Archives available at journals.mriindia.com

International Journal on Advanced Computer Engineering and Communication Technology

ISSN: 2278-5140
Volume 14 Issue 01, 2025

Programming Language Translation Using Machine Learning: A Results-Driven Study

Prof. V. S. Nalawade[1], Bhapkar Shital Rajendra[2], Raut Krupa Rajesh[3], Shaikh Diba Jamil[4]
[1]*Dean Academics & Head-AI & DS Engg. Dept. S. B. Patil College of Engineering*
[2,3,4]*Department of Computer Engineering, Savitribai Phule Pune University*
vinaynalawade2007@gmail.com, bhapkarshital2000@gmail.com, kruparaut1546@gmail.com, shaikhdiba786 @gmail.com

Peer Review Information

Submission: 15 Feb 2025
Revision: 23 March 2025
Acceptance: 27 April 2025

Keywords

Machine Learning
Programming Languages
Data Analysis
Translator

Abstract

This Paper introduces a new way to translate code between different programming language using machine learning. Traditional tool for this tasks require Complex rule and are often hard to use. Our module learns how to translate code on its own, without needing any predefined rules or example. It works for Languages like C++, Java and Python and produces accurate results that are easier to read and understand. This approach makes it simpler to converts code between languages, helping Developers update or connect different software systems more efficiently.

INTRODUTION

Transcompilers, or source-to-source compilers, translate code between languages with similar abstraction levels, differing from traditional compilers that convert high-level code to lower-level forms like assembly. Initially developed to port code between platforms (e.g., Intel 8080 to Intel 8086), they now facilitate converting newer languages (e.g., CoffeeScript, TypeScript) into widely-used ones (e.g., JavaScript), addressing language-specific limitations. They are valuable for modernizing legacy codebases (e.g., COBOL to Java) and integrating diverse codebases, but building them is complex due to differences in syntax, APIs and type systems. Recent advancements in neural machine translation (NMT) suggest potential improvements, as demonstrated by TransCoder—a model that uses monolingual code from GitHub to translate between C++, Java, and Python, achieving high accuracy and surpassing traditional methods.

LITERATURE SURVEY

Generating sequences from structured representations of code: Our model represents a code snippet as the set of compositional paths in its abstract syntax tree (AST) and uses attention to select the relevant paths while decoding. We demonstrate the effectiveness of our approach for two tasks, two programming languages, and four datasets of up to 16M examples. Our model significantly outperforms previous models that were specifically designed for programming languages, as well as state-of-the-art NMT models.[1]

Structural Language- models of code: generating a missing piece of source code in a given program without any restriction on the vocabulary or structure. We introduce a new approach to any-code completion that leverages the strict syntax of programming languages to model a code snippet as a tree - structural language modeling (SLM). SLM estimates the probability of the program's abstract syntax tree (AST) by decomposing it into a product of conditional probabilities over its nodes. We present a neural model that computes these conditional probabilities by considering all AST paths leading to a target node. Unlike previous

techniques that have severely restricted the kinds of expressions that can be generated in this task, our approach can generate arbitrary code in any programming language.[2]

A Survey On Creating Digital Health Ecosystem with Lifewellness Portal Including Hospital and Insurance Company with Cloud Computing and Artificial Intelligence: Nowadays, the development of e-health concept is offering various aspects. In this paper, we present a novel website portal with the help of cloud computing. Manage all medical data through cloud server. This website offers the ergonomic and multi-functions opportunity for an intelligent hospital and insurance company also, based on medical history, through the portal maintain the patient's records, real time treatment monitoring through this portal and for insurance claim to reimbursed for the cost of their medical treatment. Also with the help of AI (virtual) assist can help healthcare providers with a variety of tasks, such as reviewing patient documents, medical notes.[3]

