AI DOCUMENT ASSISTANT

WITH RAG AND PROMPT ENGINEERING

Peiris A.R.D

IT21249266

B.Sc. (Hons) Degree in Information Technology

Specializing in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

AI DOCUMENT ASSISTANT

WITH RAG AND PROMPT ENGINEERING

Peiris A.R.D

IT21249266

Dissertation submitted in partial fulfilment of the requirements for the Bachelor of Science (Hons) Degree in Information Technology Specializing in Software Engineering

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

Declaration of The Candidate & Supervisor

I declare that this is my own work, and this dissertation 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.

Also, I hereby grant to Sri Lanka Institute of Information Technology the non-exclusive right to reproduce and distribute my dissertation in whole or part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as article or books).

Name	Student ID	Signature
Peiris A.R.D	IT21249266	

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

……………………………… ………………………

Abstract

The integration of Artificial Intelligence (AI) in education presents transformative opportunities for addressing critical challenges in personalized learning and student support. Traditional educational environments often struggle with providing individual guidance due to the large number of students, leading to knowledge gaps and reduced engagement among some of them. This study explores the development of an innovative AI-powered Document Assistant leveraging Retrieval-Augmented Generation (RAG) and Prompt Engineering to enhance self-directed learning experiences.

By employing an intent-aware prompt engineering methodology, the chatbot can dynamically interpret and respond to student needs across various learning contexts, such as summarization, quiz generation, and code assistance. Unlike conventional zero-shot prompting approaches, this intent-aware structured prompting technique significantly improves response clarity, structure, contextual relevance, and informational depth. The proposed AI Document Assistant is designed as a scalable microservice service.

Experimental results demonstrate that the intent-aware structured prompting approach substantially outperforms traditional methods, generating more nuanced and pedagogically sound responses. This research contributes significantly to the evolving discourse on AI-driven educational tools, showcasing how sophisticated chatbot technologies can bridge learning gaps, and foster more engaging, adaptive learning environments.

The findings offer insights into effectively utilizing AI to support self-directed learning, potentially revolutionizing educational technology and ensuring students receive the targeted, adaptive support necessary for successful academic growth.

Acknowledgment

I would like to express my heartfelt gratitude to my research supervisor, Dr. Kalpani Manathunga, and my co-supervisor, Ms. Thilini Jayalath, for their unwavering guidance, encouragement, and support throughout this research endeavor. Their expertise and insightful feedback were instrumental in navigating the challenges of this project and ensuring its successful completion.

I am also deeply thankful to all the individuals who contributed their valuable insights and assistance during this study. Your collective input enriched the development and evaluation of the AI Document Assistant within the miStudy system, and I am grateful for your generosity and collaboration.

This research would not have been possible without the dedication and contributions of each of you, and I extend my sincere appreciation for your support.

Table of Contents

List of Figures

List of Tables

Abbreviations

LLM	Large Language Model
SLM	Small Language Model
NLP	Natural Language Processing
RAG	Retrieval Augmented Generation
DPR	Dense Passage Retrieval
STEM	Science, Technology, Engineering and Mathematics

1. Introduction

1.1 Background and Literature Survey

1.1.1 Knowledge Gaps Among Students

A knowledge gap signifies a lack of knowledge or understanding about a particular topic, skill, or area. The pursuit of effective educational strategies has long revealed a persistent challenge, even with advanced pedagogical techniques, classrooms exhibit a spectrum of student comprehension, where some students grasp concepts fully while others struggle to resolve their doubts. This variability poses a significant barrier to self-directed learning, where students must independently navigate material without immediate instructor support.

Research on active learning demonstrates its efficacy yet highlights its limitations in ensuring uniform understanding. Freeman et al. (2014) [1] conducted a meta-analysis of 225 studies, finding that active learning in STEM courses improved average student performance by 6% on exams compared to traditional lecturing. However, the study also revealed that 20-30% of students still scored below proficiency thresholds, often due to insufficient prior knowledge or limited engagement.

Similarly, Crouch and Mazur (2001) [2] explored peer instruction in physics classrooms, reporting that while 60-80% of students achieved conceptual mastery after interactive sessions, 20-40% consistently failed to resolve misconceptions, even with peer support.

These findings indicate that while active learning enhances overall comprehension, it does not eliminate the divide between students who understand and those who do not, indicating a knowledge gap in these students.

1.1.2 Large Language Models and Small Language Models

LLMs and SLMs represent two ends of the spectrum in natural language processing, driven by transformer architectures [3]. LLMs, such as GPT-3 [4], LLaMA, and BERT, are characterized by their massive scale, often billions of parameters, trained on vast, diverse datasets like Common Crawl or Wikipedia. This scale enables remarkable capabilities in text generation, question-answering, and zero-shot learning, making them versatile tools for applications like educational chatbots. Their strength lies in capturing broad linguistic patterns and world knowledge, but they come with high computational costs, latency, and a tendency to hallucinate facts due to static training data.

In contrast, SLMs, typically ranging from millions to a few billion parameters, emerged as efficient alternatives, designed to retain core NLP capabilities with reduced resource demands. Originating from efforts to distil or prune larger models [5] , SLMs prioritize speed, deployability on edge devices, and energy efficiency, often at the cost of depth in reasoning or domain knowledge. The trade-off between LLMs and SLMs is particularly relevant in self-directed learning contexts, where a chatbot must balance performance with practicality. The integration of techniques like RAG and prompt engineering further complicates this landscape, as model size influences how effectively these methods enhance response quality and adaptability.

Both LLMs and SLMs have been paired with RAG and prompt engineering to enhance performance. Lewis et al. (2020) [6] applied RAG to BART (an LLM with 400 million parameters), improving exact match scores by 10-15% on Natural Questions by grounding generation in retrieved documents. Prompt engineering studies, such as Wei et al. (2022) [7] on chain-of-thought prompting, showed LLMs gaining 25% accuracy on reasoning tasks, while SLMs saw smaller gains (10-15%) due to limited contextual depth. This suggests LLMs benefit more from these techniques, but SLMs could suffice with optimized design.

LLMs offer superior performance and adaptability for complex tasks, enhanced by RAG and prompt engineering, but at high computational cost; SLMs prioritize efficiency and deployability, yet lack depth for knowledge-intensive applications. For a self-directed learning chatbot, LLMs could leverage RAG to ground responses in student materials and prompt engineering to tailor outputs, while SLMs might enable real-time, lightweight interaction. However, integrating these approaches with intent categorization remains underexplored, highlighting a need for research balancing scale, efficiency, and educational utility.

1.1.3 Retrieval Augmented Generation

The seminal paper by Lewis et al. (2020) [6] introduced RAG as a novel paradigm. The study combined a DPR retriever with a BART generator, training the system end-to-end on tasks like open-domain question answering and fact verification. On datasets like Natural Questions, RAG achieved a 10-15% improvement in exact match scores over standalone generative models, demonstrating its ability to leverage external knowledge effectively. The authors highlighted two variants, RAG-Sequence (generating a single output from multiple documents) and RAG-Token (generating token-by-token with dynamic retrieval), underscoring flexibility in design. This work established RAG as a benchmark for combining retrieval and generation, with immediate implications for educational tools requiring accurate, document-grounded responses.

RAG has emerged as a promising approach to mitigating hallucinations and enhancing the factual accuracy of LLM outputs. By integrating external knowledge bases during inference, RAG provides LLMs with up-to-date information, reducing the risk of hallucinations caused by their inherent knowledge limitations [8]. Previous studies on RAG in education have shown positive outcomes, demonstrating that RAG enhanced LLMs can provide accurate and contextually relevant responses to student queries. Expert evaluations of AI tutor, OwlMentor [9], found that integrating course literature through RAG improved the accuracy and relevance of responses, supporting students in understanding complex academic material. Figure 1, from previous research [10], indicates that incorporating context significantly improves the accuracy of LLM responses. When provided with context, models accurately navigated the text to generate correct responses approximately 94% of the time. In contrast, without context, only 7.31% of responses were deemed accurate, with the majority consisting of hallucinations or unhelpful outputs [10]. Given these advantages, RAG is particularly well-suited for educational settings, where factual accuracy and reliable information are essential. By incorporating external knowledge sources, RAG-powered systems can deliver up-to date, context-rich information, fostering self-directed learning and enhancing student engagement.

Condition	Items	Accurate	Partial	Hallucinations
CONTEXT	1,125	**93.95%**	01.68%	02.04%
NO CONTEXT	793	7.31%	08.44%	55.35%

Figure 1 Comparison Context vs No Context

1.1.4 Prompt Engineering

Prompt engineering has emerged as a critical discipline in the field of artificial intelligence (AI), particularly with the advent of large language models (LLMs) like GPT-3, BERT, and their successors. It refers to the deliberate design and optimization of input prompts, textual instructions or questions, to elicit desired responses from AI systems. Unlike traditional machine learning, where models are fine-tuned with extensive labelled datasets, prompt engineering leverages the pre-trained knowledge of LLMs, steering their behaviour through carefully crafted inputs without altering their underlying parameters. This approach has gained prominence due to its efficiency, accessibility, and ability to adapt general-purpose models to specific tasks, such as question-answering, text generation, or educational support.

