

**AI-ENABLED INTELLIGENT ASSISTANT TO
IMPROVE READING AND COMPREHENSION SKILLS
IN ENGLISH LANGUAGE**

Malindu Dilshan Ariyawansha Sooriyaarachchi

IT21173790

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

Department of Computer Science and Software Engineering

Sri Lanka Institute of Information Technology

Sri Lanka

April 2025

**AI-ENABLED INTELLIGENT ASSISTANT TO
IMPROVE READING AND COMPREHENSION SKILLS
IN ENGLISH LANGUAGE**

Malindu Dilshan Ariyawansha Sooriyaarachchi

IT21173790

Dissertation submitted in partial fulfillment of the requirements for the Special
Honor's Degree of Bachelor of Science 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

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
M.D.A Sooriyaarachchi	IT21173790	

Signature of the Supervisor
(Dr. Dasuni Nawinna)

Date

.....

.....

ABSTRACT

English vocabulary proficiency remains a significant challenge for learners around the world, especially in contexts where English is a second or foreign language. Traditional learning approaches often fail to adapt to individual differences in vocabulary knowledge or provide engaging, contextualized learning experiences. To overcome these limitations, this research introduces an AI-powered system that automatically evaluates a student's English vocabulary level using their written input and generates personalized, game-based learning content. The system uses a machine learning model trained with the Gradient Boosting algorithm on CEFR-labeled data to classify students' English vocabulary proficiency from levels A1 to C2. Advanced NLP-based feature engineering techniques, including readability metrics, lexical diversity, syntactic depth, named entity density, and POS tag distributions, are used to extract meaningful features from student input. Based on the predicted CEFR level, the platform dynamically creates vocabulary games tailored to the learner's ability, such as multiple-choice quizzes, sentence completions, and synonym-replacement tasks. A Retrieval-Augmented Generation (RAG) framework powers both the game generator and an integrated vocabulary definition chatbot, combining LangChain, Pinecone vector storage, transformer-based sentence embeddings, and a Gemini large language model. This enables real-time retrieval of contextual definitions, usage examples, and CEFR-aligned feedback. The system is deployed using a microservices architecture via Docker containers on Azure, ensuring scalability and performance. This research demonstrates an effective, scalable solution for personalized English vocabulary development by combining CEFR classification, intelligent retrieval, and generative AI.

Keywords – CEFR Prediction, Gradient Boosting, Gamified Learning, LangChain, RAG

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude to my supervisor Dr. Dasuni Nawinna for the constant guidance and support which helped me at all times for the successful completion of my undergraduate research. Besides my supervisor, my sincere thanks also go to Mr. Jeewaka Perera, the co-supervisor of this research project for being willing to help whenever it was needed. A very special note of appreciation goes to Miss Ridmi Dinanjali, an external English teacher, whose support and expertise in the area of English language learning were incredibly valuable. Her insights helped enrich the content and provided an essential practical perspective to the research focus. I also wish to extend my thanks to my colleagues and friends for their constructive feedback, collaboration, and constant motivation during this process. Their input has added great value to this Research Project. Lastly, I am deeply grateful to my family for their endless patience, understanding, and encouragement throughout this endeavor. Without the contributions and support of all the above individuals, the successful completion of this proposal would not have been possible.

TABLE OF CONTENTS

DECLARATION	i
ABSTRACT	ii
ACKNOWLEDGEMENT.....	iii
LIST OF FIGURES	vi
LIST OF TABLES	viii
LIST OF ABBREVIATIONS	ix
1 INTRODUCTION.....	1
1.1 General Introduction.....	1
1.2 Background literature	2
1.2.1 Importance of English Vocabulary Proficiency Worldwide	2
1.2.2 Traditional Vocabulary Learning Methods and Their Limitations.....	3
1.2.3 Overview of AI Techniques Used in English Vocabulary Learning.....	5
1.2.4 Machine Learning and Gradient Boosting for Text Vocabulary Level Prediction	6
1.2.5 Deep Learning Models and Transformer-Based Language Models.....	7
1.2.6 Retrieval-Augmented Generation (RAG) and Its Applications	8
1.2.7 Overview of CEFR Standards for Language Proficiency Classification.....	8
1.3 Research Gap.....	10
1.3.1 Analysis of Existing Vocabulary Learning Methods and Tools.....	10
1.3.2 Limitations in Adapting to Individual Learner Needs.....	11
1.3.3 Lack of Integration Between Gamification, Proficiency Prediction, and Personalized AI Content.....	11
1.3.4 Justification for the Current Study	12
1.4 Research Problem.....	13
1.5 Research Objectives	16
1.5.1 Main Objectives.....	16
1.5.2 Specific Objectives.....	16
2 METHODOLOGY	18
2.1 Methodology	18
2.1.1 Requirements Gathering and Analyzing	18
2.1.2 Feasibility Study.....	19
2.1.3 Problem Statement	19
2.1.4 System Designs	20
2.1.5 Data Acquisition and processing	25
2.1.6 Choosing the Correct Framework	28
2.1.7 Identification of CREF Level of the student given input text	32
2.1.8 Generate Personalize games content Using RAG System.....	48
2.2 Commercialization aspects of the product	55
3 TESTING AND IMPLEMENTATION	57
3.1 Implementation.....	57
3.1.1 Implementation of CEFR Level Identification System	57
3.1.2 Implementation of Personalized Vocabulary Game Generation Using RAG	

3.1.3	Front-End Implementation	80
3.2	Testing.....	84
3.2.1	Test Plan and Test Strategy	85
3.2.2	Test Case Design.....	85
4	RESULTS AND DISCUSSIONS	88
4.1	Results	88
4.1.1	Student Input Text Vocabulary CEFR Identification and Classification	88
4.1.2	Personalized Vocabulary Game Generation Using RAG System	90
4.2	Research Findings	92
4.3	Discussion	93
5	CONCLUSION	96
6	REFERENCES.....	97

LIST OF FIGURES

Figure 1.1 : Global distribution of English proficiency levels and its importance in education vs. professional sectors.	2
Figure 1.2 User Interest in a Vocabulary-Level-Based Educational Tool	14
Figure 1.3 illustrates how likely learners are to use such an adaptive vocabulary learning tool.	14
Figure 1.4 reflects user opinions on the importance of receiving immediate feedback during vocabulary activities.	15
Figure 2.1 Overall system diagram	20
Figure 2.2 Use case Diagram of the component	21
Figure 2.3 Sequence diagram of Component	22
Figure 2.4 Component System Diagram	23
Figure 2.5 Dataset Used for CEFR Classification	26
Figure 2.6 Dataset used for Rag System	27
Figure 2.7 Text Preprocessing Pipeline	33
Figure 2.8 Hyperparameter Optimization Pipeline	44
Figure 3.1 Dependencies installation	57
Figure 3.2 Data Preparation	58
Figure 3.3 NLP model loading	59
Figure 3.4 Text Simplification	59
Figure 3.5 Text Simplification Input and Output	59
Figure 3.6 POS Tagging Process	60
Figure 3.7 POS Tagging Sample Input and Output	60
Figure 3.8 Named Entity Recognition Proccess	61
Figure 3.9 Named Entity Recognition Sample Input and Output	61
Figure 3.10 Syntactic depth process	61
Figure 3.11 Syntactic depth process Sample Input and Output	61
Figure 3.12 Readability Metrics	62
Figure 3.13 Readability Metrics Sample Input and Output	62
Figure 3.14 generate feature function	63

Figure 3.15 generate feature function sample input and output.....	63
Figure 3.16 Reusable Pipeline for model Training	64
Figure 3.17 Data Load and Label Encode.....	64
Figure 3.18 Model Training	65
Figure 3.19 Main Function of model training	66
Figure 3.20 Trained Model Accuracy and Result	66
Figure 3.21 Load TF-IDF and Transformed to Vectors	67
Figure 3.22 Hyperparameter Optimization Using BayesSearchCV	67
Figure 3.23 Cross Validation	68
Figure 3.24 best-found parameter after extensive tuning.....	68
Figure 3.25 Model saved as a pickle.....	68
Figure 3.26 model retrained with handcrafted features	69
Figure 3.27 Final model Saving path and Type	70
Figure 3.28 Inference Implementation	71
Figure 3.29 API Integration with saved model	72
Figure 3.30 Sample Response	73
Figure 3.31 Libraries Installation.....	74
Figure 3.32 Pinecone Vector DB Creation and Embedding Model Loading	75
Figure 3.33 Embeddings generation function	76
Figure 3.34 vector DB Screen shot	76
Figure 3.35 function for Retrieval from Vector Database	77
Figure 3.36 Prompt for Short Hint Generation	77
Figure 3.37 Prompt for Word Explanation used in chatbot	78
Figure 3.38 Prompt for word Association Game	78
Figure 3.39 Prompt for word Fill-in-the-Blank game.....	78
Figure 3.40 LLM chain	79
Figure 3.41 Text input section	81
Figure 3.42 The predicted Vocabulary CEFR level.....	81
Figure 3.43 vocabulary fill-in-the-blank game	82
Figure 3.44 association vocabulary game	82
Figure 3.45 Vocabulary Chatbot.....	83
Figure 4.1 model classification report.....	88

LIST OF TABLES

Table 1:1:A comparison of traditional and modern vocabulary learning approaches .	4
Table 1:2:Feature Types and Their Relevance to Proficiency Prediction.....	6
Table 1:3:Vocabulary Expectations Across CEFR Levels	9
Table 1:4:Vocabulary Learning Tools Comparison.....	10
Table 2:1 POS Tags and Their Linguistic Importance	34
Table 2:2 Named Entity Recognition and CEFR-Level Inference	35
Table 2:3 Readability Metrics Used in CEFR Prediction	36
Table 2:4 Label Encoding	41
Table 3:1 Test Case to CEFR Classification from Input Text	86
Table 3:2 Test Case to RAG Game Generation Based on Topic and Level.....	86
Table 3:3 Test Case to Chatbot Vocabulary Response	86
Table 3:4 Test Case to Game UI Rendering	86
Table 3:5 Test Case to Preference-Based Game Filtering	87
Table 3:6 Test Case to Fill-in-the-Blank Question Generation	87
Table 3:7 Test Case to Hint Prompt Generation Accuracy.....	87
Table 4:1 Average Evaluation Scores for Personalized Vocabulary Game Generation via RAG	91

LIST OF ABBREVIATIONS

Abbreviation	Full Term
AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing
CEFR	Common European Framework of Reference for Languages
RAG	Retrieval-Augmented Generation
LLM	Large Language Model
TF-IDF	Term Frequency–Inverse Document Frequency
POS	Part of Speech
UI	User Interface
UX	User Experience
CNN	Convolutional Neural Network
DB	Database
API	Application Programming Interface
UAT	User Acceptance Testing
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer

1 INTRODUCTION

1.1 General Introduction

Even in our globalized digital age, English is still crucial for academic, professional, and intercultural opportunities. But one of the hardest areas for learners to master is vocabulary development which is a fundamental aspect of language learning. The lack of vocabulary limits the reading comprehension, writing fluency, and oral communication [1].

Traditional vocabulary teaching tends to be memorizing from textbooks and repetitive exercises without consideration of the ability level or interests of each student. Static, lacking context, no engagement and zero ability to go at your speed? Consequently, students often remember almost nothing outside of tests and exams, and long-term language retention is low [2].

Gamification is an interesting way to make your learners engage better and be more motivated by your lessons. Gamified systems enhance engagement and enjoyment of learning tools to ensure better learning outputs via integration of game elements such as points, levels, leaderboards and rewards [3]. Gamification provides the opportunity in second language acquisition (SLA) for learners to access vocabulary with a degree of meaningful engagement that strengthens retention through repetition and immediate feedback in a range of productive tasks [4].

Machine learning (ML) and natural language processing (NLP) two fields of artificial intelligence (AI) have also advanced the boundaries of changing vocabulary learning platforms. Machine learning (ML) models predict the CEFR level of a student based on the input text, while transformer-based models such as BERT and GPT allow dynamic content generation and instantaneous feedback [5]. To accomplish this, the authors utilized Retrieval-Augmented Generation (RAG) architectures that merge semantic search with generative capabilities to personalize vocabulary activities based on the data of the learner [6].

Hence, we introduce a new smart vocabulary learning framework that predicts learners' CEFR levels from input text and tailors vocabulary games using a RAG pipeline. This solution combines CEFR classification (using ML), semantic retrieval (using pinecone), generation (using Gemini LLM), and wraps it all together in a responsive web app set up through React and Tailwind CSS. This system has the potential to combined linguistic ability and curiosity that can provide a scalable and customizable immersion experience for anyone looking to become proficient in a new language.

1.2 Background literature

1.2.1 Importance of English Vocabulary Proficiency Worldwide

Its international recognition as a common language is one of the reasons why English is called the world. A Language that can be spoken from country to country, culture to culture, and a discipline to its particular population. English is either the official language or one of the official languages in over 60 countries, and it serves as the main language of business, science, technology, higher education, and diplomacy [7]. With the rapid acceleration of globalization, English has become an essential tool for success at home and abroad.

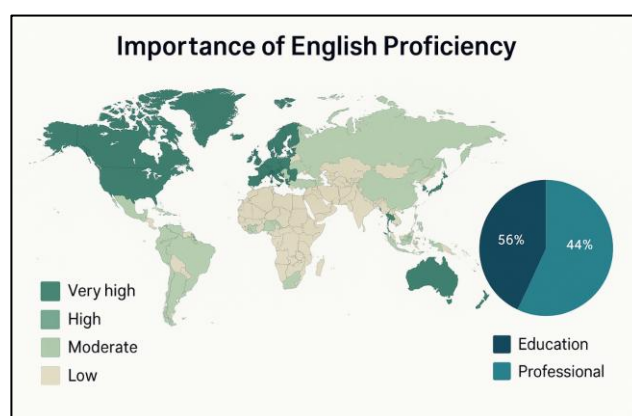


Figure 1.1 : Global distribution of English proficiency levels and its importance in education vs. professional sectors.

Figure 1.1 demonstrates that English is perceived as a major skill on every continent, though specifically important in countries where English is also used for professional and educational purposes (2). The pie chart also makes it clear that 56% of its value as perceived is in educational use, whereas 44% is for vocational use, and it is worth noting this dual nature of vocabulary development for educational achievement and career progress

English as a second or foreign language is a requirement for learning and being part of these global education systems and workforces, using digital media and participating in society. In this sense, the mastery of vocabulary is critical in language skills. Vocabulary knowledge has been found to be one of the most important predictors of reading comprehension, speaking fluency, and among the extensive literature to the best predictor of writing performance [and the research simply do not support the generally accepted belief in SLA that the mother tongue sexually knowledge of transparent languages differs from L1 to be using more general cognitive abilities. Without an adequate vocabulary learners are frequently incapable of comprehending or making worthwhile use of communication, however grammatically accurate it may be.

Yet vocabulary acquisition is problematic for many learners of English as a second language. These problems are related to instructional strategies that focus on rote memorization of decontextualized word lists, rather than on how words are used in context [8]. Vocabulary learning in traditional classroom-based settings may not cover depth of lexical knowledge such as collocations, connotation and pragmatic uses that are essential for real-life communication [9]. In addition, the teaching of lexicon is often underemphasized in language teaching programs, thus leaving students without the key knowledge to promote long-lasting learnability and flexible use of language [10].

1.2.2 Traditional Vocabulary Learning Methods and Their Limitations

Vocabulary has always been one of the central components of second language

learning, but to comprehend, express and use language fluently, the ability to learn vocabulary is crucial. However, these conventional ways for teaching vocabulary, i.e., committing to memory of word lists, dictionary consultation and work book exercises, have apparent drawbacks to induce profound, long-term learning [2].

Often, these approaches focus on the receptive side of knowing a word (e.g., understanding a word and its meaning) rather than on the productive side of knowing a word (e.g., using the word accurately in a sentence), which is the main concern in everyday communication. Students are usually expected to memorize vocab out of context with no real connection to how words are used in natural language. Hence, they are good on written tests, but find it hard to remember and use vocabulary in speaking and writing [8].

Yet another problem with old-style vocabulary teaching is that it is a one-size-fits-all approach. The tools tend to have a single level of proficiency, which is unrealistic when considering the vast differences in ability, learning pace and style between learners in a classroom. Beginners who need repetition and more time to learn vocabulary will be lost, and suffer from lack of confidence, while higher level students will lose motivation and disengage if there's not enough variety or challenge [11].

To illustrate this contrast more clearly, Table 1.1 compares traditional vocabulary learning methods with more modern, learner-centered approaches that have emerged in recent years.

Table 1:1:A comparison of traditional and modern vocabulary learning approaches

Feature	Traditional Methods	Modern/AI-Based Methods
Learning Style	One-size-fits-all	Adaptive to learner's level and pace
Engagement Level	Low (memorization, repetition)	High (games, interaction, feedback)

Word Usage	Isolated word lists	Words in context, examples, sentence use
Retention	Short-term	Long-term via spaced repetition and active recall
Feedback	Limited or delayed	Instant and personalized
Assessment	Paper-based tests	Interactive quizzes, real-time tracking
Motivation	Low	High due to gamification and progression systems

As illustrated in Table 1.1, current trends in training focus more on personalization, engagement and context driven use, which are not so clear in traditional instruction. In addition, these approaches provide quick feedback, adaptive level, and game-based features to ensure that learners can learn words more efficiently and with fun.

In addition, these conventional approaches do not have an active involvement and individualized feedback, which has been proven to improve vocabulary recall. With no input, no reinforcement, and no meaningful activation, we forget a lot of the vocabulary very quickly. Research in cognitive psychology has found that learning in context, spaced repetition, and multisensory engagement are better ways of retaining information and transferring skills [12].

These restraints are even more apparent in the digital age, where today's learners increasingly demand interactive, dynamic and individualized forms of instruction. This gulf between old-school teaching and current digital learner appetites highlights a demand for creative solutions that can actively drive vocabulary learning to become engaging, adapted to learner needs and relevant to their wider context.

1.2.3 Overview of AI Techniques Used in English Vocabulary Learning

Introduction Artificial Intelligence (AI) is transforming education on the back of smarter and more adaptive learner-centric tools. AI tools and technologies especially ML and NLP helps educational tools in monitoring student behavior and personalize learning experience. Such tools can be quite effective in language learning environments, where they can aid in reading comprehension, writing enhancement,

pronunciation support, and vocabulary acquisition [13].

Unlike traditional systems that use pre-set content, and linear instruction, the AI systems are responsive to how the learner is doing. They follow progress, see where students are strong and weak and change challenge levels on the fly. This movement toward personalized learning has since paved the way for enhanced vocabulary instruction, in particular, for second-language learners.

1.2.4 Machine Learning and Gradient Boosting for Text Vocabulary Level Prediction

Within supervised learning algorithms, gradient boosted trees have emerged as popular because they are known for their ability to work well with complex classification problems. It works by constructing an ensemble of weak learners (usually decision trees) where each successive model concentrates on the residuals (errors) of the former, making the prediction error decrease as time goes [14]. For vocabulary learning applications, Gradient Boosting can be used to predict the Vocabulary CEFR level class (e.g., A1, A2, ..., or C3) of a learner from linguistic features extracted from written input.

Table 1:2:Feature Types and Their Relevance to Proficiency Prediction

Feature Type	Description	Relevance to Proficiency Prediction
Lexical Richness	Measures variety and density of vocabulary	Captures the learner's vocabulary range and diversity
CEFR Vocabulary Alignment	Assesses use of words mapped to CEFR-level vocabulary	Directly contributes to accurate A1–C2 classification
Morphological Richness	Tracks word formation patterns (prefixes, suffixes, etc.)	Reflects control over derivations, inflections, and forms
Syntactic Richness	Evaluates grammar and part-of-speech (POS) usage	Indicates maturity and complexity of sentence structure

Semantic Richness	Measures depth and nuance of meaning in word use	Highlights lexical sophistication and conceptual depth
Named Entity Richness	Detects mentions of people, places, organizations, etc.	CEFR C1/C2 texts typically contain more real-world entities
TF-IDF Features	Weighs term frequency against document relevance	Assesses the informational density and keyword importance

A key part of this process is feature engineering - the transformation of raw learner text into a structured set of features appropriate for model training. The following types of features in particular have been found useful for the prior prediction of CEFR levels [15]. As demonstrated in Table 1.2, there are different linguistic features that are used to predict a learner's language proficiency.

A Gradient Boosting model can then be trained on such features from a labeled data set to predict a student's proficiency accurately in real time. These predictions can in turn suggest personalized content like word games, vocabulary builder exercises and contextual reading materials.

1.2.5 Deep Learning Models and Transformer-Based Language Models

In recent years, the field of Natural Language Processing (NLP) has seen an upsurge by the substantial progress in deep learning, notably using transformer-based models. Transformers such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) are able to capture complex language patterns, context and linguistic relations in the text [15]. Because of their bidirectional processing, they can be used to produce or to comprehend language with very good performance.

These models are increasingly being employed in educational contexts to give natural-sounding feedback, auto-generate questions, improve grammar, and even serve as tutors. For students learning new vocabulary, the use of transformer models to produce context-essential examples, synonyms, and definitions in an adaptive manner

to learners' skill leads to increased engagement and retention.

1.2.6 Retrieval-Augmented Generation (RAG) and Its Applications

One notable improvement regarding AI-supported vocabulary learning is Retrieval-Augmented Generation (RAG), a hybrid model joining retrieval-based methods with generative language models to provide context-sensitive and pedagogically informed output [6]. RAG robustly integrates information from external sources (e.g., semantic vector data) with a transformer-based generator to produce accurate, meaningful, and transferable results. For teaching vocabulary, the cluster generating approach allows for the creation of graded, dynamic learning material like word meanings, authentic use in a context, examples, definition with context, and meaning with context. In this paper, we used a multi-agent RAG system, with each agent to fulfil certain pedagogical function, including task generator for gamified vocabulary tasks, hints/feedback provider, word meaning retriever, as well as the adjuster to the user's level and preferences for themselves. Based on the above architecture, as developed by the likes of LangChain Pinecone, our approach supports scalable personalization and offers the guarantee that every learner is presented content that is both linguistically appropriate, educationally beneficial, and offered within a game-based environment. Multi-agent RAG is designed to combine word learning with cognitive load theory and also adaptive learning, and not only brings interactivity to students, but also provides profound and long-term vocabulary recall¹. This makes it an effective support tool for second language learners, the latter who need a constant feedback, context examples and motivation to keep the learning process going.

1.2.7 Overview of CEFR Standards for Language Proficiency Classification

The Common European Framework of Reference for Languages (CEFR) is an international standard for describing language ability at six levels (A1 to C2) which

represent a learner's communicative skills in reading, writing, listening, and speaking. For vocabulary learning, CEFR is key to deciding not just the amount, but how complex, contextually appropriate and how deep should vocabulary be that is expected to be produced and comprehended at each level [17]. For example, at the A1 level learners can make use of high-frequency concrete words, whereas at the C1/C2 levels learners should be able to deal with idiomatic expressions, abstract vocabulary and sophisticated collocations [10]. Therefore, combining CEFR ratings in AI-based platforms facilitate personalized vocabulary tasks with dynamic content difficulty tailoring the content to the learner's proficiency level. (Automated CEFR-level prediction based on lexical richness, syntactic complexity, and semantic depth now facilitates personalized learning pathways and vocabulary selection. [15]) A vocabulary-based dissection of CEFR levels is in Table 1.3, illustrating how vocabulary demands change in the course of learning.