Unsupervised statistical machine translation: This paper, we propose an alternative approach based on phrase-based Statistical Machine Translation (SMT) that significantly closes the gap with supervised systems. Our method profits from the modular architecture of SMT: we first induce a phrase table from monolingual corpora through cross-lingual embedding mappings, combine it with an n-gram language model, and fine-tune hyperparameters through an unsupervised MERT variant. In addition, iterative backtranslation improves results further, yielding, for instance, 14.08 and 26.22 BLEU points in WMT 2014 English-German and English-French, respectively, an improvement of more than 7-10 BLEU points over previous unsupervised systems, and closing the gap with supervised SMT (Moses trained on Europarl) down to 2-5 BLEU points.[4]

Sequencer: Sequence-to-sequence learning for end-to-end program repair: This paper presents a novel end-to-end approach to program repair based on sequence-to-sequence learning. We devise, implement, and evaluate a system, called SequenceR, for fixing bugs based on sequence-to-sequence learning on source code. This approach uses the copy mechanism to overcome the unlimited vocabulary problem that occurs with big code. Our system is data-driven; we train it on 35,578 samples, carefully curated from commits to open-source repositories. We evaluate it on 4,711 independent real bug fixes, as well on the Defects4J benchmark used in program repair research. SequenceR is able to perfectly predict the fixed line for 950/4711 testing samples, and find correct patches for 14 bugs in Defects4J. It captures a wide range of repair operators without any domain-specific top-down design.[5]

Voice-Enabled Traffic Sign Recognition and Alert System using ML: A Review.
The "Voice-Enabled Traffic Sign Recognition and Alert System" is an innovative application of machine learning and computer vision technologies aimed at enhancing road safety and driver awareness. In today's fast-paced world, the ability to promptly recognize and respond to traffic signs is crucial to prevent accidents and promote responsible driving. This project introduces a novel system that employs a camera installed in a vehicle to capture real-time images of the road. These images are then processed using advanced computer vision algorithms to detect and classify traffic signs. Furthermore, the system utilizes natural language processing to provide voice alerts to the driver, ensuring that they are informed about important traffic signs, speed limits, and other crucial information without taking their eyes off the road.[6]

Bert: Pre-training of deep bidirectional transformers for language understanding: We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.[7]

Codebert: A pre-trained model for programming and natural languages: We present CodeBERT, a bimodal pre-trained model for programming language (PL) and nat-ural language (NL). CodeBERT learns general-purpose representations that support downstream NL-PL applications such as natural language code search, code documentation generation, etc. We develop CodeBERT with Transformer-based neural architecture, and train it with a hybrid objective function that incorporates the pre-training task of replaced token detection, which is to detect plausible alternatives sampled from generators. This enables us to utilize both bimodal data of NL-PL pairs and unimodal data, where the former provides input tokens for model training while

the latter helps to learn better generators. We evaluate CodeBERT on two NL-PL applications by fine-tuning model parameters. Results show that CodeBERT achieves state-of-the-art performance on both natural language code search and code documentation generation tasks.[8]

Coda: An end-to-end neural program decompiler: we propose Coda1 , the first end-to-end neural-based framework for code decompilation. Coda decomposes the decompilation task into of two key phases: First, Coda employs an instruction type-aware encoder and a tree decoder for generating an abstract syntax tree (AST) with attention feeding during the code sketch generation stage. Second, Coda then updates the code sketch using an iterative error correction machine guided by an ensembled neural error predictor. By finding a good approximate candidate and then fixing it towards perfect, Coda achieves superior performance compared to baseline approaches. We assess Coda's performance with extensive experiments on various benchmarks. Evaluation results show that Coda achieves an average of 82% program recovery accuracy on unseen binary samples, where the state-of-the-art decompilers yield 0% accuracy. Furthermore, Coda outperforms the sequence-to-sequence model with attention by a margin of 70% program accuracy. Our work reveals the vulnerability of binary executables and imposes a new threat to the protection of Intellectual Property (IP) for software development.[9]

A Survey on Revolutionizing Document Security: A Comprehensive Deep Learning Approach For Signature Detection and Verification : In today's fast-paced business environment, automating signature verification is essential for efficiency. This project employs cutting-edge deep learning techniques: YOLOv5 for signature detection, CycleGAN for noise reduction, and VGG16-based feature extraction for verification. The workflow consists of three phases: signature detection, noise removal, and verification using cosine similarity with a threshold of 0.8. This interdisciplinary approach enhances operational efficiency and accuracy in document management and authentication, making it valuable for businesses.[10]

