A COMPILER BASED FRAMEWORK FOR LANGUAGE TRANSLATION

A. Subashini
IV Year CSE, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur
subashinia05@gmail.com

M. Poomari
IV Year CSE, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur
poomarisri2004@gmail.com

Dr. RR. Bhavani
Associate Professor, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur
bhavani@drsacoe.com

DOI: doi.org/10.34293/shanlax.9789361631474.ch014

Abstract

This paper presents an English–Tamil translation system designed to improve access to information for Tamil users. The proposed approach follows a hybrid model that combines rule-based translation, a local bilingual dictionary, and limited API support. Using compiler design concepts, the system processes input sentences through stages such as tokenization, parts of speech identification, phrase handling, and grammatical reordering to generate correct Tamil output. Most translations are performed offline, while online support is used only for unknown words. Experimental results show that the system produces accurate and natural Tamil translations with reduced dependency on external services, making it suitable for educational and low-connectivity environments

I. Introduction

Language is an important medium for communication, knowledge sharing, and cultural understanding. Differences between languages often create barriers in accessing information, especially when content is mainly available in English while many users prefer regional languages like Tamil. This language gap affects learning, technology usage, and digital accessibility. Most existing English–Tamil translation systems depend on cloud-based models and require continuous internet connectivity. They often focus on word-by-word translation and fail to maintain proper Tamil sentence structure. To overcome these limitations, this work proposes a hybrid translation approach that focuses on grammatical accuracy, offline usability, and meaningful sentence conversion.

II. Related Work

Language translation systems play an important role in enabling communication between people who speak different languages. Many modern translation tools use techniques such as NLP, NMT, SMT, and rule-based methods to produce accurate translations. However, neural-based

approaches usually require large datasets, high computational power, and continuous internet connectivity. Rule-based translation methods rely on predefined grammar rules and dictionaries to produce predictable and grammatically correct output. This is especially important for English–Tamil translation, as English follows SVO structure while Tamil follows SOV structure. A reliable translation system must handle both vocabulary mapping and grammatical restructuring.

1. Mohammad Faiyyaz et al. (2023) developed Polyglot, a web-based multilingual translator using NLP and NMT for text and voice translation. The system provides accurate translations but depends fully on internet connectivity. This work highlights the effectiveness of API-based translation models.
2. Siddhi Divate et al. (2023) proposed a real-time translator for text and speech used in education and travel. The system focuses on maintaining grammatical correctness during translation. It emphasizes the importance of proper sentence structure handling.
3. Sim Liew Fong et al. (2011) introduced a mobile-based offline translator using a preloaded dictionary. The system enables translation without internet access. However, it cannot adapt to new or unseen vocabulary.
4. Kasthuri and Britto Ramesh Kumar (2014) developed a rule-based English–Tamil translator using part-of-speech tagging and sentence reordering. Their system converts English SVO sentences into Tamil SOV structure. This approach ensures grammatically correct Tamil output.
5. V. Raghavan (2010) explained compiler design stages such as lexical analysis, syntax analysis, and code generation. These stages provide a structured way to process input systematically. The concepts are useful for designing grammar-based translation systems.
6. Bart Kahler et al. (2012) proposed a hybrid translation system combining offline dictionary lookup with cloud-based resources. Their approach allows translation during low connectivity. It supports dynamic vocabulary updates using online resources.
7. Sim Liew Fong et al. (2011) emphasized the importance of offline-first translation using a preloaded dictionary. The system works without internet connectivity. However, it lacks dynamic learning capability.
8. Raj Vyas et al. (2020) designed a real-time translation system combining local dictionaries with cloud translation. This hybrid approach improves translation speed and vocabulary coverage. It supports dynamic updates for new words.
9. Tutorials Point provides detailed explanations of compiler phases including tokenization and parsing. It explains how to analyze input in a step-by-step manner. These concepts

support structured language processing.

