Bridging Language Barriers in Programming Education: Java Programming Assistance Tool for Sinhala Native Speakers

Kalpani Sachie Nelanka Athukorala[1,*] and Dilshan Indraraj De Silva[2]

[1]Oloid (Pvt) Ltd, Colombo 08, Sri Lanka
[2]Department of Information Technology, Faculty of Computing, Sri Lanka Institute of Information Technology, Malabe, Sri Lanka
Email: kalpani.athukorala@oloidtechnologies.com (K.S.N.A.); dilshan.i@sliit.lk (D.I.D.S.)
[*]Corresponding author
Manuscript received October 17, 2024; revised February 10, 2025; accepted April 14, 2025; published September 12, 2025.

***Abstract*—This study presents an innovative programming assistance tool designed to address language barriers faced by Sinhala-speaking novice Java programmers. The tool provides real-time Java code generation and diagram creation based on Sinhala programming queries, enhancing conceptual understanding. Developed using a Design-Based Research methodology, the tool underwent iterative testing with 122 Sinhala-speaking learners, incorporating user feedback to refine usability and performance. Central to the system is Generative Pre-trained Transformer, version 3.5 Turbo, ensuring accurate translations and programming assistance, alongside a transformer-based model that translates Sinhala queries into English for processing. The translation model achieved 91.37% accuracy, with strong Bilingual Evaluation Understudy scores validating its contextual relevance. The tool's practical applications extend beyond academia, supporting educational institutions, self-learners, and industry professionals in learning and skill development. Statistical evaluation of user performance demonstrated significant improvements in programming comprehension, reinforcing its effectiveness. By promoting inclusivity and expanding access to programming knowledge, this research contributes to the advancement of Sri Lanka's technology sector and establishes a scalable framework for broader implementation in multilingual programming education.**

***Keywords*—Design-Based Research (DBR), pedagogical strategies, java coding, Large Language Model (LLM), Sinhala native speakers**

I. Introduction

Programming education has advanced significantly in recent years, leading to the development of various tools aimed at enhancing learning experiences. However, non-English-speaking novice programmers face a persistent challenge: the language barrier. Learners who are more comfortable in their native language often struggle to grasp English-based programming materials, which impedes their progress and creates educational inequality in computer science [1].

Language barriers introduce a significant cognitive load, as learners must simultaneously navigate complex technical concepts and a non-native language. This challenge often results in frustration and reduced learning outcomes. While existing programming assistance tools focus on syntax correction, code completion, and debugging, they rarely address the cognitive overload caused by language translation. Additionally, current machine translation solutions lack domain-specific accuracy, often producing incorrect or contextually irrelevant translations that hinder effective learning.

To address this issue, this study introduces a Sinhala-based Java assistance tool designed to mitigate the language barriers faced by Sinhala-speaking novice programmers. The tool leverages machine translation and transformer architectures to provide real-time, contextually accurate translations and programming assistance in Sinhala. By integrating these technologies, it aims to bridge the language gap, enabling learners to focus on mastering programming concepts without the added burden of translation.

This study adopts the DBR methodology, which facilitates iterative design, development, and refinement based on practical feedback [2]. DBR is particularly suitable for programming education as it allows the tool to evolve with learners' needs, ensuring continuous improvement based on real-world usability and effectiveness.

The dominance of English in programming languages and educational resources places non-English speakers at a disadvantage, limiting their ability to fully engage with and contribute to the field. This research seeks to promote inclusivity by developing a solution that accommodates diverse language preferences and learning environments. Unlike existing programming tools, the proposed solution offers structured learning support beyond mere translation, ensuring a more effective educational experience.

The primary objective of this study is to develop a comprehensive programming assistance tool that allows Sinhala-speaking learners to input queries in Sinhala and receive relevant Java code, along with a corresponding program structure diagram. This tool not only addresses the immediate educational needs of Sinhala-speaking programmers but also sets a precedent for expanding similar tools to other non-English-speaking communities. By removing linguistic barriers, this research contributes to the broader goal of democratizing programming education, ensuring language becomes a bridge to knowledge rather than a hindrance.

The remainder of this paper is organized as follows: Section II provides the background necessary to contextualize the research. It begins by exploring the linguistic challenges faced by Sinhala-speaking students in programming education, highlighting the dominance of English in programming resources and instruction. The section emphasizes how this language barrier can hinder comprehension and discourage engagement among novice learners. It then discusses the selection of Java as the target programming language, citing its widespread use in education, object-oriented structure, and compatibility with beginner-friendly tools. The potential of AI-driven translation and programming assistance tools is introduced next, illustrating how such technologies can bridge the gap

between natural language input and code generation, especially in linguistically diverse environments. Finally, the section reviews key advancements in transformer models that have enabled significant improvements in natural language understanding and translation—technologies that form the foundation of the proposed Sinhala-based programming assistance tool.

Section III reviews the existing body of literature relevant to the development of the proposed tool. It begins by examining current programming assistance tools and Sinhala-English translation systems, identifying the lack of tailored solutions for Sinhala-speaking learners. The section then explores the broader application of Artificial Intelligence (AI) and Natural Language Processing (NLP) in programming education, focusing on how these technologies can support code generation, syntax translation, and learning personalization. It further investigates the role of gamification and AI-driven feedback mechanisms in enhancing learner motivation and programming skill acquisition. Attention is also given to AI-enabled approaches for improving student engagement and self-assessment within online learning environments. Finally, the section presents a comparative analysis of existing self-assessment tools and related studies, highlighting the research gap addressed by the proposed Sinhala-based solution that combines translation, code generation, and educational reinforcement in a single tool.

Section IV details the methodology adopted in developing the Sinhala-based Java programming assistance tool. It begins with an explanation of the DBR methodology, emphasizing its iterative structure involving problem identification, solution design, real-world testing, and evaluation. The back-end Application Programming Interface (API) development is then described, highlighting how user queries in Sinhala are processed, translated into English using a custom transformer model, and passed to the Generative Pre-trained Transformer (GPT), version 3.5 Turbo, for code generation. The section also outlines the architecture and training of the transformer-based Sinhala-English translator, including its attention mechanisms, token embeddings, and fine-tuning process using a Java-specific dataset. Next, it presents the front-end development, showcasing an intuitive user interface built to reduce cognitive load by enabling real-time code and diagram generation based on Sinhala queries. Finally, the section explains the user testing and quality assurance integration process, detailing functional and usability testing conducted with 122 Sinhala-speaking novice programmers, and how iterative feedback was used to refine the tool's effectiveness and reliability.

Section V presents a detailed account of the results and discussion. It begins by outlining how the tool was developed through iterative phases of the DBR methodology, incorporating user feedback and continuous refinement. The section then examines the composition and preparation of the training, validation, and testing datasets used to develop the translation model. It explores how variations in sentence structure affect translation quality, providing insights into linguistic nuances that influence model outputs. The performance of the model is assessed through accuracy metrics and Bilingual Evaluation Understudy (BLEU) scores, supported by a comprehensive analysis of dataset distribution. Usability testing is described, detailing the evaluation methodology and findings from 122 Sinhala-speaking learners. Further, the section connects BLEU scores to user performance improvements and compares the proposed model's performance with other English–Sinhala Neural Machine Translation (NMT) systems. Finally, it discusses the practical applications of the tool, its educational impact, pedagogical benefits, and the technical soundness and reliability of the implemented system.

Section VI concludes the paper by summarizing the key findings, technical contributions, and educational benefits of the proposed tool. It also outlines future work, such as extending language support, integrating with development tools, and expanding to other programming domains.

II. Background Study

A. Sinhala and English in Programming Education

Sinhala, an Indo-Aryan language, is the national language of Sri Lanka and is spoken by the majority of the population. It possesses unique linguistic characteristics, including a distinctive script, phonetic structure, and grammatical constructs that differentiate it from other Indo-Aryan languages [3]. Understanding these features is essential for developing effective educational tools that support Sinhala-speaking learners in grasping complex subjects.

In contrast, English, originating from the West Germanic language family, has become the primary global language due to historical colonization and the impacts of modern globalization. It is the dominant language in countries such as the United Kingdom, the United States, Australia, Canada, and New Zealand, as well as in several former British colonies [4]. This widespread dominance extends to programming education, where most instructional materials and programming languages are based on English syntax and terminology. As a result, non-English speakers, including Sinhala-speaking learners, often face difficulties in grasping programming concepts, leading to an accessibility gap in computer science education.

B. Java as a Programming Language

Java, a widely used object-oriented programming language, was developed by Sun Microsystems (now Oracle) in the mid-1990s. Designed for platform independence, security, and robustness, Java eliminates complex low-level programming constructs, making it suitable for educational settings. Many institutions introduce Java as a foundational language due to its readability and maintainability. However, for non-English-speaking learners, the challenge extends beyond programming logic to understanding English-based syntax and documentation.

C. AI-Driven Translation and Assistance in Programming Education

To mitigate language barriers in programming education, this study integrates a custom transformer-based Sinhala-English translation model with AI-driven assistance. The tool provides real-time Java code generation and diagrams, ensuring an inclusive learning experience. By automating translation and contextual programming support, the system enhances comprehension and encourages broader

participation in programming education, particularly among Sinhala-speaking students.

D. Advancements in Transformer Models

The Transformer model, introduced by Vaswani *et al.* [5], has revolutionized NLP by replacing traditional Recurrent Neural Networks (RNNs). Unlike RNNs, which process input sequentially, transformers leverage self-attention mechanisms to analyze entire sequences simultaneously, improving efficiency and accuracy in capturing long-term dependencies.

Compared to RNNs and Long Short-Term Memory (LSTM) networks, transformers offer significant advantages:

- Parallel Processing: Unlike RNNs, which process tokens sequentially, transformers handle entire sentences simultaneously, improving speed and efficiency.
- Enhanced Long-Term Dependency Capture: Traditional models suffer from the vanishing gradient problem, which weakens token influence over long sequences. Transformers overcome this by establishing direct connections between all words in a sequence.
- Superior Accuracy in Translation Tasks: Gated Recurrent Units (GRU) and LSTM-based RNN models struggle with complex language patterns, whereas multi-head self-attention and positional encodings in transformers enable precise translations [5, 6].

By leveraging these advancements, the proposed tool provides contextually accurate programming assistance for Sinhala-speaking learners, ensuring that language is no longer a barrier to programming education.