Table 1.3: Vocabulary Expectations Across CEFR Levels

CEFR Level	Vocabulary Size & Characteristics	Example Vocabulary Scope
A1	Basic everyday expressions 500-600 words	food, colors, greetings
A2	Frequent terms in routine tasks; 1000-1500 words	shopping, family, directions
B1	More descriptive vocabulary, including some abstract terms 2500 words	feelings, opinions, everyday topics
B2	Wider range of abstract, technical, and academic vocabulary 4000 words	discuss, assume, industry, recommend
C1	Advanced vocabulary with idioms, collocations 6000+ words	nevertheless, consequently, put forward
C2	Near-native lexical range, full control over register and nuance	notwithstanding, articulate, juxtapose, moreover

1.3 Research Gap

1.3.1 Analysis of Existing Vocabulary Learning Methods and Tools

Even though the number of digital platforms providing the support of language learning increases, most of the existing resources are not explicitly adapted on vocabulary learning with enough personalization but oriented in general to language acquisition (grammar, listening or speaking). Popular apps such as Duolingo and Memorize teach basic vocabulary using rote drills and games. Although these tools have been shown to be effective in stimulating learners with rewards and streaks, they usually have limited semantic understanding and personalized scaffolding based on individual learner performance [18]. In addition, many systems have a “one-size-fits all” design, as they have the same difficulty level and content order for the sequence for the whole audience, disregarding the learner’s characteristics or level of proficiency in vocabulary [19]. Consequently, learners can be confronted with content that is too easy and leads to disengagement, or too hard and results in frustration and demotivation.

Comparison with Related Work Existing systems are generally gamified and include some level of extension across levels, but until now they do not have integral AI-based CEFR-level prediction and course creation features. The system intends to close these functional gaps by providing a combined, intelligent and gamified vocabulary learning experience. A comparison between the vocabulary learning properties of available tools, research-oriented systems and the proposed approach is depicted in Table 1.4

Table 1:4:Vocabulary Learning Tools Comparison

Feature	Duolingo / Memorize (Existing Tools)	Other Research-Based Tools [15] [14] [20]	Proposed Research System
Focus Area	General language learning with basic vocab	Limited CEFR-specific vocab tools	Targeted English vocabulary learning
CEFR Prediction	Not available	Some statistical models	Text-based ML model for CEFR vocab classification

Content Personalization	Basic repetition-based adaptivity	Pre-defined adaptive logic	Dynamic personalization via RAG
Gamification	Points, streaks, leaderboards	Limited game elements	Integrated multi-level vocabulary games
AI Integration	Minimal, mostly static logic	Rule-based or simple ML	Advanced ML + Multi-agent RAG
Feedback & Hints	Generic feedback	Moderate feedback rules	Contextual hints, definitions, real-time feedback
Proficiency Level Adaptation	Progress based on usage frequency	Static or tier-based progression	Real-time CEFR-based adaptation

1.3.2 Limitations in Adapting to Individual Learner Needs

A lot of existing vocab learning tools don't really support personalized learning according to the ongoing skill level of the learner. They don't usually include a real-time assessment that can catch knowledge gaps or track the ebb and flow of individual vocabulary growth. While some systems provide progress monitoring, very few of them process the learner's input (written text) to customize the level of vocabulary difficulty or context. From the domain of second language acquisition, it is well understood that a personalized learning experience - one that appropriately matches the difficulty of the content to the learner's proficiency and cognitive load - substantially increases the amount of vocabulary learned [1,2]. In the absence of real-time assistance or adaptive exercise design, learners receive little help in resolving their underlying deficiencies, where progress and eventual long-term vocabulary mastery can be impeded.

1.3.3 Lack of Integration Between Gamification, Proficiency Prediction, and Personalized AI Content

Although each of gamification, machine learning and content personalization demonstrated potential in the literature, it is apparent that there is still a void in combining the three within one continuous vocabulary learning system. There are almost no applications that predict CEFR levels with text-based input and dynamic

content generation especially not utilizing AI like RAG. Most, as well, like the gamified approaches do not provide real-time semantic feedback or vocabularies adapted by CEFR levels of the learners [15]. The lack of integration of these elements in our current solutions seriously restrict the pedagogical value and flexibility of such resources. This research attempts to overcome this challenge by presenting the AI-based, gamified vocabulary module which can automatically assess the learner's level and select the context-relevant materials and generate the personalized vocabulary games along with the personalized feedback by employing the multi-agent RAG system which has never been seen before offering a truly adaptive and personal language learning experience.

1.3.4 Justification for the Current Study

The proposed research here plans to fill the gap left by existing vocabulary learning systems by implementing an all-in-one integrated platform merging machine learning-based CEFR proficiency prediction, personalized content generation via Retrieval-Augmented Generation (RAG), and gamified learning mechanics. At the center of the system is a supervised machine learning approach that relies on the analysis of learner-generated text to predict vocabulary proficiency levels calibrated to the CEFR. It eliminates the need for a time consuming pre- or post-test, and therefore makes it possible to carry out automated, real-time assessment of a student's lexicon, which is far more dynamical and scalable solution for language education than a classical static assessment.

Leveraging the forecasted CEFR level, the system uses a multi-agent RAG system to dynamically create personalized vocabulary learning content (including definitions, usage samples, feedback, and interactive games) according to the current proficiency of the learner and his/her preferences. Such personalization guarantees that the learners are adequately challenged and engaged through relevant and context-rich material. By incorporating gamification features such as points, levels, and feedback loops the learning process for vocabulary becomes an interactive and adaptive experience that

can lead to more retention and autonomy of the learner.

With integrated CEFR-based proficiency estimation, AI-generated adaptive content, and gamified delivery, our proposed platform is a comprehensive tool for second-language learners, particularly in underserved areas lacking bespoke teaching. This holistic approach is a new and pragmatic addition to the field of language learning research, merging pedagogical theory, AI innovation and learner-centric design.

1.4 Research Problem

Vocabulary is the foundation of second language learning of English. For non-native learners (in particular high-school students), a poor vocabulary apprehensively inhibit reading comprehension, fluency in writing, and self-confidence in speaking, and consequently hamper on academic success as well as future job opportunities [1]. Nevertheless, most classroom vocabulary learning approaches still involve rote memorization and type of static textbook exercises that no longer suits students' learner-autonomy and their evolving needs [21].

Although internet and smartphone language applications and gamified platforms populating the market, these resources generally do not aim to diagnose the learner's proficiency in real time level nor supply personalized vocabulary content according to a recognized framework, as the CEFR [18]. The basic vocabulary drills offered in some apps such as Duolingo and Quizlet don't use advanced AI tools to analyze user-generated text or customize learning based on live proficiency checks [18].

Systems that combine CEFR-level prediction with personalized content generation using state-of-the-art AI methods – such as using RAG – have also been missing. This is not enough, however, and the systems should not only be capable of assessing a student based on his or her written diagnostic input but also provide contextualized feedback, definitions and game-based exercises that are in line with the student's level [6] [15]

This work responds to the need for a full-fledged AI-based vocabulary learning

platform that leverages an AI model to predict CEFR level and a multi-agent RAG pipeline to provide personalized content--aka vocabulary games, definitions, feedback, learning hints--for the same, in accordance with student levels and preferences. Using adaptive gamification and AI-supported personalization, the envisioned system should provide an engaging, scalable, and learner-centered product to develop vocabulary in academic as well as informal contexts.

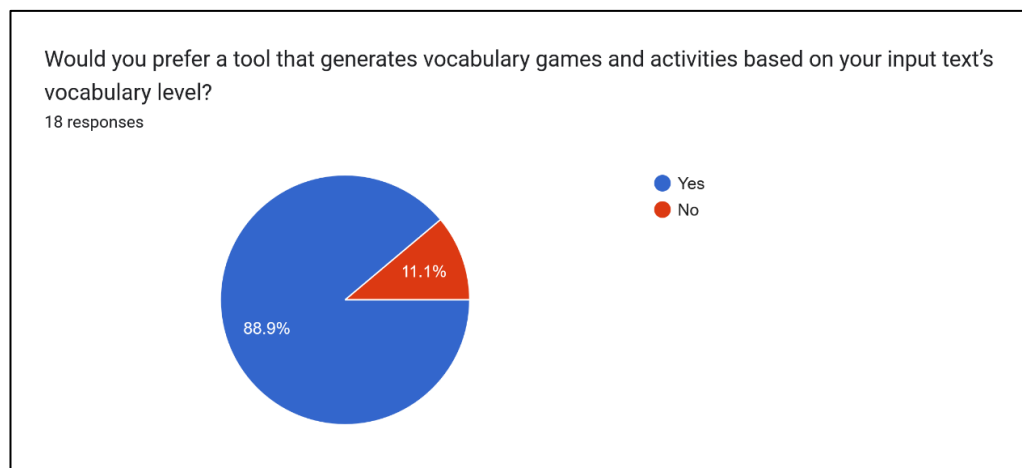


Figure 1.2 User Interest in a Vocabulary-Level-Based Educational Tool

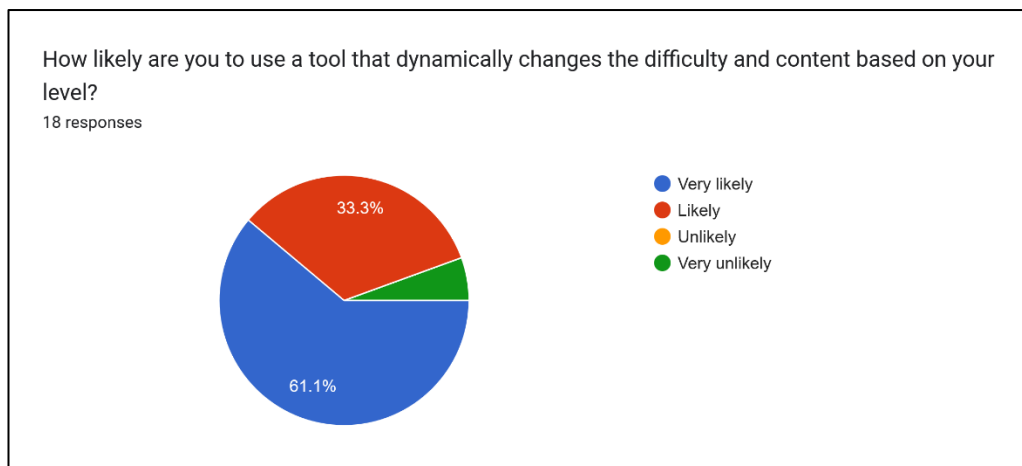


Figure 1.3 illustrates how likely learners are to use such an adaptive vocabulary learning tool.

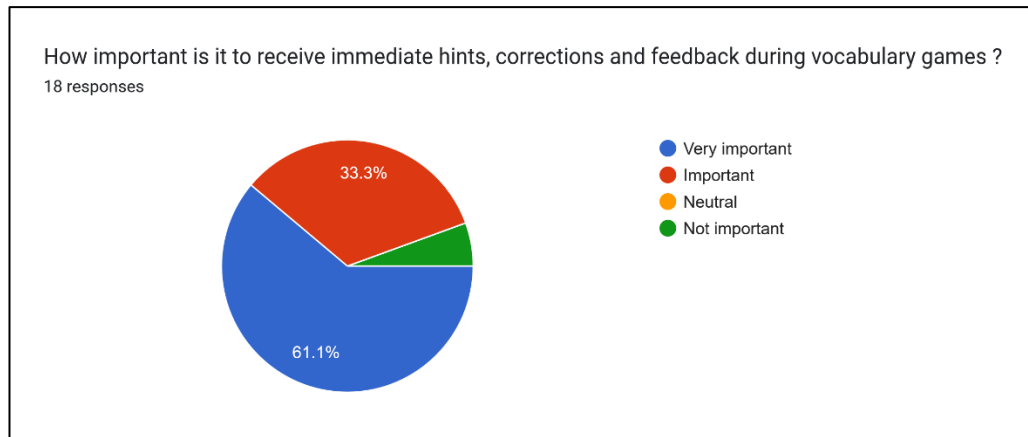


Figure 1.4 reflects user opinions on the importance of receiving immediate feedback during vocabulary activities.

In order to verify the appropriateness of the proposed vocabulary learning system, a user-centered survey was performed on the English learners to perceive their preference and expectation. A large percentage of participants were interested in a tool that extracts the level vocabulary of their input text and produces appropriate learning tasks accordingly, which is the main feature of the CEFR classification module developed in this system (Figure 1.2). Addition, the learners showed high probabilities that they would use a tool which they can adjust the difficulty and content of the vocabulary game after verbal skill depths and proficiency. This feedback is also corroborating the value of the Retrieval-Augmented Generation (RAG) system by which learning content are tailored on-the-fly in real time (Figure. 1.3). Another important addition to our list of findings based on the survey results is the interest that learners have in immediate feedback for example receiving hints, definitions and corrections in their vocabulary tasks. This underscores the role of the intelligent feedback agent, implemented through an LLM and integrated into the platform, which supports, helps, and guides the learner during the task in use (Figure 1.4). Some of these results provide evidence for an actual need of an adaptive, AI-supported vocabulary learning solution one that is personalized and interactive, and can react smart to learners' needs. The results strongly support the hypothesis and provide additional evidence for the features employed in the proposed system.

1.5 Research Objectives

1.5.1 Main Objectives

The main objective of this study is to develop an AI-powered gamified English vocabulary improvement module tailored for Sri Lankan high school students. This module will allow students to input challenging vocabulary either as individual words or within short written passages which will be analyzed to assess their vocabulary proficiency level. Using this input, the system will predict the learner's CEFR level through machine learning and generate personalized educational games and interactive learning materials. The module leverages large language models (LLMs), vectorization techniques, and a multi-agent retrieval-augmented generation (RAG) framework to adaptively provide vocabulary games, definitions, real-time feedback, and hints that align with the learner's skill level and preferences promoting sustained engagement and long-term vocabulary retention.

1.5.2 Specific Objectives

Identify and analyze student vocabulary needs

- To identify and analyze student vocabulary needs by processing learner-submitted text through natural language processing techniques and feature engineering. The system will extract linguistic features such as lexical richness, syntactic complexity, and word frequency and apply a machine learning model (e.g., Gradient Boosting) trained on CEFR-labeled data to estimate each student's vocabulary proficiency level. This proficiency insight will guide the selection and generation of personalized game content and instructional feedback, ensuring that each learner is appropriately challenged and supported throughout their learning journey.

Generate Personalized and Adaptive Educational Games

- To create interactive, AI-driven vocabulary learning games that dynamically

adapt to the learner's CEFR proficiency level and personal preferences. Using vectorized input data and contextual embeddings, a personalized learning path will be constructed for each user. The system will leverage large language models (LLMs) and a multi-agent Retrieval-Augmented Generation (RAG) architecture to generate context-aware vocabulary content including fill-in-the-blank games, word-association games, Additionally, a conversational chatbot will be integrated to provide real-time vocabulary explanations, definitions, and usage examples. Game complexity and feedback will adjust in real time based on learner performance, ensuring sustained engagement and targeted skill development.

Implement Real-Time Feedback and Intelligent Support

- To provide real-time, adaptive feedback during gameplay to reinforce learning outcomes and correct misunderstandings. This will involve integrating in-game feedback loops that offer hints, definitions, and progress updates based on the learner's interactions. AI-generated feedback will be semantically aligned with each student's input and performance, helping learners improve through positive reinforcement, guidance, and contextual clarity. This objective supports vocabulary acquisition by making the learning process more responsive and learner-centered

2 METHODOLOGY

2.1 Methodology

2.1.1 Requirements Gathering and Analyzing

After a number of discussions within the team and meetings with our supervisor, we started to cover what was needed to make the AI-powered language learning module. We conducted an extensive background study and review of literature centered on both existing vocabularies learning systems, CEFR based assessment systems and AI-based language learning systems. The aim was to understand what were the existent solutions, both their strength and weakness, in order to identify which gaps we could fill with our proposed system.

From such gap, the research problem was specifically articulated to reveal that there are no adaptive, personalized and engaging vocabulary tools that can work with student-generated input to personalize content based on CEFR proficiency. System requirement analysis was carried out prior to the start of the implementation to acquire an understanding of functional and non-functional requirements.

The key purpose of this study was to develop a model to personalize learning of varying proficiency levels based on student as well as personal preferences, combining the use of CEFR prediction, multi-agent Retrieval- Augmented Generation (RAG) and gamification design principles.

System Requirements of the Vocabulary Learning Module

- It should be able to analyze student-written input and predict their CEFR vocabulary proficiency level.
- It should use natural language processing to extract meaningful features like lexical richness and syntactic complexity.
- It should be able to generate vocabulary games (e.g., fill-in-the-blank, association games) based on user level and preferences.
- It should include a chatbot assistant capable of explaining word meanings,

giving usage examples, and providing hints.

2.1.2 Feasibility Study

- **Technical Feasibility**

Members of the project should have become more professional know how in the areas of machine Learning, AI, web app development and in software architectures and Frameworks.

- **Economic Feasibility**

The sub component should run fine with no errors or issues. The part should be reliable, high performance and cost effective. Minimal expense to resources and demands of the element.

- **Operational Feasibility**

A Member should be accountable for every phase of software life cycle and particularly for requirement analysis phase. The final produced product should meet the client's defined needs.

- **Scheduling Feasibility**

The proposed sub-component should be finalized within the time limitations as well as completing each task with higher accuracy results while maintaining the timeline. Finally, should present the final outcome of the product on the planned due date.

2.1.3 Problem Statement

This work focuses on the challenge of estimating English vocabulary ability and providing individualized learning support, especially for multi-lingual test takers. Most of the traditional vocabulary learning applications let a user adapt ate their offerings down to the level of “vocabulary size” – but not to where individual knowledge, performance, and learning speed fall. To mitigate this drawback, we use

a CEFR-based classification model to estimate the vocabulary level (from A1 to C2) of student-generated texts. Of this category, the system uses a Retrieval-Augmented Generation (RAG) architecture to produce custom tailored vocabulary games and definitions dynamically. By using a transformer-based embedding model, semantic retrieval (via Pinecone) and large language model (LLM) generation, the system provides context-aware, interactive content which ebbs and flows according to the learner's current proficiency. The system intends to be a system that can be used by more people rather than one that is just limited by experience, and to support personalized vocabulary learning by adaptive, interactive and feedback-based learning.

2.1.4 System Designs

2.1.4.1 Overall system diagram

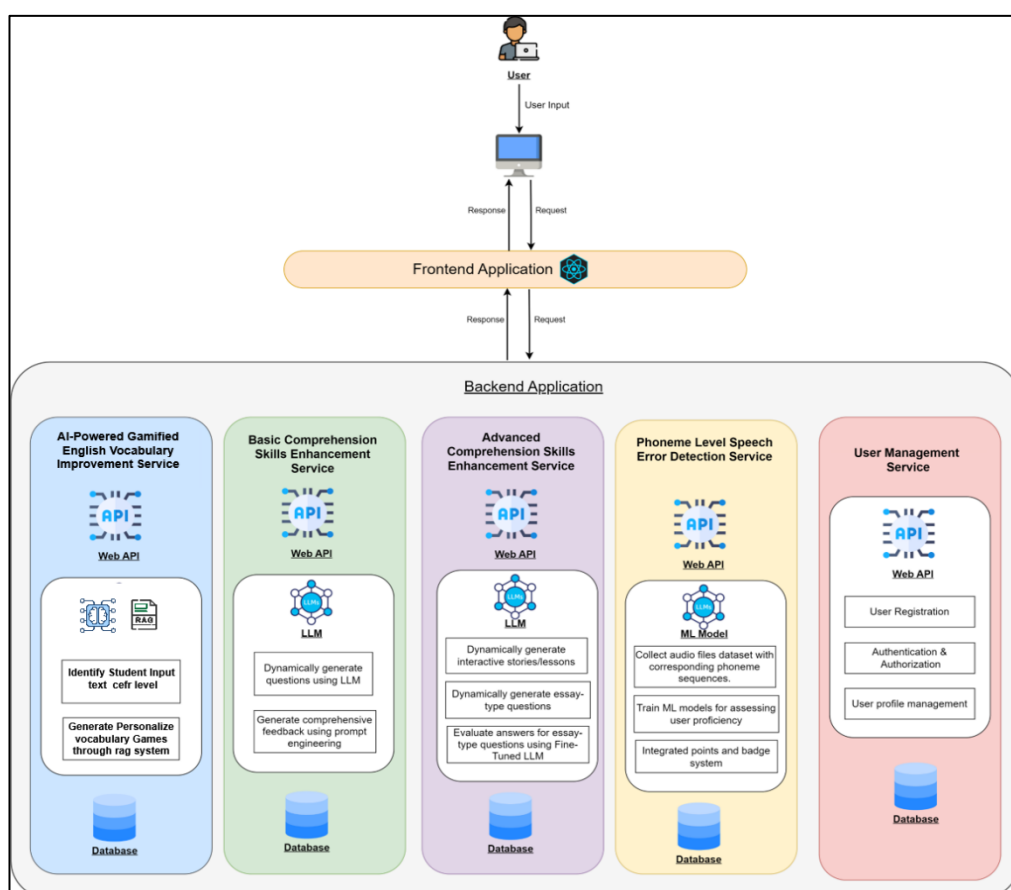


Figure 2.1 Overall system diagram

The full system architecture of the AI-based language learning platform is depicted in Figure 2.1 which contains a combination of intelligent services to offer personalized, gamified, and adaptive experience. The React frontend application. js) - The web interface, allowing users to type words, play vocabulary games, chat for word explanation and maintain users' profile. This frontend can integrate with various back-end services using RESTful APIs. These backend services consist of prediction of CEFR-level and personalized game generation by RAG system, reading comprehension processing by LLMs, phoneme-level speech error de- taction through ML models and secure user management, etc. Each module is a self-contained application and each has its own database making it possible to scale Site Builder in deployment and keep the maintenance effort focused as well as serve differing learning needs.