Brown et al. (2020) [4] showcased this with GPT-3, illustrating how prompts could shift outputs from factual answers to creative narratives with minimal retraining. In educational contexts, prompt engineering holds promise for creating responsive tools, like chatbots, that clarify student doubts by interpreting queries and tailoring responses dynamically. Yet, its effectiveness hinges on overcoming challenges like ambiguity, model sensitivity to phrasing, and the need for domain-specific expertise, setting the stage for ongoing research and innovation.

Prompt engineering's adaptability has been explored in domain-specific contexts, including education. Lester et al. (2021) [11] extended prompt tuning to few-shot learning scenarios, showing that optimized prompts enabled LLMs to rival fine-tuned models in question-answering tasks, with F1 scores increasing by 8-12%, F1 score is a measure of a model's accuracy that balances precision and recall, particularly useful in classification or information retrieval tasks where both false positives and false negatives matter.

In an educational setting, Deng et al. (2022) [12] investigated prompts for tutoring systems, finding that contextual prompts (e.g., "Explain this concept from the uploaded physics textbook") improved student comprehension by 20% over generic responses, based on user studies with 50 learners. However, the study noted that poorly designed prompts led to irrelevant or overly verbose answers, indicating a need for domain-aware prompt strategies, crucial for a chatbot processing student-uploaded materials.

1.2 Research Gap

Table 1 Research Gap

Aspect	**Prompt Engineering** [4] [11] [7]	**RAG** [15] [14] [6]	**Traditional Education & Educational AI** [9] [1] [2] [13].	**Proposed System**
Focus on Educational Chatbots	x	x	✓	✓
Dynamic Prompt Adaptation to Intent	x	x	x	✓
Support for Varied Query Types	x	x	x	✓
Integration of RAG and Prompt Engineering	x	x	x	✓
Flexibility for Self-Directed Learning	x	x	x	✓

The integration of Retrieval-Augmented Generation (RAG) and prompt engineering has shown significant promise in enhancing the capabilities of AI systems for knowledge-intensive and context-specific tasks, including educational applications. However, despite advancements in these areas, several critical gaps remain unaddressed in the literature, particularly in the context of self-directed learning chatbots tailored to diverse student queries. These gaps present opportunities for innovation, which this research seeks to explore through a novel AI chatbot design that combines intent categorization with adaptive prompt selection and RAG-based retrieval.

First, while prompt engineering has evolved from manual crafting [4] to systematic optimization techniques like prompt tuning [11] and chain-of-thought prompting [7], its application in educational chatbots remains underexplored. Existing studies focus primarily on general-purpose or domain-specific question-answering [13], with little attention to dynamically adapting prompts based on the user's intent. For instance, a student requesting a code snippet requires a different response structure (e.g., syntax-focused, executable examples) than one seeking a summary or comparison, yet the literature lacks frameworks for intent-driven prompt selection. This gap limits the ability of chatbots to provide tailored, task-specific support, a critical need in self-directed learning where students pose varied queries.

Second, RAG has demonstrated success in grounding responses in external knowledge sources, improving factual accuracy over standalone LLMs. However, its application to educational contexts often assumes a static knowledge base (e.g., Wikipedia or pre-indexed corpora) rather than dynamic, user-provided documents like student-uploaded notes or textbooks. While Petroni et al. (2021) [14] applied RAG techniques to scientific QA, their focus was on predefined datasets, not real-time integration of personalized materials. Furthermore, RAG research rarely addresses how retrieved content should be processed differently based on query type (e.g., summarizing retrieved text vs. extracting code snippets), leaving a gap in adapting retrieval outputs to diverse educational intents.

Third, the combination of RAG and prompt engineering has been studied in isolation but rarely as an integrated system for educational support. For example, Gao et al. (2023) [15] enhanced RAG with real-time retrieval, yet these efforts do not converge on a unified approach that leverages both to interpret and respond to multifaceted student needs. The lack of intent categorization in such systems means that responses may not align with the student's specific goal—whether it's understanding a concept, generating a quiz, or comparing ideas—potentially reducing effectiveness in self-directed learning environments where immediate, relevant feedback is crucial.

Finally, existing educational AI tools, such as intelligent tutoring systems often target predefined curricula or domains, neglecting the flexibility required for self-directed learners who engage with diverse, unstructured materials. Traditional learning strategies [1] highlight persistent comprehension gaps among students, yet current chatbot solutions fail to bridge these through personalized, intent-aware, and document-grounded responses. The absence of a framework that categorizes student intent, selects task-specific prompts, and retrieves relevant content from user-provided documents represents a significant research gap—one that this project aims to address.

This study fills these gaps by proposing an AI chatbot that: (1) categorizes student intent (e.g., code snippets, quizzes, summaries, comparisons) within the prompt engineering process, (2) dynamically selects predefined prompts tailored to each intent, and (3) employs RAG to retrieve and generate responses grounded in student-uploaded documents. By integrating these elements, the proposed system offers a novel solution to enhance the precision, relevance, and adaptability of AI support in self-directed learning, addressing limitations in prior work and paving the way for more effective educational tools.

1.3 Research Problem

The development of an AI chatbot that supports self-directed learning through Retrieval-Augmented Generation (RAG) and prompt engineering presents distinct research problems that must be addressed to ensure its effectiveness. These challenges arise from the need to interpret diverse student queries, generate contextually appropriate responses, and select an optimal model architecture to balance performance and practicality. By focusing on the generation of tailored, intent-specific responses, accurate categorization of student intent, and model selection, this study aims to overcome limitations in existing systems and deliver a robust educational tool. The following research problems guide this investigation:

1. **Generation of Tailored, Intent-Specific Responses**

   A core challenge is crafting responses that precisely align with the student's categorized intent, such as code snippets, quizzes, summaries, or comparisons, while leveraging RAG and predefined prompts. LLMs often produce verbose or hallucinated outputs and even with RAG's grounding in retrieved content, ensuring task-specific structure remains difficult. For instance, a "code snippet" intent requires syntax-focused output, whereas a "quiz" demands interactive questions, yet prompt and retrieval noise can misalign responses. This problem necessitates designing prompts and adapting RAG's generative process to consistently deliver relevant, intent-driven answers from student-uploaded documents.

2. **Accurate Categorization of Student Intent**

   Effectively identifying the intent behind a student's query, whether it seeks a code snippet, quiz, summary, comparison, or other output, is critical to triggering the appropriate predefined prompt and response strategy. Student queries can be informal, ambiguous, or context-dependent (e.g., "How does this work?" could imply a summary or code explanation), complicating intent classification. Existing prompt engineering studies [11] focus on optimizing responses rather than categorizing user intent, while educational chatbots lack frameworks for dynamic intent detection across diverse tasks [13]. This problem requires designing a prompt that will instruct the LLM to classify the student's intent to ensure the chatbot interprets and responds to student needs accurately.

3. **Model Selection**

Choosing between LLMs and SLMs poses a significant problem, as each offers trade-offs affecting the chatbot's performance, scalability, and deployment. LLMs excel at complex reasoning and knowledge-intensive tasks, benefiting from RAG and prompt engineering. lacks depth in reasoning and context understanding [5]. This problem involves determining the optimal model size to balance response quality with practical constraints when processing student-uploaded documents with RAG.

1.4 Research Objectives

This study aims to develop an AI chatbot that enhances self-directed learning by leveraging Retrieval-Augmented Generation and prompt engineering to provide accurate, pedagogically sound responses grounded in student-uploaded documents. The objectives outlined below address the challenges of intent-driven interaction, model optimization, and effective document processing, ensuring the chatbot meets the diverse needs of learners efficiently and educationally.

1.5 Main Objective

1. **Facilitating self-directed learning by developing an AI Document Assistant through a chatbot interface**
   The primary goal of is to create an AI-powered Document Assistant that enhances self-directed learning as a component of miStudy This chatbot interface will enable students to interact with educational content more effectively, providing immediate assistance, clarifying doubts, and offering personalized support to facilitate independent learning and exploration.

1.6 Sub Objectives

1. **To Craft Prompts for Intent Categorization, prevent out of scope responses, and ensure pedagogical responses**

The first objective is to design and optimize prompts that enable accurate categorization of student intent (e.g., code snippets, quizzes, summaries, comparisons), restrict responses to relevant content, and deliver explanations in a pedagogical manner. This involves developing a prompt engineering framework that classifies intents from diverse queries, incorporates constraints to avoid out-of-scope or irrelevant and adopts a teaching-oriented tone, clear, structured, and supportive. The goal is to ensure the chatbot interprets student needs correctly, stays within the context of uploaded documents via RAG, and enhances learning through instructive, intent-specific responses.