III. Literature Review

A. Existing Programming Assistance Tools and Sinhala-English Translation Systems

A range of tools has been developed to support programming education and overcome language barriers between Sinhala and English. Some tools primarily assist with programming education, while others focus on facilitating translation between Sinhala and English for better accessibility. Additionally, certain tools integrate both programming assistance and language translation, offering a comprehensive solution for Sinhala-speaking programmers. The following is an overview of existing tools in these categories:

- **Helaa [1]**: A Sinhala programming assistance tool that introduces a novel programming language tailored specifically for Sinhala speakers. This Java-based tool integrates Sinhala syntax and compiler exceptions, making programming more accessible to non-English speakers. Future development aims to expand its compatibility with JavaScript and Python.
- **Interactive Programming Assistance Tool (iPAT) [7]:** A tool designed to assist students and instructors in managing computer labs through functionalities like error handling, remote access, and PC inventory management. While it enhances the learning environment, it does not specifically address the linguistic barriers faced by Sinhala-speaking programmers.
- **CodeMage [8]**: An educational platform that provides real-time programming guidance, automatic code generation, and best practices. Although robust in supporting novice programmers, it does not specifically cater to Sinhala speakers.
- **SimpliTrans [9]**: A bilingual coding tool allowing users to switch between English and Sinhala. It translates programming commands and instructions, making coding more accessible to Sinhala-speaking learners by integrating localized programming terminology.
- **Web Programming Assistance Tool (WPAT) [10]**: A debugging-focused tool that interprets compiler error messages and suggests fixes. While beneficial for reducing frustration during programming, it does not address language barriers.
- **Sinhala to English Language Translator [11] and Sinhala to English and English to Sinhala (SEES) [12]**: These tools facilitate Sinhala-English translation, including Singlish recognition, making them useful for general language translation but not specifically optimized for programming-related terminology.
- **NMT for Sinhala-English Code-Mixed Text [13]**: This research utilizes advanced neural network architectures, including LSTM units and Sequence to Sequence (Seq2Seq) models, to translate Sinhala-English mixed text. The integration of a normalization pipeline and Teacher Forcing mechanism aims to improve the accuracy and fluency of translations, supporting bilingual communication effectively.
- **Example-Based Machine Translation for English-Sinhala Translations [14]**: This model leverages a knowledge database to perform accurate translations between English and Sinhala by utilizing inter-language matching techniques. This approach helps in understanding and generating contextually relevant translations, thereby facilitating smoother communication between English and Sinhala speakers in educational and professional settings.

B. AI and NLP Applications in Programming Education

1) AI-driven pedagogical strategies

Recent studies have explored the integration of AI to support non-native English speakers in programming education. AI-driven pedagogical strategies are increasingly being developed to enhance multilingual learners' comprehension and engagement in coding. For instance, research by Long *et al.* [15] introduced innovative instructional techniques such as storytelling, role-playing, and AI-driven text-to-speech-to-text mechanisms to foster AI literacy among multilingual students. These methods leverage linguistic scaffolding and translanguaging to bridge language gaps, ensuring that programming concepts are effectively conveyed regardless of the learner's native language.

2) AI-based programming assistants

AI-based programming assistants are being assessed for their role in aiding multilingual education. Wang *et al*. [16] evaluated the effectiveness of AI-powered tools like

ChatGPT in solving diverse computer science problems. Their study explored how AI-generated explanations and code suggestions can be optimized to support learners with varying levels of English proficiency, highlighting the importance of adapting instructional materials to AI capabilities while incorporating instructor insights. Furthermore, Piech and Abu-El-Haija [17] developed "CodeInternational," an NLP-based tool designed to facilitate code translation between human languages. By localizing programming instructions and offering context-aware explanations, this system aims to make computer science education more accessible to non-English speakers.

3) Advancements in machine translation and NLP

While several tools aim to address programming education challenges for non-English speakers, the integration of advanced AI techniques like NLP offers new avenues for innovation. For instance, Shaik *et al*.'s study [18] delves into the potential of NLP in educational feedback analysis, emphasizing its role in bridging language barriers and enhancing student engagement. By employing methods such as sentiment analysis, entity recognition, and topic modeling, NLP provides actionable insights from textual feedback, which could be adapted to evaluate programming education tools tailored for Sinhala speakers. Furthermore, the study highlights challenges such as domain-specific language ambiguity and proposes strategies to overcome these limitations, showcasing its relevance to creating effective educational technologies in linguistically diverse contexts.

The following studies exemplify key advancements in machine translation and NLP:

- **Attention Is All You Need [19]:** This research paper introduced the Transformer model, discussing its architecture and improvements over Seq2seq models. The Transformer model minimizes the need for recurrent or convolution layers, focusing primarily on the attention mechanism. This mechanism enables efficient parallelization, significantly reducing training times. The paper covers various aspects of the Transformer model, including the encoder, decoder, attention, feed-forward layers, embedding, and positional encoding. The model demonstrated exceptional performance in machine translation, achieving state-of-the-art BLEU scores on the WMT 2014 English-to-German and English-to-French translation benchmarks. The study highlights the Transformer's scalability and robustness, establishing it as a cornerstone in machine learning.
- **Machine Translation on Dravidian Languages [20]:** This study explores translation resources for Dravidian languages, predominantly spoken in southern India. Despite their popularity, Dravidian languages have not received adequate attention due to a lack of translation resources. The paper discusses several machine translation models, including rule-based machine translation, NMT, example-based machine translation, hybrid machine translation, transformer model, statistical machine translation, and LSTM model. It explores the characteristics, importance, and advantages of these models, highlighting the most effective model for improving information access and generation for monolingual speakers in the region.
- **Improving English to Sinhala NMT using Part-of-Speech (POS) Tag [21]:** This study identifies techniques to improve the English to Sinhala translation model. Byte Pair Encoding (BPE) and Character Segmentation were utilized to enhance translation quality. The study sheds light on optimizing NMT for languages with complex linguistic structures and addressing the challenges of low-resource constraints.

Although AI-based tools enhance learning through intelligent assistance and translation, they do not inherently address student motivation and engagement. To further enhance learning outcomes, gamification strategies have been integrated into AI-driven programming education, providing interactive and competitive learning experiences. The next section examines these strategies.

C. Gamification and AI-Driven Learning Insights in Programming Education

1) Gamification approaches

Gamification has emerged as a transformative approach in programming education. Zhan *et al*. [22] conducted a meta-analysis assessing its effectiveness, demonstrating how reasoning strategy games and competitive mechanisms can significantly enhance student motivation and learning outcomes. The study underscores the importance of balancing cognitive load and maintaining intrinsic motivation to create inclusive, gamified educational platform.

2) Predictive Learning Analytics (PLA)

PLA has emerged as a powerful tool for improving educational outcomes by identifying students at risk. Hlosta *et al*. [23] investigates the errors made by PLA systems, particularly False Positives (FP) and False Negatives (FN), in predicting assignment submissions. Through qualitative interviews with students, the study reveals that factors such as unexpected life events, shifting responsibilities, and technical issues significantly contribute to these errors. The findings underscore the limitations of machine learning algorithms in capturing contextual nuances and highlight the need for integrating human intelligence, such as teacher insights, into AI-driven systems. By addressing these challenges, PLA can play a crucial role in enhancing not only engagement but also the accuracy and inclusivity of educational tools, including those aimed at overcoming language barriers in programming education.

3) Explainable AI (XAI) in education

XAI has gained prominence in the educational sector due to its potential to improve trust and usability in AI-driven systems. Khosravi *et al*. [24] propose the XAI-ED framework, which identifies six key aspects of explainability tailored to educational contexts. These aspects include stakeholder needs, benefits, explanation delivery methods, AI model types, human-centered interface designs, and potential pitfalls. The framework is exemplified through four case studies, demonstrating its applicability in designing educational AI tools that support metacognitive processes like self-monitoring and reflection. By addressing

Fairness, Accountability, Transparency, and Ethics (FATE), the study emphasizes the importance of mitigating biases and ensuring that AI tools align with educational goals. The insights from this work are particularly valuable for integrating XAI into tools aimed at reducing linguistic and accessibility barriers in programming education, ensuring that explanations foster both trust and effective learning outcomes.

While gamification strategies enhance motivation and engagement, effective learning also requires students to assess their own progress and identify areas for improvement. AI-driven self-assessment tools complement gamification by providing structured feedback, enabling students to track their learning outcomes systematically. The next section explores these self-assessment tools and their impact on online learning environments.

D. AI-Driven Approches to Student Engagement and Self-Assessment in Online Learning Education

A study by Yang *et al*. [25] explored how students' self-assessment behaviors in online learning environments impact their academic performance. This research specifically looks at online self-assessment behaviors in the context of a 6-week accounting course where students completed formative quizzes after class. The study investigates students' patterns of online self-assessment behavior using AI techniques, specifically hierarchical clustering algorithms. By analyzing factors such as test attempt frequency, question views, submission rates, and hint usage, the researchers identified three distinct behavioral patterns: students who rarely completed assessments, those who engaged in nonstandard behaviors, and those who consistently completed assessments with standard behaviors. AI clustering enabled profiling of these behaviors, revealing that students who relied heavily on hints or engaged in extensive, unspaced practice were less likely to benefit from the testing effect, thus impacting their learning performance.

As AI continues to reshape self-assessment in online learning, evaluating existing tools and related studies is essential for understanding their effectiveness across various educational contexts. The following section provides a comparative analysis of the various tools and studies discussed in the paper, focusing on several key factors to provide the reader with an overall summary.

E. Comparative Analysis of Self-Assessment Tools and Studies

Table 1 provides a structured comparison of the tools and developments discussed in this paper, highlighting key factors such as target audience, core features, language support, and primary focus areas. This analysis offers valuable insights into the suitability of each tool for various educational needs and contexts.

Table 1. Overview of tools and research papers discussed in the study

Tool / Research Paper	Target Audience	Primary Features	Input Language	Focus Areas
Helaa	Sinhala-speaking novice programmers	Sinhala-based coding, compiler support	Sinhala	Introduces a Sinhala programming language to support non-English speakers
iPAT	Students, Instructors	Remote error handling, lab management	English	Supports computer lab management, but does not address linguistic barriers
CodeMage	Novice Java programmers	Real-time programming guidance, automated code generation	English	Supports Java programming but lacks Sinhala language support
SimpliTrans	Bilingual programmers	Bilingual programming support, localized error messages	English	Improves accessibility by translating programming keywords between Sinhala and English
WPAT	Novice C programmers	GUI-based debugging, compiler error analysis	English	Focused on enhancing error handling in C programming
Sinhala to English Language Translator	Sinhala speakers	General text translation, grammar checking, Sinhalese dictionary	Sinhala	Translates general Sinhala text to English, not specialized for programming
SEES	Sinhala and English speakers	Sinhala-English bidirectional translation	Sinhala, English	Supports linguistic translation but lacks programming-specific adaptations
Neural Machine Translator	Singlish speakers	AI-driven translation for mixed-language text	Singlish	Convert Singlish-based Sinhala to structured Sinhala text
Example Based Machine Translator	Sinhala speakers	Rule-based translation for structured documents	English	Primarily facilitates formal government documentation
AI-Driven Pedagogical Strategies	Non-native English speakers in programming education	AI-assisted storytelling, role-playing, and text-to-speech-to-text learning	Multilingual	Enhances programming comprehension for multilingual learners by integrating linguistic scaffolding and translanguaging techniques.
AI-powered tools	Educators, multilingual learners	AI-generated explanations and code suggestions, adaptation to instructional materials	Multilingual	Optimizes AI-generated content through instructor insights

CodeInternational	Non-English speaking programming students	NLP-based tool, localization of programming instructions, context-aware explanations	Non-English languages	May not fully capture nuanced programming terminology across all languages
NLP for Educational Feedback Analysis	Sinhala speakers	Sentiment analysis, topic modeling, linguistic adaptation	Sinhala	Uses NLP to evaluate student responses and provide contextual feedback
Attention Is All You Need	AI & NLP advancements	Attention-based model improving translation accuracy	Multilingual support (English, German, French)	Machine Translation, NLP Model Architecture
Machine Translation on Dravidian Languages	Dravidian Languages Speakers	Translation Dravidian Language of text to English	Dravidian Language	Translate Dravidian Language into English and improve information access and generation for monolingual speakers in the region
Improving English to Sinhala NMT using POS Tag	Sinhala Speakers	Translation of English text to Sinhala	English	Develop an efficient domain-specific English to Sinhala NMT system using the Transformer architecture, incorporating POS information as an additional linguistic feature.
Gamification in Programming Education	Novice programmers	AI-driven competitive coding, interactive assessments	English	Enhance motivation and knowledge retention through AI-assisted gamification
PLA	Educators, Students	AI-based student behavior analysis, risk prediction	English	Identifies at-risk students and personalizes educational strategies
Explainable AI (XAI-ED)	Educational technology designers	AI-driven personalized feedback with transparency	English	Focus on integrating XAI into educational tools to improve trust and outcomes, reducing linguistic and accessibility barriers in programming education.
AI-Based Self-Assessment Systems	Online learners, Educators	AI-powered real-time feedback, behavioral profiling	English	Uses hierarchical clustering to assess student self-learning patterns

IV. Methodology

This study presents the development of an innovative Java assistance tool specifically tailored for novice programmers who are native Sinhala speakers. The primary objective of this tool is to transcend the language barriers that hinder the effective learning of Java programming by providing real-time code generation and diagrams directly in Sinhala. This reduces reliance on external resources and facilitates independent learning. The methodology employed in this research integrates cutting-edge techniques, notably the use of the LLM model GPT-3.5 Turbo, which has been custom-trained to generate relevant code and diagrams based on user queries. This model also maintains a history of user interactions to refine responses and improve accuracy over time. By leveraging this advanced language model, the tool ensures precise translations and effective programming assistance, aligning with the unique linguistic and educational needs of Sinhala-speaking users and fostering a more inclusive and supportive learning environment. To offer a clear understanding of the system's operational dynamics, Fig. 1 illustrates the system architecture, highlighting the interactions between its three core components: the front-end user interface, the back-end APIs, and the Sinhala-English translators optimized for Java programming. Each component is meticulously designed to work in harmony, ensuring that users experience a seamless and intuitive interface while receiving accurate, context-specific programming support.