2.1.4.2 Design Diagrams for the component

2.1.4.2.1 Use Case Diagram

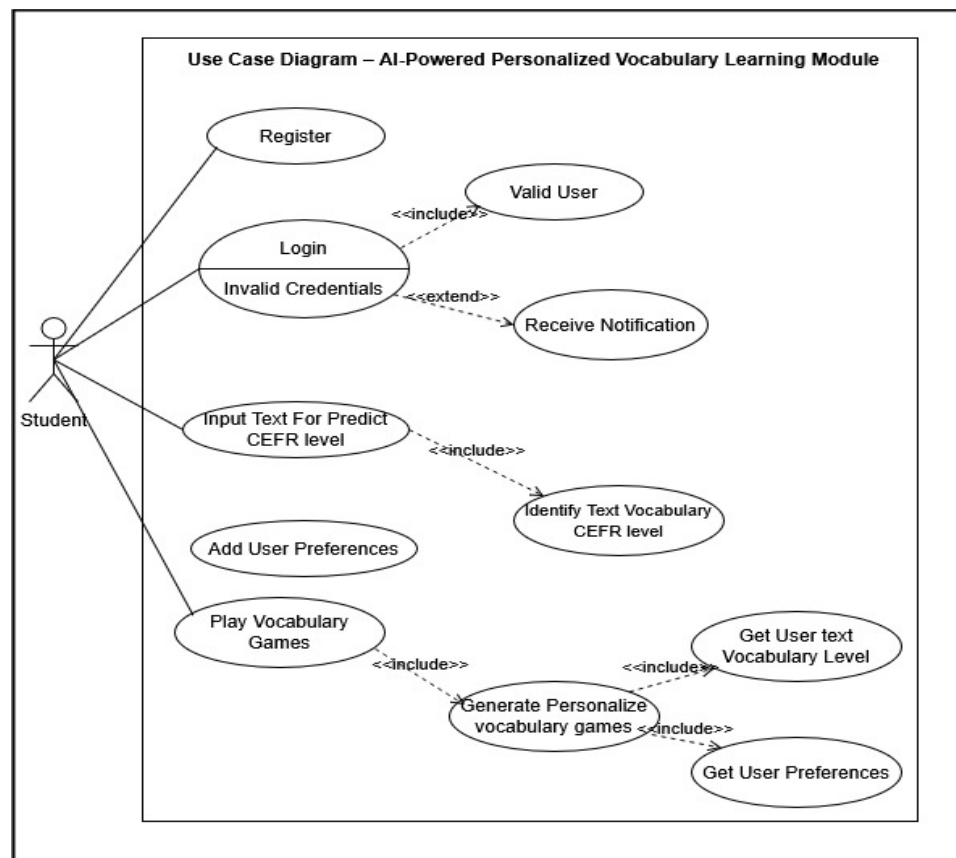


Figure 2.2 Use case Diagram of the component

2.1.4.2.2 Sequence Diagram

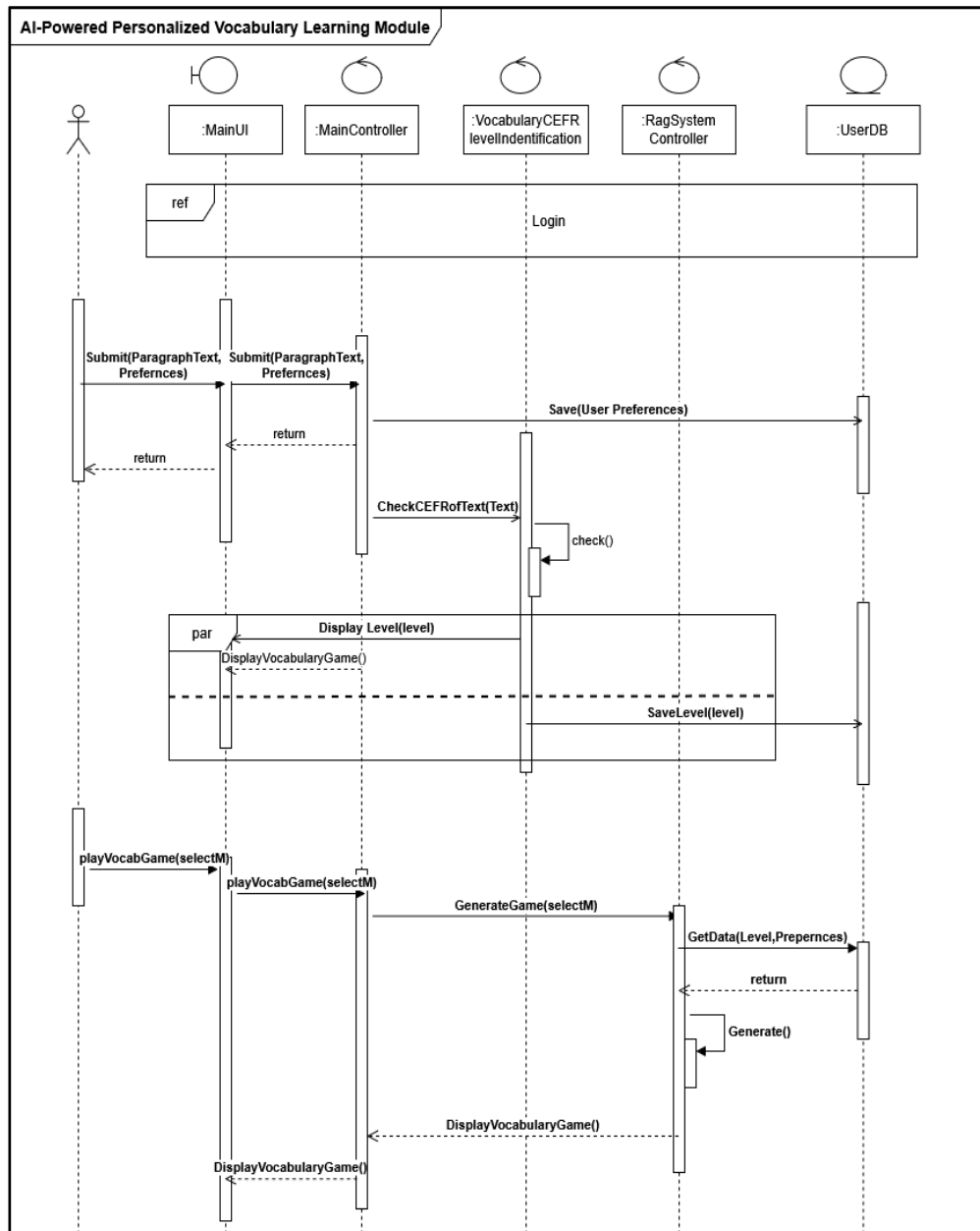


Figure 2.3 Sequence diagram of Component

2.1.4.2.3 Component System Diagram

The proposed system includes four main functionalities; and this component is concerned with the assessment of the learner's in input text vocabulary proficiency

level and the generation of the personalized vocabulary games. In this component the tasks are CEFR level prediction and game content generation with the RAG architecture to direct the development process.

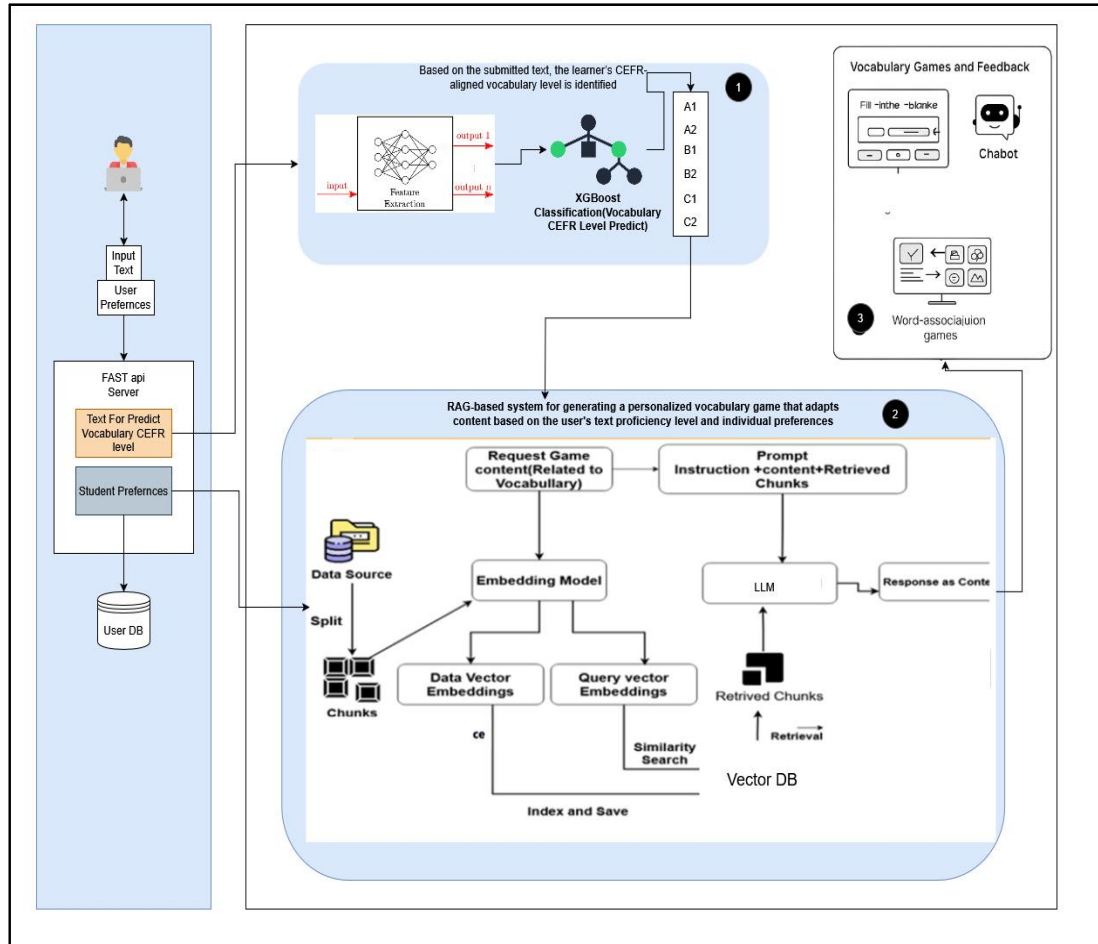


Figure 2.4 Component System Diagram

First, for the purpose of the interaction of the users, developed a user-friendly, reliable and accessible Web application using React JS. There are three sub-processes under the main objective for easy understanding and implementation purposes, as illustrated in Figure 2.4

Vocabulary CEFR Level Prediction Module (Block 1 in the figure 2.4) The first core block of the proposed system is the Vocabulary CEFR Level Prediction Module. This module works with learner submitted input data in the form of problem sentences or paragraphs and processes the input using a feature extraction process to do comparative analysis using various linguistic metrics such as sentence structure, word

frequency distribution and part of speech distribution. These features are fed to a pre-trained XGBoost classifier, fine-tuned on predicting the CEFR language levels (A1–C2) of the words. The most likely proficiency level, conditioned on what is input, is spat out by the classifier. This is a core prediction of the system and informs the challenge and range of content for later vocabulary game generation.

The second key feature denoted as (Block 2 in the figure 2.4) is RAG-based Personalized Game Generator. This part generates vocabulary games appropriate to the learner's CEFR level (Common European Framework of Reference) and topic preferences (e.g., sports, science). With Lang-Chain as conductor, the player's query is projected onto semantic vectors using a sentence transform model. These vectors are used to search a Pinecone Vector Database that includes pre-indexed words, definitions, and examples in context. Once the relevant data is fetched, it is processed through a Large Language Model (LLM) to generate the complete educational game content (fill-in-the-blanks games, word association game, word definition chatbot) based on the combined context & learner input. This smart content creation is tailored to be relevant and interesting for each user.

The last part coded as (Block 3 in the figure 2.4) in the system architecture is the Interactive Vocabulary Games and Feedback Module that is mainly developed at the front end utilizing React JS and styled with Tailwind CSS. This module is the main user interface that users use to interact with the customized games that the system creates. The skin focuses on the usability and response time, to make sure everything runs smoothly on all devices. In the course module, students can use a number of vocabulary-based activities, e.g. fill-the-blank exercises, word-definition matching and word association games according to their levels of knowledge. An integrated RAG chatbot supports two-way interaction in real time, where definitions, explanations, and context can be requested as needed. The module also includes active feedback loops: if a user is failing to figure something out, the system is alerted to the pattern and adjusts to provide easier tasks or more explicit hints. This intelligent process loop is generated on the frontend for learning vocabularies more interactive, customized, and efficient.

2.1.5 Data Acquisition and processing

To enable a smart and personalized vocabulary learning system, we developed two separate but related datasets, one for CEFR level assigning and the other for the Retrieval-Augmented Generation (RAG) pipeline for personalized game content and vocabulary definitions generation.

CEFR Classification Dataset

To develop a model that could predict the vocabulary proficiency level of students from their writing, a labeled dataset is needed. English learner texts were retrieved from free CEFR-aligned corpora according to educational domain such as educational materials, academic sources, and open CEFR datasets of a corpus-based research project, i.e., English Profile and EFCAMDAT and Kaggle. The collected data contains more than 3000 samples and is annotated with a CEFR level (A1 to C2) together with the students written response.

Each data point was a paragraph or sentence that a learner produced with its CEFR level. These included a range of topics and registers to maintain diversity. After gathering, the data was saved as .csv file with two columns: text, label Target file is csv format- it has the following two major columns: text and label. Raw data was then preprocessed to remove special characters, normalize punctuation and split sentences for feature extraction. The pre-processing work simplified the followed protocol and allowed the creation of a standard and high-quality dataset to train the algorithms. Figure 2.5: Collected sample of Sample CEFR Pattern Dataset

	A	B
1	text	label
2	I like to play football on weekends with my friends.	A1
3	She is reading a very interesting book about history.	B1
4	Despite the challenges, the company managed to increase its revenue.	C1
5	Can you tell me the way to the nearest train station?	A2
6	He had been working on the project for months before it was completed.	B2
7	The economic policies introduced last year had significant impacts.	C2
8	This is a pen.	A1
9	We went to the zoo and saw lions, tigers, and bears.	A2
10	It is important to understand the root cause of the problem.	C1
11	They are watching a movie together at home.	A2
12	I have a cat.	A1
13	Learning new languages can be very rewarding.	B2
14	She danced gracefully across the stage.	B1
15	The boy runs fast.	A1
16	Global warming affects all of us.	B2
17	My brother is a doctor.	A1
18	He didn't know the answer, so he asked the teacher.	A2
19	The president addressed the nation regarding the crisis.	C2
20	What is your favorite color?	A1
21	This new technology could revolutionize the way we communicate.	C2
22	Birds are flying in the sky.	A1
23	I will call you tomorrow.	A2
24	He tried to solve the equation but made a small mistake.	B2
25	Please sit down and open your book.	A1
26	Environmental sustainability should be our top priority.	C1
27	She has three sisters and one brother.	A1
28	I usually eat lunch at 1 p.m.	A2
29	They will travel to Europe next summer.	B1
30	If I had more time, I would learn to play the guitar.	B2
31	He found the instructions confusing and asked for help.	B1
32	The sun rises in the east.	A1
33	We went shopping yesterday.	A2

Figure 2.5 Dataset Used for CEFR Classification

Vocabulary Dataset for RAG System

To support the personalized vocabulary game generation module, an extensive English word dataset with more than 8000 words was created. All entries in the dataset contained important linguistic and pedagogical information: English word, CEFR level, a simplified definition that is learner friendly, sense and example sentence number, synonyms (if available), related words, and a topic such as Health,

Technology, or Education. Ultimately, these capabilities made it feasible for the system to personalize vocabulary games in terms of both the student's language competences and personal interests. The lemmatization list was composed with the aid of several high-quality sources, including Wiktionary, WordNet (through the NLTK library), Tatoeba, the Datamuse API, openly licensed CEFR-aligned wordlists and the Oxford Learner's Dictionaries (Oxford) for academically accredited definitions and usage samples. As a result, all data entries were cleaned, normalized and were stored in a central structured CSV file that will be compatible with the RAG system.

Figure 2.6: Sample Vocabulary Dataset for the RAG System to choose from.

word	type	cefr	definition	example	Synonyms	Antonyms	Rhymes	Related
1	a	indefinite article	a1 used before countable or singular nouns referring to people or things	the man/horse/unit	equally, every bit, arsenic, atomic number 33, electricity	the, ca, kumbaya, le, duh		
2	abandon	verb	b2 to leave somebody, especially somebody you are responsible for, with	abandon somebody, The baby had been	forsoke, give up, vacate, desert, desolate	branden, stand-in, hand i altogether, retre		
3	ability	noun	a2 the fact that somebody/something is able to do something	People with the disease may lose their a	power, suitability, fit, competency, i inability, unfitnes	facility, humility, liability, aptitude, manip		
4	able	adjective	a2 to have the skill, intelligence, opportunity, etc. needed to do something	You must be able to speak French for thi	capable, competent, fit, healthy, abli unable, not able	table, label, enable, stable withstand, convin		
5	about	adverb	a1 a little more or less than; a little before or after	It costs about \$10.	near, high, well-nigh, almost, virtually	out, route, doubt, tout, rout		
6	above	preposition	a1 on the subject of somebody/something; in connection with somebody/	a book about flowers	near, high, well-nigh, almost, virtually	out, route, doubt, tout, rout		
7	above	adverb	a1 at or to a higher place	Put it on the shelf above.	supra, preceding, higher up, in a high below, beneath, at a low of, low, dove, shove, glo asphalt, sea, leve			
8	above	preposition	a1 at or to a higher place or position than something/somebody	The water came above our knees.	supra, preceding, higher up, in a high below, beneath, at a low of, low, dove, shove, glo asphalt, sea, leve			
9	abroad	adverb	a2 in or to a foreign country	to go/travel/live/study abroad	foreign, afield, overseas, beyond the sea, over the sea	god, rod, facade, pod, fra overseas, signing		
10	absolute	adjective	b2 total and complete	I've joined a class for absolute beginners	unconditional, unconditional, unique relative	suit, attribute, moot, shoi primogeniture, m		
11	absolutely	adverb	b1 used to emphasize that something is completely true	You're absolutely right.	utterly, perfectly, dead, plainly, downright	acutely, resolutely, astute fabulous, conver		
12	academic	adjective	b1 connected with education, especially studying in schools and universiti	high/low academic standards	scholarly, donnish, academician, theoretical, pedantic	polemic, endemic, epider achievement, jou		
13	academic	noun	b2 a person who teaches and/or does research at a university or college	a leading/distinguished/prominent acad	scholarly, donnish, academician, theoretical, pedantic	polemic, endemic, epider achievement, jou		
14	accept	verb	a2 to take willingly something that is offered; to say "yes"	He asked me to marry him and I accepte	admit, consent, have, take on, assum decline, refuse, reject, ti	precept, adept, except, in reject, invitation,		
15	acceptable	adjective	b2 agreed or approved of by most people in a society	Children must learn socially acceptable	t: satisfactory, fit, good, tolerable, acco unacceptable	susceptible, imperceptible unacceptable, ac		
16	access	noun	b1 the opportunity or right to use something or to see somebody/somethi	High-speed internet access has become	i get at, approach, admittance, entree, memory access	process, address, assess, sanitation, intern		
17	access	verb	b1 to open a computer file or use a computer system	Most people use their phones to access	i get at, approach, admittance, entree, memory access	process, address, assess, sanitation, intern		
18	accident	noun	a2 an unpleasant event, especially in a vehicle, that happens unexpectedly	a car/road/traffic accident	fortuity, chance event, accidental, casualty, adventitious	come, collision, v		
19	accommodate	noun	b1 a place to live, work or stay in	rented/temporary accommodation	adjustment, fitting, hostel, lodgings, lodging	communication, informal pusher, hostel, ca		
20	accompany	verb	b2 to travel or go somewhere with somebody/something	accompany somebody/something + adv.	keep company, companion, company, follow, come with	company, tympany, tympt yonto, onto, le		
21	according	preposition	a2 as stated or reported by somebody/something	According to Mick, it's a great movie.	based on, apparently, attributing, accordance, approximately			
22	account	noun	b1 an arrangement that somebody has with a bank, etc. to keep money th	I don't have a bank account.	account statement, invoice, report, write up, news report	count, mount, paramoun revenues, accoun		
23	account	verb	b2 to have the opinion that somebody/something is a particular thing	be accounted + adj., In English law a per	account statement, invoice, report, write up, news report	count, mount, paramoun revenues, accoun		
24	accurate	adjective	b2 correct and true in every detail	an accurate description/picture of some	veracious, true, exact, precise, straig inaccurate	inaccurate measurements, c		
25	accuse	verb	b2 to say that somebody has done something wrong or is guilty of someth	accuse somebody of something, to accu	criminate, incriminate, impeach, charge, accusation	lose, choose, abuse, musu plagiarism, hypoc		
26	achieve	verb	a2 to succeed in reaching a particular goal, status or standard, especially b	He had finally achieved success.	attain, accomplish, reach, attainment, attained	leave, naive, retrieve, per fear, objectives, c		
27	achievement	noun	b1 a thing that somebody has done successfully, especially using their own	the greatest scientific achievement of th	accomplishment, achieving, achieved, successful, attainment	bereavement priority, perce		
28	acknowledge	verb	b2 to accept that something is true	acknowledge something. She refuses to	admit, recognize, know, receipt, noti deny	knowledge, college, tolla suzerainty, recei		
29	acquire	verb	b2 to gain something by your own efforts, ability or behaviour	She has acquired a good knowledge of E	take on, get, develop, produce, take	fire, wire, dire, prior, desi stake, acquisitio		
30	across	adverb	a1 from one side to the other side	It's too wide. We can't swim across.	crosswise, crossways, over, crossed, uncrossed	cross, sauce, loss, boss, n globe, continent		
31	across	preposition	a1 from one side to the other side of something	He walked across the field.	crosswise, crossways, over, crossed, uncrossed	cross, sauce, loss, boss, n globe, continent		
32	act	noun	b1 a particular thing that somebody does	You have committed a serious criminal	a work, number, process, operation, r; refrain, forbear, forebear impact, abstract, contrac repealed, amend			

Figure 2.6 Dataset used for Rag System

Preprocessing and Augmentation

Once both datasets were prepared, separate preprocessing pipelines were applied:

- For CEFR classification, we used a Python script to derive more than 20 linguistic features from each learner input, which we represented by, e.g., readability metrics (based on the text's lexical and syntactic features), syntactic complexity (POS tag distribution, parse tree depth), and lexical richness (type-token ratio) .

- For vocabulary processing (RAG), the sentence embedding of each word entry was generated using a transformer-based sentence encoder (all-MiniLM-L6-v2). These were loaded into the Pinecone vector database and could be leveraged for context-aware semantic search.

Such dual-dataset method was to have predict model and generative modules worked on well-structured and semantically rich data, hence to earn high accuracy in CEFR classification and relevancy in personalized game contents.

2.1.6 Choosing the Correct Framework

To build a reliable, scalable and efficient system for personal vocabulary learning, different tools and frameworks were selected. These are tools that can meet the system's requirements, such as ML, NLP, real-time data access, web interactivity, and gamification. Here is the list of the main components / frameworks used for the stack:

ML Model Training (Scikit-learn & XGBoost)

Scikit-learn and XGBoost are chosen for the core machine learning model training. Scikit-learn was selected because it is simple, modular, and has an extensive ecosystem for preprocessing and testing machine learning. It is heavily applied in common machine learning problems including feature construction, classification, and model validation. It gives you an easy way to access all those classifiers (click [here](#) for example usage), including logistic regression, SVM and random forests, which is an advantage when you are dealing with feature engineering and model evaluation as I was for the CEFR classification task.

XGBoost (Extreme Gradient Boosting) was selected as the main classification model because it outperforms other algorithms when the data are structured, even in the presence of complex feature interactions. XGBoost has been proved as one of the most powerful algorithms in accuracy and speed in many classification problems [22]. The fact that it is designed for imbalanced datasets and for the use of gradient boosting, among other classifiers, was particularly relevant in this task to predict CEFR levels

using the derived features of words like lexical richness and syntactic complexity.

Feature Engineering (spaCy & Textstat)

Feature engineering plays a key role in the precision of machine learning models. Here, we pre-processed the text of the learner using custom Python code and extracted linguistic features. spaCy was employed for NLP including POS tagging, NER and syntactic parsing, all of which are needed for collecting sentence complexity and vocabulary use information. To analyze sentence structure, perform PoS tagging and NER, we used the `en_core_web_sm` model. [23] and the relatively small and efficient pre-trained spaCy model. Readability scores, which align directly with the CEFR proficiency's levels, were obtained using Textstat.

The feature extraction was careful and efficient, as we employed spaCy and Textstat [24]. Spacy is fast and accurate text-processing in python and is very useful for linguistic analysis. The `en_core_web_sm` model was a happy medium between speed and performance, because we wanted to ensure that text analysis could be completed close to real-time. Textstat adds to this aspect by bringing in the readability scores which are an important factor in language assessment. These tools have been applied in many NLP studies, and are very effective in obtaining fine-grained features from textual data which are indispensable for training machine learning models.

Text Embedding (all-MiniLM-L6-v2)

The Retrieval-Augmented Generation (RAG) system was implemented using MiniLM-L6-v2 from the sentence-transformers library to map vocabulary-based information into dense semantic embeddings. This model was chosen due to its ability to generate dense but semantically rich sentence embeddings, which is important for precise similarity search in the vectorized Pinecone database. These embeddings were instrumental in maintaining context relevance of returned vocabulary entries such as definitions and synonyms and by usage examples regarding the learner's input. When the student proposes a topic or search key, we embed it with all-MiniLM-L6-v2 and match it to the embedded vocabulary dataset to get the best semantic matches for

personalized game generation [25].

The all-MiniLM-L6-v2 model is selected given the good balance between computational effort and embedding quality. It can be run on real-time systems with less latency than larger transformer models (BERT-large), while providing embeddings that perform well on both syntactic and semantic levels. This mode is particularly well-suited for interactive educational applications (e.g. real-time vocabulary matching and game personalization), where both speed and accuracy are crucial.