Two new evaluation datasets for low-resource machine translation: Nepali-english and sinhala-english: In this work, we take sentences from Wikipedia pages and introduce new evaluation datasets in two very low resource language pairs, Nepali-English and Sinhala-English. These are languages with very different morphology and syntax, for which little out-of-domain parallel data is available and for which relatively large amounts of monolingual data are freely available. We describe our process to collect and cross-check the quality of translations, and we report baseline performance using several learning settings: fully supervised, weakly supervised, semi-supervised, and fully unsupervised. Our experiments demonstrate that current state-of-the-art methods perform rather poorly on this benchmark, posing a challenge to the research community working on low resource MT.[11]

Deep code comment generation: This paper proposes a new approach named DeepCom to automatically generate code comments for Java methods. The generated comments aim to help developers understand the functionality of Java methods. DeepCom applies Natural Language Processing (NLP) techniques to learn from a large code corpus and generates comments from learned features. We use a deep neural network that analyzes structural information of Java methods for better comments generation. We conduct experiments on a large-scale Java corpus built from 9,714 open source projects from GitHub. We evaluate the experimental results on a machine translation metric. Experimental results demonstrate that our method DeepCom outperforms th

Towards neural decompilation: Present a novel approach to decompilation based on neural machine translation. The main idea is to automatically learn a decompiler from a given compiler. Given a compiler from a source language S to a target language T, our approach automatically trains a decompiler that can translate (decompile) T back to S. We used our framework to decompile both LLVM IR and x86 assembly to C code with high success rates. Using our LLVM and x86 instantiations, we were able to successfully decompile over 97% and 88% of our benchmarks respectively.[13]

LIMITATIONS OF EXISTING WORK

- **Rule-Based and Complex:** They rely on manually created rules to translate code, which makes them difficult to build and maintain.
- **Hard to Understand:** The translated code is often messy, hard to read, and usually requires a lot of manual corrections.
- **Requires Expert Knowledge:** Creating and updating these systems needs experts who understand both programming languages deeply.
- **Limited to Specific Languages:** They usually work well only for a few language pairs and struggle with others.

- **High Cost and Time-Consuming:** Because of all the manual effort and expertise needed, these systems can be expensive and slow to developer.

PROBLEM STATEMENT

The goal of this paper is to develop a programming language translator that leverages machine learning models to automatically convert source code written in one programming language into semantically equivalent code in another language. The translator should maintain the logic, structure, and functionality of the code across different languages, and handle various programming paradigms, syntax differences, and library dependencies.

PROPOSED SYSTEM

The **proposed system** for a **Programming Language Translator using Machine Learning** aims to automate the translation of code from one programming language to another, leveraging the power of machine learning (ML) techniques. This system is intended to serve both educational and practical purposes, allowing users to translate code snippets, understand programming concepts across languages, and assist in migrating codebases between languages.

- **Machine Learning Model**:

The heart of the system will be an ML-based model trained on large parallel corpora of code in multiple programming languages. The model will learn the syntactical and semantic mappings between different programming languages.

- **Preprocessing and Representation**:

Abstract Syntax Tree (AST) parsing will be used to represent the structure of code in a more language-agnostic manner. This allows the model to understand code logic and structure, independent of language-specific syntax.

- **Translation Pipeline:**

The user will input code in one language (e.g., Python), and specify the target language (e.g., Java).

The system will process the input code, generating an AST and using the trained model to translate the code to the target language while maintaining functionality.

- **Context-Aware Translation**:

The system will incorporate context-awareness to handle different translation scenarios. For example, if the model encounters language-specific constructs (e.g., Python's list comprehension), it will look for an equivalent construct in the target language (e.g., a for loop in Java)

- **Testing and Validation**:

The translated code will undergo automated testing to ensure that it preserves the functionality of the original code. This could include unit tests or functional tests.