A. DBR Methodology

DBR is a methodology focused on enhancing educational practices through iterative design and development [2]. The process begins with a thorough analysis of specific problems or challenges within an educational context. In this study, this involves identifying the language barriers faced by Sinhala-speaking novice programmers and understanding the cognitive load associated with navigating English-based programming materials. Based on this analysis, a solution is designed and developed to address these issues. For this research, a Sinhala-based Java assistance tool is created, leveraging machine translation technologies to provide programming support in Sinhala. The tool undergoes iterative testing, where it is implemented in real-world settings and used by learners. Feedback from these testing phases is crucial for refining the tool and making necessary improvements. Finally, the effectiveness of the tool is evaluated to determine how well it meets its objectives, such as reducing language barriers and improving learning outcomes. DBR ensures that the tool is both theoretically grounded and practically effective, as it is continually adjusted based on real-world use and feedback, leading to a more robust solution that specifically addresses the needs of Sinhala-speaking novice programmers. Fig. 2 illustrates the iterative process of identifying challenges, designing solutions, testing prototypes, and refining tools based on user feedback to enhance educational practices.

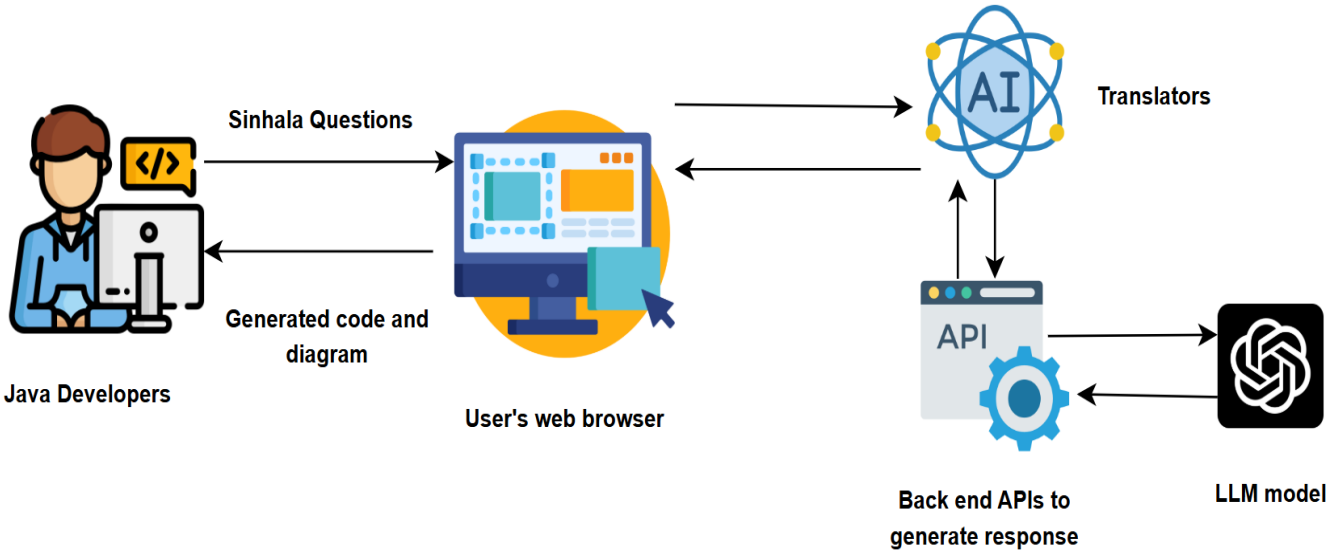

Fig. 1. System diagram of the proposed Sinhala-based programming assistance tool.

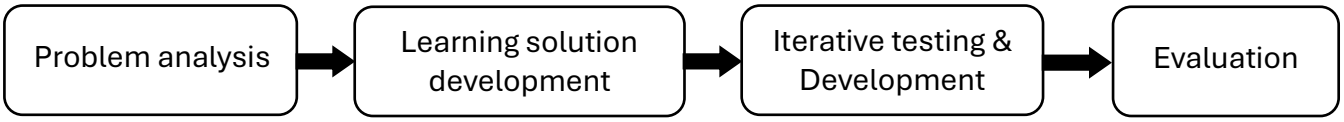

Fig. 2. Iterative stages of the DBR methodology.

B. Back-end API Development

The back-end API of the system serves dual functions. Initially, it processes incoming Sinhala queries by translating them into English using a transformer-based model. The translated queries are then passed to the LLM model (GPT-3.5 Turbo) with the aid of custom prompt templates and an output parser to generate the corresponding Java code.

Within the Flask framework, specialized libraries and parsers are employed to accurately distinguish Java code from text, supporting robust input validation and effective translations. The architecture of this system is illustrated in Fig. 1, which highlights the interactions between the front-end user interface, the back-end API, and the LLM model optimized for Java programming. This diagram underscores the coordinated workflow that enables the tool to deliver precise and contextually relevant programming support, bridging the language gap for Sinhala-speaking users.

C. Transformer-Based Sinhala-English Translator

At the core of the system lies the transformer-based translation model, meticulously trained for the specific task of translating Java programming content between Sinhala and English. This model has been fine-tuned to handle the unique linguistic challenges and technical terminology associated with Java programming, ensuring high accuracy and contextual relevance in translations.

The development of this transformer model was underpinned by a dataset uniquely tailored to the Java programming domain, which significantly enhances the model's ability to translate and interpret Java programming principles with precision. The choice of a transformer model was driven by its superior capability in managing long-range dependencies and parallel processing, making it the optimal solution for creating an effective and efficient translation system within the context of programming education.

Fig. 3 provides an overview of the entire model architecture, detailing the stages from input embedding and positional encoding through to the attention mechanisms and final training and inferencing phases. Fig. 4 delves deeper into the flow of data through the system's encoders and decoders, visually demonstrating the encoding and decoding processes that are pivotal to the model's functionality.

These processes involve transforming raw data into encoded formats suitable for specific tasks, such as transmission and storage, and then reversing this transformation to restore the data to its original form. The system's design, which includes multiple encoders and decoders, is specifically tailored to manage the complexities of Java programming translation, ensuring that the output is both technically accurate and contextually appropriate for educational purposes.

In this research, the transformer model plays a crucial role in the machine translation process, translating Java queries to English with a high degree of accuracy. The following steps detail how the transformer model was implemented and fine-tuned to maximize the accuracy and effectiveness of the Java-specific dataset used in this study:

- **Input Embedding:** The methodology begins with the critical process of input embedding, where the input text is tokenized into individual units called tokens. Each token is then transformed into a numerical array that encapsulates its semantic meaning. This transformation is essential for enabling the model to interpret and process the linguistic nuances of the Sinhala language, particularly in the context of Java programming. The accurate representation of these tokens in vector form ensures that the subsequent stages of the model can effectively capture and utilize the underlying semantics.

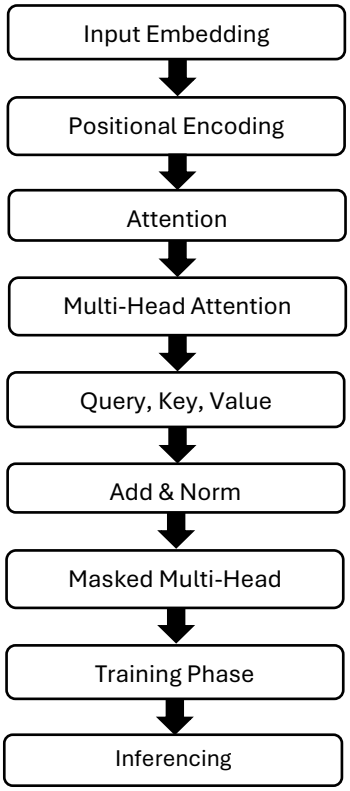

Fig. 3. Transformer model architecture.

- **Positional Encoding:** Unlike traditional models, transformers do not inherently recognize the order of tokens in a sequence. To address this, positional encoding is incorporated, which provides the model with essential information about the position of each token within the sequence. This step is crucial for maintaining the syntactic structure of the input, as it allows the model to understand and respect word ordering. For instance, when breaking down a sentence like "Write a Java class to check if a given number is prime," the positional encoding ensures that the sequence is interpreted correctly, preserving the logical flow necessary for accurate translation.
- **Attention Mechanism**: The attention mechanism is a key feature of the transformer model that enables it to selectively focus on different parts of an input sentence when generating output. By calculating attention scores, the model identifies how much importance it is to assign to each word in relation to others, effectively understanding their connections. This capability is especially useful for capturing relationships between words that are far apart in a sentence. For example, as shown in Fig. 5, the model assigns significant attention to the word "given" in the context of the entire sentence, ensuring the translation not only preserves the meaning but also aligns with the sentence structure and overall context, making it accurate and appropriate for the situation.
- **Multi-Head Attention:** To enhance the model's ability to understand complex linguistic structures, multi-head attention is employed. This technique involves running several attention mechanisms in parallel, each specializing in different aspects of the token relationships. By combining the outputs of these attention heads, the model generates a more comprehensive and multi-faceted representation of the input. This process is particularly effective in capturing the intricacies of Java programming terminology and syntax, ensuring that the translation is both precise and contextually relevant.

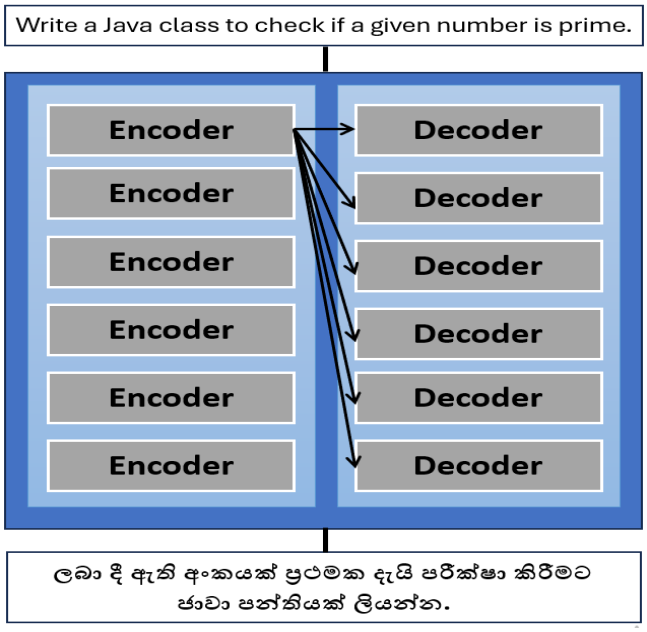

Fig. 4. Illustration of encoders and decoders in a transformer model, demonstrating the process with a Java prompt to check if a number is prime, accompanied by explanatory text in Sinhala.