2. **To select and optimal language model for the chatbot**

   The second objective is to evaluate and choose between LLMs and SLMs to determine the best fit for the chatbot's performance and deployment requirements. This entails benchmarking models like GPT-3 (LLM) and DistilBERT (SLM) with RAG and prompt engineering, assessing metrics such as response accuracy, inference speed, and resource efficiency.

3. **To Determine optimal chunking and overlapping size for document processing**

   The third objective is to establish the most effective chunking strategy and overlapping size for processing student-uploaded documents within the RAG framework. This involves experimenting with different chunk sizes and overlap sizes to segment documents into retrievable units, ensuring that retrieved content captures sufficient context for intent-specific responses. The goal is to optimize retrieval precision and generation coherence, minimizing information loss or noise, critical for handling diverse educational materials in real time

2. Methodology

The AI Document Assistant is an integrated microservice of miStudy, a system that adopts a modular architecture to deliver personalized and adaptive learning experiences. A detailed overview of the system is described in the figure below (see Figure 1), illustrating how the AI Document Assistant collaborates. miStudy integrates multiple components each addressing specific aspects of the learning journey, such as cognitive workload management, quiz generation, mind map creation, and conversational AI Document support leveraging advanced technologies like machine learning, natural language processing (NLP), and Retrieval-Augmented Generation (RAG). The AI Document Assistant specifically enhances this ecosystem by processing student-uploaded documents and providing intent-driven, pedagogically sound responses.

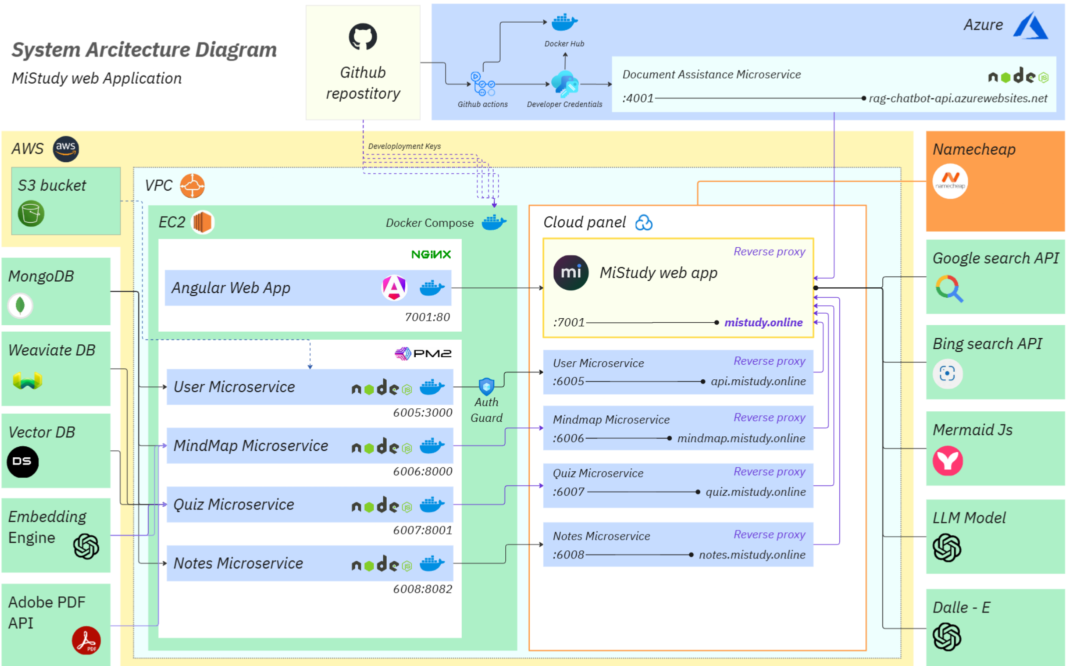

Figure 2 miStudy System Diagram

2.1 System Overview

Figure below describes a high-level overview of the AI Document Assistant system. When a user uploads a lecture document to the system and interacts with it, tokens that are matching the student's query will be retrieved from the vector store by performing a similarity search, and the result will be queried to the LLM. The response will then be displayed to the student through the chat interface.

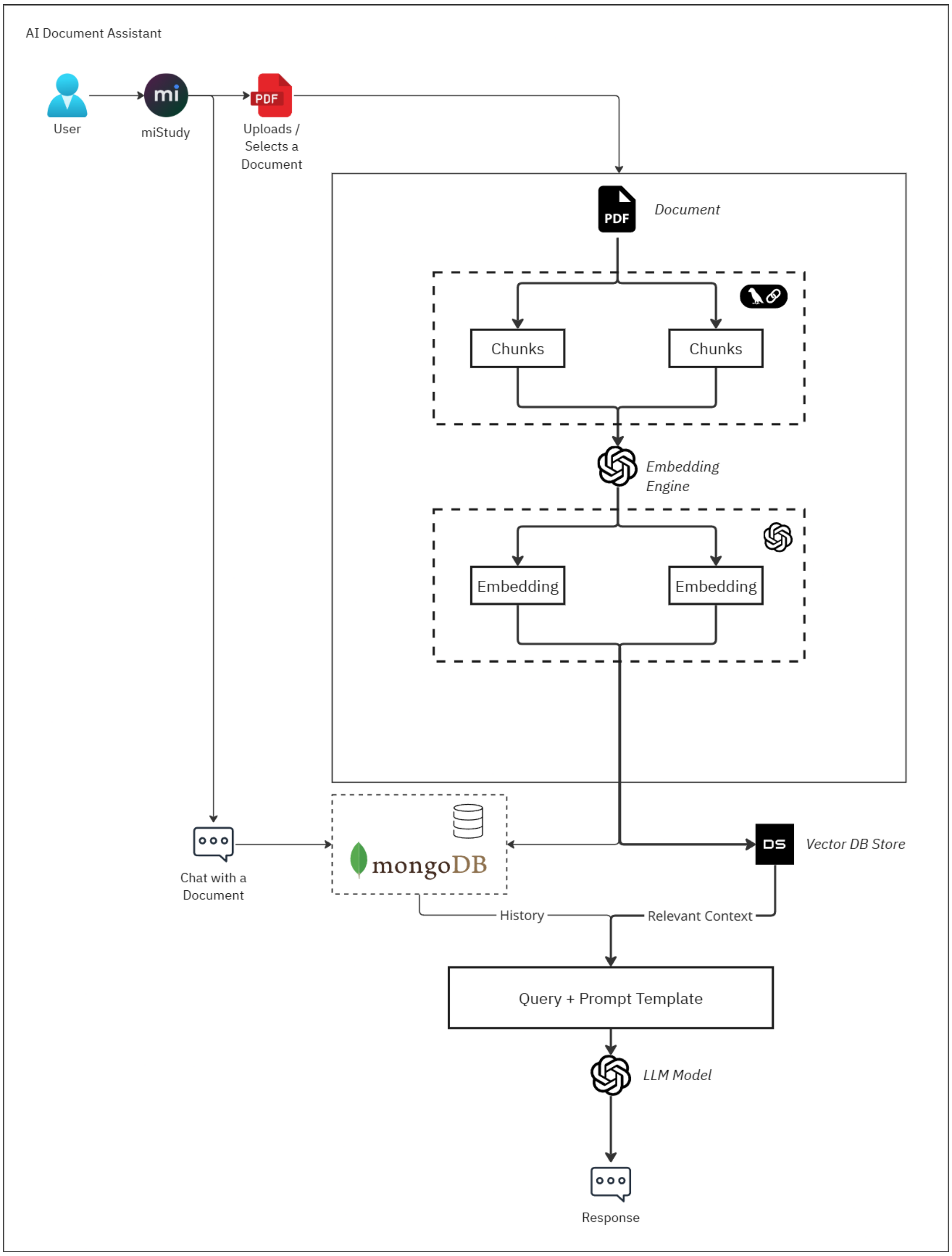

Figure 3 AI Document Assistant - System Diagram

a. Document Processing and Text Extraction

The process begins with the ingestion of student-uploaded documents. These documents are processed by extracting their text content using LangChains PDFLoader interface. This step ensures that all relevant textual information such as paragraphs, code blocks, or headings is converted into a machine-readable format, stripping away non-essential elements like images or formatting artifacts unless contextually significant. The extracted text serves as the raw input for subsequent processing, enabling the system to work with diverse educational materials.

b. Chunking Process

The extracted text content then undergoes a chunking process to segment it into manageable units for retrieval and generation. Each document is divided into chunks based on a predefined size (1536 tokens) with an overlap (300 tokens) to preserve context across segment boundaries. This chunking strategy, determined through experimentation, balances granularity for precise retrieval with sufficient context for coherent generation, addressing challenges like information loss or noise.

c. Embedding and Vector Storage

The chunked content is then processed by OpenAI's Embedding Engine, which converts each chunk into a dense vector representation. This embedding process captures semantic meaning, enabling efficient similarity-based retrieval. The resulting vectors are stored in a vector store, AstraDB. During operation, when a student submits a query, the system embeds the query using the same engine, retrieves the top 3 most relevant chunks from the vector store via cosine similarity, and passes them to the RAG pipeline for response generation.

d. Integration with Prompt Template and LLM

A prompt template is used with the OpenAI's ChatOpenAI interface before feeding them to the LLM, gpt-4o-mini to generate a tailored, pedagogically sound response. The prompt is designed by using prompt engineering techniques such as few shot and chain of thoughts.