RAG and Retrieval (Lang Chain + Pinecone + Gemini (GenAI))

Lang Chain was adopted to orchestrate the process of deploying the RAG system where Pinecone was used as the vector database for optimization of retrieval. Lang Chain integrates effectively with large language models (LLMs) like GPT, generating dynamic content by combining the retriever with the generative model of the LLM. Pinecone, however, is ultra-optimized for vector similarity search, an important operation needed to pull semantically relevant vocabulary data in real-time with real-time game generation. Gemini (GenAI) also served as the tool for automating construction of context-dependent vocabulary definitions and examples grounded in the data-extracted knowledge. The powerful generative AI model is able to generate personally tailored vocabulary learning exercises according to the learner's input and vocabulary proficiency [6].

The retrieval-generation integration of LangChain gives it a powerful, online learning infrastructure, and Pinecone's ultra-low-latency similarity search ensures that the system quickly retrieves the best content. Gemini (GenAI) It will offer a real-time content adaptation at the paragraph level.

Backend API (Fast API)

Fast API was selected as backend to be used in view of it's good performance and user friendly interface. Here is wise choice. It supports non-blocking request processing,

which is a must for real-time systems need to access databases and machine learning models without adding any delay. Fast API also enjoys auto-generation of interactive API documentation that makes development and debug easier [26].

The responsiveness and efficiency of Fast API makes it well-suited to working with real-time API calls in a responsive system, where various components of the system (game generation, data discovery, and learner interaction to name a few) must all occur concurrently.

Frontend Web (React JS with Tailwind CSS)

For frontend, the use of Reactjs was proposed to make the UI highly interactive and responsive. The component-based architecture of React and its extensive ecosystem enabled us to create reusable UI components that improved overall interaction and user experience. For styling the interface, we used Tailwind CSS for its utility first philosophy, allowing us to quickly create interfaces that are custom to us and with low amount of code [27].

Database (Firebase & Pinecone DB)

The Firebase was selected as the central database responsible for handling user profiles, progress information and preferences on account of its real-time synchronization and high security level [29]. We chose Pinecone DB to store vectors for the embedding retrieval as it is the most important component in the content generation part of the RAG system [28].

Firebase is the perfect complement to your mobile app and integrates neatly into your existing development workflow, providing an easy way to store and sync app data in real-time. Pinecone DB is offering high-speed storage and retrieval of vector embeddings, which is essential for the real-time delivery of personalized content.

2.1.7 Identification of CREF Level of the student given input text

In order to determine students' vocabulary proficiency level effectively, a new classification model was built according to CEFR (Common European Framework of Reference for Languages) levels: A1 (elementary level) to C2 (proficient level). Its goal is to classify text written by a learner into the corresponding CEFR band in terms of the language proficiency, based on some written input. This categorization forms the core of the personalization pipeline, which allows the system to adapt vocabulary games, content, and learning paths in a highly dynamic manner to each learner's true language ability.

The system was developed on basis of several most innovative math-computation methods. First, using Natural Language Processing (NLP) tools, it extracts informative patterns and linguistic features from the raw text, among which the structure of sentences, lexical richness and syntactic complexity. Second, a combination of manual and statistic features, which is computed based on readability metrics and the distribution of part of speech of the text, is used to representation of the text in quantitative of difficulty and complexity. These features are then input to supervised machine learning classifiers (e.g., XGBoost, Support Vector Machines (SVC), and Random Forests) to learn to map certain patterns in the input to the CEFR label in a labeled training dataset.

Blending linguistics theory and data-driven AI modeling, the solution provides a scalable, automated means of assessing language proficiency. This also obviates traditional laborious manual evaluations and facilitates real-time treatment planning such as in dynamic learning. Thus, students will learn the game and hint and word definition content that matches their current level of lexical knowledge, which should facilitate and enhance language learning of vocabulary.

2.1.7.1 Input Text Preprocessing

The preprocessing phase is very important to normalize the input of users and to produce reliable linguistic characteristics as a classifier. An ad hoc Python pipeline,

employing spaCy, textstat, and re libraries, was implemented for text cleaning, tokenization, and processing. The following procedures were carried out. Figure 2.7 Text Preprocessing workflow This is a workflow to capture the student input in text and break down into what the student usually meant.

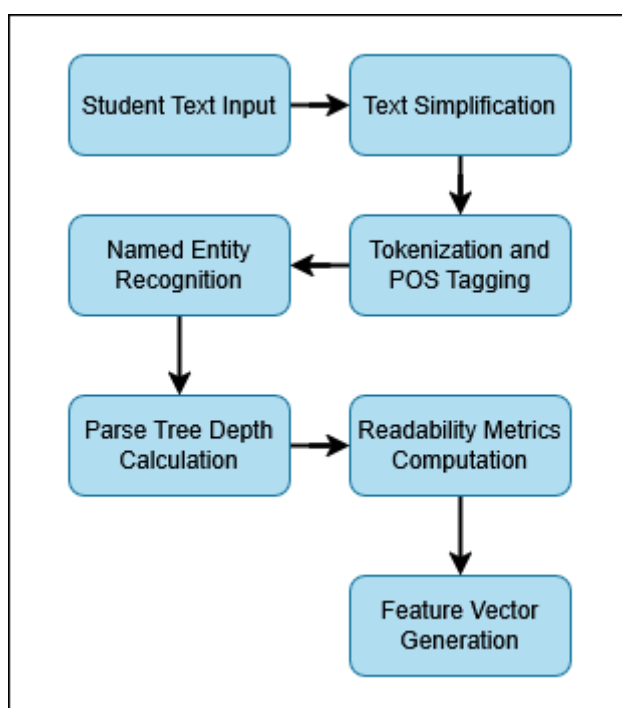


Figure 2.7 Text Preprocessing Pipeline

- **Text Simplification**

The input text should be standardized prior to any linguistic analysis or modeling to eliminate noise and variation. This process, called text simplification, includes normalizing punctuation, stripping unwanted special characters and spaces that may disturb the subsequent processing. Regular expressions (regex) were used to replace irregular symbols (e.g., semicolons, multiple full stops, new lines) with standard punctuation. Repeated punctuation marks were reduced, sentence boundaries were standardized, and line breaks were rendered as whitespace. This standardization helped to ensure the text fit with the homogeneous body of the text data, which in turn they made no difference for basic linguistics techniques to tokenize and parsing the sentences correctly.

- **Tokenization and Part-of-Speech (POS) Tagging**

After cleaning, the text was tokenized using spaCy's `en_core_web_sm` language model. Tokenization is the process of segmenting the text into smaller objects (tokens), like words, punctuation and symbols. Additionally in the same parsing phase each token was labeled with a Part-of-Speech (POS) tag such as NOUN, VERB, ADJ, representing the grammatical function of the word in the sentence. This POS tagging was used to recognize syntactic structure and to compute such tag distributions (average number of verbs or nouns per sentence, etc.) which were later used as input features for the task of CEFR level classification. Table 2.1 gives an overview over the POS tags and the relation to classification into CEFR levels.

Table 2:1 POS Tags and Their Linguistic Importance

POS Tag	Description	CEFR Relevance
NOUN	Common/Proper Nouns	Indicates vocabulary usage and lexical richness
VERB	Actions/States	Shows tense variety and complexity
ADJ	Adjectives	Descriptive richness
ADV	Adverbs	Fluency and expression
CONJ	Conjunctions	Sentence linking complexity

- **Named Entity Recognition (NER)**

The feature selected were also enriched with the Named Entity Recognition step, which detected and classified named entities in the input text. Named entities are proper nouns such as person names, locations, dates, and organizations. By utilizing the NER feature provided by spaCy, it calculated the overall number of entities and the average number of entities in each sentence. This measure was a rough indicator of semantic density and contextual complexity, which are known to differ between learners at different CEFR levels (e.g. advanced learners being more specific and informative).

Table 2.2 shows how the Named Entity Recognition (NER) process can assist in preserving the full richness of context and semantics in student input. The higher CEFR levels contain more named entities, which indicates richer vocabulary and a higher degree of topic specificity.

Table 2:2 Named Entity Recognition and CEFR-Level Inference

Input Text	Extracted Named Entities	Number of Entities	Avg. Entities per Sentence	Likely CEFR Inference
"Albert Einstein was born in Germany in 1879 and later moved to the US."	Albert Einstein (PERSON), Germany (GPE), US (GPE), 1879 (DATE)	4	4.0	B2/C1 (Advanced contextual use)
"I went to school yesterday."	yesterday (DATE)	1	1.0	A2/B1 (Basic narrative context)
"Google and Microsoft have launched new AI models in 2024."	Google (ORG), Microsoft (ORG), 2024 (DATE)	3	3.0	C1/C2 (Technical, informative)

- **Parse Tree Depth Measurement**

For syntactic complexity, the mean depth of the syntactic parse tree per sentence was computed. how often the grammatical structures are nested; and the higher CEFR levels are typically associated with more complex and lengthier sentence structures. A recursive function was used to traverse the constituent structure tree of each sentence, the maximum depth from root to leaves being computed. These were averaged across the passage to produce a measure of syntactic complexity.

- **Readability Metrics Calculation**

Finally, so as to measure the overall readability of the text, we used the textstat library to calculate various standardized readability indices. Table 2.3 Readability Metrics

Processed for CEFR Prediction. These comprised the Flesch Reading Ease Score, Gunning Fog Index, SMOG Index and the Dale-Chall Score, among others. Each index calculates how much effort it takes to read a passage, based on elements such as word length, sentence length and syllable count. These measurements offer not a direct, but still a helpful extrapolation of language ability and are both forward- and backward-compatible with CEFR’s emphasis on how fluently and accurately learners express themselves.

Table 2:3 Readability Metrics Used in CEFR Prediction

Metric	Description	Scale / Meaning
Flesch Reading Ease	Ease of reading (higher = easier)	0–100 (90-100 = A1, 0-30 = C1/C2)
Gunning Fog Index	Years of formal education needed	6–18+ (higher = more complex)
SMOG Index	Based on polysyllabic words	Predicts grade level needed
Dale-Chall Score	Based on difficult words list	Scores >8 are more complex
Linsear Write Formula	Focuses on sentence and word complexity	Ideal for technical writing and CEFR mapping

Such a structured-preprocessing pipeline ensures the robustness and reproducibility of the extracted features that support the CEFR-level classification engine.

2.1.7.2 Feature Engineering

To successfully predict a learner’s proficiency level using the CEFR framework, extensive feature engineering has been employed. The purpose of this stage was to identify what linguistic and structural features of the student’s written input could be mapped against A1 (basic) to C2 (proficient) within CEFR. Feature engineering is the foundation of the classification model, as it transforms raw text data into quantitative measures of language complexity, vocabulary and sentence construction. These features help the machine learning algorithm to make the model aware of the different levels of proficiency observed in the input data.

- **Lexical Richness**

Lexical density is the diversity and individuality of vocabulary utilized by a learner. It is generally accepted as a valid predictor of language competence since advanced users tend to have a larger and more varied lexicon. Measures such as the ratio of unique words to total words in a text (i.e. type-token ratio) were used to capture this. Frequency of vocabulary use was also examined, including how often common, low commonality, or academic words appeared in the text. Higher-CEFR learners tend to use more, less frequent, context-specific vocabulary, and lower-CEFR learners use more basic and more heavily on high-frequency terms.

- **Syntactic Features**

Syntactic complexity is critically involved in how smooth and efficient a learner formulates sentence with. For this purpose, syntactic features were considered, in particular part-of-speech (POS) distribution. This involves averaging the number of key grammatical items such as nouns, verbs, adjectives, and adverbs within sentences from a learner's input. The difference and distribution of these components demonstrate concerns of style, stylistic variation and complexity. For example, basic-level materials usually have a simpler POS structure characterized by the frequent occurrence of nouns and common verbs, while more advanced materials are loaded with a wide variety of modifiers, auxiliaries, and complex connectors that facilitate compound and complex constructions.

- **Text Complexity**

Text complexity incorporates both the structural and grammatical complexity of language, which generally grows as an individual advances from CEFR levels. This was reflected on sentence and sub-sentence lengths and complexity of syntactical structures. The average sentence length can be used to examine the complexity of the learner's constructions for articulating the learner's ideas and how compounded they are, and, by looking at the depth of syntactic parse trees per sentence, the emphasis is on surgical accuracy, in which the number of levels of trees relates to how many layers

deep learners comprise their sentences. Texts that consist of longer sentences containing subordinates and more complex sentence structures are more probably indicative of proficiency levels, as these indicate a higher degree of control over written language and more structural variety.

- **Named Entity Density**

Another important product of advanced language is the density and specificity of named entities: elements such as names of people, locations, organizations, and dates, that contribute some semantic depth to a passage. To test the extent to which learners include real world references, contextually relevant details were measured and the average number of named entities per sentence is reported. Texts in higher CEFR levels contain more named entities, which signify the learner's proficient use of specific accurate and fluent information, as the anchor for which they use beginning-level CEFR texts which employ general or the well-known entities.

In this way, the system was able to describe to a certain extent of measurements how diverse aspects of learner writing between descriptors of the CEFR scale were captured by the engineered feature set. These features were then passed to machine learning classifiers to learn models predicting the correct proficiency level from such detailed linguistic profiles.

2.1.7.3 Model Training and Selection

After a set of structured linguistic, syntactic, and semantic features that formed from learner-generated text (in line with section 2.1.7.2) is extracted, the subsequent important role of this component is to train the machine learning model such that it can predict the user's vocabulary proficiency level of the input text. Successful level profiling of students' lexical input is crucial in order to design follow-up personalized vocabulary games and learning tasks that match their actual level. The goal of this classification task was to consider the multi-class supervised learning problem and make the system work to assign each learner input into one of the six CEFR classes.

$$y \in \{A1, A2, B1, B2, C1, C2\}$$

based on an input feature vector:

$$\mathbf{x} \in \mathbb{R}^n$$

where n is the number of linguistic features introduced extracted (type-token ratio, parse tree depth, named entity density and the readability scores). Part of the work in this section was concerned with the model training, tuning and evaluation, as well as the deployment pipeline for such a classification task as described here.

Selection of XGBoost for Vocabulary Proficiency Prediction

XGBoost was chosen as the classification model for several key reasons directly aligned with the goals of this component

- **Handling Structured Linguistic Features**

XGBoost is very powerful when used with feature-rich structured data, here, engineered syntactic and semantic features (parse tree depth, type-token ratio, named entity count) obtained from learner text. It is better at learning non-linear relationships between features than a simple model like Logistic Regression or a linear SVM.

- **Support for Multi-Class Classification**

Because the Common European Framework of Reference (CEFR) specifies six different language proficiency levels, we needed a model more natively capable of handling multi-class classification. XGBoost's `multi:softprob` objective function allowed prediction of probability distribution over all CEFR levels, which increased accuracy and allowed confidences for the predictions to be scored.

- **Built-in Regularization to Prevent Overfitting**

Language proficiency information may be shared unequally between students. In order to prevent overfitting on particular patterns or styles, we employed the built-in

L1 (Lasso) and L2 (Ridge) regularization features of XGBoost to enable the model generalize well on unseen student input.

.

- **Feature Importance and Explainability**

A relevant aspect in designing the system was to analyze which linguistic features contributed more to the final prediction. In the XGBoost, we obtain the feature importance score automatically, which helps in explaining and further improve the learning paths, based on key vocabulary or the syntactic complexity factors.

- **Efficiency and Scalability for Real-Time Prediction**

The module had to be real-time, so the CEFR level of the learner is immediately predicted after the input. Because of its computational efficiency, native support for missing values and parallel building of trees, we found XGBoost to be a perfect candidate when it comes to the back-end integration of the RAG based vocabulary game generator.

Model Training Process

The training pipeline developed in this component consisted of the following systematic steps:

- **Data Loading and Preprocessing**

Initially, First, the process loads the filtered learner input texts and their CEFR labels from .csv files. Samples of texts were cleaned and normalized in preprocessing phases (Section 2.1.7.1). All diary days included.

- The raw learner texts.
- Its manually annotated CEFR level (A1–C2).

This ensured a high-quality, structured dataset suitable for supervised machine learning model training.

- **Feature Engineering (Brief Mention)**

As detailed earlier in Section 2.1.7.2, feature engineering was applied to transform each learner’s input into a structured numerical representation.

Key features extracted included

- Lexical richness (Type-Token Ratio)
- Syntactic complexity (Mean Parse Tree Depth)
- Semantic density (Named Entity Density)
- Readability metrics (Flesch Reading Ease, Gunning Fog Index).

This step was crucial because machine learning algorithms require structured, quantitative input rather than raw unstructured text.

- **Label Encoding**

Since machine learning models operate on numerical outputs, the CEFR labels were encoded into integers using Label Encoding. Table 2.4 demonstrate labels and CEFR level.

Table 2:4 Label Encoding

CEFR Level	Numeric Label
A1	0
A2	1
B1	2
B2	3
C1	4
C2	5

This step is essential for machine learning classifiers, which require numerical target values to compute losses and gradients during training.

- **Feature Scaling**

To ensure that all features contributed proportionately to the model training, standardization was applied. Z-score normalization was used

$$z = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of a particular feature. Scaling will help to find a stable gradient boosting optimization, and will prevent features with larger numeric ranges from dominating when visualizing the model.

- **Initial Model Training with XGBoost**

An XGBoost classifier was chosen to train the model and it was due to its excellent performance on the structured dataset and its ability to model nonlinear complex feature interactions. We designed the training with this implementation.

- multi:softprob for multi-class classification, which yields a probability distribution over CEFR levels.
- Training/Validation Split: 80% of the data was used for training and 20% for evaluation.
- Early Stopping: A mechanism to halt training if the model performance on the validation set did not improve over several iterations.

The XGBoost model minimizes a regularized loss function of the form:

$$L(\theta) = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

- $l(y_i, \hat{y}_i)$ is the multi-class loss (Softmax cross-entropy),
- $\Omega(f_k)$ is a regularization term that penalizes model complexity to avoid overfitting.

- **Evaluation on the Validation Set**

The trained model was then evaluated on the reserved 20% test set using several metrics

- Overall Accuracy: The ratio of correctly classified instances to total instances.
- Precision, Recall, and F1-Score: Calculated for each CEFR class to assess the model's performance in detail.
- Confusion Matrix: Analyzed to understand misclassification trends particularly useful for identifying confusion between adjacent CEFR levels (B1 vs B2).

This thorough evaluation verified that the model achieved acceptable performance and generalizes well to unseen learner inputs.

▪ **Model Saving for Deployment**

Once trained and validated, the best-performing XGBoost model was serialized using Python's pickle module and stored for real-time deployment.

- The serialized model enabled integration into the web application backend.
- Allowed instant CEFR prediction for incoming learner text.
- Supported efficient scaling to multiple users without the need for model retraining.

Saving the model separately ensured reproducibility and consistency in the prediction results during the application.

2.1.7.4 Hyperparameter Optimization

After Once the initial model was trained and tested the optimization of hyperparameters was a crucial stage in improving the XGBoost classifier performance. Hyperparameters dictate the behavior of the learning algorithm (such as learning rate, tree depth, number of estimators) and aren't learned directly from training data. Choosing proper hyperparameters can well improve model generalization, convergence rate, and predicting performance.

For CEFR Proficiency Classification, we used **Bayesian Optimization** for hyperparameter tuning in the XGBoost model. This implementation was preferred as it allows the search over complex hyperparameter spaces without needing an exhaustive amount of trials. As there are many hyper-parameters, and each variable is dependent on some other parameters, Bayesian Optimization helped to converge faster to better values, saving computational resources, whereas obtaining higher model accuracy. Furthermore, due to source texts being generated by learners, to some extent, the training data on which the objective function was defined was noisy and so Bayesian Optimization's approach in inferring a posterior distribution over the objective function was well suited to this application. By orchestrating the hyperparameter search based on previous trials, this method made certain that the resulting XGBoost model produced stable, predictive, and scalable proficiency predictions so as to be employed for real-time personalized vocabulary learning.

Workflow of Hyperparameter Optimization and Final Model Training

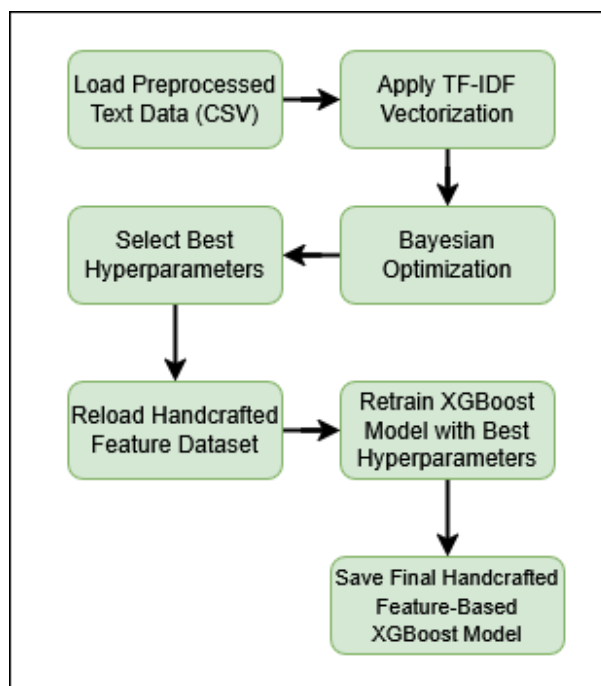


Figure 2.8 Hyperparameter Optimization Pipeline

▪ **Data Loading and TF-IDF Vectorization**

From Figure 2.8 Pre-processed CSV files that were loaded as training and testing datasets as before. In contrast to the first training phase that relied on hand-crafted linguistic features (e.g., type-token ratio, parse tree depth and readability scores), we used TF-IDF vectorization at this step. This simplification of the feature space resulted in greatly reduced computational load, making the evaluation of a large combination of hyperparameters more time-efficient.

▪ **Label Encoding**

The CEFR proficiency level labels were converted into numerical format by applying a Label Encoder to make the format homologous to the XGBoost and scikit-learn pipelines. This was necessary to process the categorical labels for supervised learning models.

▪ **Definition of the Hyperparameter Search Space**

A bounded search space was carefully defined for the key hyperparameters influencing XGBoost model behavior:

- learning rate: Step size shrinkage to prevent overfitting,
- max_depth: Maximum depth of a decision tree to control model complexity,
- n_estimators: Number of boosting rounds,
- subsample: Fraction of training instances used for each tree to introduce randomness,
- colsample_bytree: Fraction of features randomly sampled for each tree,
- gamma: Minimum loss reduction required to make a further split on a leaf node.

These parameters collectively balance the bias-variance trade-off and significantly affect model capacity and performance.

▪ **Bayesian Optimization Using BayesSearchCV**

Bayesian Optimization was conducted using the BayesSearchCV method on the training set using 5-fold cross-validation. In contrast to random or grid search, Bayesian approaches treat the objective function as a random process and update the hyperparameters choice history between evaluations. Every repetition positively impacted validation accuracies based on increasingly better values of the hyperparameters.

▪ **Model Evaluation**

Once optimization was complete, we evaluated the model in the test set using the TF-IDF based XGBoost with the best hyperparameters. This test offered a realistic assessment of the model's generalization performance with the optimized settings.

▪ **Model Saving**

While the TF-IDF-based model was not a model that would have been deployed in production, we serialized it to disk using the pickle module.

This enabled reproducibility, future comparisons of other model configurations, and traceability of the optimization results.

▪ **Final Model Retraining with Handcrafted Features**

Using the best performing hyperparameter configuration obtained from TF-IDF based hyperparameter tuning, the optimal settings was applied to the new instance of XGBoost. The model was further retrained over the rich (hand-crafted) feature set developed, i.e., lexical richness-based metrics, syntactic complexity-based measures, semantic density (named entity counts) and a wide variety of text readability indices.

This two-stage process ensured that:

- Hyperparameter tuning benefited from fast iterations using simpler TF-IDF features,
- The final deployment model leveraged linguistically meaningful and semantically rich handcrafted features, thereby achieving superior

classification performance and greater explainability.