- **Query (Q), Key (K), Value (V):** Within the attention mechanism, each token is categorized into three vectors: Q, K, and V. These vectors are used to calculate attention scores, which determine the emphasis that should be placed on each token during translation. For example, when translating the English phrase "Write a Java class to check if a given number is prime" into Sinhala, the model utilizes these vectors to accurately map the relationships between words, ensuring that the most contextually significant terms are appropriately weighted in the translated sentence.
- **Add & Norm:** Following the attention mechanism, the model applies a residual connection, adding the original input to the output of the attention layer. This step is followed by normalization, which stabilizes the training process and preserves the integrity of the input information. By maintaining a balance between the original input and the processed output, this technique ensures that the model continues to generate accurate and stable translations throughout the training and inference phases.
- **Masked Multi-Head Attention:** In the decoding phase, masked multi-head attention is used to prevent the model from attending to future tokens during training. This technique ensures that the model's predictions for a given position are based solely on the known outputs up to that point, thereby enhancing the accuracy and reliability of the translation process.
- **Training Phase:** The training phase is a critical component of the methodology, where the model is fed with pairs of Sinhala Java explanations and their English counterparts. The objective during training is to minimize the difference between the predicted

translations and the actual translations, achieved through a loss function and the optimization of the model's parameters via backpropagation. This phase is iterative, allowing the model to gradually improve its translation accuracy and contextual understanding.

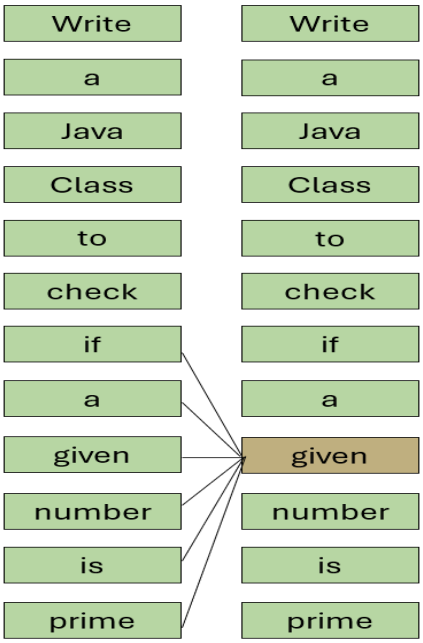

Fig. 5. Illustration of the attention mechanism in a Transformer model, emphasizing its ability to capture long-range dependencies and selectively focus on various segments of an input sequence.

- **Inferencing:** In the inferencing stage, the trained model is applied to new input sequences, translating them with high accuracy. When presented with a Sinhala explanation, the model generates the corresponding English Java explanation, enabling Sinhala-speaking students to engage with Java programming in a language they understand. This inclusive approach ensures that the educational tool meets its primary objective of bridging the language gap in programming instruction.
- **Final Model Architecture:** Fig. 6 showcases the complete transformer model architecture, illustrating the flow from input embeddings through multi-head attention, feed-forward networks, and positional encoding layers, culminating in the generation of output probabilities via a Softmax function. This architecture is specifically designed to handle the complexities of language translation within the Java programming domain, ensuring that the output is both contextually appropriate and semantically precise.
- **Closing the Language Gap:** By leveraging cutting-edge AI and transformer technology, this research provides Sinhala-speaking novice programmers with real-time, contextually accurate translations and programming diagrams. The integration of these advanced methodologies ensures that the tool not only addresses the immediate language barrier but also contributes to a more inclusive and equitable learning environment in the field of computer science.

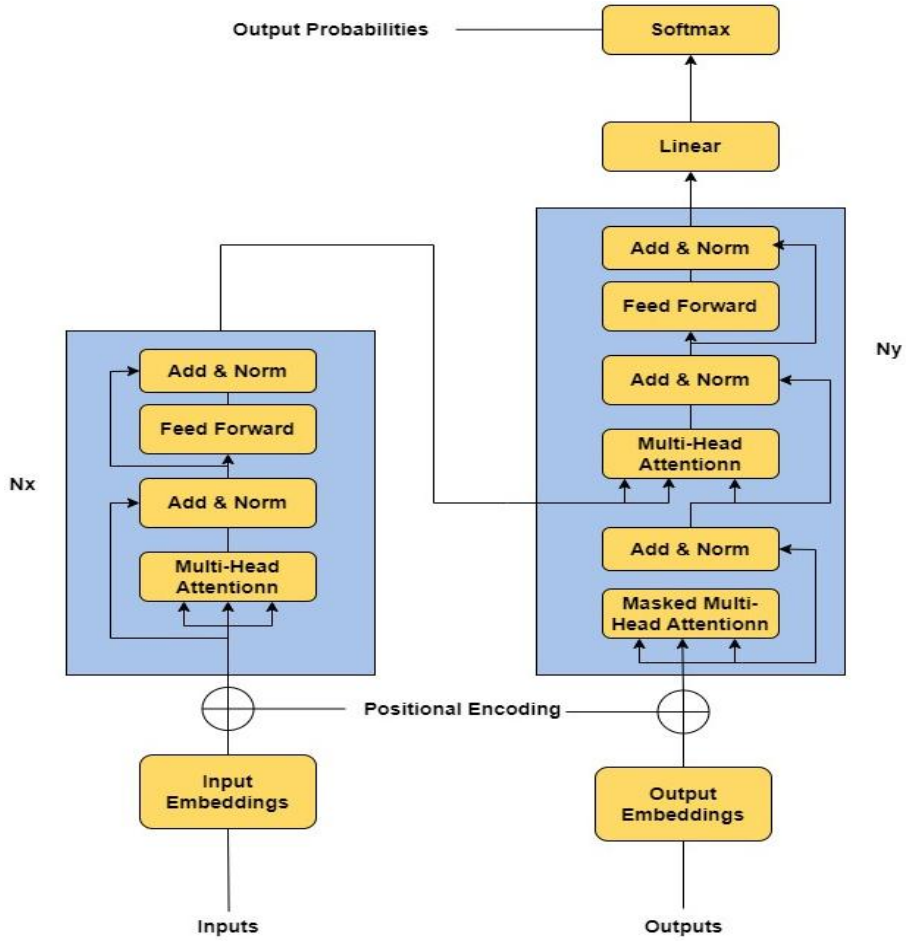

Fig. 6. Diagram of a Transformer model, showcasing the progression from input embeddings through multi-head attention, feed-forward networks, and positional encoding, culminating in output probabilities via a SoftMax function.

D. Front End Development

The front-end of the system serves as the primary user interface, designed to facilitate a seamless interaction experience for novice programmers. Built using the Flask framework, the front-end prioritizes ease of use, ensuring that users can effortlessly submit their programming queries or Java code snippets. The interface processes user inputs and transmits them via HTTP requests to the back-end API, which then returns the relevant output—either Java code generated from Sinhala queries and diagrams of Java code translated into Sinhala.

The user interface is intentionally designed to reduce cognitive load and language barriers by providing direct, contextually appropriate code outputs. As depicted in Fig. 7, the interface of the Sinhala-based Java assistance tool enables users to input queries in Sinhala and receive real-time Java code along with corresponding visual diagrams. This feature allows learners to instantly visualize the structure and logic of their code, reinforcing conceptual understanding. The tool automatically converts the generated code into Mermaid diagram code, which is then rendered as visual diagrams on the front end. This user-centric design ensures that the interface not only meets functional requirements but also enhances the overall learning experience by catering to the specific linguistic needs of Sinhala-speaking users.

To further demonstrate the tool's versatility and its applicability to advanced programming tasks, the following examples highlight its ability to generate code for data structure operations, recursive functions, and object-oriented design patterns.

Fig. 8 illustrates the tool's capability to find the second largest element in an array by sorting the array and comparing elements in reverse order. The operation emphasizes the importance of array manipulation in data structures, showcasing the tool's effectiveness in handling tasks related to sorting and array traversal. The approach ensures the identification of the second largest element, demonstrating the tool's practical use in real-world programming scenarios.

Fig. 9 showcases how the tool generates Java code for recursive functions based on user input. Recursion is a fundamental yet challenging programming concept, often used for problems like factorial calculation, Fibonacci sequence generation, and tree traversals. By successfully generating recursive solutions, the tool demonstrates its capability to handle complex algorithmic problems.

Fig. 10 showcases the tool generating Java code for the Singleton design pattern. Design patterns are critical in software development, providing reusable solutions to common problems. The Singleton pattern ensures a class has only one instance and provides a global access point to it. Successfully generating this pattern demonstrates the tool's ability to support OOP principles and best practices.

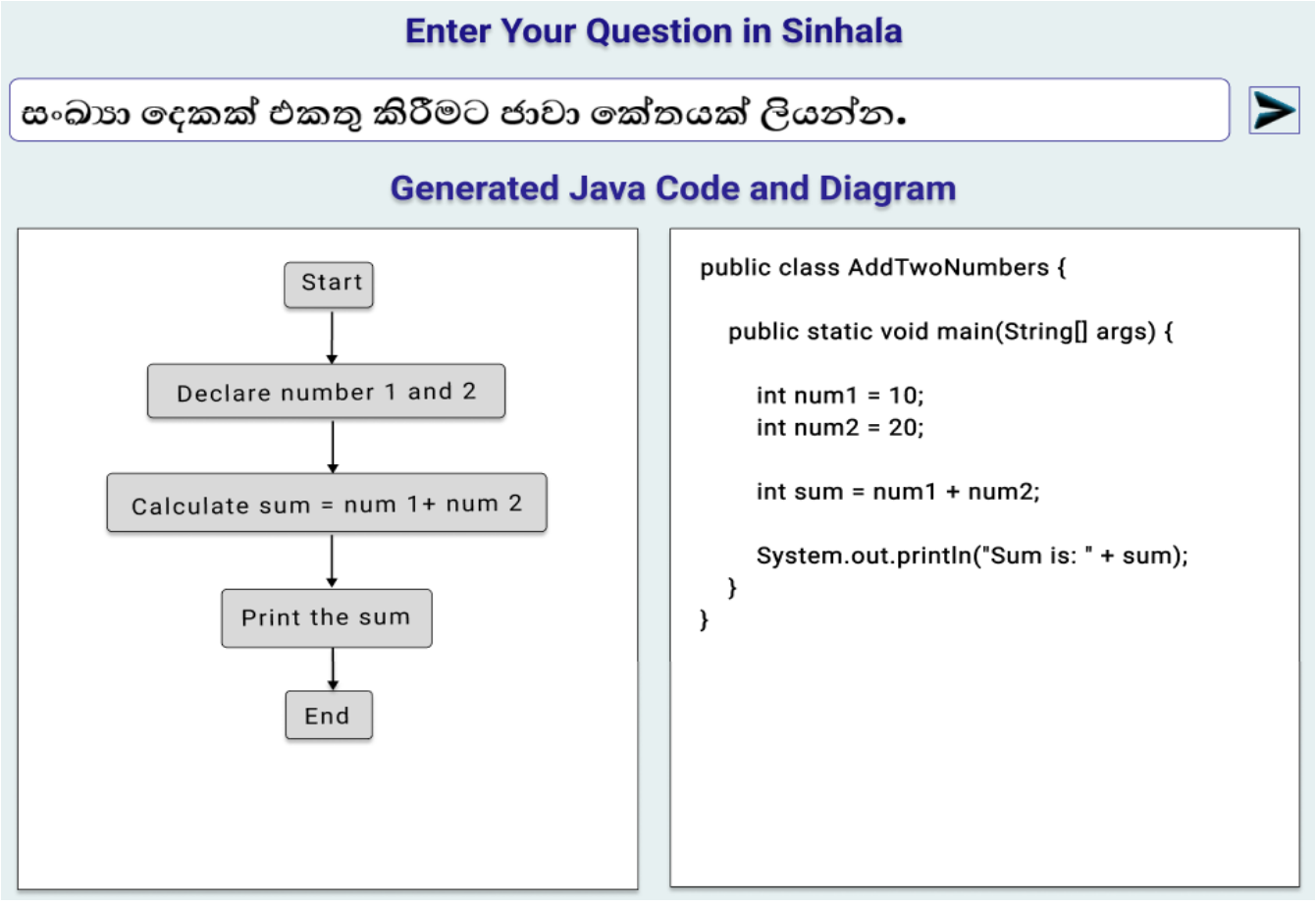

Fig. 7. User interface of the Sinhala-based Java assistance tool enabling real-time code and diagram generation.

