2401.03401, 2024.

[8] M. A. Hussein, H. A. Hassan, & M. Nassef, "Automated language essay scoring systems: a literature review," PeerJ Computer Science, vol. 5, 2019.

[9] Pawar, Purushottam, “AI-Enhanced Education: Personalized Learning and Educational Technology”, 2023, doi:10.25215/9358791152.01.

[10] Wu, H., Li, S., Gao, Y. et al. “Natural language processing in educational research: The evolution of research topics”, Educ Inf Technol, 2024, doi: 10.1007/s10639-024-12764-2.

[11] Yuhua Li, "AceReader pro and reading comprehension," Proceeding of the International Conference on e-Education, Entertainment and e-Management, Bali, 2011, pp. 198-201, doi: 10.1109/ICeEEM.2011.6137784.

[12] R. P. Maulida, F. M. Ivone, and A. N. Wulyani, “ReadyRead: App-based Supplementary Materials for Reading Comprehension”, KSS, vol. 5, no. 3, pp. 350–364, Mar. 2021. doi: 10.18502/kss.v5i3.8557

[13] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling, “Corrective Retrieval Augmented Generation,” arXiv.org, 2024. https://arxiv.org/abs/2401.15884 (accessed Mar. 25, 2025).

[14] Meta Open Source, “React,” react.dev, 2025. https://react.dev/

[15] FastAPI, “FastAPI,” fastapi.tiangolo.com, 2023.

[16] LangChain, “Introduction | 🦜🔗 LangChain,” Langchain.com, 2024. https://python.langchain.com/docs/introduction/

[17] “🦜🕸️ LangGraph,” langchain-ai.github.io. https://langchain-ai.github.io/langgraph/

[18] Supabase, “The Open Source Firebase Alternative,” Supabase. https://supabase.com/

[19] “Pinecone Documentation - Pinecone Docs,” Pinecone.io, 2024. https://docs.pinecone.io/guides/get-started/overview

[20] Li, Y. (2023). A Practical Survey on Zero-Shot Prompt Design for In-Context Learning. In Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing, pages 641–647, Varna, Bulgaria. INCOMA Ltd., Shoumen, Bulgaria.

[21] M. Bahrami, M. Mansoorizadeh and H. Khotanlou, "Few-shot Learning with Prompting Methods," 2023 6th International Conference on Pattern Recognition and Image Analysis (IPRIA), Qom, Iran, Islamic Republic of, 2023, pp. 1-5, doi: 10.1109/IPRIA59240.2023.10147172.

[22] “Towards Understanding Chain-of-Thought Prompting: An Empirical Study of What Matters,” Research.google, 2023. https://research.google/pubs/towards-

understanding-chain-of-thought-prompting-an-empirical-study-of-what-matters/

[23] PrajnaAI, “Semantic Chunking in RAG: Balancing Context and Relevance,” Medium, Nov. 25, 2024. https://prajnaaiwisdom.medium.com/semantic-chunking-in-rag-balancing-context-and-relevance-2325451b4507 (accessed May 25, 2025).

[24] “Text Inspector,” Textinspector.com, 2019. https://textinspector.com/

[25] “LLM-as-a-judge: A complete guide to using LLMs for evaluations,” Evidentlyai.com, 2025. [Online] Available: https://www.evidentlyai.com/llm-guide/llm-as-a-judge#evaluation-by-criteria [Accessed May 01, 2025].