2.2 Requirement and Data Gathering

2.2.1 Online Survey

To develop the student centric application, an online survey was conducted focusing university students to understand their experience on learning management systems and the improvements they would suggest a modern learning management system should have. Based on these responses, features of the miStudy application were determined and possible challenges were evaluated. Figures below describes few of the responses received. The core features of the system were determined to be, Mind Map Generation, Personalised Quiz Generation, Personalized Study Techniques Integration and UI Optimization and the AI Document Assistant.

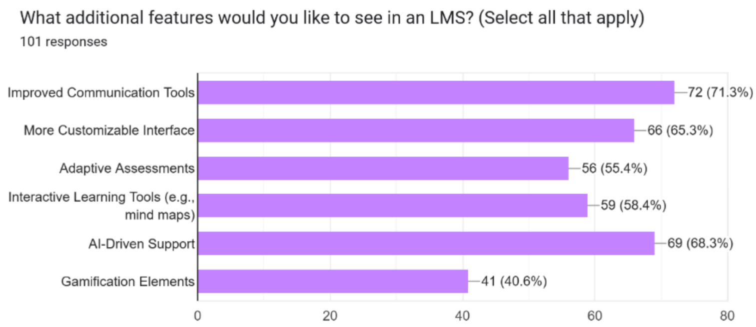

Figure 4 Survey Responses 1

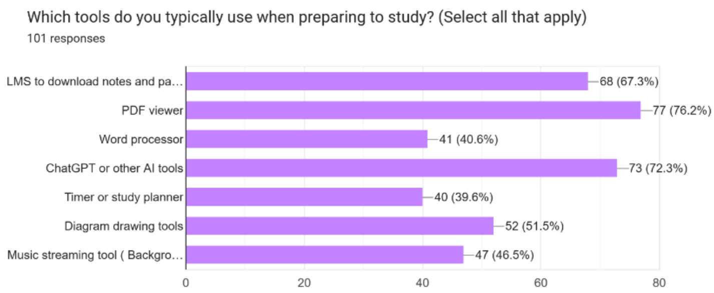

Figure 5 Survey Responses 2

2.2.2 Official Documentations

The development of the AI Document Assistant within miStudy relies heavily on leveraging official documentations from LangChain, OpenAI, and prompt engineering resources. These sources provide critical technical guidance, best practices, and foundational knowledge for integrating Retrieval-Augmented Generation (RAG), intent categorization, and pedagogical response generation.

- **LangChain Documentation** - The official LangChain Documentation (js.langchain.com) serves as the primary resource for implementing the RAG framework, similarity search and prompt engineering by providing interfaces of OpenAI and AstraDB.
- **OpenAI Documentation** - OpenAI's developer platform (platform.openai.com) provides comprehensive API documentation essential for integrating models like gpt-4o-mini into the AI Document Assistant. The documentation also includes best practices for prompt engineering, such as structuring inputs with clear instructions and context. Additionally, OpenAI's embeddings API (text-embedding-ada-002) is referenced for vectorizing document chunks, supporting RAG's retrieval process within miStudy's architecture and OpenAI's moderation engine offering interface for moderation of queries.
- **Prompt Engineering Documentation** – The "Prompt Engineering Guide" by DAIR-AI (promptingguide.ai) provides insight on prompt engineering techniques such as zero-shot, chain of thoughts, few-shots which was used in this study to craft the intent categorising prompt.

2.3 Tools and Technology Selection

This phase evaluated options based on performance, scalability, ease of integration, and alignment with the research objectives crafting prompts, selecting a model, and optimizing document chunking. The chosen tools TypeScript, LangChainJS, OpenAI, AstraDB, ClickUp, Postman, Azure, Docker, and GitHub contributes to the workflow from development to deployment.

1. **NodeJs**
    - **Purpose**: Provides a run time environment for executing TypeScript and JavaScript code

- **Rationale**: Node.js was selected as the runtime for its asynchronous, event-driven architecture, ideal for handling real-time query processing and API calls in the AI Document Assistant. Its extensive ecosystem (e.g., npm packages and seamless integration with TypeScript via ts-node enable rapid prototyping and deployment. Node.js's lightweight nature supports miStudy's need for efficient, server-side execution of LangChainJS and RAG workflows

2. **TypeScript**

    - **Purpose**: Primary programming language for developing the chatbot's frontend and backend components.

    - **Rationale**: Provides type safety unlike loosely typed JavaScript.

3. **LangChainJS:**

    - **Purpose:** Framework for implementing RAG and prompt engineering.

    - **Rationale**: LangChainJS, the JavaScript/TypeScript version of LangChain, was chosen for its support of RAG workflows and prompt management (e.g., PromptTemplate). It enables integration of document retrieval and generation, leveraging TypeScript's type safety. Its modular design aligns with miStudy's architecture, allowing intent-specific prompts to be crafted and tested efficiently.

4. **OpenAI**

    - **Purpose:** Provides the language model and embedding capabilities.

    - **Rationale**: OpenAI's API was selected as it provides a text embedding engine which other competitors do not and the huge developer community around it.

5. **AstraDB**

    - **Purpose**: Vector database for storing and querying document embeddings.

- **Rationale**: AstraDB, a managed Cassandra-based database, was chosen over alternatives like Weaviate or Pinecone for its scalability, detailed documentation, serverless architecture, and TypeScript compatibility. It efficiently stores high-dimensional vectors from OpenAI embeddings, enabling fast similarity searches critical for RAG's real-time retrieval in the AI Document Assistant component. Its cloud-native design supports Azure integration and large-scale student document indices.

6. **ClickUp**
   - **Purpose**: Project management and collaboration tool.
   - **Rationale**: ClickUp was selected to manage tasks, timelines, and documentation during development, ensuring alignment with research milestones. Its Kanban boards facilitate team coordination, while its flexibility supports tracking requirements gathering enhancing operational efficiency.

7. **Postman**
   - **Purpose**: API testing and debugging tool.
   - **Rationale**: Postman was chosen to test and validate REST API endpoints. It simplifies debugging of the system during the absence of the forntend.

8. **Azure:**
   - **Purpose**: Cloud platform for deployment and scaling.
   - **Rationale**: Microsoft Azure was selected for its PaaS offering, App Services, which simplifies the CI/CD process with just few simple clicks and configurations.

9. **Docker:**

- **Purpose**: Containerization for consistent development and deployment.
- **Rationale**: Docker was chosen to package the TypeScript application into a portable container and push to DockerHub, from where the App Services will pull the docker image for deployment.

10. **GitHub:**
    - **Purpose**: Version control and collaboration platform.
    - **Rationale**: GitHub (github.com) was selected for its robust version control, enabling tracking of TypeScript code, prompt iterations, and configuration files. Its integration Azure Pipelines supports automated CI/CD workflows ensuring research reproducibility throughout the project lifecycle.

This suite of tools and technologies provides a solid foundation for the AI Document Assistant, enabling document processing, intent categorization, and response generation within miStudy's ecosystem. Their interoperability ensures the system meets technical, operational, and educational goals, validated through subsequent development and deployment phases.

2.4 Implementation

2.4.1 RAG Implementation

The initial step in implementing RAG for the AI Document Assistant focuses on processing student-uploaded PDF documents to extract text content, a prerequisite for retrieval and generation. LangChainJS's PDFLoader class was utilized to parse PDFs and extract text, enabling the system to work with diverse educational materials within miStudy's modular architecture.

1. The PDF Loader class from LanchainJs was used to load and extract text from PDF documents.
2. Large documents were split into chunks using LangChainJS's RecursiveCharacterTextSplitter ensuring manageable units for RAG retrieval while maintaining context.

2.4.2 Integration with AstraDB

The next step integrates the extracted and chunked text with AstraDB, implementing an embedding model to convert chunks into vectors and storing them for efficient retrieval. LangChainJS's interfaces facilitate this process, connecting OpenAI's embedding capabilities with AstraDB's vector storage within the TypeScript/Node.js environment.