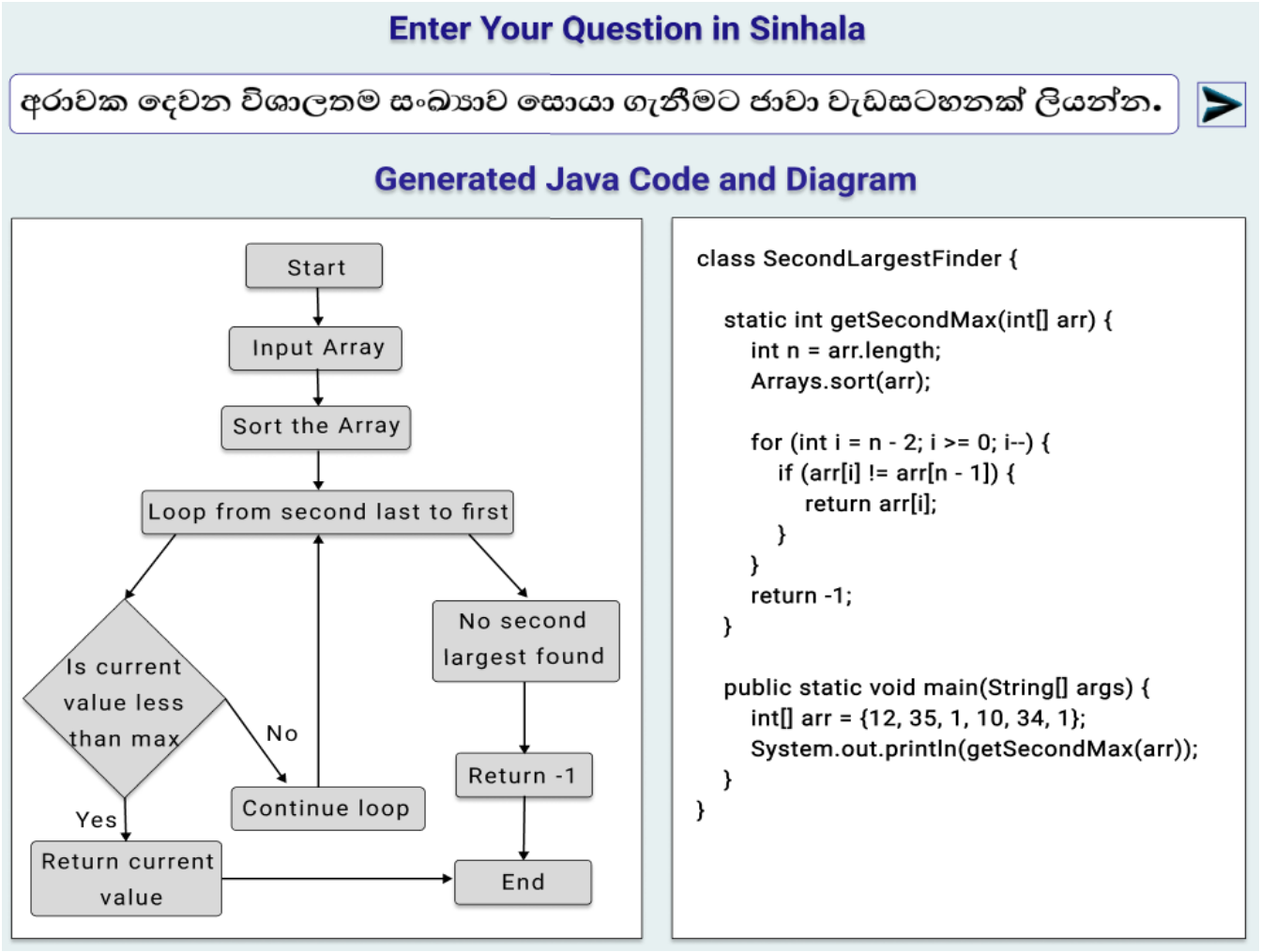

Fig. 8. User interface displaying a Java code to find the second-largest element in an array, generated based on a Sinhala-language user query.

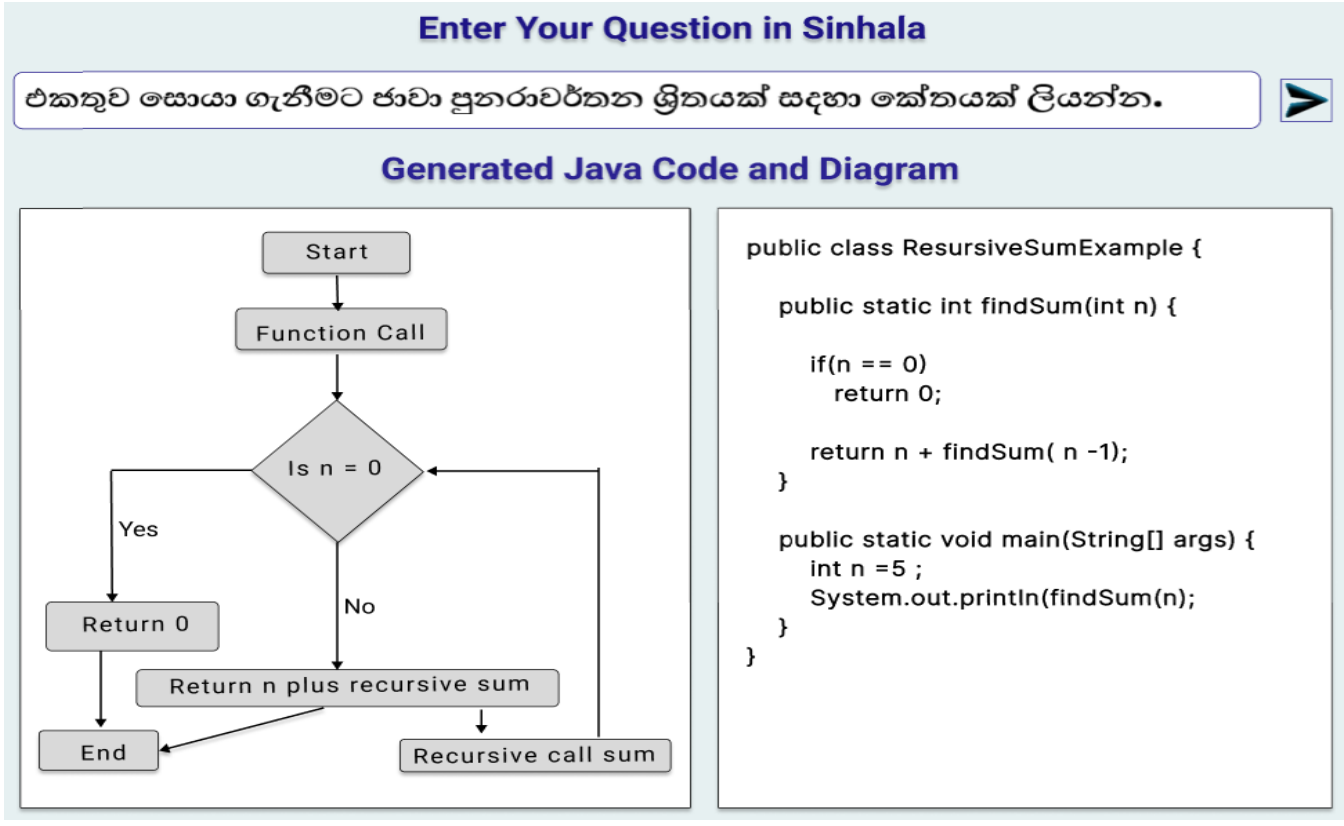

Fig. 9. User interface displaying a Java code for a recursive function, generated based on a Sinhala-language user query.

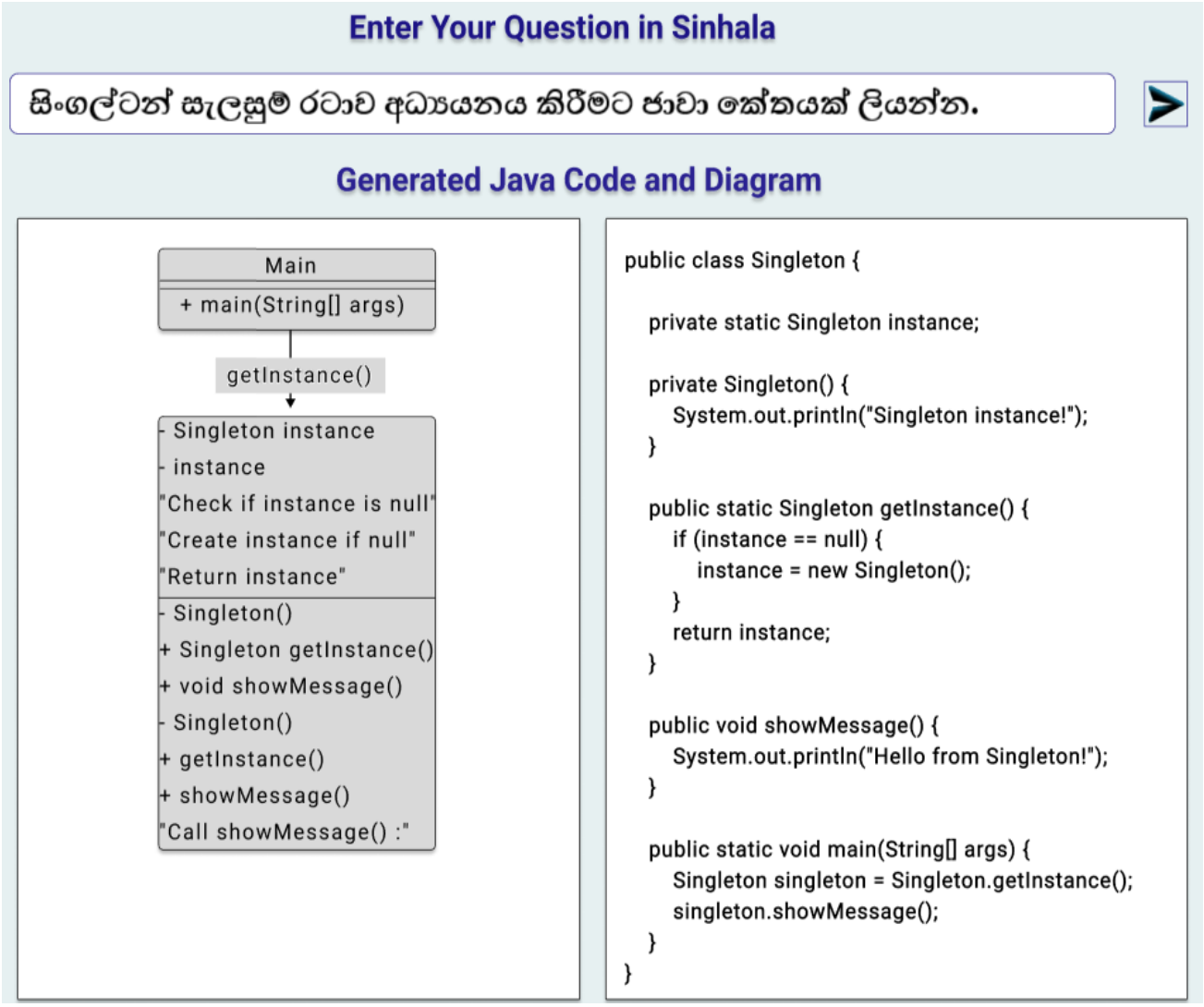

Fig. 10. User interface displaying a Java code to understand the Singleton design pattern, generated based on a Sinhala-language user query.