10. Geeks for Geeks offers practical tutorials on tokenization, parsing, and syntax analysis. It explains how to classify input and apply transformations. These ideas are helpful for grammar-aware translation systems.

III. Flow Chart

IV. Methodology

The proposed English–Tamil translation system follows a compiler-based hybrid approach to produce accurate and grammatically correct Tamil output. The translation process is divided into structured stages similar to compiler phases, making the system systematic and easy to process.

(I) Lexical Analysis

In a compiler, lexical analysis breaks source code into tokens by removing unnecessary characters .Similarly, in our translator, the input sentence is split into individual words after removing punctuation and converting all text to lowercase.

Example: Input: *I am writing a letter*, **Tokens**: [i, am, writing, a, letter]

(II) Syntax Analysis

In compilers, syntax analysis checks whether tokens follow grammatical rules.

In our system, each token is tagged using Part-of-Speech (POS) to identify subject, verb, and object. This step is important because English follows SVO order while Tamil follows SOV order. **Example: Subject:** *I*, **Verb**: *writing*, **Object:** *letter*

(III) Semantic Analysis

Semantic analysis ensures correct meaning. In our translator, each English word is mapped to its Tamil equivalent using a local bilingual dictionary. If a word is missing, an API is used as a fallback to fetch the translation. **Example:** I → நான், writing → எழுதுகிறேன், letter → கடிதம்

(IV) Intermediate Code Generation

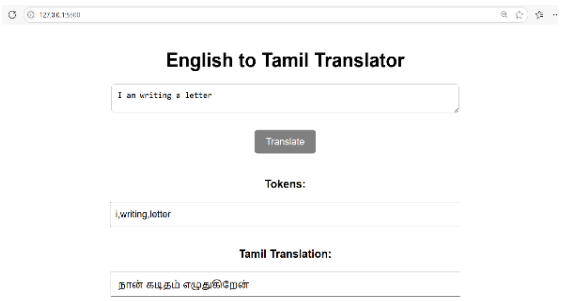
After translation, the Tamil words are temporarily stored in an array, similar to intermediate code in a compiler. This stage allows further processing before final sentence formation. **Example:** Tokens after translation: [நான், கடிதம், எழுதுகிறேன்]

(V) Code Optimization

In compilers, optimization improves efficiency. In our translator, this stage refines the sentence by removing unnecessary words and improving word placement to make the output natural and fluent.

(VI) Code Generation

This is the final stage where the reordered Tamil words are combined based on Tamil grammar rules. The final translated sentence is then displayed to the user. **Final Output:** *நான் கடிதம் எழுதுகிறேன்.*



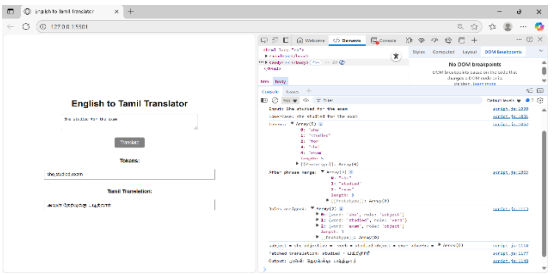
V. Results and Discussion

The translator works well for basic English sentences. It gives quick and offline output.

Example:

Input: "She studied for the exam"

Output: "நான் கடிதம் எழுதுகிறேன்"



In this example, the input sentence "she studied for the exam" is processed step by step to demonstrate the internal working of the translator. Initially, the sentence is converted into lowercase and tokenized during the lexical analysis phase, producing the tokens *she, studied, for, the,* and *exam*. Common articles and prepositions such as *for* and *the* are removed, retaining only the meaningful words required for translation.