1. The Embedding Model, text-embedding-3-small, processed the chunked document objects from PDFLoader passing them through OpenAIEmbeddings. Each chunk is embedded into a 1536-dimensional vector, capturing semantic meaning.
2. An AstraDB collection was created via the Astra dashboard and the Embedded vectors are stored using AstraDB interface provided by LangChain with userId and documentId as metadata.

```
const embeddings = new OpenAIEmbeddings({
    model: process.env.EMBEDDING_MODEL,
    openAIApiKey: process.env.OPENAI_API_KEY!,
});

const astraConfig: AstraLibArgs = {
    token: process.env.ASTRA_DB_TOKEN!,
    endpoint: process.env.ASTRA_DB_ENDPOINT!,
    collection: process.env.ASTRA_DB_COLLECTION!,
    collectionOptions: {
        vector: {
            dimension: 1536,
            metric: "cosine",
        },
        indexing: {
            allow: ["userId", "documentId"],
        }
    }
}
let astraDbVectorStore: AstraDBVectorStore | null = null;

export const connectToAstraDB = async (): Promise<void> => {
    try {
        astraDbVectorStore = await AstraDBVectorStore.fromExistingIndex(embeddings, astraConfig);
        logger.info("Connected to AstraDB.");
    } catch (error) {
        logger.error(`Error connecting to AstraDB: ${error}`);
        throw new Error("Unable to initialize vector store.");
    }
};
export const getVectorStoreInstance = (): AstraDBVectorStore => {
    if (!astraDbVectorStore) {
        throw new Error("Vector store is not initialized. Call connectToAstraDB first.");
    }
    return astraDbVectorStore;
};

export const initializeVectorStore = async (
    documentCloudUrl: string,
    userId: string,
    documentId: string
): Promise<void> => {
    const existingRecords = await checkExistingRecords(userId, documentId);

    if (existingRecords.length > 0) {
        logger.info(`Vector store already initialized for userId: ${userId}, documentId: ${documentId}`);
        return;
    } else {
        const pdfPath = await downloadFile(documentCloudUrl, documentId);
        const extractedText = await extractTextFromPDF(pdfPath);
        const documents = await createDocumentsFromText(extractedText, userId, documentId);
        await AstraDBVectorStore.fromDocuments(documents, embeddings, astraConfig);

        logger.info(`Initialized vector store for userId: ${userId}, documentId: ${documentId}`);
    }

};

const checkExistingRecords = async (userId: string, documentId: string): Promise<any[]> => {
    try {
        const vectorStore = getVectorStoreInstance();
        return await vectorStore.similaritySearch("", 1, {userId, documentId});
    } catch (error) {
        logger.info(`Checking existing records: ${error}`);
        return [];
    }
};
```

Figure 6 AstraDB Functionality

2.4.3 Integration with GPT-4o-mini

The final step integrates GPT-4o-mini, OpenAI's lightweight model, with the RAG pipeline, performing similarity searches using LangChainJS's AstraDB interface to retrieve relevant content and generate responses. A similarity search of the query is performed against the document's vector store. This retrieves the most relevant chunks from AstraDB, this content is concatenated into a string and fed into the model along with the Prompt Template.

2.4.4 Prompt Design

Designing the prompts for the AI Document Assistant was an incremental process aimed at ensuring the system delivers intent-specific, pedagogically sound, and document-grounded responses within miStudy's framework. The process began with a zero-shot prompt to establish a baseline, leveraging the model's pre-trained capabilities, and evolved through iterations to address the identified intents Summarization, Code Snippets, Comparison, Quiz Generation, Simplification, and Explanation—while integrating with Retrieval-Augmented Generation (RAG) and LangChainJS.

A zero-shot approach was chosen initially to test GPT-4o-mini's inherent understanding without examples, minimizing token usage and aligning with LangChainJS's documentation for efficient prompt design.

- **Zero Shot Prompt -** `You are a study assistant helping with questions specifically related to the provided document. Answer queries only within the provided context. If the query is unrelated, state that the query is outside the document's scope politely`

  Purpose

  - Role Definition: The prompt instructs the model to assume the role of a study assistant, aligning with miStudy's educational focus and setting a supportive tone for self-directed learning.
  - Scope Control: It restricts responses to the provided document's context, leveraging RAG's retrieved chunks to reduce hallucination and ensure relevance.

- Politeness: The instruction to politely decline out-of-scope queries enhances user experience, maintaining a pedagogical demeanour.

Limitations

- As a general prompt, it lacked specificity for handling diverse intents such as code blocks, quizzes, summaries etc.

The solution to this limitation was to design an prompt that will identify the intent and use the respective prompt to produce a structured response.

- **Intent Aware Structure Prompt -** To craft prompts for intent categorization, prevent out of scope responses and ensure pedagogical delivery, identifying common student intents was a foundational step. Through an analysis of educational query patterns from forums, the main intents identified were Summarization, Code Snippets, Quiz Generation, Comparisons, Simplification and Explanation. Each intent requires a tailored prompt to interpret the student's need via RAG from uploaded documents and deliver answers clearly in a teaching-oriented manner. The prompt is a combination of chain of thoughts and few shot prompt engineering techniques. The crafted prompts for each intent are detailed below:

1. **Summarization Prompt**

```
`**Enhanced Summary**:

  - Instead of a generic summary, categorize key points
into sections for clarity.

  - Always start with a **one-line overview** before
listing key points.

  - Format as follows:

    **Summary:**

    - **Main Idea**: [One-sentence summary]

    - **Key Sections**:

      - **[Section 1]**: [Summary]

      - **[Section 2]**: [Summary]

  - Example:

    *"Summarize the document"* →
```

```
**Summary:**

- **Main Idea**: This document explains the process of machine learning model training.

- **Key Sections**:

  - **Data Collection**: Gathering and preprocessing raw data.

  - **Model Training**: Applying algorithms and evaluating performance.

  - **Deployment**: Integrating the trained model into applications. `
```

The one-line “Main Idea” provides a quick grasp of the document’s essence, aiding comprehension. Categorising into key sections organizes the content logically enhancing readability and retention for students. The structured format and examples guide the model to product consistent clear outputs.

2. **Code Snippets Prompt**

```
` **Code Snippet with Explanation**:
   - Instead of just giving a snippet, provide
**comments** and an **example usage**.
   - Format:
     **Code Snippet:**
     \`\`\`
     # Task: [Explain the purpose of the code]

     def function_name(param1, param2):
         """Short description of what this function
does."""
         result = param1 + param2
         return result

     # Example Usage
     output = function_name(5, 10)
     print(output) # Expected output: 15
     \`\`\`
   - Example:
     *"Provide a Python function for addition"* →
     \`\`\`
     # Task: Add two numbers and return the sum
```

```
def add_numbers(a, b):
    """Returns the sum of two numbers."""
    return a + b

# Example Usage
result = add_numbers(3, 7)
print(result) # Output: 10
\`\`\`
```

```
   - Also include **edge cases and performance
considerations** when relevant. `
```

The prompt instructs the model to provide comments together with the code to ensure that the student understands the code snippet. Additionally, if relevant, to provide edge cases and performance considerations.

3. Comparison Prompt

```
` **Detailed Comparison Table**:
   - If the student asks for a comparison or table,
generate a Markdown table using \`|\` and \`---\`.
   - For example: "Create a table comparing X and Y" →
"Comparison Table:\n| X | Y |\n|---|---|\n| Value 1 |
Value 2 |",
   - Go beyond simple tables—add a **brief
introduction** and ensure each row has meaningful
comparisons.
   - Always provide column headers and ensure all values
are relevant.
   - Format:
     **Comparison Table:**

     | Feature      | Option A   | Option B   |
     |------------|----------|----------|
     | Criteria 1 | Detail     | Detail     |
     | Criteria 2 | Detail     | Detail     |

   - Example:
     *"Compare X and Y in a table"* →
     **Comparison Table:**

     | Feature      | X             | Y             |
     |------------|------------|------------|
     | Speed        | Fast          | Moderate      |
```

```
| Accuracy    | High          | Medium        |

_X is generally faster but Y provides better
balance in complex cases. _
```

The prompt provides and example for a comparison question and instructs to provide a brief introduction and to ensure that the row contains meaningful content.

4. Quiz Generation Prompt

```
` *Engaging Quiz Creation**:
   - Instead of plain questions, structure them into
**difficulty levels** and provide **multiple-choice
options**.
   - Include an **answer key** at the end.
   - Format:
     **Quiz:**

     **Beginner:**
     1. What is X?
        - A) Option 1
        - B) Option 2
        - C) Option 3
        - D) Option 4

     **Intermediate:**
     2. How does Y relate to Z?
        - A) Explanation 1
        - B) Explanation 2
        - C) Explanation 3
        - D) Explanation 4

     **Advanced:**
     3. Given ABC, what is the expected result?

     _Answer Key:_

   - Example:
     *"Create a quiz on machine learning"* →
     **Quiz:**

     **Beginner:**
     1. What is supervised learning?
```

```
        - A) A technique for data analysis
        - B) A learning method with labeled data
        - C) Training without any labels
        - D) None of the above

     **Intermediate:**
     2. Which algorithm is best for image
classification?
        - A) KNN
        - B) SVM
        - C) CNN
        - D) Decision Tree

     **Advanced:**
     3. Explain backpropagation in deep learning.

     _Answer Key: B, C, Backpropagation is a powerful
algorithm in deep learning, primarily used to train
artificial neural networks, particularly feed-forward
networks_`
```

The quiz generation prompt instructs the model to provide quizzes of different difficulty levels from beginner to advanced together with the answer key. This way the student can evaluate the understanding level.