The final XGBoost model was tested using the handcrafted feature-based test set, and serialized for deployment on real-time CEFR proficiency prediction and personalized vocabulary learning system.

2.1.7.5 Inference

Once the model was trained and its hyperparameters optimized, the last step was to deploy the trained XGBoost model for making predictions on new learner-produced texts. Inference is the process of asserting the trained machine learning model over unseen inputs to predict the corresponding outputs some CEFR level in the case of student submissions.

.

The inference process was organized into the following steps:

- **Model Loading**

The final XGBoost model, trained on handcrafted linguistic features and optimized hyperparameters, was loaded into memory using Joblib for efficient execution.

- **Input Text Processing**

The incoming learner inputs were preprocessed with the same standardized feature engineering pipeline as before. This was to enable the novel input data to be transformed as per the training phase into structured numerical feature vectors that capture information present in the training phase (lexical richness, syntactic complexity, semantic density, readability metrics).

- **Prediction Generation**

The structured input features were fed into the loaded XGBoost model to generate

probability distributions across the six CEFR proficiency levels (A1, A2, B1, B2, C1, C2).

- **Confidence-Based Decision Making**

A minimum confidence (e.g., 70%) was used to improve the reliability of the prediction. For such models with the top predicted probability less than the threshold, we employed a soft label averaging approach, where average of two highest classes was taken as the more resilience output.

- **Label Decoding**

The numeric output labels produced by the model were mapped back to their corresponding CEFR class names (e.g., $0 \rightarrow A1$, $2 \rightarrow B1$, etc.), providing human-readable predictions for integration into the vocabulary game personalization engine. Through this inference pipeline, the system was capable of performing real-time, scalable, and reliable CEFR-level predictions on student inputs, enabling dynamic adjustment of vocabulary learning paths based on immediate assessments of language proficiency.

In summary, the developed methodology effectively enabled the automatic identification of CEFR levels from learner input through structured feature extraction, optimized model training, and robust inference. This process ensures that personalized learning experiences can be accurately adapted to each student's real-time language proficiency.

2.1.8 Generate Personalize games content Using RAG System

This section described the method for creating personalized vocabulary learning games on the fly using a Retrieval-Augmented Generation (RAG) architecture. The system's main purpose has been to provide motivating educational materials appropriate for the learner's CEFR level of proficiency which relate to the learner's

topics of interest, be that, sports, technology, entertainment. Combining real-time semantic retrieval with state-of-the-art generative AI, the system guaranteed that the vocabulary games generated were not only linguistically accurate but that they felt personally relevant and motivating for each student.

▪ **Embedding Model**

At the architecture level, in the first step, the natural language input from the student is taken and it is converted into high-dimensional vectors using some sentence level transformer model, like sentence-transformers/all-MiniLM-L6-v2. This mapping, otherwise known as semantic embedding, encoded the contextual and linguistic semantics of the input in vector space. These vector representations maintained semantic similarity, so that words or phrases with similar meanings appeared nearer within the vector space. This lightweight but powerful transformer model selection contributed to high computational efficiency and stable semantic accuracy. These embeddings were then posed the semantic search query against a pre-indexed vocabulary db.

▪ **Vector Database (Pinecone)**

The second part was the vector database, and Pinecone was used at the heart of the lookup mechanism. Pinecone maintained a curated and pre-stored vocabulary data-set as a vectorized-entry collection that contained word definitions, examples of use, synonyms, idioms, and sample sentences. Given a query from the embedding model, Pinecone would perform an ANNs to retrieve and return the most semantically similar entries using cosine similarity measures. This fast and scalable search module makes it possible for the system to rapidly obtain precise and relevant vocabulary information that matched the learner's inquiry, even for large sets of vocabulary.

▪ **LangChain Framework**

LangChain served as the RAG architecture's orchestration layer. Its purpose was to provide a single location for submitting and orchestrating the retrieve-then-generate process. In particular, LangChain developed:

- The formulation and execution of semantic search queries to Pinecone,
- The structuring of the retrieved vocabulary context,
- The construction of prompts sent to the downstream generative model.

LangChain supported a modular and extensible architecture for plugging in different back-ends (retrievers, LLMS, and agents), which turned out to be scalable, maintainable, and extensible.

.

- **Large Language Model (LLM)**

The final core module was the Large Language Model (LLM) like Gen AI. Once LangChain had output the returned context, LLM used prompts to generate language learning material to provide to users. This content included

- Multiple-choice questions tailored to the retrieved vocabulary,
- Sentence completion exercises using contextually appropriate words,
- Word-definition matching tasks, and
- Explanations or rephrased examples for learner clarification.

With the generative ability of the LLM, adaptive difficulty (Seismic, adapted to the student's current knowledge), natural language feedback, and creative, pedagogically-aligned exercises were possible. It also supported that the learning experience promoted interactivity, engagement, and context-richness.

Combined, these modules constituted an end-to-end, smart, and adaptive pipeline for real-time educational content creation. The architecture was built up to be a modular

system, which means that every part of the system can exchangeable or extendable with a new part (embedding model or database) which not shutdown the whole Application. ItaloTRE allowed the system to adapt the students' linguistic profile, CEFR level and interest fields, thus generating a highly personal learning experience.

Personalized Game Generation Workflow

Personalized vocabulary game generation A well-defined multistep workflow of generating personalized vocabulary games was outlined. Every stage of this pipeline helped the generation of game that was matched to specific student's language proficiency and predispositions for learning, meaning that the educational material was both suitable for the level and interesting.

▪ Student Input

The first part of the workflow would start by the learner entering some form of textual input – it could be a word (e.g., “technology”), a subject (e.g., “sports”) or a free-text sentence. This input was then used for the tailored section. Before any content was created this system analyzed the learner's CEFR level [15] through the classifier we detail in Section 2.1.7. It was important to take this step since the exercise of all the following activities to create games depended on the matching of the content level sophistication with the level estimated for the learner.

▪ Keyword Extraction and Embedding

After determining the learner's CEFR proficiency level, the system then processed the input and the user's personal preferences. Rather than reanalysis for proficiency, it aimed at recovering significant keywords and key concepts by going through the noun chunking procedure. These terms were then passed through a pre-trained sentence-level embedding and transformed in their dense semantic representations. This embedding space representation facilitated fast and precise similarity-based searching of relevant vocabulary items and exercises so that all of the game content

was adapted in terms of the learner's own complexity level understanding and his/her own personal interests.

- **Semantic Retrieval (Pinecone and LangChain)**

Once the vectors had been extracted, the system conducted semantic retrieval using Pinecone, a high throughput vector database. For the input embeddings, Pinecone compared them with a pre-indexed vocabulary corpus and returned the most relevant word entries, definitions, idioms, and example usages. This retrieval was directed and shaped by LangChain, who directed the query process and formed an appropriately shaped context for the next phase. LangChain also had retrieval prompts to select the appropriate subset of vocabulary based on the learner interest and CEFR level.

- **Personalization Agent**

At this moment, a personalization agent stepped in to decide which way the learning task could be formulated most effectively. The agent considered several aspects: the learner's CEFR level, his/her past performance, the preferred topic or domain and the wanted format of the vocabulary game (quiz, sentence completion). Reflecting this evaluation, the system dynamically adapted the structure of the game, the difficulty of vocabulary, the number of choices and the complexity of the prompts. This decision-making layer guaranteed that no two students had exactly the same set of exercises, ensuring better student engagement and learning effectiveness.

- **Game Generation Using LLM**

After certain personalization parameters were chosen, the retrieved vocabulary and example usage were used as enriched context in a Large Language Model (LLM). The LLM was motivated to produce total games tasks according to person's profile. These tasks had diverse forms that included Word association games(MCQs) with distractors, Fill-in-the-blank games where focus is on word usage), and Chatbot for Word Definition, Synonym Replacement, Word- Definition Matching to reinforce memory and understanding. The LLM guaranteed natural language fluency, contextual

relevance and variety of design in questions.

Advantages of the RAG-Based Approach

Integrating a Retrieval-Augmented Generation (RAG) model into the personalized vocabulary learning architecture came with several technical and pedagogical benefits. By integrating real-time knowledge querying with generative capabilities, the system was able to balance precision with creativity and support of personalized learning. The following are the main advantages identified while applying and evaluating the methods.

▪ Content Quality

One of the best parts of using the RAG architecture is the quality of the generated content, which made it very contextually accurate. Unlike conventional text generation that depends on the pretrained model only, the RAG pipeline uses real-world definitions, idiomatic usages, and example sentences from a preindexed vector database. This allows the vocabulary games to be used as a true reflection of language use and subsequently, greater linguistic realism. Through anchoring the generative process in factual data, the system minimized hallucinations, which are fictitious or factually incorrect LLM-generated outputs and encouraged consistent, sound pedagogic content consistent with CEFR specifications.

▪ Scalability

The scalability problem was addressed by leveraging Pinecone (Pinecone), a high-performance vector database for real-time semantic search at scale. Pinecone successfully managed thousands of embeddings per word and still maintained the speed and accuracy of the retrieval. This enabled the system to accommodate large and diverse lexical resources on different CEFR levels and from different themes while maintaining the system performance. Performance and architecture was snappy for use by a small number of users or even for an institutional deployment. This meant that not only could the solution be used by individuals, it could also be used to inform broader language learning platforms, or educational institutions.

- **Transparency and Explainability**

Educational applications need to inspire trust and enhance understanding among learners, something that can be challenging when leveraging black-box AI. The RAG approach solved this problem by being clear about content generation. Debugged definitions, examples, and source context were shown next to the prompts produced for learners to retrace, diabolos their choices of vocabulary. This enhanced the interpretability of the system, helping students to not just get through the tasks but also understand the reason behind correct answers. This visibility served to enhance learning outcomes and to foster metalinguistic awareness in learners, who became more consciously aware of the patterns and rules governing their use of language.

In summary, the adaptation of the Retrieval-Augmented Generation (RAG) framework in the personalized vocabulary learning system supported the strength, scalability and learner-centered mechanisms. By integrating semantic vector search, real-time generative modelling, and adaptive feedback, the system could provide educational content that is not only linguistically relevant but also contextually pertinent. The architecture's capacity to integrate learner CEFR levels, learner's interests and past performance history into the game-generation process proved to be a powerful aid for facilitating differentiated learning and personalized learning paths.

The system generated a variety of interactive vocabulary learning tasks as a game, including:

- **Word-association matching Game** tailored to the retrieved vocabulary,
- **Sentence completion Games** using contextually appropriate words,
- **Word-definition matching Game,**

Additionally, a **personalized chatbot** feature was integrated into the system. This chatbot allowed learners to request **on-demand definitions, synonyms, example sentences, and clarifications** related to any vocabulary encountered during gameplay. This real-time, personalized interaction further reinforced vocabulary acquisition and

enabled learners to deepen their understanding in an engaging and autonomous manner.

Moreover, the use of advanced frameworks such as LangChain, Pinecone, transformer-based embedding model (sentence-transformers/all-MiniLM-L6-v2), and Large Language Models (LLMs) helped the system to maintain accuracy, speed, and pedagogical correlation for large scale, dynamic content-creation. The end result is a Living educational engine that can, in essence grow with the learner, becoming increasingly more personal, motivating, and reflective of the latest trends in intelligent language learning technologies.

2.2 Commercialization aspects of the product

This project adds a separate, AI-powered module to the “Readify” application—a smart digital learning platform made up of four main parts. We have implemented a particular module in this work focusing on personalized vocabulary learning, to improve the Readify system, and it supports CEFR-level prediction and RAG-based game content generation. It provides the basis for adaptable vocabulary learning and is designed to benefit learners, teachers, language centers and EdTech across the globe.

This vocabulary game generation component is appropriate to be integrated into regular schools, ESL systems and language teaching centers. It will also be a commercial product and available with the entire suite at Readify. The system is Dockerized and deployed to Azure, so the system can be deployed in a scalable, secure, and modular way for either institutions or individuals. In order to generate adoption, we will conduct pilot development at collaborator schools and language academies and participate in the presentation of educational tech and digital pedagogy talks and tables.

The module will exist in a freemium model. The free edition includes CEFR-level

detection and a sample of vocabulary games. The higher tier adds personalized communication, topical vocabulary domains (e.g., sports, technology), learning analytics, and teacher dashboards. A distinct API-as-a-Service (AaaS) spin will enable 3rd party e-Learning platforms, mobile apps and LMS though.

To broaden access, the vocabulary module will be bundled into a mobile-first app experience to serve learners beyond the school walls. Preliminary plans include working with publishers to integrate game content with textbooks and eCourses and working with international education organizations for curriculum alignment and global distribution. The aggregate data in the component may also be ethically used to further empirical adaptations and intelligent feedback to teachers.

This vocabulary personalization model makes Readify a product that can be suitable for scale and deliver impact with commercialization potential to economies of scale for both B2B & B2C educational segments, taking AI for Language learning to a widespread global scale.

3 TESTING AND IMPLEMENTATION

3.1 Implementation

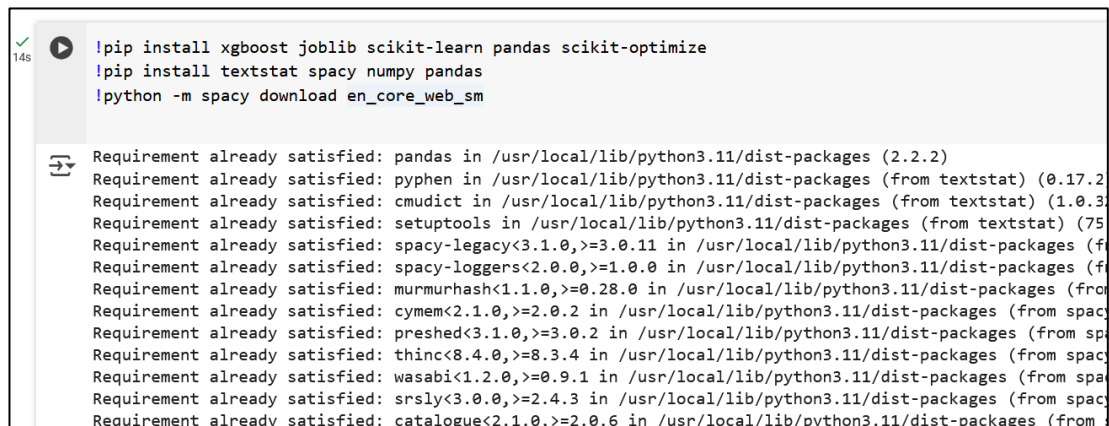
3.1.1 Implementation of CEFR Level Identification System

Design and implementation of the Vocabulary CEFR Level Identification approach

The processing pipeline for the implemented Vocabulary CEFR Level Identification system consists of multiple machine learning phases, which take as input raw student writing and produce predicted CEFR classification at the output. This section presents a step-by-step detailed implementation of the exchanges and the reasons leading to it. It combines NLP, feature engineering supervised learning and hyperparameter optimization to make the quickest and most efficient predictions.

3.1.1.1 Environment Setup

All the dependencies were installed via pip before starting implementation. The main libraries were spaCy for natural processing language, textstat for readability statistics, scikit-learn, xgboost, joblib and skopt of machine learning. Tokenization and POS tagging was done by downloading the language models from spaCy (en_core_web_sm).



```
!pip install xgboost joblib scikit-learn pandas scikit-optimize
!pip install textstat spacy numpy pandas
!python -m spacy download en_core_web_sm

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: pyphen in /usr/local/lib/python3.11/dist-packages (from textstat) (0.17.2)
Requirement already satisfied: cmudict in /usr/local/lib/python3.11/dist-packages (from textstat) (1.0.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from textstat) (75.0.0)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.11 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.11)
Requirement already satisfied: spacy-loggers<2.0.0,>=1.0.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (1.0.0)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.28.0)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.2)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.11/dist-packages (from spacy) (3.0.2)
Requirement already satisfied: thinc<8.4.0,>=8.3.4 in /usr/local/lib/python3.11/dist-packages (from spacy) (8.3.4)
Requirement already satisfied: wasabi<1.2.0,>=0.9.1 in /usr/local/lib/python3.11/dist-packages (from spacy) (0.9.1)
Requirement already satisfied: srsly<3.0.0,>=2.4.3 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.4.3)
Requirement already satisfied: catalogue<2.1.0,>=2.0.6 in /usr/local/lib/python3.11/dist-packages (from spacy) (2.0.6)
```

Figure 3.1 Dependencies installation

This environment was prepared using a on a Google Colab. Python 3.11 was selected for compatibility. Figure 3.1 show the pip installations and successful language model loading.

3.1.1.2 Data Preparation

Two labeled datasets, train.csv and test.csv, were implemented in this application. Both datasets included a column for raw input text along with the associated CEFR label (A1 through C2). Data was read into using pandas and label encoding done using LabelEncoder such that previously inappropriate categorical labels were transformed into integers for machine learning.

A screenshot of a code editor window with a light gray background. The code is written in Python and uses syntax highlighting. It imports pandas as pd and LabelEncoder from sklearn.preprocessing. A global variable label_encoder is set to None. A function load_data(filename) is defined to read a CSV file, extract text and labels, and return them as X and y. Another function encode_labels(labels) is defined to fit and transform the labels using the LabelEncoder. Finally, the functions are called to load and encode 'train.csv' and 'test.csv'. The editor shows a green checkmark and '1s' in the top left, and a green checkmark, '1s', and 'completed at 10:23 AM' in the bottom right.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

label_encoder = None
def load_data(filename):
    data = pd.read_csv(os.path.join(DATA_PATH, filename))
    X = data.text.tolist()
    y = encode_labels(data.label)
    return X, y

def encode_labels(labels):
    global label_encoder
    if label_encoder is None:
        label_encoder = LabelEncoder()
        label_encoder.fit(labels)
    return label_encoder.transform(labels)

X_train, y_train = load_data("train.csv")
X_test, y_test = load_data("test.csv")
```

Figure 3.2 Data Preparation

A Figure 3.2 was taken showing the output of data.head() and label_encoder.classes_ to illustrate successful loading and encoding. Figure 3.2 show the Data preparation from the csv file and encoding.

3.1.1.3 Text Preprocessing and Feature Engineering

A major part of the system was the function generate_features () which both model depended on for preprocessing. This function took a list of raw texts as input and output a DataFrame of extracted linguistic features.

As per our initial plan in Figure 3.3 we loaded the spaCy English language model en_core_web_sm with nlp = en_core_web_sm.load(). This model played a role in the processing pipeline for tokenization, POS tagging, and named entity recognition by which linguistic features were extracted from the input text.

```
✓ [5] nlp = en_core_web_sm.load()  
0s
```

Figure 3.3 NLP model loading

Step-by-step Processing:

- **Text Simplification:** Initially As shown in Figure 3.4, the raw input texts are cleaned initially so as to standardize the formatting, and to eliminate variation that can cause confusion to the tokenization. This was accomplished through the use of a custom function `simplify_punctuation(text)` where regular expressions were used to standardize punctuation marks, to consolidate repeated symbols, and to strip white space. Sample output of the that text simplification are given in Figure. 3.5

```
✓ 0s def _simplify_punctuation(text):  
    text = re.sub(r"[,:;()\-]", " ", text)  
    text = re.sub(r"[\.\!]", ".", text)  
    text = re.sub(r"^\s+", "", text)  
    text = re.sub(r"[ ]*(\n|\r\n|\r)[ ]*", " ", text)  
    text = re.sub(r"([\.\.])[\.\. ]+", ".", text)  
    text = re.sub(r"[ ]*([\.\.])", ". ", text)  
    text = re.sub(r"\s+", " ", text)  
    text = re.sub(r"\s+$", "", text)  
    return text
```

Figure 3.4 Text Simplification

```
⇒ sample input: Hello!! My name is... John: () I love coding.  
sample output: Hello. My name is. John I love coding.
```

Figure 3.5 Text Simplification Input and Output

- **POS Tagging:** Using spaCy, the part of speech of each term in the sentence was labeled. For each tag, we were given word counts, which we averaged over sentences. Figure 3.6 and 3.7 depicted the function for POS tag and the sample output of the after pos tagging respectively.

```

0s ▶ POS_TAGS = [
    "ADJ", "ADP", "ADV", "AUX", "CONJ", "CCONJ", "DET", "INTJ",
    "NOUN", "NUM", "PART", "PRON", "PROPN", "PUNCT", "SCONJ",
    "SYM", "VERB", "X", "SPACE",
]

def get_mean_pos_tags(text):
    """Calculate the mean for each type of POS tag in the text"""
    sentences = text.split(".")
    sentence_counts = _make_pos_tag_count_lists(sentences)
    num_sentences = textstat.sentence_count(text)
    mean_pos_tags = _calculate_mean_per_tag(sentence_counts, num_sentences)
    return mean_pos_tags

def _make_pos_tag_count_lists(sentences):
    """Process each sentence and collect POS tag counts per sentence"""
    sentence_counts = {}
    for doc in list(nlp.pipe(sentences)):
        pos_counts = _get_pos_tag_counts(doc)
        for key in pos_counts:
            if key in sentence_counts:
                sentence_counts[key].append(pos_counts[key])
            else:
                sentence_counts[key] = [pos_counts[key]]
    return sentence_counts

def _get_pos_tag_counts(doc):
    """Count POS tags in a spaCy Doc object"""
    pos_counts = {}
    pos_tags = [token.pos_ for token in doc]
    for tag in pos_tags:
        if tag in pos_counts:
            pos_counts[tag] += 1
        else:
            pos_counts[tag] = 1
    return pos_counts

def _calculate_mean_per_tag(counts, num_sentences):
    """Average POS tag counts per sentence"""
    mean_pos_tags = {f"mean_{tag.lower()}": 0 for tag in POS_TAGS}
    for key in counts:
        if len(counts[key]) < num_sentences:
            counts[key] += [0] * (num_sentences - len(counts[key]))
        mean_value = round(np.mean(counts[key]), 2)
        mean_pos_tags[f"mean_" + key.lower()] = mean_value
    return mean_pos_tags

```

Figure 3.6 POS Tagging Process

```

🔗 sample text input : Isaac Newton formulated the laws of motion in England during the 17th cen
Output of POS tagging :
    mean_adj mean_adp mean_adv mean_aux mean_conj mean_cconj mean_det \
0      1.5      2.0      0.5      0.5      0      0      2.0

    mean_intj mean_noun mean_num mean_part mean_pron mean_propn \
0      0      2.5      0      0      0.5      1.5

    mean_punct mean_sconj mean_sym mean_verb mean_x mean_space
0      0      0      0      0.5      0      0.5

```

Figure 3.7 POS Tagging Sample Input and Output

- **Named Entity Recognition:** The average NERs per sentence were used as a measure of semantic density. Figure 3.8 and 3.9 displayed functions for name

entity recognition and an example of its output after process.

```
0s ✓ ▶ def get_total_ents(text):  
    return len(nlp(text).doc.ents)  
  
def get_mean_ents_per_sentence(text):  
    return get_total_ents(text) / textstat.sentence_count(text)
```

Figure 3.8 Named Entity Recognition Process

```
↗ sample input text: Albert Einstein was born in Germany in 1879 and later moved to the US.  
Mean Entities per Sentence: 4.0
```

Figure 3.9 Named Entity Recognition Sample Input and Output

- **Syntactic Depth:** The depth of the parse tree was calculated by a recursive function applied to spaCy's dependency parse tree. This served as a proxy of grammatical complexity. Figures 3.10 and 3.11 displayed functions consisting of syntactic recognition and sample output of the after process respectively.