E. User Testing and Quality Assurance Integration

To ensure the robustness, reliability, and user-friendliness of the proposed tool, a comprehensive user testing and quality assurance protocol was meticulously designed and implemented. This protocol was critical in validating that each system component—the front-end interface, the back-end API, and the custom transformer-based translation model—performed optimally and met predefined standards for functionality, accuracy, and user satisfaction.

The testing process began with functional tests, aimed at verifying the technical performance of each system component. These tests assessed the translation accuracy of the Java-specific Sinhala-English model, the responsiveness and intuitiveness of the front-end interface, and the robustness of the back-end API in processing and managing user queries and code snippets. Ensuring that each component was individually effective was essential in achieving a cohesive and reliable overall user experience.

Following the functional testing phase, a series of usability tests were conducted with a targeted group of 122 novice programmers who were fluent in Sinhala. These participants were asked to engage with the tool by inputting Java code and queries in Sinhala, allowing the development team to assess the tool's ease of use, intuitiveness, and effectiveness in assisting programming tasks. The feedback gathered from these sessions was invaluable, highlighting areas for improvement in user interaction and satisfaction. The usability testing also provided insights into how the tool could be optimized to better meet the needs of its target audience.

Quality assurance was integrated into the development process through continuous iterative cycles. These cycles involved refining the tool's functionality, enhancing the user interface, and improving translation accuracy based on real-time feedback. This iterative approach ensured that the tool evolved in response to user needs and external changes, such as updates to services like ChatGPT or modifications in the Java programming environment. The iterative testing and refinement process played a crucial role in maintaining the tool's relevance, reliability, and effectiveness, ensuring that it could be seamlessly deployed in a variety of educational contexts.

User feedback was a driving force behind the iterative development of the tool. Adjustments such as optimizing translation accuracy, refining code generation algorithms, and improving the user interface were directly influenced by the experiences and suggestions of the target user base. This ongoing process of enhancement ensured that the tool

remained attuned to the needs of Sinhala-speaking novice programmers, delivering a high-quality, user-friendly learning experience. The rigorous approach to user testing and quality assurance not only confirmed the tool's readiness for broader deployment but also underscored its potential to significantly enhance the programming education landscape for non-English speaking learners.

V. Results and Discussion

A. DBR Methodology in Tool Development

The development of the Sinhala-based Java assistance tool followed the DBR methodology, which emphasizes iterative design, collaboration, and real-world testing to address practical challenges in educational contexts. This approach ensured that the tool effectively mitigated language barriers and cognitive load experienced by Sinhala-speaking novice programmers. The DBR methodology was implemented through the following four key phases, which guided the tool's design, development, and refinement:

- **Phase 1: Identifying Practical Problems:** The initial phase focused on identifying the challenges faced by Sinhala-speaking novice programmers in understanding English-based programming resources. Authors collaborated with practitioners, including educators and novice programmers, to identify specific obstacles, such as language barriers and cognitive load associated with translating and interpreting Java programming content. Through this process, the problem was clearly defined, laying the foundation for the development of the Sinhala-based Java assistance tool.
- **Phase 2: Analyzing Data and Developing Solutions:** Once the problem was identified, researchers analyzed learner experiences and existing design principles to develop a suitable solution. Data was collected from real-world settings to understand the difficulties Sinhala-speaking learners face while interacting with Java programming materials. Leveraging insights from machine translation technologies and educational tools, the team designed a Sinhala-based Java assistance tool that provides real-time programming support in the native language. The tool's design prioritized ease of use and effectiveness in improving learning outcomes.
- **Phase 3: Testing and Refining the Solution:** The tool underwent usability testing with 122 Sinhala-speaking novice programmers, who interacted with it by entering Java code and Sinhala queries. User feedback played a crucial role in refining interface design, translation accuracy, and interaction mechanisms. Several iterations were conducted to enhance functionality, ensuring that the tool effectively supports programming tasks and meets user needs.
- **Phase 4: Evaluating the Effectiveness of the Solution:** Following iterative testing and refinement, the final phase involved evaluating the tool's overall effectiveness. The analysis focused on user feedback, usability, and the tool's impact on reducing language barriers and cognitive load. Results demonstrated that the tool successfully addressed the identified challenges and significantly improved programming comprehension for Sinhala-speaking learners. Additionally, areas for further development and optimization were identified, ensuring continuous improvement. While the solution was deemed effective, ongoing user feedback and refinements will be crucial for maintaining long-term success and adaptability.

B. Training, Validation, and Testing Data

Table 2 presents a detailed breakdown of the dataset used for training, validation, and testing the transformer-based model. Comprising 223,334 sentences, the dataset ensures that the model is exposed to a wide range of linguistic structures and contexts, strengthening its ability to handle diverse translation tasks. The careful allocation of data across these phases supports the model's generalization capability, enabling accurate performance in real-world scenarios.

Table 2. Breakdown of sentences across the training, validation, and testing datasets

Dataset Type	Sentence Count
Training set size	156,334
Validation set size	33,500
Testing set size	33,500
Total Sentences	**223,334**

C. Impact of Sentence Structure on Translation Quality

The effectiveness of the dataset distribution is evaluated through BLEU scores, a key metric for assessing machine translation quality. Sentence length and complexity significantly influence BLEU scores, as observed in model testing.

- Shorter sentences with common structures tend to achieve higher BLEU scores, as they pose less linguistic ambiguity and are easier to translate.
- Longer and structurally complex sentences, particularly those in the Sinhala dataset, present greater challenges, often resulting in lower BLEU scores.

This analysis underscores the importance of dataset composition in translation accuracy.

D. Model Performance and Accuracy

The transformer-based model developed in this study achieved an overall accuracy of 91.37%, demonstrating its effectiveness in translating Java programming content between Sinhala and English. The BLEU scores indicate that the model produces translations that closely align with reference outputs, ensuring both accuracy and fluency.

Given the linguistic differences between Sinhala and English, as well as the technical nature of programming content, this performance is significant. The model successfully adapts to complex sentence structures, maintaining syntactic accuracy and semantic coherence in Java-related translations. These results highlight the potential of transformer-based models in mitigating language barriers for Sinhala-speaking programmers.

E. Dataset Distribution Analysis

Fig. 11 provides a visual representation of the dataset distribution, offering insights into the sentence structures in both Sinhala and English. The dataset was analyzed using two key visualization techniques:

- Box Plot: Illustrates variation in sentence lengths and highlights differences between Sinhala and English datasets.
- Histogram: Displays the frequency distribution of sentence lengths, identifying common patterns and anomalies in dataset composition.

The analysis reveals that most sentences in both languages fall within the 5 to 10-word range, ensuring consistency in linguistic complexity across training, validation, and testing sets. However, the presence of outliers, particularly longer Sinhala sentences, suggests that translation quality may be affected by complex syntactic structures.

This dataset composition analysis is critical for refining translation models, as it ensures balanced training data that can enhance the model's ability to handle both common and complex sentence structures effectively. Additionally, a statistical evaluation was conducted to measure the tool's effectiveness in bridging the language barrier and improving Java programming comprehension. The evaluation also included a pre- and post-tool user performance comparison, validating its impact on learning outcomes.

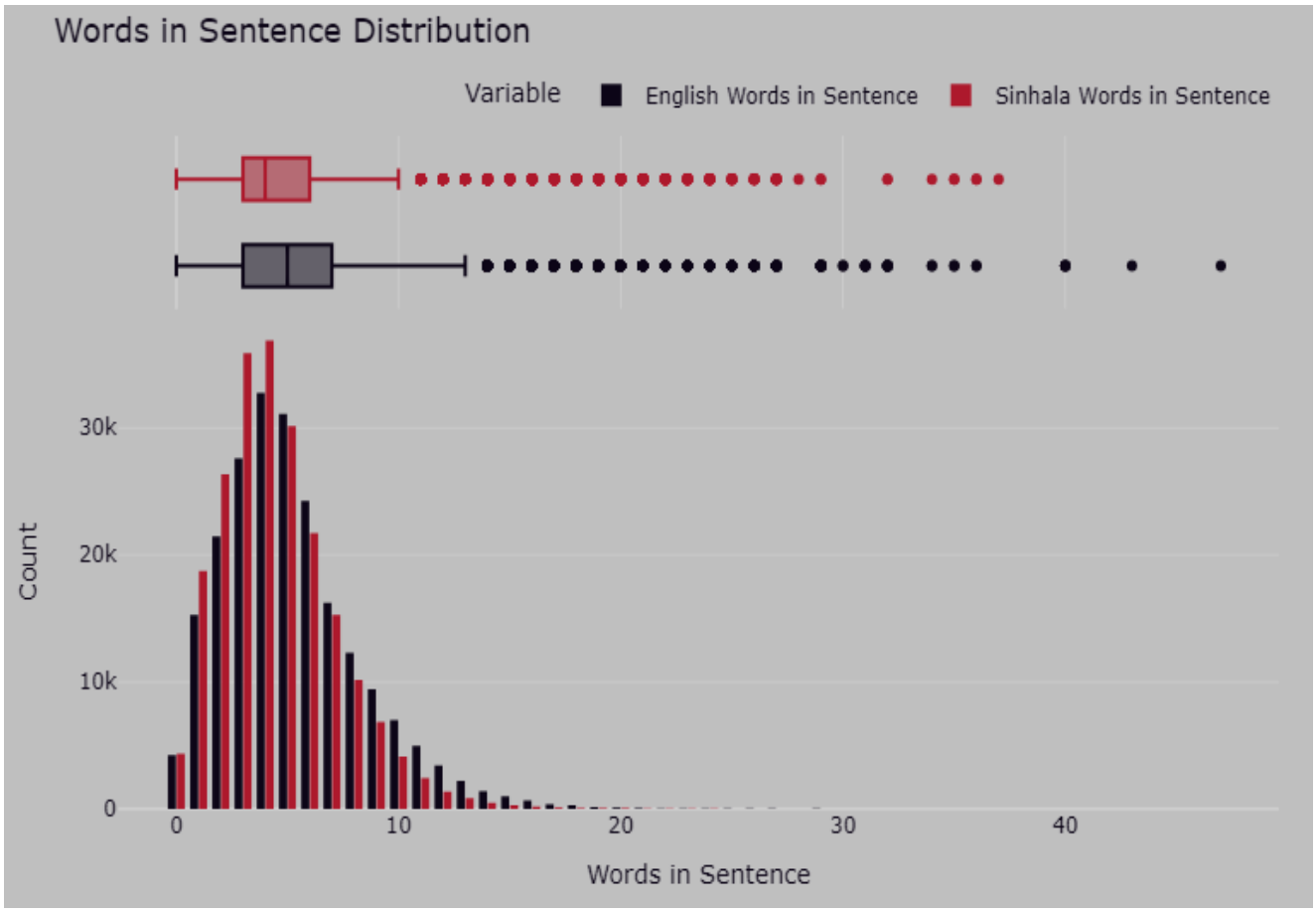

Fig. 11. Distribution of words per sentence in English and Sinhala.