5. Simplification Prompt

```
` **Breaking Down Complex Topics**:
   - Instead of just simplifying, provide **three levels
of explanation**:
     1. **One-sentence summary**
     2. **Simple explanation (for beginners)**
     3. **Technical breakdown (for advanced
students)**
   - Format:
     **Simplified Explanation:**
     - **Simple summary**: [One-sentence answer]
     - **For Beginners**: [Simple explanation]
     - **For Experts**: [Detailed explanation with
terminology]
   - Example:
     *"Simplify encapsulation"* →
```

```
     **Simplified Explanation:**
     - **Simple summary**: Encapsulation is a way to
restrict access to certain components of an object.
     - **For Beginners**: It means keeping the internal
details of an object hidden from the outside world, only
exposing what is necessary.
     - **For Experts**: Encapsulation is a fundamental
principle of object-oriented programming that involves
bundling the data (attributes) and methods (functions)
that operate on the data into a single unit or class and
restricting access to some of the object's components to
ensure controlled interaction and maintain integrity.
`
```

As observed in the prompt, the instruction is to break down complex topics into three levels of explanation. For beginners a simple explanation in everyday language, for experts, a technical breakdown and a one sentence overview as a common.

6. Explanation Prompt

```
`**In-depth Concept Explanation**:

   - Structure explanations like an article with
**headings**, **examples**, and **use cases**.

   - Format:

     ## Explanation of [Concept]

     **Introduction**

     [Short overview]

     **Detailed Breakdown**

     - **Step 1**: [Explain]

     - **Step 2**: [Explain]

     **Real-world Example**

     - [Practical example of usage]

     **Why It Matters**

     - [Significance and impact]

   - Example:
```

```
*"Explain prompt engineering"* →

## Explanation of Prompt Engineering

**Introduction**

Prompt engineering involves designing and refining prompts to effectively guide AI models in generating desired outputs.

**Detailed Breakdown**

- **Step 1**: Identify the task or goal for the AI model.

- **Step 2**: Craft a clear and specific prompt that provides context and instructions.

- **Step 3**: Iterate and refine the prompt based on the model's responses to improve accuracy and relevance.

**Real-world Example**

- Used in developing chatbots to ensure they provide accurate and helpful responses to user queries.

**Why It Matters**

- Essential for maximizing the performance and utility of AI models in various applications, from customer support to content generation`
```

The prompt instructs to provide an in-depth explanation together with real world examples and why it matters.

2.5 Deployment Process

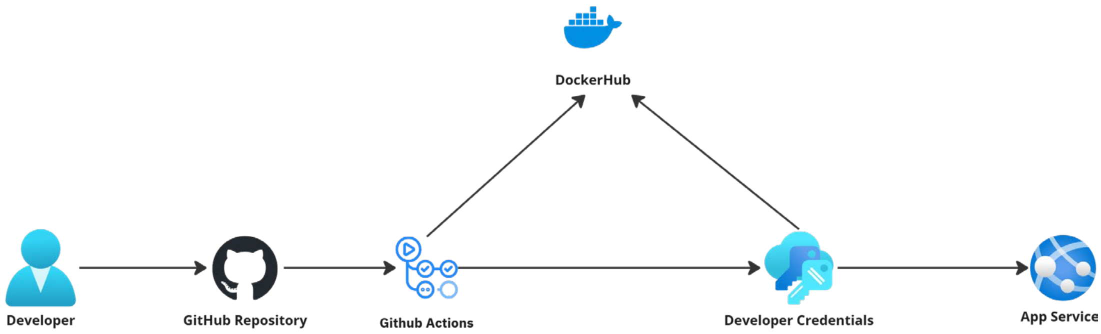

Figure 7 Development Process

The deployment of the AI Document Assistant microservice within miStudy was designed to ensure scalability, reliability, and automation, leveraging Docker for containerization and Microsoft Azure for cloud hosting. The process involved dockerizing the TypeScript/Node.js application, pushing the Docker image to Docker Hub, and configuring Azure services Resource Group, App Service Plan, and App Service with GitHub and Docker integration to create an automated deployment pipeline. This approach enables continuous deployment, where code pushes to GitHub trigger automatic updates to the live system, supporting miStudy's goal of delivering a dynamic, real-time learning experience.

1. Dockerization ensures consistency across development and production environments encapsulating the dependencies of the system. The docker image was built locally on Node 20 Alpine.
2. The image was tagged for DockerHub and pushed making it accessible for Azure deployment.
3. A resource group was created in Azure Portal to organize all related resources such as App Services and App Service Plan
4. An App Service Plan was configured with the Basic tier, opting for Linux to support Docker
5. An App Service was created linking the Docker Hub image for pulling the containerized application ad to expose it via a public URL.

6. The App Service was then connect to the GitHub repository under the deployment centre opting in for continuous deployment and authorizing Azure to access GitHub.
7. A GitHub Actions workflow was then added to the repository to automate the build push deploy cycle.

2.6 Testing

- **Automated Continuous Deployment** - Pushed a code update, verifying the GitHub workflow completed, redeploying the updated app seamlessly.

Table 2 Test case for GitHub Workflow

Test Case Id	01
Test Case	Automated Continuous Deployment
Test Scenario	Verify automatic deployment by GitHub actions workflow after a code change is pushed
Input	Push a code change to GitHub repository
Expected Output	Trigger GitHub actions workflow and successful build and deployment to Azure App Service.
Actual Result	Success with changes reflecting on the deployed application.
Status (Pass/Fail)	Pass

- **Validation** - Tested with 40 different questions to ensure the responses are generated without any issue.

Table 3 Scope test case

Test Case Id	02
Test Case	Out of Scope Questioning
Test Scenario	Testing out of scope questioning.
Input	Who won the FIFA world cup in 2022
Expected Output	A response conveying it is out of scope and the question cannot be answered
Actual Response	A polite response conveying the question is out of scope and to question only within the content provided.
Status (Pass/Fail)	Pass

Table 4 Harmful content test case

Test Case Id	03
Test Case	Harmful Questioning
Test Scenario	Testing questions that may provide harmful content
Input	How to commit a crime by harming someone
Expected Output	A response conveying it is violation of content guidelines
Actual Response	A response conveying it is violation of content guidelines
Status (Pass/Fail)	Pass

- **Response Evaluation** – The effectiveness of the two different prompt approaches, zero shot, and intent aware structured prompt was observed during the validation process of the 40 questions queried in all categorisations. The queries questions are categorised in the Table and the Results are discussed in the Results and Discussion section

Table 5 Query Types

Query Type	**Description**
General Queries	Basic factual questions based on the document
Summarisation	Query for a summary of key topics
Table Generation	Query to organise information into structured tables
Quiz Creation	Query to generate quiz questions
Code Snippets	Query for relevant code examples
Simplify a concept	Query to explain complex ideas in simple terms
Explanation	Detailed explanations
Unrelated Queries	Queries unrelated to the document

2.7 Commercialisation

The modular design of this component, intentionally decoupled from tight integration with miStudy, presents a significant opportunity for independent commercialization. Unlike system-specific solutions, its flexibility allows it to be adapted for use in other platforms or applications by tailoring the prompt engineering to align with diverse business goals. This adaptability enables seamless integration into various systems, whether for educational tools, customer service interfaces, or content generation platforms, offering a scalable, customizable solution that can address unique operational needs while leveraging the core capabilities of the component.

Commercialisation Steps

1. **Market Analysis and Segmentation** - Identify target industries, such as education, e-commerce, healthcare, and assess their specific needs for AI-driven solutions, ensuring the component's value proposition aligns with market demands.
2. **Prompt Customization –** Customise the Prompt to suit the business needs and goals.
3. **Pilot Testing with Partners** - Collaborate with early adopters in select industries to deploy the component, gather feedback, and refine its integration process, demonstrating proof of concept and building case studies.
4. **Licensing and Pricing Model Establishment** - Develop a commercial model, such as subscription-based licensing or per-use pricing, tailored to the scalability and modular appeal of the component, making it accessible to both small businesses and large enterprises.
5. **Marketing and Distribution -** Launch a targeted marketing campaign highlighting the component's modularity and adaptability, leveraging partnerships, online platforms, and industry events to reach potential customers and establish a distribution network.

3. Results and Discussion

Approach 1 – Zero shot prompt template

Approach 2 – Intent Aware Structure Prompt template

1. General Query – What is this document about?

Table 6 Response Evaluation 1

Approach 1

This document is about the Web Services Description Language (WSDL) and its structure, including elements such as WSDL Types, Messages, and Ports. It describes how WSDL is used to define and locate web services, detailing the components like operation types, message structure, SOAP header and body, as well as discussing OAuth and OpenID for authorization and authentication.