Next, during the phrase merging stage, semantically relevant tokens are grouped to improve contextual clarity, resulting in the refined token set *she, studied,* and *exam*. In the syntax and semantic analysis phase, grammatical roles are assigned where *she* acts as the subject, *studied* as the verb, and *exam* as the object.The system then searches for Tamil equivalents of each token in the local bilingual dictionary. While the translations for *she* and *exam* are found locally, the verb *studied* is missing. To handle this, the fallback mechanism automatically invokes the Google Translate API to retrieve the correct Tamil translation "படித்தார்." This ensures uninterrupted translation even when local data is incomplete.

Finally, Tamil grammatical reordering rules are applied to convert the English Subject–Verb–Object structure into the Tamil Subject–Object–Verb pattern. The final translated output displayed is "அவள் தேர்வுக்கு படித்தார்." This example clearly demonstrates how the system maintains grammatical correctness, semantic accuracy, and transparency throughout the translation process.

VI. Conclusion

The proposed English–Tamil Translator using Compiler Design Concepts successfully demonstrates how compiler architecture can be adapted to solve real-world linguistic translation

problems. Each stage of the compiler — from lexical analysis to code generation — has been intelligently reinterpreted to perform linguistic tasks such as tokenization, syntax recognition, semantic mapping, and grammar reordering. This structured, phase-based approach ensures that the translation process is logical, modular, and error-free.

The translator effectively combines rule-based translation with API-assisted adaptability, creating a hybrid model that works both offline and online. The built-in dictionary ensures quick, local translations, while the Google Translate API acts as a fallback to handle missing or uncommon words. This hybrid system enhances accuracy and continuity of translation even when dealing with complex or unfamiliar input sentences.

Experimental results confirm that the translator produces accurate, meaningful, and grammatically correct Tamil outputs while maintaining contextual relevance. The system successfully identifies parts of speech, merges meaningful phrases, and reconstructs Tamil sentences according to SOV grammar rules. The implementation, built using HTML, CSS, and JavaScript, operates entirely in the browser, making it lightweight and platform independent.

In conclusion, the proposed model provides a practical and efficient solution for bilingual translation by combining linguistic analysis with compiler logic. The system bridges the gap between human language understanding and computational translation, establishing a foundation for future multilingual translation tools. Further improvements in context handling, phrase interpretation, and neural grammar correction could make this hybrid translator even more robust and intelligent.

VII. Reference

1. Mohammad Faiyyaz et al., "Web-Based Language Translator", IRJMETS, April 2023.
2. Siddhi Divate et al., "Real Time Language Translator", IRJMETS, December 2023.
3. Mobile-based Translator Project (Student Research Paper) “https://ieeexplore.ieee.org/document/6140723”
4. Rule-Based English to Tamil Translator (Academic Project Paper). “https://ieeexplore.ieee.org/document/6755127”
5. V. Raghavan, Principles of Compiler Design, Tata McGraw Hill Education, 2010.
6. Language translation of web-based content- Authors: Bart Kahler; Brian Bacher; K. C. Jones Published in: 2012 IEEE National Aerospace and Electronics Conference (NAECON), DOI: 10.1109/NAECON.2012.6531026, link: https://ieeexplore.ieee.org/document/6531026
7. Mobile language translator-Authors: Sim Liew Fong; Abdelrahman Osman Elfaki; Md Gapar bin Md

Johar; Kevin Loo Teow Aik Published in: 2011 Malaysian Conference in SoftwareEngineering,DOI: 10.1109/ MySEC.2011.6140723,link:https://iee explore.ieee.org/document/6140723

8. Real Time Machine Translation System for English to Indian language-Authors: Raj Vyas; Kirti Joshi; Hitesh Sutar; Tatwadarshi P. Nagarhalli, Authors Published in: 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) DOI: 10.1109/ICACCS48705.2020.907 4265,link:https://ieeexplore.ieee.org /document/9074265
9. "Compiler Design Tutorial," Tutorials Point: https://www.tutorialspoint.com/co mpiler_design/index.htm
10. "Compiler Design Tutorials," GeeksforGeeks: https://www.geeksforgeeks.org/co mpiler-design-tutorials/