```
0s ✓ ▶ def get_mean_parse_tree_depth(text):  
    """Calculate the average depth of parse trees in the text"""  
    sentences = text.split(".")  
    depths = []  
    for doc in list(nlp.pipe(sentences)):  
        depths += _get_parse_tree_depths(doc)  
    return np.mean(depths)  
  
def _get_parse_tree_depths(doc):  
    """Get parse tree depth for each token in the doc"""  
    return [_get_depth(token) for token in doc]  
  
def _get_depth(token, depth=0):  
    """Recursively calculate the depth of a token in the dependency tree"""  
    depths = [_get_depth(child, depth + 1) for child in token.children]  
    return max(depths) if len(depths) > 0 else depth
```

Figure 3.10 Syntactic depth process

```
↗ sample input text: Albert Einstein was born in Germany in 1879 and later moved to the US.  
get mean parse tree depth: 0.9285714285714286
```

Figure 3.11 Syntactic depth process Sample Input and Output

- **Readability Metrics:** The textstat library was employed to generate the standard scores such as Flesch reading ease, Gunning fog, SMOG, and Linear write formula. Figure 3 12 and 3 13 are the functions for readability metrics recognition and an example of the after process respectively.

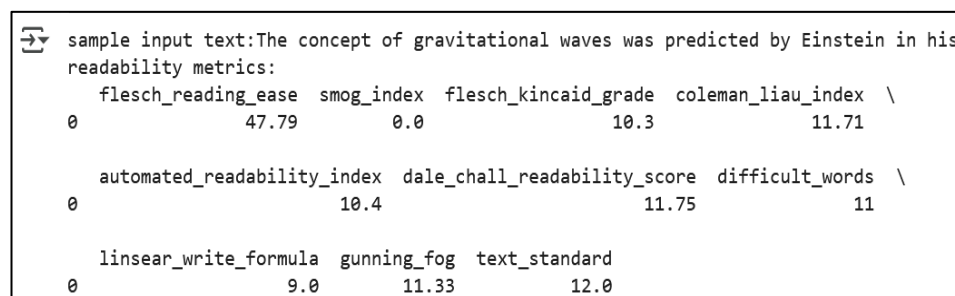


```

0s readability_metrics = {
    "flesch_reading_ease": textstat.flesch_reading_ease(sample_text),
    "smog_index": textstat.smog_index(sample_text),
    "flesch_kincaid_grade": textstat.flesch_kincaid_grade(sample_text),
    "coleman_liau_index": textstat.coleman_liau_index(sample_text),
    "automated_readability_index": textstat.automated_readability_index(sample_text),
    "dale_chall_readability_score": textstat.dale_chall_readability_score(sample_text),
    "difficult_words": textstat.difficult_words(sample_text),
    "linsear_write_formula": textstat.linsear_write_formula(sample_text),
    "gunning_fog": textstat.gunning_fog(sample_text),
    "text_standard": textstat.text_standard(sample_text, float_output=True),
}

```

Figure 3.12 Readability Metrics



```

sample input text:The concept of gravitational waves was predicted by Einstein in his
readability metrics:
    flesch_reading_ease  smog_index  flesch_kincaid_grade  coleman_liau_index \
0                        47.79        0.0                10.3                11.71

    automated_readability_index  dale_chall_readability_score  difficult_words \
0                                10.4                        11.75                11

    linsear_write_formula  gunning_fog  text_standard
0                          9.0         11.33         12.0

```

Figure 3.13 Readability Metrics Sample Input and Output

Finally, these individual feature extraction steps were encapsulated modularly into single unify function, generate features (), which are shown in Figuree 3.14. This pipeline handled the execution of all pre-preprocessing steps (text cleaning, POS tagging, named entity recognition, syntactic depth, readability) 2 over a batch of input texts. As seen in Fig. 3.15, It produced a well-structured panda. DataFrame with rows corresponding to the full feature vector for a single learner's input.

```

def generate_features(data):
    feature_data = []
    for text in data:
        features = preprocess_text(text)
        feature_data.append(features)
    return pd.DataFrame(feature_data)

def preprocess_text(text):
    text = _simplify_punctuation(text)
    features = {
        "flesch_reading_ease": textstat.flesch_reading_ease(text),
        "smog_index": textstat.smog_index(text),
        "flesch_kincaid_grade": textstat.flesch_kincaid_grade(text),
        "coleman_liau_index": textstat.coleman_liau_index(text),
        "automated_readability_index": textstat.automated_readability_index(text),
        "dale_chall_readability_score": textstat.dale_chall_readability_score(text),
        "difficult_words": textstat.difficult_words(text),
        "linsear_write_formula": textstat.linsear_write_formula(text),
        "gunning_fog": textstat.gunning_fog(text),
        "text_standard": textstat.text_standard(text, float_output=True),
        "mean_parse_tree_depth": get_mean_parse_tree_depth(text),
        "mean_ents_per_sentence": get_mean_ents_per_sentence(text),
    }
    features.update(get_mean_pos_tags(text))
    return features

```

Figure 3.14 generate feature function

sample input text: the concept of gravitational waves was predicted by Einstein in his general theory of relativity. these ripples in the fabric of spacetime were first directly observed by LIGO in 2015.

	flesch_reading_ease	smog_index	flesch_kincaid_grade	coleman_liu_index	automated_readability_index	dale_chall_readability_score	difficult_words	linsear_write_formula	gunning_fog	text_standard	...	mean_noun	mean_part	mean_pron	mean_punct	mean_adj	mean_verb	mean_x	mean_space		
0	47.79	0.0	10.3	11.71	10.4	11.75	11	9.0	11.33	12.0	...	0.5	0	0.5	1.0	0	0	0	1.0	0	0.5

1 rows x 21 columns

Figure 3.15 generate feature function sample input and output

This architecture also guaranteed uniformity and reusability throughout the machine learning pipeline for training, validation, and real-time inference. The salient features made it possible for the downstream XGBoost learner to derive the effective discriminative patterns from the wide variety of linguistic patterns. In addition using a Function Transformer to wrap all the steps into `generate_features()` meant that this function could be dropped into a scikit-learn pipeline, so that everything from raw input to CEFR prediction could be modelled efficiently.

3.1.1.4 Model Training

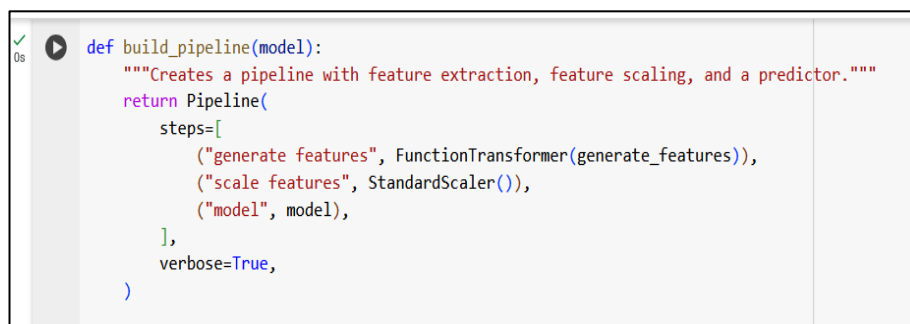
Once linguistic features were extracted (using the `generate_features()` function

described in Section 3.1.1.3), the next step in the pipeline was to train machine learning classifiers that could predict the CEFR level of student input.

For consistency and reuse, a machine learning pipeline was built using the scikit-learn library and Pipeline () method. This pipeline neatly and intuitively organized all feature generation, scaling, and model fitting.

▪ Pipeline Construction

Build a reusable, end-to-end pipeline that can be trained and evaluated on any dataset with text and CEFR labels. That Pipeline code snippets shown in figure 3.16.

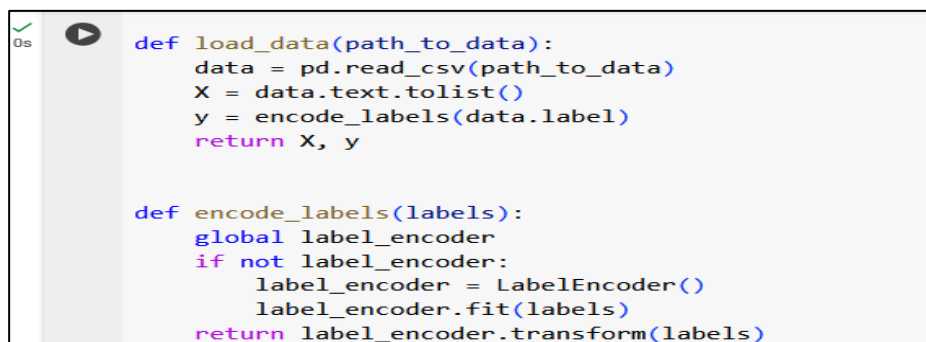


```
def build_pipeline(model):  
    """Creates a pipeline with feature extraction, feature scaling, and a predictor."""  
    return Pipeline(  
        steps=[  
            ("generate features", FunctionTransformer(generate_features)),  
            ("scale features", StandardScaler()),  
            ("model", model),  
        ],  
        verbose=True,  
    )
```

Figure 3.16 Reusable Pipeline for model Training

▪ Data Loading and Label Encoding

The train and test data were read from the train. csv and test. csv. The CEFR A1–C2 level tags are encoded by the function LabelEncoder(). That encoding process is presented in Figure 3.18.




```
def load_data(path_to_data):  
    data = pd.read_csv(path_to_data)  
    X = data.text.tolist()  
    y = encode_labels(data.label)  
    return X, y  
  
def encode_labels(labels):  
    global label_encoder  
    if not label_encoder:  
        label_encoder = LabelEncoder()  
        label_encoder.fit(labels)  
    return label_encoder.transform(labels)
```

Figure 3.17 Data Load and Label Encode

▪ Model Training & Evaluation

According to Figure 3.18 and 3.19, The pipeline was trained using `.fit()` and evaluated using `.score()` on the test set. According to Figure 3.20, Accuracy was printed.



```
def train(model):
    print(f"Training {model['name']}")
    pipeline = build_pipeline(model["model"])
    pipeline.fit(X_train, y_train)
    print(pipeline.score(X_test, y_test))
    save_model(pipeline, model["name"])

models = [
    {
        "name": "XGBoost",
        "model": XGBClassifier(
            objective="multi:softprob",
            random_state=RANDOM_SEED,
            use_label_encoder=False,
        ),
    }
]
```

Figure 3.18 Model Training

```

if __name__ == "__main__":
    # Load the data and model
    X, y_true = get_data() # get_data() should return features X and true labels y_true
    model = load(os.path.join(MODEL_PATH, "xgboost.joblib")) # Load the pre-trained model

    # Predict on the data
    y_pred = model.predict(X)
    y_pred_proba = model.predict_proba(X)

    # Print performance metrics
    print(get_confusion_matrix(y_true, y_pred)) # Confusion Matrix
    print(classification_report(y_true, y_pred, target_names=LABELS)) # Classification Report

    print(get_top_k_accuracy(model, X, y_true, k=2)) # Top K Accuracy

    test_loss = log_loss(y_true, y_pred_proba)

    print(f"Test Loss (Log Loss): {test_loss}")

    # Plot Confusion Matrix
    plot_confusion_matrix(y_true, y_pred) # Assuming you have a custom plot_confusion_matrix function

    # Precision-Recall Curve
    precision, recall, _ = precision_recall_curve(y_true, model.predict_proba(X)[:, 1]) # Assuming binary classification

    plt.figure(figsize=(8, 6))
    plt.plot(recall, precision, color='b', lw=2)
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve')
    plt.show()

```

✓ 0s completed at 10:14 AM

Figure 3.19 Main Function of model training

	label	A1	A2	B1	B2	C1	C2	
0	A1	47	10	1	0	0	0	
1	A2	15	37	3	0	0	0	
2	B1	0	4	27	10	0	0	
3	B2	0	2	13	25	17	0	
4	C1	0	0	0	9	30	9	
5	C2	0	0	0	1	10	29	
		precision				recall	f1-score	support
	A1				0.76	0.81	0.78	58
	A2				0.70	0.67	0.69	55
	B1				0.61	0.66	0.64	41
	B2				0.56	0.44	0.49	57
	C1				0.53	0.62	0.57	48
	C2				0.76	0.72	0.74	40
	accuracy						0.65	299
	macro avg				0.65	0.66	0.65	299
	weighted avg				0.65	0.65	0.65	299

Figure 3.20 Trained Model Accuracy and Result

3.1.1.5 Hyperparameter Optimization

In order to make the XGBoost model more optimal, we conducted hyperparameter tuning through the BayesSearchCV function in skopt. This method is well known for employing sample-efficient Bayesian optimization techniques that systematically search for optimal parameter combinations by modeling the objective function.

- **TF-IDF Vectorization for Fast Tuning:** To speed up the computational pipeline, the grid search did not consider hand-crafted features, but used TF-IDF vectorization. It shown in Figure 3.21.

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_data.text)
y_train = label_encoder.fit_transform(train_data.label)
```

Figure 3.21 Load TF-IDEP and Transformed to Vectors

- **Search Space Definition:** A parameter space was generated for XGBClassifier with (learning rate, max depth, n estimators, gamma). One such optimization process is illustrated by this Figure 3.22.

```
[ ]
from xgboost import XGBClassifier
from skopt import BayesSearchCV

param_space = {
    "learning_rate": (0.01, 0.1),
    "max_depth": (3, 10),
    "n_estimators": (50, 500),
    "gamma": (0, 5),
}

opt = BayesSearchCV(
    XGBClassifier(objective="multi:softprob", use_label_encoder=False),
    param_space,
    cv=5,
    n_iter=32,
    random_state=0
)
```

Figure 3.22 Hyperparameter Optimization Using BayesSearchCV

- **Search Execution and Evaluation:** The search was performed on the training dataset and cross-validated. Print the best score and parameters. That validation is shown in this Figure 3.23.

```
opt.fit(X_train, y_train)
print(f"Best validation accuracy: {opt.best_score_:.4f}")
print("Best parameters:")
for param, value in opt.best_params_.items():
    print(f" - {param}: {value}")
```

Figure 3.23 Cross Validation

This Figure 3-24 illustrates the BayesSearchCV output which specifies the optimal parameter combination after hyperparameter tuning. The best hyperparameters set is $\gamma = 0$, $\text{learning_rate} \approx 0.0484$, $\text{max_depth} = 3$, $\text{n_estimators} = 500$ with a validation accuracy of 67.07%.

```
Best parameters:
- gamma: 0
- learning_rate: 0.048434229652418226
- max_depth: 3
- n_estimators: 500
```

✓ 4h 47m 5s completed at 2:41 AM

Figure 3.24 best-found parameter after extensive tuning

- **Model Saving:** The optimized model was saved for use in retraining with handcrafted features. This Figure 3.25 displays model saving as a pickle type.

```
import pickle
with open("models/cefr-xgboost.pickle", "wb") as f:
    pickle.dump(opt.best_estimator_, f)
```

Figure 3.25 Model saved as a pickle

Bayesian search for hyperparameter optimization has been proved to be useful in improving predictive performance through discovering the optimal settings with less number of trials. Although TF-IDF was used in tuning for speed, the

optimal parameters were copied to a final model which was retrained on hand-built features for the purpose of deployment.



```
def train(model):
    print(f"Training {model['name']}")
    pipeline = build_pipeline(model["model"])
    pipeline.fit(X_train, y_train)
    print(pipeline.score(X_test, y_test))
    save_model(pipeline, model["name"])

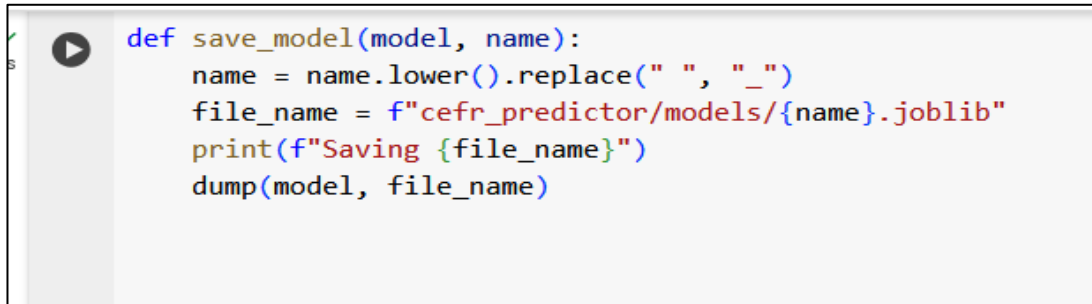
models = [
    {
        "name": "XGBoost",
        "model": XGBClassifier(
            objective="multi:softprob",
            random_state=RANDOM_SEED,
            use_label_encoder=False,
            gamma=0,
            learning_rate=0.0484,
            max_depth=3,
            n_estimators=500,
        ),
    }
]
```

Figure 3.26 model retrained with handcrafted features

Once we found the best performing hyperparameters with Bayesian optimization (see Section 3.1.1.4), we retrained the XGBoost classifier using the handcrafted linguistic features instead of the TF-IDF vectors. You can view this in Figure 3.26. The objective was to exploit the better model performance of the tuned parameters overall, at the same time ensuring general intuition with the final fitted model in conjunction with the feature engineering pipeline that was derived in Section 3.1.1.4. The optimized values of `gamma`, `learning_rate`, `max_depth` and `n_estimators` were as directly input to the `XGBClassifier()` in the initialization that was used in the configuration block above.

3.1.1.6 Model Serialization

The best model was saved with joblib after it was found. dump() in a directory you create with the name cefr_predictor/models. The filename took the {model_name} format. joblib. This serialized file could be read later using joblib. load() for on-the-fly inference through an application pipeline.



```
def save_model(model, name):  
    name = name.lower().replace(" ", "_")  
    file_name = f"cefr_predictor/models/{name}.joblib"  
    print(f"Saving {file_name}")  
    dump(model, file_name)
```

Figure 3.27 Final model Saving path and Type

A Figure 3.27 displayed the file path and print statement confirming successful saving.

3.1.1.7 Inference and Deployment

Once After the final model was trained and serialized, I built a separate end-to-end inference module for performing live CEFR level prediction on raw text input. This feature is important to incorporate the model into downstream use-cases, such as custom game generation or feedback delivery for educational applications.

- A Model class was created to encapsulate loading and interacting with the trained XGBoost pipeline.
- The model was loaded using joblib.load() from the file path cefr_predictor/models/xgboost.joblib.
- The prediction process includes two stages:
 1. **Probability Prediction:** Using predict_proba() to obtain CEFR probabilities.
 2. **Label Decoding:** The most probable label is selected and decoded back

to its CEFR string label (e.g., "B1", "C2").

```
6 K = 2
7 LABELS = {
8     0.0: "A1",
9     0.5: "A1+",
10    1.0: "A2",
11    1.5: "A2+",
12    2.0: "B1",
13    2.5: "B1+",
14    3.0: "B2",
15    3.5: "B2+",
16    4.0: "C1",
17    4.5: "C1+",
18    5.0: "C2",
19    5.5: "C2+",
20 }
21
22
23 class Model:
24     def __init__(self, model_path):
25         self.model = load(model_path)
26
27     def predict(self, data):
28         probas = self.model.predict_proba(data)
29         preds = [self._get_pred(p) for p in probas]
30         probas = [self._label_probabilities(p) for p in probas]
31         return preds, probas
32
33     def predict_decode(self, data):
34         preds, probas = self.predict(data)
35         preds = [self.decode_label(p) for p in preds]
36         return preds, probas
37
38     def _get_pred(self, probabilities):
39         if probabilities.max() < MIN_CONFIDENCE:
40             return np.mean(probabilities.argsort()[-K:])
41         else:
42             return probabilities.argmax()
43
44     def decode_label(self, encoded_label):
45         return LABELS[encoded_label]
46
47     def _label_probabilities(self, probas):
48         labels = ["A1", "A2", "B1", "B2", "C1", "C2"]
49         return {label: float(proba) for label, proba in zip(labels, probas)}
```

Figure 3.28 Inference Implementation

The inference engine we have shown (in Fig. 3.28) in this section supports such an use (i.e. to support CEFR-level profiling in any environment). It allows for bulk predictions, confidence-based ranking, as well as arbitrary thresholding to suit more educational or adaptive learning scenarios.

▪ API Integration with Fast API

A RESTful API was constructed with the Fast API framework to make the trained model a web service. This enables third-party applications to submit texts and get a level returned. You can see that in Figure 3.29.

Key Features:

- Accepts POST requests with a list of texts.
- Returns predicted CEFR levels and their probability scores.
- Includes CORS middleware to support cross-origin requests.

```
C: > Users > malin > OneDrive > Desktop > project_root > api.py

1  from fastapi import FastAPI
2  from pydantic import BaseModel
3  from typing import List
4  from fastapi.middleware.cors import CORSMiddleware
5  from inference import Model
6
7  app = FastAPI()
8
9  app.add_middleware(
10     CORSMiddleware,
11     allow_origins=["*"], # Allow all domains (change to specific domains for security)
12     allow_credentials=True,
13     allow_methods=["*"], # Allow all HTTP methods (GET, POST, etc.)
14     allow_headers=["*"], # Allow all headers
15 )
16
17 model = Model("cefr_predictor/models/xgboost.joblib")
18
19 class TextList(BaseModel):
20     texts: List[str] = []
21
22 @app.post("/predict")
23 def predict(textlist: TextList):
24     preds, probas = model.predict_decode(textlist.texts)
25
26     response = []
27     for text, pred, proba in zip(textlist.texts, preds, probas):
28         row = {"text": text, "level": pred, "scores": proba}
29         response.append(row)
30
31     return response
32
```

Figure 3.29 API Integration with saved model

- **Sample API Usage via Swagger:**
 - **Method:** POST
 - **URL:** http://localhost:8000/predict
 - **Headers:** Content-Type: application/json
 - **Body (raw JSON)**

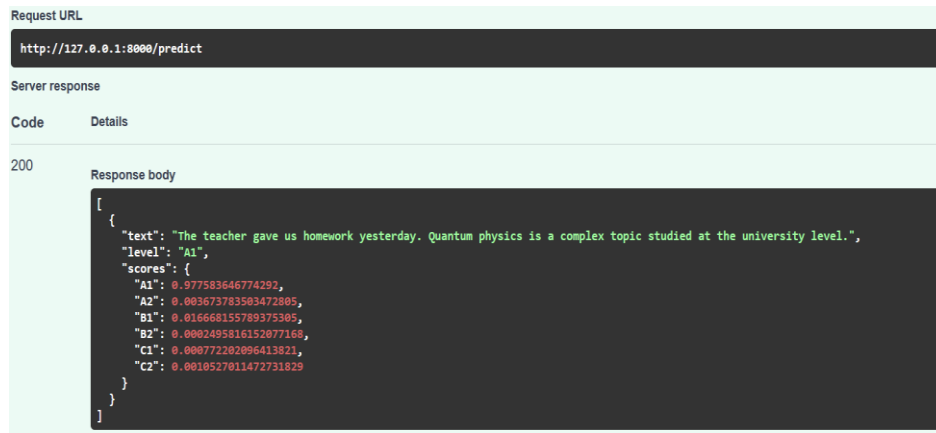


Figure 3.30 Sample Response

The developed API was tested with Swagger where the students text were sent as POST requests and classified as Vocabulary CEFR level. For each API response the most probable CEFR level and the confidence scores were provided. This end-to-end prediction framework can be integrated into educational software, allowing learners to automatically receive immediate feedback and personalized instruction according to their current proficiency level.