Approach 2

Explanation of the Document

Introduction
This document primarily discusses the structure and purpose of WSDL (Web Services Description Language) and its related concepts, such as SOAP (Simple Object Access Protocol) and authorization protocols like OAuth and OpenID.
Detailed Breakdown

- **WSDL Overview:** Describes how WSDL is an XML-based document that details a web service's location, operations, and data types.
- **Major Elements of WSDL:**
  - **Types:** Defines the data types used in the web service, utilizing XML Schema syntax for maximum compatibility.
  - **Messages:** Specifies the data elements for an operation, akin to parameters in a function call.
  - **Ports:** Describes the legal operations a web service exposes and classifies them into different operation types (one-way, request-response, etc.).
- **SOAP Structure:** Discusses the organization of a SOAP message into headers and bodies, the purpose of each, and the limitation of SOAP not specifying the body content.
- **Authorization Protocols:**
  - **OAuth:** Focuses on delegated authorization, allowing third-party services to access personal data without sharing passwords.
  - **OpenID:** Concentrates on authentication and provides a single sign-on identity solution.

Real-world Application
WSDL documents are used extensively in web services, allowing for seamless communication and interaction between different applications, especially in distributed systems.
Conclusion
This document serves as a comprehensive guide to understanding WSDL and its associated protocols, software interactions, and their practical applications in modern web services.

2. Summarisation - Summarize the main points about SOAP and how it works.

Table 7 Response Evaluation 2

Approach 1

SOAP (Simple Object Access Protocol) is an XML-based protocol used for exchanging information between applications, primarily focusing on Remote Procedure Calls (RPCs) transported via HTTP or SMTP. Here are the main points:

- **XML-Based Protocol**: SOAP is designed for exchanging information using XML for message formatting, making it platform and language independent.
- **Communication Protocols**: SOAP typically utilizes HTTP or SMTP for transporting messages.
- **RPC Mechanism**:
    - On the client side, an RPC call is converted into an XML document.
    - This XML data is transported over HTTP/SMTP to the server.
    - The server processes the XML document into a procedure call and then transforms the response back into an XML document.
    - The client then converts this XML data into a response.
- **Structure of SOAP Messages**:
    - SOAP messages consist of an envelope that contains a header (optional) and a body (mandatory).
    - The **header** may include metadata such as security information or transaction identifiers.
    - The **body** contains application-specific data, including a Fault entry section for error reporting.

Overall, SOAP facilitates structured information exchange between diverse systems in a standardized manner, supporting RPCs effectively.

Approach 2

Summary of SOAP and How It Works

Main Idea

SOAP (Simple Object Access Protocol) is an XML-based protocol that enables the exchange of structured information in a decentralized, distributed environment, primarily using HTTP.

Key Sections

- **Definition and Purpose**:
    - SOAP is a protocol designed for exchanging messages between applications, focusing on remote procedure calls (RPC) over the network.
    - It uses XML to define its message structure, which allows for platform and language independence.
- **Communication Protocol**:
    - SOAP messages are typically transmitted over HTTP or SMTP.
    - Examples include a Java client making a call to a SOAP server running on Windows, which demonstrates interoperability across different platforms.
- **Message Structure**:
    - A SOAP message consists of two main parts:
        - **Header**: Optional and contains metadata like security information, identifiers, and coordination data.
        - **Body**: Mandatory and contains the application-specific data, including responses and error messages (Fault entries).
- **How SOAP Works**:
    1. **Client Side**:
        - Converts the RPC call into an XML document and sends it via HTTP or SMTP to the server.
    2. **Server Side**:
        - Receives the XML message, processes it into a procedure call, and sends back a response encapsulated in an XML document.
    3. **Response Handling**:
        - The client turns the returned XML data back into a meaningful response for the original request.

Why It Matters

SOAP's XML-based messaging and RPC capabilities enable different applications, regardless of their platform or language, to communicate effectively and securely. This interoperability is crucial for building complex distributed systems.

3. **Table Comparison – Provide a table comparing different authentication methods (OpenID, OAuth, SSL)**

Table 8 Response Evaluation 3

Approach 1

The query regarding the comparison of different authentication methods (OpenID, OAuth, SSL) is outside the document's scope. The document provides information specifically on OpenID and OAuth, discussing their purposes, scenarios, and some details, but does not include SSL or a comparative analysis of these methods. If you have any questions related to OpenID or OAuth, feel free to ask!

Approach 2

Comparison Table: Different Authentication Methods

Feature	OpenID	OAuth	SSL (Secure Sockets Layer)
Purpose	Authentication (verifying the identity of a user)	Authorization (granting access without sharing passwords)	Securing communication (encrypting data between client and server)
Workflow	User logs into a site using an identifier from their provider	User authorizes a third-party app to access their data from a service provider	Establishes a secure connection between client and server using certificates
User Interaction	Minimal; user authenticates once with the identity provider	More involved; requires user consent to grant access to data	Seamless; users do not interact directly with SSL processes
Data Sharing	Does not share user passwords; just confirms identity	Shares access tokens with third-party applications	Encrypts the entire data transmission between parties
Use Cases	Single sign-on across multiple sites	Integration of third-party applications (e.g., social media sharing apps)	E-commerce transactions, secure login pages
Complexity	Moderate; involves setup of identity provider and relying party	High; requires multiple steps including access token management	Low for end-users; complexity lies in setup and maintenance by site owners

OpenID is primarily for identity verification, OAuth for access delegation, and SSL for secure data transmission. Each method serves distinct roles in the realm of web security and user authentication.

4. Quiz Generation - Make a short quiz about the differences between WSDL and SOAP.

Table 9 Response Evaluation 4

Approach 1
Sure! Here's a short quiz focusing on the differences between WSDL and SOAP based on the provided context: **Quiz: Differences between WSDL and SOAP** 1. **What does WSDL stand for?** a) Web Services Definition Language b) Web Services Description Language c) Web Service Development Language d) Web Services Data Language 2. **What is the primary purpose of WSDL?** a) To define the structure of SOAP messages b) To specify the location of a web service and its operations c) To provide security information for web services d) To manage transactions in web services 3. **Which component of a SOAP message is optional?** a) Header b) Body c) Fault entry d) Message 4. **In WSDL, what does the element define?** a) The operations a web service can perform b) The data types used by the web service c) The details of the SOAP message d) The synchronization method of web services 5. **Which statement is true regarding SOAP body?** a) It contains security information. b) It is intended for infrastructure level data. c) It is for application level data. d) It includes identifiers for transactions. **Answers:** 1. b 2. b 3. a 4. b 5. c Feel free to use or modify these questions for your study purposes!
Approach 2
Quiz on the Differences Between WSDL and SOAP **Beginner:** 1. What does WSDL stand for? ◦ A) Web Services Development Language ◦ B) Web Services Description Language ◦ C) Web Socket Description Language ◦ D) Wireless Services Description Language **Intermediate:** 2. Which of the following describes the primary function of WSDL? • A) Encrypt data transmission • B) Describe web services and their operations • C) Send and receive SOAP messages • D) Define data types using XML **Intermediate:** 3. In terms of message structure, which part is mandatory in a SOAP message? • A) Header • B) Body • C) Both Header and Body • D) None of the above **Advanced:** 4. What is the main purpose of the SOAP header? • A) To contain application-specific data • B) To provide security information and identifiers • C) To describe service operations • D) To define data types

5. Code Snippets - Give a sample HTTP request for a RESTful web service.

Table 10 Response Evaluation 5

Approach 1

A sample HTTP request for a RESTful web service can look like this:

```
GET /resourceName HTTP/1.1
Host: http://example.com
Accept: application/json
```

In this example:

- `GET` is the HTTP method used to retrieve information.
- `/resourceName` is the URI of the resource being requested.
- `Host` specifies the domain of the web service.
- `Accept` indicates the type of response data the client is willing to accept, such as JSON.

Approach 2

Give a sample HTTP request for a RESTful web service.

Sample HTTP Request for a RESTful Web Service

Introduction

A RESTful web service communicates via standardized HTTP methods. Below is an example of a typical HTTP GET request to retrieve information about a resource, such as user details from a web service.

Sample HTTP GET Request

```
GET /api/users/123 HTTP/1.1
Host: example.com
Accept: application/json
Authorization: Bearer your_access_token
```

Explanation of the Request

1. **Method:** `GET` - This method is used to retrieve information from the specified resource.
2. **URI:** `/api/users/123` - This is the endpoint targeting the specific user with the ID of 123.
3. **HTTP Version:** `HTTP/1.1` - Specifies the HTTP version being used.
4. **Headers:**
   - **Host:** `example.com` - The domain name of the server to which the request is sent.
   - **Accept:** `application/json` - Indicates that the client expects the response in JSON format.
   - **Authorization:** `Bearer your_access_token` - Contains the token used for authenticating the request, assuming token-based authentication.

Real-World Use Case

In a web application, this request might be sent when a user wants to view their profile information. The server processes the request, retrieves the relevant data for the user with ID 123, and responds with the user details in JSON format.

Why It Matters

RESTful web services provide a flexible and efficient way to interact with resources over the internet, enabling data e between clients and servers using standard protocols like HTTP. This approach is widely adopted due to its ease of u scalability.
If you have any further questions or need additional examples, feel free to ask!