F. Usability Test Evaluation Methodology

- **Participant Selection:** To ensure a diverse representation of students with varying educational backgrounds, participants were recruited from two distinct categories of universities in Sri Lanka that offer University Grants Commission (UGC)-affiliated IT degrees:
  - UGC-affiliated government universities
  - UGC-affiliated private universities

An initial target of at least four students per institution was set, covering 25 universities across these two categories, with the aim of recruiting 100 participants. However, 122 responses were successfully collected, exceeding the anticipated sample size.

Participants were further categorized into three groups based on their programming proficiency:

  - Group A (Minimal programming experience): First and second semester students with limited prior exposure to programming.
  - Group B (Moderate programming experience): Third to fifth semester students with intermediate programming knowledge.
  - Group C (Advanced programming experience): Sixth semester and beyond students with substantial programming experience.
- **Testing Procedure:** The evaluation was conducted in three phases to systematically assess the effectiveness of the tool:
  - Pre-Test: Participants were assigned a set of Java programming tasks in English to establish their baseline understanding and performance. These tasks included writing Java code snippets, interpreting existing code, and solving fundamental programming problems.

- Tool Interaction: Participants were introduced to the tool and given two weeks to explore its functionalities. During this period, they utilized the tool to translate Sinhala programming queries into Java code and visualize corresponding diagrams.
- Post-Test: After the tool interaction phase, participants completed a new but comparable set of Java programming tasks in Sinhala, requiring them to use the tool. Performance was assessed based on three key metrics: task accuracy (percentage of correctly solved problems), completion time (time taken to complete assigned tasks), and perceived difficulty (self-reported ease of use).

- **Data Collection:** To evaluate user performance and tool usability, the following quantitative data points were collected:
  - Task accuracy
  - Average task completion time
  - Self-reported ease of use (measured using a 5-point Likert scale)
- **Statistical Analysis:** To assess the significance of performance improvements, paired t-tests were conducted, comparing pre-test and post-test scores across participant groups. The statistical findings, detailed in Tables 3–5, highlight the tool's impact on task accuracy, completion time, and user experience.

Table 3. Task accuracy metrics before and after using the proposed tool across participant categories

Metric	Pre-Test	Post-Test	Progress
Task Accuracy (Students belonging to UGC-affiliated government universities in Group A)	59.26%	77.78%	18.52%
Task Accuracy (Students belonging to UGC-affiliated private universities in Group A)	55.17%	79.31%	24.14%
Task Accuracy (Students belonging to UGC-affiliated government universities in Group B)	66.67%	83.33%	16.67%
Task Accuracy (Students belonging to UGC-affiliated private universities in Group B)	76.19%	80.95%	4.76%
Task Accuracy (Students belonging to UGC-affiliated government universities in Group C)	78.95%	89.47%	10.53%
Task Accuracy (Students belonging to UGC-affiliated private universities in Group C)	76.92%	84.62%	7..69%

Table 4. Task completion time metrics before and after using the proposed tool across participant categories

Metric	Pre-Test (Minutes)	Post-Test (Minutes)	Progress (Minutes)
Task Completion Time (Students belonging to UGC-affiliated government universities in Group A)	40	32	8
Task Completion Time (Students belonging to UGC-affiliated private universities in Group A)	51	38	13
Task Completion Time (Students belonging to UGC-affiliated government universities in Group B)	28	22	6
Task Completion Time (Students belonging to UGC-affiliated private universities in Group B)	32	28	4
Task Completion Time (Students belonging to UGC-affiliated government universities in Group C)	25	20	5
Task Completion Time (Students belonging to UGC-affiliated private universities in Group C)	23	19	4

Table 5. User experience metrics before and after using the proposed tool across participant categories

Metric	Pre-Test	Post-Test	Progress
User Experience (Students belonging to UGC-affiliated government universities in Group A)	3	4	1
User Experience (Students belonging to UGC-affiliated private universities in Group A)	2	4	2
User Experience (Students belonging to UGC-affiliated government universities in Group B)	3	4	1
User Experience (Students belonging to UGC-affiliated private universities in Group B)	2	3	1
User Experience (Students belonging to UGC-affiliated government universities in Group C)	3	5	2
User Experience (Students belonging to UGC-affiliated private universities in Group C)	4	5	1

G. Connecting BLEU Scores to Translation Performance and User Improvement

The BLEU score, a widely recognized metric for evaluating translation quality, was utilized to measure the effectiveness of Sinhala-to-Java translations produced by the tool.

The analysis revealed a strong correlation between higher BLEU scores and improved student performance, indicating that superior translation quality directly enhances programming comprehension. Specifically, translations with higher BLEU scores resulted in fewer syntactic and semantic errors, enabling students to understand Java programming concepts more effectively.

By incorporating BLEU score-based evaluation, this study establishes a robust relationship between translation accuracy and learning outcomes, further validating the tool's practical effectiveness in improving programming education for Sinhala-speaking learners. The results underscore the importance of high-quality translations in reducing cognitive load, facilitating knowledge retention, and improving students' ability to write Java code accurately. To contextualize these findings, the following section compares BLEU scores from related studies and highlights how the proposed model outperforms existing approaches.

H. BLEU Score Comparison Across English–Sinhala NMT Models

To contextualize the performance of the proposed transformer-based model, this section compares its BLEU score with those reported in previous English–Sinhala NMT studies. These comparisons highlight advancements in dataset size, model design, and translation accuracy over time.

Sen *et al.* [26] introduced two multilingual NMT models based on the Transformer architecture—one for English-to-Indic language translation and another for the reverse direction. Sinhala was among the supported languages, and their English-to-Sinhala bilingual baseline model achieved a BLEU score of 12.75.

Expanding on this foundation, Guzman *et al.* [27] developed an NMT system trained on open-domain datasets using both supervised and semi-supervised methods. However, due to a domain mismatch between the training and test data, the English-to-Sinhala translation yielded a significantly lower BLEU score of just 1.2, revealing the sensitivity of translation accuracy to domain-specific data.

To address this challenge, Nguyen *et al.* [28] employed a data diversification strategy using backward and forward peer models to enrich the dataset. This method improved the BLEU score to 2.2, surpassing the result reported by Guzman *et al*., yet still reflecting limitations due to dataset quality and volume.

A notable improvement was achieved by Fonseka *et al.* [29], who trained an English-to-Sinhala NMT model using BPE and a Transformer architecture on a closed-domain dataset comprising 18,000 official Sri Lankan government document pairs. This model attained a BLEU score of 28.28, marking a significant step forward in domain-specific translation quality.

Building upon these efforts, Naranpanawa *et al.* [30] explored the impact of various sub-word segmentation techniques, including BPE, Unigram Language Model, and Character Segmentation. Their best-performing model, trained on a dataset of 54,000 sentences, achieved a BLEU score of 29.92 using BPE, demonstrating the effectiveness of sub-word techniques in handling morphologically rich languages like Sinhala.

While these studies have contributed meaningfully to the development of English–Sinhala NMT, they are often constrained by limited training data and scope. In contrast, the proposed model in this study achieves a BLEU score of 33.17—substantially higher than previously reported results.

This improvement is attributed primarily to the significantly larger dataset of 156,334 training sentences, complemented by balanced validation and test sets of 33,500 sentences each. The larger training corpus enables better learning of grammar patterns, contextual alignments, and vocabulary distributions, while the expansive test set enhances the statistical reliability of performance evaluation. The use of BPE, coupled with task-specific tuning and transformer-based architecture, further contributes to the model's high accuracy and fluency in translation.

Table 6 summarizes the BLEU scores and methodologies of the reviewed English–Sinhala NMT models for quick reference and comparison.

This comparative analysis underscores the effectiveness of the proposed model in delivering superior translation quality for English–Sinhala NMT tasks. By overcoming limitations in dataset size and leveraging advanced transformer techniques, the model sets a new benchmark in BLEU score performance for this language pair.

Table 6. Comparison of BLEU Scores for English–Sinhala NMT Models

Model	BLEU Score	Methodology
Sen *et al.'s* Model [26]	12.75	Multilingual NMT using Transformer
Guzman *et al.'s* Model [27]	1.2	Open-domain NMT, supervised training
Nguyen *et al.'s* Model [28]	2.2	Data diversification with peer models
Fonseka *et al.'s* Model [29]	28.28	Transformer with BPE
Naranpanawa *et al.'s* Model [30]	29.92	Transformer with various sub-word techniques
Proposed Model	33.17	Transformer with BPE and enhanced training on a significantly larger dataset (156,334 sentences)

I. Significance and Practical Applications

This research makes a significant contribution to programming education by addressing the unique challenges faced by Sinhala-speaking novice programmers. By developing a transformer-based Sinhala-to-Java programming assistance tool, this study bridges a critical gap in educational resources for non-English-speaking learners.

Beyond resolving immediate linguistic challenges, this tool establishes a scalable model that can be adapted for other linguistic communities worldwide, breaking language

barriers in programming education and fostering a more inclusive and equitable learning experience.

By enabling the generation of Java code and visual representations in Sinhala, the tool democratizes programming education, ensuring that non-English-speaking learners have access to high-quality learning resources. This innovative approach has the potential to transform programming education for underrepresented linguistic groups, making technical knowledge more accessible and empowering.

J. Key Finding and Pedagogical Benifits

- **Enhanced Learning and Code Quality:**
  - Students demonstrated substantial improvements in writing syntactically correct Java code.
  - The tool contributed to a reduction in common syntax errors, leading to cleaner and more efficient coding practices.
- **Improved Programming Comprehension:**
  - By allowing students to articulate problems in Sinhala, the tool enhanced their understanding of Java programming concepts.
  - Code structure visualization and logic diagrams reinforced conceptual clarity, resulting in better learning outcomes.
- **Reduction in Cognitive Load:**
  - Eliminating the language barrier enabled students to focus on programming logic, rather than struggling with English-based syntax and terminology.
  - This was particularly impactful for first and second-semester students, who had limited exposure to English-based programming resources.
- **Increased Engagement and Confidence:**
  - The tool significantly boosted student motivation, with post-test surveys reporting higher confidence levels.
  - A structured two-week interaction period allowed students to gain familiarity with the tool, fostering greater competence in Java programming.

K. Technical Soundness and Reliability

The technical robustness of the research is demonstrated through the meticulous development and seamless integration of a custom transformer-based translation model. Specifically trained on a curated dataset tailored for Java programming, the model ensures translations that are both accurate and contextually relevant.