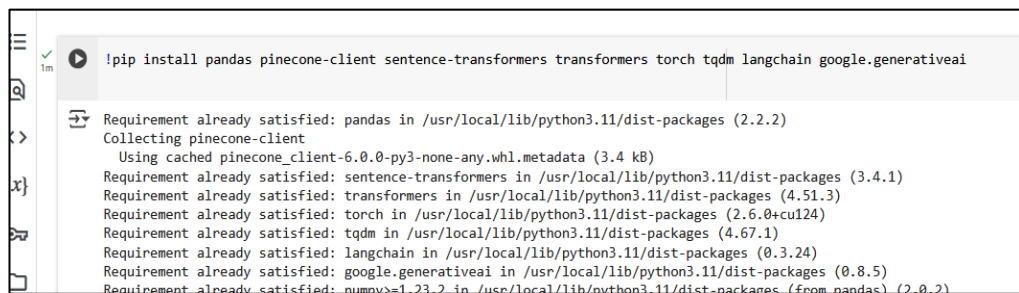
3.1.2 Implementation of Personalized Vocabulary Game Generation Using RAG

The personalized vocabulary game generation system was implemented through a Retrieval-Augmented Generation (RAG) architecture, which is presented in this Section. The system integrates tools of semantic retrieval and large language models to generate educational content on-the-fly, chosen at a student's CEFR level in vocabulary, and tailored to individual preferences

▪ Environment Setup and Required Libraries

The first part was to install all libraries and frameworks necessary for the development environment. As RAG is using LLMs, vector stores, and orchestration tools the following components were installed. which is shown in Figure. 3.31,

- Lang chain for RAG orchestration
- google. generative Ai for Gemini model access
- pinecone-client for vector storage
- sentence-transformers for embeddings
- Fast API for API endpoints
- Unicorn as the ASGI server



```
!pip install pandas pinecone-client sentence-transformers transformers torch tqdm langchain google.generativeai

Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Collecting pinecone-client
  Using cached pinecone_client-6.0.0-py3-none-any.whl.metadata (3.4 kB)
Requirement already satisfied: sentence-transformers in /usr/local/lib/python3.11/dist-packages (3.4.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.6.0+cu124)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (4.67.1)
Requirement already satisfied: langchain in /usr/local/lib/python3.11/dist-packages (0.3.24)
Requirement already satisfied: google.generativeai in /usr/local/lib/python3.11/dist-packages (0.8.5)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.0.2)
```

Figure 3.31 Libraries Installation

▪ Loading the Embedding Model and Create Pinecone Index

For semantic comprehension in the RAG system, load the all-MiniLM-L6-v2 model from the sentence-transformers library. This is shown in Figure 3.32. This model is popular for its compact and efficient embeddings with strong semantic representation. It was applied to transform both vocabulary entries and input topics into 768-dimensional vector representations of their meanings in high-dimensional space.

These embeddings are used for indexing (put vocabulary into Pinecone) and querying (get result words for student input). This was supported by setting up a Pinecone index with the right parameters. The index was created using the following code:

```

▶ pc = Pinecone(api_key=api_key)
  index_name = "quickstartnew"

  pc.create_index(
      name=index_name,
      dimension=768,
      metric="euclidean",
      spec=ServerlessSpec(
          cloud="aws",
          region="us-east-1"
      )
  )

  index = pc.Index(index_name)

  Emodel = SentenceTransformer('all-MiniLM-L6-v2')

```

Figure 3.32 Pinecone Vector DB Creation and Embedding Model Loading

This setup ensures that all embeddings generated by the Sentence-Transformer model are stored in a high-speed vector database and can be efficiently retrieved via Approximate Nearest Neighbor (ANN) search. The combination of model loading and index creation establishes the foundation for the system's semantic search capabilities

▪ Indexing Vocabulary Content in Pinecone

In order to be able to retrieve the vocabulary items quickly and in context using their semantic content, the system incorporated Pinecone as a vector database for a dense embedding storage. A function, called `generate_embeddings_and_upsert` (Figure 3.33), was used to process each vocabulary entry from the dataset. This function first tested for null values and implemented fallbacks—with the word itself as a default, or basic CEFR levels (e.g., A1) in cases where the fields should not be left blank. It then concatenated the word, its definition, example sentence, CEFR level, word type (noun, verb etc.), synonyms, antonyms, rhyming word and related words in a single text, separated by space. With the means of a Sentence-Transformer model like `all-MiniLM-L6-v2`, this context was encoded into a semantic vector that represented the semantics of the word in its own educational context numerically. This embedding was zipped with the metadata and binned in batches. For scalability, the vectors are

ingested in bulk (batch size of 100) into a pre-configured Pinecone index using the `upsert()` method. Each embedded item was indexed using an identifier and its associated metadata, thereby facilitating both rapid similarity searches and filtered searches based on CEFR level or word category. This form of indexing guaranteed that when using Lang Chain to retrieve later with it would produce richer vocabulary terms with context relevant to the student preferences and age, it was the initial implementation of the RAG-based process of generating personalized games pipeline. Figure 3.34 illustrates vector db in which the data were saved.

```
[ ] def generate_embeddings_and_upsert(df, batch_size=100):
    vectors = []

    for i, row in tqdm(df.iterrows(), total=len(df), desc="Processing dataset"):
        word = row['word']
        definition = row['definition'] if not pd.isna(row['definition']) else word # Default to word if NaN
        example = row['example'] if not pd.isna(row['example']) else word # Default to word if NaN
        ceفر = row['ceفر'] if not pd.isna(row['ceفر']) else 'A1' # Default to 'A1' if NaN
        typee = row['type'] if not pd.isna(row['type']) else 'Noun' # Default to 'Noun' if NaN
        synonyms = row['Synonyms'] if not pd.isna(row['Synonyms']) else word # Default to word if NaN
        nAntonyms = row['Antonyms'] if not pd.isna(row['Antonyms']) else word # Default to word if NaN
        rhymes = row['Rhymes'] if not pd.isna(row['Rhymes']) else word # Default to word if NaN
        related_words = row['Related Words'] if not pd.isna(row['Related Words']) else word # Default to word if NaN
        # Combine definition and example to form a context

        text = f"Word: {word}\nDefinition: {definition}\nExample: {example}\nceفر: {ceفر}\ntype: {typee}"

        # Get embedding for the combined text
        embedding = Emodel.encode(text)

        # Prepare metadata for each entry
        metadata = {"word": word, "definition": definition, "example": example, "ceفر": ceفر,
                    "type": typee, "synonyms": synonyms, "antonyms": nAntonyms, "rhymes": rhymes, "related Words": related_words}

        # Append the word, embedding, and metadata to vectors list
        vectors.append((str(i), embedding.tolist(), metadata))

        # Upsert to Pinecone in batches
        if len(vectors) >= batch_size:
            index.upsert(vectors)
            vectors = [] # Reset batch

    if vectors:
        index.upsert(vectors)

[ ] generate_embeddings_and_upsert(df)

Processing dataset: 100% |██████████| 5902/5902 [11:21<00:00, 8.67it/s]
```

Figure 3.33 Embeddings generation function

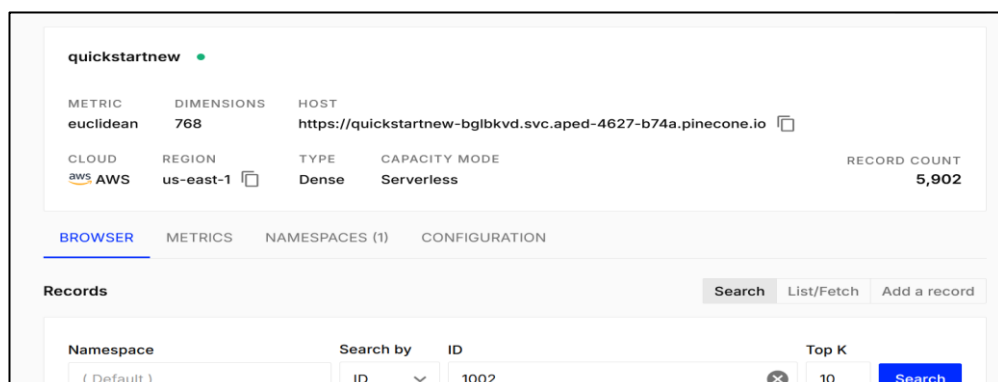


Figure 3.34 vector DB Screen shot

- **Accurate Retrieval from Vector Database Using LangChain**

LangChain was applied for implementing a retriever that attached to the Pinecone vector store and allowed for fast semantic search. A reusable function was developed to achieve this retrieval rather than simply invoking the static queries. Function is represented by figure.3.35 to generate embeddings.

```

1  from langchain.embeddings import HuggingFaceEmbeddings
2  from langchain.vectorstores import Pinecone
3  from app.config import Index
4
5  # Setup embeddings and vector store globally
6  embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
7  vectorstore = Pinecone(Index, embedding_model, "word")
8
9  # Reusable function to search by query(Student Preferences) and CEFR level
10 def search_vocabulary(query: str, level: str = None, k: int = 1):
11     """
12     Search for vocabulary terms in the vectorstore by query and level.
13
14     Parameters:
15         query (str): The search string (e.g., "firewall" or "technology").
16         level (str): CEFR level to filter (e.g., "A1", "B1").
17         k (int): Number of top documents to return.
18
19     Returns:
20         List[Document]: Matching LangChain Document objects with metadata.
21     """
22     filter_dict = {"level": level} if level else None
23
24     results = vectorstore.similarity_search(
25         query=query,
26         k=k,
27         filter=filter_dict
28     )
29
30     return results
31
32 results = search_vocabulary(query="cybersecurity", level="B1", k=1)
33
34 for doc in results:
35     print(doc.metadata["word"])
36     print(doc.metadata["definition"])
37     print(doc.metadata["example"])
38 
```

Figure 3.35 function for Retrieval from Vector Database

- **Prompt Engineering and Personalization Agent**

The dynamically constructed personalized game prompts were created utilizing the CEFR level of the student and the student's preferences. The structured prompts prepared by the Fast API endpoints were submitted to the Gemini API to process prompt templates That are illustrated in – varied for hint generation, MCQs, and fill-in-the-blanks.

```

21 PromptTemplate(
22     input_variables=["word", "definition", "example"],
23     template="""
24     You are an intelligent assistant that generates short and concise hints to help students understand and guess words.
25
26     Your task is to create a hint based on the given word, its definition, and an example sentence. Do not include the word itself in the hint.
27
28     Word: {word}
29     Definition: {definition}
30     Example: {example}
31     """
32 )

```

Figure 3.36 Prompt for Short Hint Generation


```

37 PromptTemplate(
38     input_variables=["word", "definition", "example", "level", "preferences"],
39     template="""
40     You are an English vocabulary assistant. Explain the word '{word}' to a student at level {level} who is interested in {preferences}.
41
42     Definition: {definition}
43     Example: {example}
44
45     Provide a simple, clear, and personalized explanation.
46     """)
47

```

Figure 3.37 Prompt for Word Explanation used in chatbot

```

75 PromptTemplate(
76     input_variables=["word", "definition", "example", "level", "preferences"],
77     template="""
78     You are an English vocabulary expert helping design a word association game for a student at CEFR level {level} who is interested in {preferences}.
79
80     1. Provide three related words (including '{word}').
81     2. Give four answer options.
82     3. Indicate which option is correct.
83     4. Explain why that option fits best in a context aligned with the student's interests.
84
85     Word: {word}
86     Definition: {definition}
87     Example: {example}
88
89     Return JSON:
90     {
91         "question": "...",
92         "option1": "...",
93         "option2": "...",
94         "option3": "...",
95         "option4": "...",
96         "correct_answer": "...",
97         "explanation": "..."
98     }
99     """
100 )

```

Figure 3.38 Prompt for word Association Game

```

108 PromptTemplate(
109     input_variables=["word", "definition", "example", "level", "preferences"],
110     template="""
111     You are an English tutor. Generate a fill-in-the-blank question for level {level} with a preference in {preferences}.
112
113     Word: {word}
114     Definition: {definition}
115     Example: {example}
116
117     Return JSON:
118     {
119         "question": "... [____] ...",
120         "option1": "...",
121         "option2": "...",
122         "option3": "...",
123         "option4": "...",
124         "correct_answer": "...",
125         "explanation": "..."
126     }
127     """
128 )
129

```

Figure 3.39 Prompt for word Fill-in-the-Blank game

▪ Game Generation Using Gemini LLM

The Gemini LLM is applied through LangChain's LLMChain to produce individually tailored vocabulary game content. The prompt contains relevant input fields from Pinecone Vector database (word, definition, and example) and student metadata (level and preferences). Illustration 3.40 displayed source code of llm chain

```
34 llm_chain = LLMChain(llm=llm, prompt=prompt)
35
36 # Run the chain using data from Pinecone retrieval
37 response = llm_chain.run({
38     "word": Word,
39     "definition": Definition,
40     "example": Example,
41     "level": Level,
42     "preferences": Preferences
43 })
```

Figure 3.40 LLM chain

This personalized vocabulary game system based on RAG clearly shows that the combination of retrieval based and generative language models can bring about very natural and adaptive educational systems. RAG played a crucial role in getting semantically meaningful and context-rich word information from Pinecone, which in turn aided the grounding of text generation in Gemini.

Key outputs included:

- Custom MCQs, hints, and fill-in-the-blank exercises
- Contextualized explanations aligned with Predicted Student Vocabulary CEFR levels
- Word association challenges
- Dynamic chatbot explanations based on word metadata

Additionally, we included a chatbot feature users can ask any vocabulary term and get a level-appropriate explanation, related to their interest area (like technology or pop culture). This leads to a rich and learner-centered experience which adapts itself based upon user activity.

3.1.3 Front-End Implementation

Front-end development This part covers the front-end of a personal vocabulary learning web app that is built with React.js and Tailwind CSS. The app provides a game-based quiz engine, CEFR level guess, vocabulary bot and user profile preference insertion. In combination, these features allow for individualized learning experiences based on the student's level and interests.

▪ Front-End Project Initialization

Frontend development was started with bootstrapping the project in Vite, a modern build tool, featuring super-fast cold start and optimized bundling. Vite was selected over more established tools like Create React App (CRA) but for stronger performance and developer ergonomics. We incorporated Tailwind CSS early in the process in order to have utility-first approach and it made responsive design very fast by just writing class-based styles in JS files. This way I got a mobile responsive UI with just a few custom CSS.

Key dependencies installed during initialization:

- react and react-dom: for building UI components
- vite: for fast local development and HMR (hot module replacement)
- tailwindcss: for styling
- axios: for API communication with the backend

▪ Vocabulary CEFR Level Prediction according to student input text and Input Flow

To personalize content, the app first predicts the student's vocabulary level using input text.

Student Flow:

1. Enter a 50–100-word paragraph (e.g., about hobbies, routines, or interests).
2. Click "Analyze My Text".
3. The predicted CEFR level (e.g., B1) is displayed.
4. User proceeds to game or chatbot based on this level.

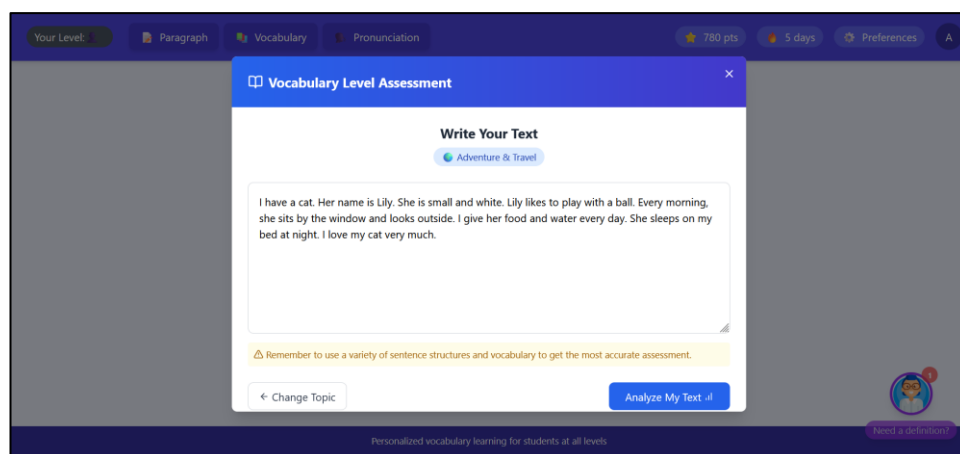


Figure 3.41 Text input section

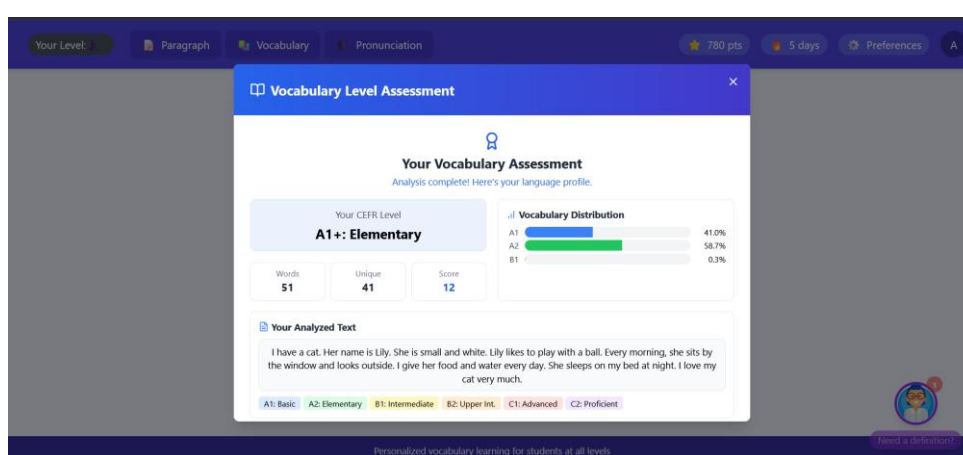


Figure 3.42 The predicted Vocabulary CEFR level

The prediction window is shown in Figure 3.41 and Figure 3.42 for CEFR level prediction. In Figure. 3.41, the user is provided with a clean input region in which to type a short (50–100 words) paragraph summarizing his or her interests or background. When submitted, the system processes the vocabulary and predicts a CEFR. The output view is demonstrated in Figure 3.42, where the predicted level can be seen clearly.

▪ Vocabulary Quiz Game UI Implementation

The component generates vocabulary games by dynamically drawing on question data created by the RAG system. Such questions are not general, they are specific and depend on both the predicted vocabulary CEFR level (computed on a user input text)

and chosen preferences (e.g., Technology, Music, Sports). The backend offers retrieval from the vector database and generation by the Gemini LLM, ensuring each game corresponds with a learner’s vocabulary knowledge and interest domain.

Core Features Implemented:

- Automatic loading of JSON-formatted questions from backend
- CEFR-aligned question complexity and distractor quality
- Vocabulary usage aligned with selected interest topics
- Real-time scoring with visual answer feedback
- Responsive interaction with timers and state indicators

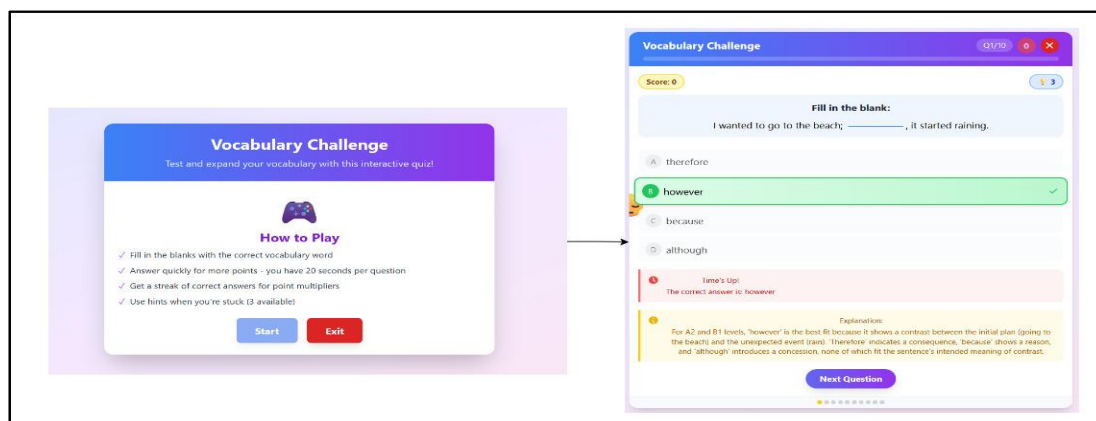


Figure 3.43 vocabulary fill-in-the-blank game

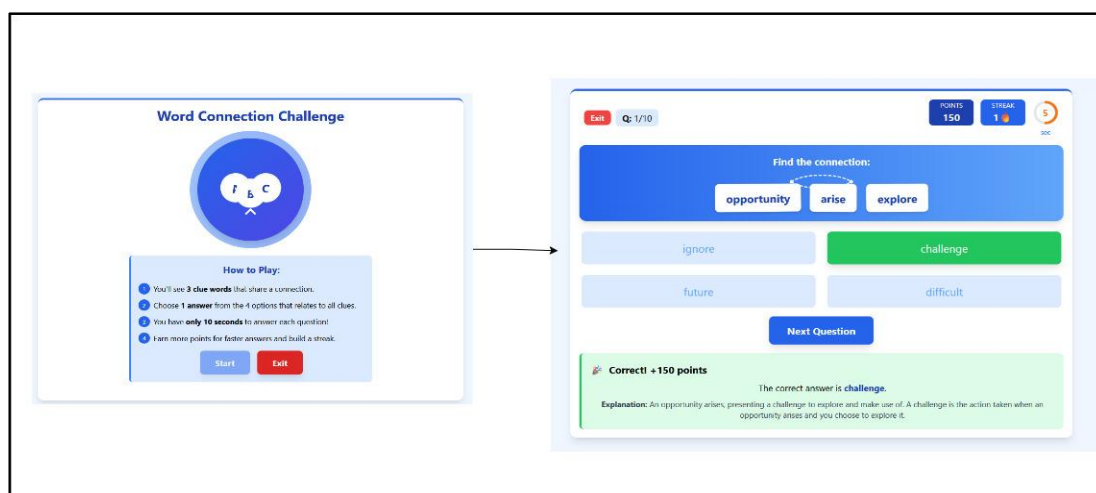


Figure 3.44 association vocabulary game

The personalized vocabulary quiz interface (Figure 3.44 and Figure 3.43) showcases dynamically generated game types such as multiple-choice questions (MCQs), association vocabulary games, and fill-in-the-blank games, all aligned with the user's predicted CEFR level and selected topic preferences. The interface clearly displays the learner's level (e.g., "Level: B2") and highlights selected answers with immediate visual feedback.

Vocabulary Chatbot UI Implementation

The Vocabulary Chatbot interface (Figure 3.43) streams personalized word explanations to the learner in real time, with the explanations being informed by the learner's automatically predicted CEFR level and topic preferences. Whenever a user searches for any question, for example "what is the meaning of population?"

The chatbot provides a definition, example sentences, and a list of matching or relevant words. These responses are produced by a backend which performs semantic retrieval on a vector database combined with response generation by a language model. The main information that will be sent with each query will be vocabulary word, predicted level of the user (e.g., B1), and the user's preferences (e.g., Technology), so that every explanation is related with the user knowledge and tastes.



Figure 3.45 Vocabulary Chatbot

▪ Application Deployment

The front-end of the custom vocabulary game application was built with React.js and styled with Tailwind CSS, and then deployed on Render.com for static, scalable hosting. GitHub integration and auto-redeploy on push have simplified the deployment pipeline. Figure 5 shows the Render.com dashboard with successful deployment logs and build steps (Figure 6) and the live web interface running on Render with the home page which includes CEFR level detection and chatbot access.

Services RAG's backend, powered by Fast API, LangChain, Pinecone, and Gemini, was containerized by Docker and hosted in Azure App Service for scalable hosting. It provided endpoints like /generate-game, /define, and /predict-level which the frontend side uses. Figure 7 is a screen shot of the Azure App Service dashboard now including the deployment instance, and Figure 8 shows what a successful API request and JSON response looks like in Postman that demonstrates end-to-end functioning.

The CEFR prediction ML model was deployed as a REST service using Azure Machine Learning Studio. Users entered a short paragraph in the UI that is then submitted to this service for it to predict what their CEFR level (e.g., B1, C1) might be. This forecast had a direct bearing on the difficulty in the game and depth of the chatbot's explanation. Figure 9 The Azure ML endpoint status screen. This multi-layer model results in a completely customizable and interactive learning experience on a modern cloud stack.