6. Simplification - Simplify the concept of WSDL bindings.

Table 11 Response Evaluation 6

Approach 1

WSDL bindings define how the messages for a web service are formatted and what protocols are used for communication. Essentially, a binding connects the operations of the web service to the specific protocols that will be used to send and receive messages.
In more detail:

- Each binding has a name (to identify it) and a type (which tells you which port it connects to).
- The soap:binding element specifies the style (how the message is structured) and the transport method (like HTTP).
- Operations within the binding define the specific actions the web service can perform and detail how the inputs and output for those operations are encoded.

So, WSDL bindings essentially serve as a set of rules for communicating with a web service.

Approach 2

Simplified Explanation of WSDL Bindings

- **Simple summary:** WSDL bindings define how a web service's messages are formatted and how they communicate over the network.
- **For Beginners:**
  - WSDL (Web Services Description Language) is like a contract that specifies what a web service can do and how to interact with it.
  - Bindings are essential because they explain the specifics: which protocols to use (like HTTP or SOAP) and the format of messages (like XML).
  - Think of it as instructions on how to make a phone call to a specific service; it tells you the number to dial and the language to use.
- **For Experts:**
  - A WSDL binding consists of a `<binding>` element that includes attributes such as `name` and `type`, where `name` identifies the binding and `type` links to the corresponding port type.
  - Within the binding, the `soap:binding` element specifies transport protocols (SOAP over HTTP) and message styles (RPC or document-based).
  - Each operation in the port type is linked to its respective SOAP action, ensuring that the input and output messages are correctly encoded and processed according to the specified formats.

This layered explanation should help different audiences grasp the concept of WSDL bindings effectively!

7. Explanation - Explain the role of UDDI in web services?

Table 12 Response Evaluation 7

<table>
<tr><td>Approach 1</td></tr>
<tr><td>The role of UDDI (Universal Description, Discovery, and Integration) in web services is to serve as a centralized directory of services. It pro a central place where developers can publish new services or find existing ones. Essentially, UDDI functions as a clearinghouse for compani and their services, enabling service requestors to search for and discover available web services. This facilitates communication and interoperability between different applications and services on the internet.</td></tr>
<tr><td>Approach 2</td></tr>
<tr><td>Explanation of UDDI in Web Services

Introduction
UDDI (Universal Description, Discovery, and Integration) plays a crucial role in web services by providing a mechanism for service providers to publish their services and for service requesters to discover those services.
Detailed Breakdown
Service Registry: UDDI acts as a centralized directory where businesses can register their web services. This registry allows for the cataloging of services with associated descriptions, making them searchable by potential users.
Service Discovery: It enables clients (service consumers) to search for available web services based on specific criteria, including service name, provider information, and service type. This discovery process facilitates easier integration between different systems and applications.
Interoperability: By standardizing how services are published and discovered, UDDI promotes interoperability among diverse applications, allowing them to communicate and exchange data seamlessly.
Integration with Other Specifications: UDDI works in conjunction with WSDL (Web Services Description Language) and SOAP (Simple Object Access Protocol) to provide a full-fledged environment for service orientation. While WSDL defines what the service does, UDDI describes where to find it.
Real-world Example
In a business scenario, a company may offer a service to manage inventory. By publishing this service in a UDDI registry, other companies can search for inventory management services and find the appropriate service provider. They can then use the information from WSDL to consume the service effectively.
Why It Matters
UDDI enhances the ecosystem of web services by simplifying the service discovery process. It fosters a more collaborative environment where organizations can leverage existing services, reduce redundancies, and innovate quickly by building upon available services. This approach ultimately leads to increased efficiency in integrating disparate systems.</td></tr>
</table>

8. Unrelated Queries - Who won the FIFA World Cup in 2018?

Table 13 Response Evaluation 8

<table>
<tr><td>Approach 1</td></tr>
<tr><td>I'm sorry, but that query is outside the document's scope.</td></tr>
<tr><td>Approach 2</td></tr>
<tr><td>I'm sorry, but that question is out of context based on the provided document. If you have any questions regarding the content related t online security, OAuth, OpenID, or RESTful web services, feel free to ask!</td></tr>
</table>

3.1 Research Findings

- **Zero Shot Prompt Limitations**

  The zero-shot approach showed signs of hallucination and inconsistency as it struggled to accurately scope queries, incorrectly labelling 12.5% of the document related questions as out-of-scope due to its lack of intent-specific guidance. While it provided factually accurate answers, its responses lacked structure and clarity, limiting its pedagogical effectiveness.

- **Intent-Aware Structured Prompt Advantages**

  The intent-aware structured prompt outperformed the zero-shot approach, particularly in delivering well-formatted responses. It excelled in structuring summaries, tables, code snippets, and quizzes, achieving consistent organization as instructed in the prompt. Both prompts maintained factual accuracy, but the intent-aware approach offered better clarity and coherence, with responses tailored to student intentions.

- **Handling Unrelated Queries**

  Both prompts effectively managed unrelated queries, politely rejecting them with 100% consistency. However, the intent-aware structured prompt maintained a more refined and pedagogically appropriate tone, enhancing the student experience.

These findings demonstrate that while the zero-shot prompt offers simplicity, the intent-aware structured prompt significantly enhances response quality and relevance, aligning better with miStudy's educational goals.

4. Conclusion

This study examines the impact of structured, intent-aware prompting on the effectiveness of an RAG-based chatbot in self-directed learning. Although zero-shot prompting remains the most widely used approach in RAG chatbots due to its simplicity, our findings indicate that incorporating multiple prompt engineering techniques substantially improves response quality, coherence, and relevance. The intent-aware structured prompt outperforms the zero-shot approach in several key areas, particularly in generating well-organized responses such as summaries, tables, quizzes, and code snippets. This approach enhances the chatbot's ability to interpret user intent, leading to improved student engagement and learning outcomes. Moreover, while both prompt strategies effectively managed unrelated queries, the structured prompt maintained a more refined and pedagogically appropriate tone.

These findings underscore the potential of integrating multiple prompt engineering techniques to enhance AI-driven educational tools. Future research could focus on refining prompt structures, incorporating adaptive learning strategies, and expanding the chatbot's capabilities to support more complex educational interactions. Additionally, further exploration could involve implementing a similar approach using LangChain for Python or alternative frameworks that facilitate image-based text extraction. This would enable the chatbot to process a broader range of educational materials, including scanned documents and handwritten notes. This research contributes to the growing discourse on AI in education, demonstrating how intent-aware structured prompt engineering can bridge the gap between generative AI and effective, structured learning assistance within the miStudy framework.

5. References

[1] S. Freeman, S. L. Eddy, M. McDonough, M. Smith, N. Okoroafer, H. Jordt and M. P. Wenderoth, "Active Learning Increases Student Performance in Science, Engineering, and Mathematics," *Proceedings of the National Academy of Sciences,* 2014.

[2] C. H. Crouch and E. Mazur, "Peer Instruction: Ten years of experience and results," *American Journal of Physics,* vol. 69, no. 9, 2001.

[3] A. Vaswani, N. Shazeer, . N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . Ł. Kaiser and I. Polosukhin, "Attention Is All You Need," in *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[4] T. B. Brown, B. Mann, N. Ryder and M. Subbiah, "Language Models are Few-Shot Learners," in *NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing Systems*.

[5] S. Victor, . L. Debut, J. Chaumond and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," in *Energy Efficient Machine Learning and Cognitive Computing - NeurIPS*, 2019.

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goval, H. Kurrler, M. Lewis, W.-t. Yih, S. Riedal and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing System*.

[7] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le and D. Zhou, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *NIPS'22: Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022.

[8] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin and T. Liu, "A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions," *ACM Transactions on Information Systems,* 2024.

[9] D. Thüs, S. Malone and R. Brünken, "Exploring generative AI in higher education: a RAG system to enhance student engagement with scientific literature," *Frontiers in Psychology,* 2024.

[10] P. Feldman, J. R. Foulds and S. Pan, "RAGged Edges: The Double-Edged Sword of Retrieval-Augmented Chatbots," 2024.

[11] B. Lester, R. Al-Rfou and . N. Constant, "The Power of Scale for Parameter-Efficient Prompt Tuning," *Association for Computational Linguistics,* vol. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021.

[12] Y. Goldberg, Z. Kozareva and Y. Zhang, "RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning," *Association for Computational Linguistics,* vol. Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, 2022.

[13] Y. Deng, J. Cherian, K. Kumari, U. Comite, B. Gavurova and J. Popp, "Family and Academic Stress and Their Impact on Students' Depression Level and Academic Performance," *Frontiers in Psychiatry,* vol. 13, 2022.

[14] F. Petroni, . T. Rocktäschel, S. Riedel, . P. Lewis, A. Bakhtin, Y. Wu and A. Miller, "Language Models as Knowledge Bases?," *Association for Computational Linguistics,* Vols. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019.

[15] Y. Gao, Y. Xiong, X. Gao and K. Jia, "Retrieval-Augmented Generation for Large Language Models: A Survey," 2023.