The system architecture, encompassing a front-end interface, back-end API, and integration with ChatGPT, underwent rigorous testing to guarantee optimal performance. Comprehensive user testing and quality assurance protocols further validate the tool's reliability and effectiveness, positioning it as a valuable resource for both educational institutions and professional programming environments.

VI. Conclusion and Future Work

A. Conclusion

This research has successfully developed and validated a transformer-based programming assistance tool designed to overcome language barriers faced by Sinhala-speaking novice Java programmers. With a translation accuracy of 91.37% and strong BLEU scores, the tool demonstrates its ability to generate precise and contextually relevant Java code from Sinhala queries. The integration of GPT-3.5 Turbo for code generation, combined with a transformer-based translation model, has proven to be an effective solution for bridging linguistic gaps in programming education.

Beyond its immediate educational benefits, this tool enhances inclusivity and accessibility by enabling real-time code and diagram generation. By fostering a more equitable learning experience, it democratizes access to technology education, particularly in regions where English proficiency poses a barrier. Its technical robustness and reliability have been validated through rigorous testing and iterative refinements, making it a valuable resource for both academic institutions and industry professionals. The tool's applications extend beyond classrooms, benefiting self-learners, educators, and software developers.

Furthermore, this research establishes a scalable framework that can be adapted to support other non-English-speaking communities, setting a precedent for the development of similar educational technologies. By fostering inclusivity and expanding access to programming knowledge, this study contributes to a more diverse and innovative technology landscape.

While the tool significantly enhances accessibility in programming education, several limitations must be addressed in future iterations. A key area for improvement is expanding the dataset to further improve translation accuracy and model robustness. Additionally, increasing the response count during testing is essential to validate the tool's performance across a broader range of scenarios and use cases, ensuring its effectiveness in diverse real-world applications.

B. Future Work

The future directions for this research offer significant advancements and broader applications beyond its current scope. The following key areas highlight promising avenues for further development:

1) Enhanced language support

Expanding the tool's language capabilities to include additional languages such as Tamil, Arabic, and Hindi would make programming education accessible to a broader demographic. By offering multilingual support, the tool could effectively bridge the language barrier in programming education, particularly in regions where English is not the first language. This enhancement would foster greater inclusivity and increase global adoption.

2) Integration with specialized models

Instead of relying solely on ChatGPT, integrating multiple specialized AI models could enhance performance. Incorporating models for code completion, error detection, and algorithm optimization would provide task-specific support and improve accuracy. This approach would create a more robust system capable of addressing diverse programming needs and styles.

3) Broader Programming Language and Domain Support

Adapting the tool to support other programming languages (e.g., Python, C++, JavaScript) would increase its relevance to a wider audience. Additionally, extending its capabilities to domains like web development, mobile app development, and machine learning would enhance its versatility and scalability, making it a comprehensive resource for learners across various fields.

4) Integration with development tools and platforms

Connecting the tool with popular Integrated Development Environments (IDEs) such as IntelliJ IDEA, Eclipse, and Visual Studio Code, as well as version control systems like Git, could streamline programming workflows. Additionally, integration with online learning platforms would further enhance its educational value, providing real-time feedback, personalized recommendations, and a seamless learning experience.

5) Improved usability and user experience

Enhancing the tool's user interface and incorporating personalized feedback based on user progress would improve accessibility and engagement. An intuitive design with targeted guidance would empower learners to better understand their mistakes, boost confidence, and accelerate skill development.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Kalpani Sachie Nelanka Athukorala was responsible for developing the Sinhala-to-English translation model and designing the overall structure of the Sinhala-based Java programming assistance tool. Dilshan Indraraj De Silva integrated the translation model with the Java programming assistant and conducted tool evaluations to enhance its usability for students. Both authors contributed to the writing and critical revision of the manuscript and approved the final version for submission.

REFERENCES

[1] R. Yasasri and D. Karunarathna, "Helaa: A Sinhala language-based programming," *in Proc. 4th International Conference on Innovations in Info-business & Technology*, Colombo, Sri Lanka, Aug. 2023, pp. 1–11.

[2] F. Wang and M. Hannafin, "Design-based research and technology-enhanced learning environments," *Educational Technology Research & Development*, vol. 53, no. 4, pp. 5–23, 2005.

[3] Editors of Encyclopaedia Britannica (2012). Sinhalese language. *Britannica*. [Online]. Available: https://www.britannica.com/topic/Sinhalese-language

[4] R. Nordquist. (2024). English language: History, definition, and examples. ThoughtCo. [Online]. Available: https://www.thoughtco.com/what-is-the-english-language-1690652

[5] R. Kulshrestha. (2020). Transformers. *Medium: Towards Data Science*. [Online]. Available: https://towardsdatascience.com/transformers-89034557de14

[6] Z. Xu and E. Martin. (2024). From RNNs to Transformers. [Online]. Available: https://www.baeldung.com/cs/rnns-transformers-nlp

[7] M. Amaratunga, G. Wickramasinghe, M. Deepal, O. Perera, D. De Silva, and S. Rajapakse, "An Interactive Programming Assistance tool (iPAT) for instructors and novice programmers," in *Proc. 8th International Conference on Computer Science & Education*, Colombo, Sri Lanka, 2013, pp. 680–684.

[8] S. J. Whittall, W. A. C. Prashandi, G. L. S. Himasha, D. I. De Silva, and T. K. Suriyawansa, "CodeMage: Educational programming environment for beginners," in *Proc. 9th International Conference on Knowledge and Smart Technology*, Chonburi, Thailand, 2017, pp. 311–316.

[9] P. Perera and S. Ahangama, "SimplyTrans: A simplified approach to sinhala-based coding and introductory programming language localization," in *Proc. 16th International Conference on Industrial and Information Systems*, Kandy, Sri Lanka, 2021, pp. 318–323. doi: 10.1109/ICIIS53135.2021.9660709

[10] A. More, J. Kumar, and R. V. G. "Web based programming assistance tool for novices," in *Proc. International Conference on Technology for Education*, Chennai, India, 2011, pp. 270–273.

[11] D. De Silva, A. Alahakoon, I. Udayangani, V. Kumara, D. Kolonnage, H. Perera, and S. Thelijjagoda, "Sinhala to English language translator," in *Proc. International Conference on Information and Automation for Sustainability*, Colombo, Sri Lanka, 2008, pp. 419–424.

[12] L. Wijerathna, W. L. S. L. Somaweera, S. L. Kaduruwana, Y. V. Wijesinghe, D. I. De Silva, K. Pulasinghe, and S. Thellijjagoda, "A translator from Sinhala to English and English to Sinhala (SEES)," in *Proc. International Conference on Advances in ICT for Emerging Regions,* Colombo, Sri Lanka, 2012, pp. 14–18.

[13] A. Kugathasan and S. Sumathipala, "Neural machine translation for Sinhala-English code-mixed text," *International Journal on Advances in ICT for Emerging Regions (ICTer),* vol. 15, no. 3, pp. 60–71, 2022.

[14] A. M. Silva and R. Weerasinghe, "Example based machine translation for English-Sinhala translations," in *Proc. 9th International IT Conference*, Colombo, Sri Lanka, Oct. 2008, pp. 27–28.

[15] L. Long, J. Liang, and A. Smith, "AI literacy for multilingual learners: Storytelling, role-playing, and programming," *The CATESOL Journal*, vol. 35, no. 1, pp. 1–11, 2024.

[16] T. Wang, D. V. Diaz, C. Brown, and Y. Chen, "Exploring the role of ai assistants in computer science education: Methods, implications, and instructor perspectives," in *Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Washington, DC, USA, 2023, pp. 92–102. doi: 10.1109/VL-HCC57772.2023.00018

[17] C. Piech and S. Abu-El-Haija, "Human languages in source code: Auto-translation for localized instruction," in *Proc. Seventh ACM Conference on Learning @ Scale*, USA, 2020, pp. 167–174. doi: 10.1145/3386527.3405916

[18] T. Shaik, X. Tao, Y. Li, C. Dann, J. Mcdonald, P. Redmond, and L. Galligan, "A review of the trends and challenges in adopting natural language processing methods for education feedback analysis," *IEEE Access*, vol. 10, pp. 56720–56739, 2022.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. International Conference on Neural Information Processing Systems*, California, USA, 2017, pp. 6000–6010.

[20] B. V. Kiranmayee, R. S. Priya, R. Vijaya, P. Suresh, and R. V. Goutham, "Machine translation on dravidian languages," *International Journal of Recent Technology and Engineering*, vol. 12, no. 1, pp. 1–14, 2023.

[21] R. Perera, T. Fonseka, R. Naranpanawa, and U. Thayasivam, "Improving English to Sinhala neural machine translation using part-of-speech tag," arXiv Print, arXiv:2202.08882, 2022. doi: 10.48550/arXiv.2202.08882

[22] Z. Zhan, L. He, Y. Tong, X. Liang, S. Guo, and X. Lan, "The effectiveness of gamification in programming education: Evidence from a meta-analysis," *Computers and Education: Artificial Intelligence*, vol. 3, no. 100096, pp.1–11 ,2022.

[23] M. Hlosta, P. Bergamin, C. Herodotou, T. Papathoma, and A. Gillespie, "Predictive learning analytics in online education: A deeper understanding through explaining algorithmic errors," *Computers and Education: Artificial Intelligence*, vol. 3, no. 100108, pp.1–12, 2022.

[24] H. Khosravi, Y. S. Tsai, S. B. Shum, J. Kay, S. Knight, G. Chen, R. Martinez-Maldonado, and D. Gašević, "Explainable artificial intelligence in education," *Computers and Education: Artificial Intelligence*, vol. 3, no. 100074, pp. 1–22, 2022.

[25] A. C. M. Yang, I. Y. L. Chen, B. Flanagan, and H. Ogata, "How students' self-assessment behavior affects their online learning performance," *Computers and Education: Artificial Intelligence*, vol. 3, no. 100058, pp.1–8,2022.

[26] S. Sen, K. K. Gupta, A. Ekbal, and P. Bhattacharyya, "IITP-MT at WAT2018: Transformer-based multilingual indic-english neural machine translation system," in *Proc. 32nd Pacific Asia Conference*

Language, Information and Computation: 5th Workshop Asian Translation, Hong Kong, Dec. 2018, pp. 1003–1007.

[27] F. Guzmán, P. Chen, M. Ott, J. Pino, G. Lample, P. Koehn, V. Chaudhary, and M. Ranzato, "The FLORES evaluation datasets for low-resource machine translation: Nepali–English and Sinhala–English," in *Proc. Conference Empirical Methods in Natural Language Processing and the 9th International Joint Conference Natural Language Processing*, Hong Kong, China, 2019, pp. 6097–6110.

[28] X. P. Nguyen, S. Joty, W. Kui, and A. T. Aw, "Data diversification: An elegant strategy for neural machine translation," arXiv preprint, arXiv: 1911.01986, 2019.

[29] T. Fonseka, R. Naranpanawa, R. Perera, and U. Thayasivam, "English to Sinhala neural machine translation," in *Proc. International Conference on Asian Language Processing*, Kuala Lumpur, Malaysia, 2020, pp. 305–309. doi:10.1109/IALP51396.2020.9310462

[30] R. Naranpanawa, R. Perera, T. Fonseka, and U. Thayasivam, "Analyzing subword techniques to improve English to Sinhala neural machine translation," *International Journal of Asian Language Processing*, vol. 30, no. 04, pp. 2050017:1–2050017:13, 2020. doi:10.1142/s2717554520500174