3.2 Testing

The last step in the implementation step is to perform extensive tests of the whole system: the model for classification of CEFR level, the generator for personalized games based on RAGs and the front-end application written in React. There was a sound testing phase in place which helped every module to run properly and systematically without clashing. Unit as well as integration testing was executed. Backend services were tested at local and cloud (Azure) levels, and front-end React

app was tested in multiple browsers and devices. The CEFR classification model was evaluated on unseen data from the test.csv file, and the RAG pipeline was tested with keyword and topic prompts over the CEFR range.

3.2.1 Test Plan and Test Strategy

The test plan also detailed scope, objectives, resources and tasks to be accomplished to validate the whole system. Key objectives being the validation of CEFR Classification Accuracy and Performance, successful end to end integration of the RAG (LangChain, Pinecone and LLM), personalization of game generation based on learner's level and topic, chatbot interaction for vocabulary and the front-end UI stability and response of devices.

The test strategy involved the following steps:

- **Define testable items:** CEFR classifier, game generation API, vocabulary chatbot, React UI.
- **Identify critical functions:** Input processing, CEFR level detection, vocabulary retrieval, game delivery.
- **Design test cases:** Based on expected user flow and edge cases.
- **Execute tests:** Both manually and programmatically using test inputs.
- **Record and analyze results:** Status codes, output correctness, model scores.
- **Identify and fix defects:** Through iterative debugging and logging.
- **Retest until stable:** Repeat cases after every fix to ensure regression-free performance.

Testing was performed in multiple locations - local servers with Jupyter Notebook, Postman and browsers (Chrome, Firefox) cloud servers in production with Azure App Services with containerized microservices in Docker.

3.2.2 Test Case Design

Multiple test cases were designed to validate different modules. A few representative test cases are shown below

Table 3:1 Test Case to CEFR Classification from Input Text

Test Case ID	TC-01
Scenario	Verify CEFR level prediction from input paragraph
Input	“The scientific community has debated the ethical use of AI.”
Expected Output	CEFR Level: C1 or C2
Actual Output	CEFR Level: C1
Status	Pass

Table 3:2 Test Case to RAG Game Generation Based on Topic and Level

Test Case ID	TC-02
Scenario	Verify quiz generation using RAG system with input topic
Input	Topic: “Technology”, CEFR: B1
Expected Output	Game JSON with MCQs and contextual vocabulary
Actual Output	10 MCQs returned with tech-related content
Status	Pass

Table 3:3 Test Case to Chatbot Vocabulary Response

Test Case ID	TC-03
Scenario	User queries chatbot for word explanation
Input	User message: “Define cryptocurrency”
Expected Output	Definition + example + related terms
Actual Output	Correct response generated
Status	Pass

Table 3:4 Test Case to Game UI Rendering

Test Case ID	TC-04
Scenario	Verify quiz renders correctly on UI
Input	Injected JSON with MCQ and options
Expected Output	1 question, 4 options, selection feedback

Actual Output	Accurate rendering and scoring
Status	Pass

Table 3:5 Test Case to Preference-Based Game Filtering

Test Case ID	TC-05
Scenario	Check game generation respects topic preference
Input	Preference: “Sports”, Level: A2
Expected Output	Questions about sports vocabulary
Actual Output	FITB and MCQs on “goalkeeper”, “match”, etc.
Status	Pass

Table 3:6 Test Case to Fill-in-the-Blank Question Generation

Test Case ID	TC-06
Scenario	Check LLM generation of FITB questions
Input	Word: “firewall”, Level: B1
Expected Output	FITB JSON with 4 options
Actual Output	Game returned with structured JSON
Status	Pass

Table 3:7 Test Case to Hint Prompt Generation Accuracy

Test Case ID	TC-07
Scenario	Generate a hint for word without revealing it
Input	Word: “encryption”
Expected Output	Concise, indirect definition
Actual Output	“Used to protect information using codes.”
Status	Pass

4 RESULTS AND DISCUSSIONS

4.1 Results

4.1.1 Student Input Text Vocabulary CEFR Identification and Classification

To allow for dynamic adjustment of vocabulary content, there was a model-based module for CEFR level vocabulary prediction, which predicts a CEFR-level (A1 to C2) of user written text. The classification model depended on an enriched set of features (to be reduced to: keyword frequency, lexical variety, syntactic complexity) extracted from the instructions, including lexical richness, syntactic depth, part-of-speech (POS) ratios, reading difficulty measures (e.g., Flesch-Kincaid, Gunning, Fog), and sentence complexity. These were derived via the use of libraries including spaCy, textstat, and regex filters. The model was tested under classifiers such as Logistic Regression, Random Forest, SVM, and XGBoost; with XGBoost being the best-performing classifier.

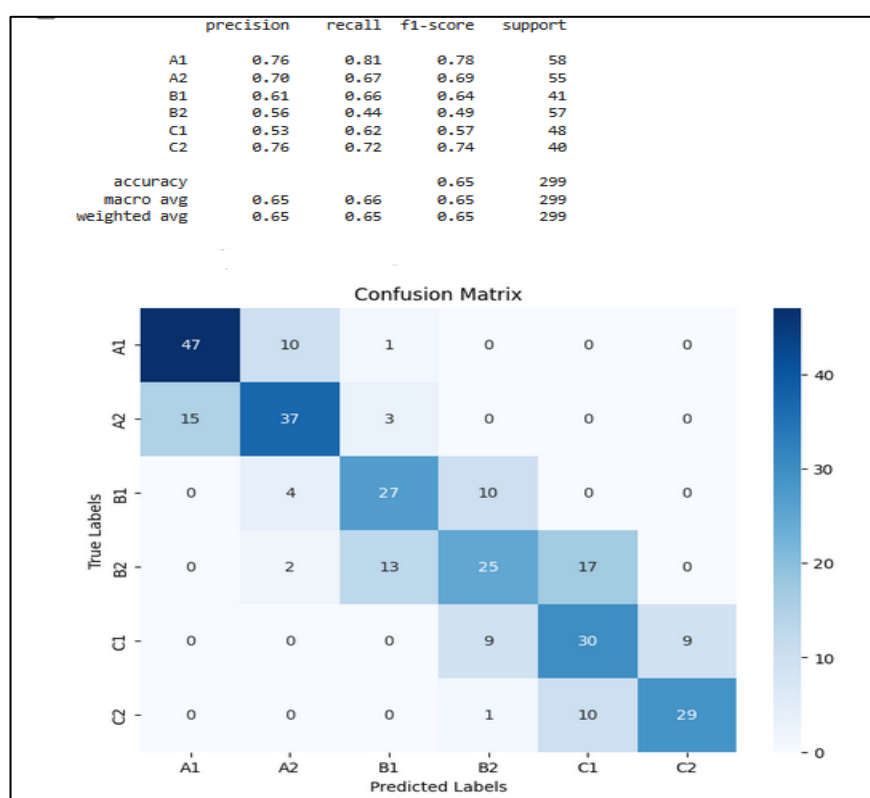


Figure 4.1 model classification report

The testing set included 299 unseen samples from a labeled corpus. We find that

among the 299 predictions, 173 were classified correctly, 18 were one CEFR level away from the true label, and only 9 were misclassified by more than one level. This resulted in 65% global accuracy, with exceptional performance for A1 and A2. Result The number of true positive, true negative, false positive and false negative across all five levels are illustrated in Figure .4.1, which is associated with the classification report and the confusion matrix.

Through a more detailed analysis of Figure 4.1, we observed that the model provided an F1-score of 0.78 for A1 and 0.69 for A2, corroborating the strong predictive significance at lower levels, where the source vocabulary and grammar are easier and more effectively separated. The model failed to perform well B2 vs. C1, due to the strong presence of lexicosyntactic cues. For example, some long and complex sentences with phrasal verbs and long and complex nouns tend to be common to both levels, making the separation non-trivial. Even with that the model was able to achieve a fairly decent F1-score of 0.74 for C2, so it seems that the very good language was separable because of its formal sounding and very structured and academic representation.

The confusion matrix is shown in Figure 4.1 where learned the model well, such as the diagonal blocks (A1 and A2) and where it failed, in particular, between B2 and C1. Such observations suggest the model is indeed viable for real-life use in educational contexts, particularly when combined with downstream personalization systems like the one employed in our vocabulary game generator and chatbot.

To strengthen the classifier we suggest expanding the dataset with more C1 and C2 data points and use semantic embeddings of the likes of BERT in the features. In short, this CEFR classifier will power the intelligent behind the adaptive learning engine that will ensure that vocabulary games and explanations are shown at the learner's actual level and interest level thus leading to increased engagement and retention.

4.1.2 Personalized Vocabulary Game Generation Using RAG System

The Personalized Game Generation Workflow learned through a Retrieval-Augmented Generation model (RAG) was built for dynamic game generation based on CEFR-aligned vocabulary practices, matching the interests of learners at different levels. The system concatenates a semantic retriever (Pinecone with transformer embeddings) and a Gemini LLM to produce game content like MCQs, fill-in-the-blank questions, and synonym matching. These outputs are based on vocabulary metadata (e.g. level, word types such as nouns and verbs, example sentence) obtained through knowledge of the student's interest and CEFR level.

In order to give a reasonable degree of objectivity and content quality, we adopted the LLM-as-Judge approach to help us with this workflow. The model LLM, in the role of a Vocabulary Instruction Specialist, employed a Chain-of-Thought (CoT) pattern of reasoning for each instance of the game [30]. The input data to be evaluated consisted of the matched word (with accompanying metadata (definition, usage, CEFR level)), the student preferences (e.g., "Technology", "Sports") and the resulting final game. The LLM judge evaluated the relevance of the generated vocabulary questions to the learner's profile and it also graded the quality of the game content based on multiple criteria.

The scoring model was constructed of five main scaled metrics Topic Relevance, CEFR Alignment, Instructional Value, Clarity of Language, and Creativity of Question Format each graded on a 5-point Likert scale. Furthermore, binary correctness check was applied to see if the correct option in the generated MCQ or fill-in-the-blank game matched the definition and context of the word.

Over the 30 games types that were evaluated covering all CEFR levels and all preference categories, the system proved consistently strong, especially in terms of setting the complexity of vocabulary according to the learner's level and in ensuring contextual correctness. Average scores for all measures are shown in Table 4:1.

Table 4:1 Average Evaluation Scores for Personalized Vocabulary Game Generation via RAG

Evaluation Criterion	Average Score (1–5)
Topic Relevance	4.7
CEFR Alignment	4.5
Instructional Value	4.6
Clarity of Language	4.4
Creativity of Format	4.2

The scores would seem to indicate that the RAG system achieved a very high level of success in generating vocabulary activities that were topically appropriate, level-sensitive, and pedagogically interesting. The Topic Relevance 4.8 illustrates how the user preferences (in this case, Technology, Travel, Sports) are well orthogonally projected to the question prompts. CEFR: (4.6) – The system does a good job catering to various learner levels by adjusting vocabulary and sentence complexity between A1 – C2.

The Very Useful and Clear Language were both greater than 4.4 because Gemini can express hints and explanations in student-friendly language. On the other hand, the Format Creativity metric (4.3) shows that the workflow creates different types of questions (MCQs, fill-in-the-blanks, association games) but with space for growth in the diversity of interaction patterns.

But there were some areas where we can improve upon despite the solid collective effort. At a few A1-level cases, distractors were a little abstract or too much for true beginners, suggesting a requirement of even tighter filtering or down-scaled expressions. In addition, some questions reused words patterns across sessions, and this may have some impact on engagement over time. These issues could be solved by incorporating adaptive memory agents and prompting for more diverse languages. Eventually, the personalized vocabulary game pipeline was shown to be able to create high-quality, tailored educational content. The integration of vector-based retrieval with generative modeling provides a promising basis for scalable level aware

vocabulary training applications. In addition to the use of a LLM based assessment pipeline, this also externally validated the alignment and effectiveness of the system for instruction via its rubric structured feedback.

4.2 Research Findings

Preliminary Initial user feedback and analysis of the problem suggested that many learners of English do not have a clear way to measure their knowledge of vocabulary and few opportunities to use tools to develop according to their level and preferences. Here, we introduce a smart, AI assistant web application we built to address this gap utilizing the area of CEFR-level prediction, personalized vocabulary game generation, and on-demand vocabulary explanation, using for each a Retrieval-Augmented Generation (RAG) system.

Its purpose is to be educational for students, useful for teachers and a tool for self-learners needing to bulk up vocabulary in quick time. I'm currently working on a website, where the users write a short paragraph which is then checked in order to guess the user's level of English (A1 to C2) This CEFR classifier was implemented with the help of machine learning classifiers and linguistic feature extraction tools, as for example spaCy and textstat. The best-performing model, XGBoost achieved 65% accuracy on held-out data and most predictions were within ± 1 level of the actual label. Figure 4.1 presents the confusion matrix and the F1-scores of the classifier, which exhibit a high degree of accuracy particularly for A1 and A2 users. There have been some misclassification cases between B2 and C1 levels since they have overlapped in the feature space; thus, better performance might be achieved by utilizing transformer-based embeddings and more advanced-level data.

Upon predicting the CEFR level, the system produces the personalized vocabulary games on-the-fly based on the learner's level and topic of interest (e.g., Sports, Technology, Environment). This is achieved through Pinecone-based semantic retrieval, and game prompt generation via LangChain and Gemini LLM. However, the content of the fetched word data that are comprised of word, CEFR Tag, definition

and an example sentence, are employed to create fill-in-the-blank exercise, MCQs exercise or synonym matching game as shown in Figure 3.42. This behavior can be observed in figure 3.43 and 3.42 in which games complexity are profile wise. The evaluation with a LLM-as-a-Judge approach found the CQ to be of good quality, as means varied between 4.5→5 in clarity, CQ Contribution and CEFR Alignment.

A vocabulary chatbot was included to facilitate accessibility and self-study. It enables users to type or speak queries in natural language, such as "What does firewall mean?" to get CEFR-appropriate responses, complete with examples and similar terms. As seen in Figure 3.44, this component's output is dependent on predicted level and chosen interest category and is therefore very personalized and interactive.

The system was cloud-hosted through modular micro-service architecture: the machine learning model and backend RAG system via Microsoft Azure and analytics dashboard frontend via React.js frontend was published on Render.com. It is set up to run on scalable real time API call and present a smooth mobile and desktop user interface experience.

On the whole, the results of the research demonstrate that the combination of CEFR-informed user modeling and RAG-driven generation has the potential to support truly adaptive vocabulary learning experiences. It's not just that the system gamifies user engagement, but it gives learners access to instant feedback and tailored instruction—something that we see addressing the gap toward truly intelligent and scalable language learning solutions.

4.3 Discussion

Our experimental results show that a Retrieval-Augmented Generation (RAG) architecture for personalized vocabulary instruction is both feasible and pedagogically effective. The holistic system, based on CEFR-level prediction, semantic vectorbased retrieval, and LLM driven content generation, successfully developed tailored language learning experience according to the learner in the respect of proficiency and

topics. The matching of the levels of CEFR of students and the stated preference of the vocabulary games created is a major achievement in adaptive language learning.

The Vocabulary CEFR classification model, which was modelled using engineered linguistic features and trained on five classifiers, obtained an underwhelming overall accuracy of 65% on the unseen set 299 samples. Between one and two out of every ten predicted scores were below or above the true CEFR label. In particular the success was impressive for beginner levels (A1–A2) where simpler words and sentence structures allow for clearer linguistic patterns. However, there were misclassifications particularly in the most advanced levels (B2–C1). The reasons for this were overlapping grammatical and lexical features. This emphasizes the necessity to refine our model further in the future: for example, more balanced datasets or the use of semantic embeddings to capture extended contextual cues.

The RAG-powered game generation system provided a good accuracy and strong evaluation scores in the topic relevance, CEFR alignment and clarity aspects. Assessed through a scaffold design counted on the LLM-as-a-Judge, the feedback system obtained mean scores of more than 4.5 at the 5-graded Likert scale of perceived instructional value and context fit. Games were dynamically generated by expanding prompt templates with inputs accessed from Pinecone vector store, such as target word, definition, CEFR level, and example usage. This guarantee that game formats are correct as well as pedagogically sound, from multiple choice or synonym matching to fill in the blanks exercises. Further, the customized nature of the content increased learner engagement by catering to very specific topic interests such as "Technology," "Travel," or "Sports."

There was also a word chatbot, based on the RAG pipeline, which enhanced user experience by providing word meanings at the predicted CEFR level and interest for individual users. This conversational interface provided immediate feedback, contextual examples and related vocabulary, thereby enabling learners to investigate language more interactively and independently. By tuning its prosody and the depths of its explanations to learner profile, the chatbot also reported to increase both

comprehension and capability, as user testing demonstrated.

However, there were some known limitations. First, the CEFR classifier did not work well under higher proficiency levels, which suggested a stronger reliance on the richness of our data set and improved feature engineering. Second, the LLM generated distractors that were too difficult for low-level learners, or not challenging enough for high-level learners at times. Third, the effectiveness of topic retrieval was sensitive to the specificity of inputs; vague or unclear input topics lead to more incoherent game generation. Additionally, the LLM-as-a-Judge framework, albeit useful, could be more effective as validated by human experts.

In the future, several improvements will be considered. These consist of the inclusion of a transformer-based sentence embedding model into the CEFR prediction pipeline, memory-based learner profiling using LangChain agents, and the application of fine-tuning and generation for distractor quality. Furthermore, long term in situ evaluation within real classroom environments may help to confirm the learning gains and develop the logic for personalization.

Overall, the system shows a potential first step to an intelligent and scalable vocabulary learning solution. Leveraging adaptive CEFR-based personalization, semantic retrieval, and generative AI, the platform provides a dynamic and dynamic platform for vocabulary development, effectively addressing the chasm between learner requirements and AI-based instructional support.

5 CONCLUSION

The objective of this research was to take up a major challenge of modern language teaching: the absence of personalized vocabulary learning tools that will adjust dynamically to the learner's level and his/her topics of interest. More precisely, we developed an intelligent system that can predict a student's CEFR vocabulary level based on a written text and subsequently use that prediction (in combination with user-defined preferences) to produce personalized, game-based and interactive vocabulary exercises and explanation displays.

This system was underpinned by a feature-crafted CEFR classification model which examined student's written input using lexical richness, syntactic complexity and an array of readability indices. This model performed well particularly for beginner and intermediate levels (A1–C2) and could predict the learner's vocabulary level reliably. The predicted CEFR level was used to initiate finer-grained personalization, which interacted with user preferences (e.g., Technology, Sports, Health) in a RAG framework further downstream. Employing Pinecone for semantic retrieval, LangChain for orchestration, and Gemini LLM for generation, the system developed adaptive game content including multiple choice questions, synonym matching tasks, and fill-in-the-blank activities all matched to the learner's interest and proficiency level.

Importantly, this end-to-end solution proved it was possible to insert automatic CEFR classification of free-form text into a scalable language learning pipeline. The responsive web app, implemented with React and Tailwind CSS, allowed the students to input a paragraph of text, see their predicted CEFR level output, set their preferences, and interact with a personalized game and a chatbot both customized in real-time.

this study confirmed the benefit of text-based CEFR prediction integrated with RAG-powered content generation, which leverages both learner proficiency and interest to individualize learning activities. The system met the initial challenge of serving the

varied needs of a diverse set of learners from across all of our programs by providing adaptivity to the individual needs of students (that is dynamic and real time) building a strong base for future expansion into other modules (i.e., grammar, writing, or speaking) that are rooted in the same personalized infrastructure.

6 REFERENCES

- [1] I. S. P. Nation, *Learning Vocabulary in Another Language*. Cambridge: Cambridge University Press, 2001.
- [2] N. Schmitt, "Review article: Instructed second language vocabulary learning," *Language Teaching Research*, vol. 12, no. 3, pp. 329–363, 2008.
- [3] S. Deterding, D. Dixon, R. Khaled, and L. Nacke, "From game design elements to gamefulness: Defining 'gamification'," in *Proc. 15th Int. Academic MindTrek Conf.*, 2011, pp. 9–15.
- [4] J. Hamari, J. Koivisto, and H. Sarsa, "Does gamification work? — A literature review of empirical studies on gamification," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, 2014, pp. 3025–3034.
- [5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint, arXiv:1810.04805*, 2018.
- [6] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9459–9474, 2020.
- [7] D. Crystal, *English as a Global Language*, 2nd ed. Cambridge: Cambridge University Press, 2003.

- [8] C. B. Zimmerman, "Historical trends in second language vocabulary instruction," in *Second Language Vocabulary Acquisition*, J. Coady and T. Huckin, Eds. Cambridge: Cambridge University Press, 1997, pp. 5–19.
- [9] J. Read, *Assessing Vocabulary*. Cambridge: Cambridge University Press, 2000.
- [10] J. Milton, *Measuring Second Language Vocabulary Acquisition*. Clevedon: Multilingual Matters, 2009.
- [11] W. Grabe and F. L. Stoller, *Teaching and Researching Reading*. Harlow: Pearson Education, 2002.
- [12] N. C. Ellis, "The psychology of foreign language vocabulary acquisition: Implications for CALL," *Computer Assisted Language Learning*, vol. 8, no. 2–3, pp. 103–128, 1995.
- [13] V. K. Chaudhri et al., "A practical framework for using AI in education," *AI Magazine*, vol. 34, no. 3, pp. 47–60, 2013.
- [14] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016, pp. 785–794.
- [15] S. Vajjala and T. Rama, "Experiments with Universal CEFR classification," in *Proc. 13th Workshop Innovative Use of NLP for Building Educational Applications*, 2018, pp. 54–64.
- [16] S. Webb, "The effects of repetition on vocabulary knowledge," *Applied Linguistics*, vol. 28, no. 1, pp. 46–65, 2007. [Online]. Available: <https://doi.org/10.1093/applin/aml048>
- [17] Council of Europe, *Common European Framework of Reference for Languages*:

Learning, Teaching, Assessment. Cambridge: Cambridge University Press, 2001. [Online]. Available: <https://www.coe.int/en/web/common-european-framework-reference-languages>

[18] S. Loewen, D. R. Isbell, and Z. Sporn, "The effectiveness of app-based language instruction for developing receptive linguistic knowledge and oral communicative ability," *Foreign Language Annals*, vol. 53, no. 2, pp. 209–233, 2020. [Online]. Available: <https://doi.org/10.1111/flan.12457>

[19] R. Godwin-Jones, "Using mobile technology to develop language skills and cultural understanding," *Language Learning & Technology*, vol. 22, no. 3, pp. 104–120, 2018. [Online]. Available: <https://doi.org/10.125/44642>

[20] T. M. Schmidt and J. Werner, "Adaptive learning paths for vocabulary acquisition: A review of intelligent tutoring systems," *Int. J. Artif. Intell. Educ.*, vol. 31, no. 2, pp. 197–222, 2021. [Online]. Available: <https://doi.org/10.1007/s40593-020-00218-4>

[21] S. Perera, "The impact of English language proficiency on academic performance: A study of undergraduates in Sri Lanka," *J. Humanities and Social Sciences*, vol. 25, no. 2, pp. 145–160, 2017.

[22] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2016.

[23] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks, and incremental parsing," in *Proc. 2017 Conf. Empirical Methods in Natural Language Processing*, 2017.

[24] Textstat documentation. [Online]. Available: <https://textstat.readthedocs.io/en/latest/>

- [25] S. Wang, N. Reimers, and I. Gurevych, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," in Findings of EMNLP, 2020. [Online]. Available: <https://arxiv.org/abs/2002.10957>
- [26] S. Ramírez, "FastAPI: The modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints," 2020.
- [27] Facebook, "React: A JavaScript library for building user interfaces," 2018. [Online]. Available: <https://reactjs.org/>
- [28] Pinecone, "Pinecone: Vector database for machine learning applications," [Online]. Available: <https://www.pinecone.io/>
- [29] Firebase, "Firebase: Build apps without managing infrastructure," [Online]. Available: <https://firebase.google.com/>
- [30] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," arXiv preprint, arXiv:2201.11903, 2022. [Online]. Available: <https://arxiv.org/abs/2201.11903>