SYSTEM REQUIREMENTS

Software Requirements

Operating System-Windows 10/11, Linux (Ubuntu,etc.), or macOS

Programming Language: Python (version 3.7 or higher)

Deep Learning Frameworks- TensorFlow or PyTorch

Natural Language Processing (NLP) Libraries-NLTK, spaCy

IDE: Jupyter Notebook or PyCharm.

Additional Libraries: NumPy, Pandas.

Hardware Requirements:

Processor (CPU)Intel core i3 or above

Memory (RAM) 8 GB and above

Storage –SSD(500GB) or HDD(1 TB).

METHODOLOGY

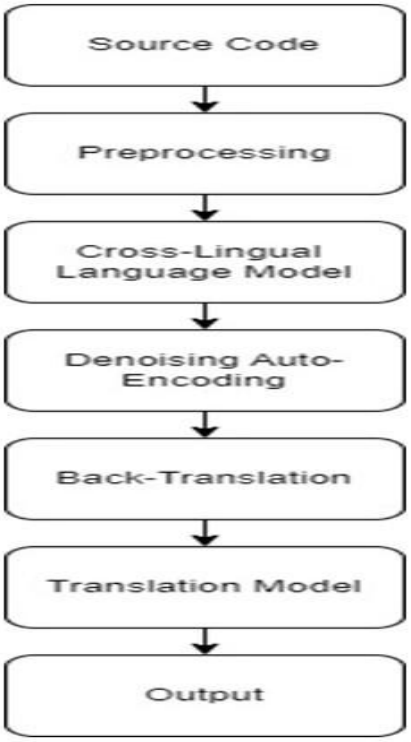


Fig 1. Architecture Diagram

Teacher User (A)

Logs in via the React Frontend (F):

The teacher logs in to the platform, just as before.

Uploads Resources (e.g., notes, question papers, videos):In addition to uploading resources, the teacher can now upload code in various programming languages (Python, Java, C++, etc.) that they want translated or need to be made accessible in different languages.

Defines Translation Tasks:The teacher can request the system to translate the code into

different programming languages. The system might offer an option for teachers to upload specific translation tasks, where they specify which programming languages should be translated into which.

Student User (B)
Logs in to View and Download Resources: Students can access resources uploaded by the teacher, as before.

Interacts with AI/Auto-Help Features: Students can also use the AI/Auto-Help module for additional help, including learning programming concepts or translating code between languages.

Translation Requests:
Students can request translation of code snippets into different programming languages, and the AI module will automatically translate the code using machine learning models.

React Frontend (F)
Single-Page Application (SPA): The SPA interface remains the same, but with additional features for interacting with the programming language translator and AI functionalities.

User Interface for Code Upload and Translation:A new interface is provided for uploading code snippets and requesting translation to other languages. The React Frontend will display translation options and allow users to input parameters (e.g., which language to translate code to).

Communicates with Python Backend (P) via RESTful APIs:Frontend sends requests for translations and code uploads to the Python Backend (P), which processes these tasks

Python Backend (P)
Handles Business Logic, Authentication, File Operations, and Communication with the Database:
As before, the Python Backend will manage authentication and handle file uploads.

Machine Learning-Based Language Translation: The Python Backend integrates a **Machine Learning Model** that has been trained on programming language syntax and patterns. It handles translating code from one programming language to another.

Database (D)
Stores User Information, Resource Metadata, and Uploaded Files:Database stores user profiles (teacher and student data), metadata about uploaded code resources, and file contents (e.g., text files, code snippets).

Stores Translated Code:The system stores a log of all translations performed, including both the source and target languages for future reference.

NoSQL or Relational Database:Depending on the nature of the data and scale, a NoSQL (MongoDB) or relational database may be used. NoSQL is ideal for flexible, document-based storage (like code and translation logs), while relational databases might be used for structured data like user profiles and metadata.

AI / Auto-Help Module (AI)
Provides Advanced Features:The AI/Auto-Help module is enhanced to assist with
Automated Code Translation: The AI module could offer code translation suggestions based on the context (e.g., language-specific idioms).
Pattern Recognition: Recognize common patterns in code to help translate them accurately.
Chatbot for Code Explanation: The chatbot can also help students understand why certain code blocks are translated the way they are or suggest optimizations for the translated code.

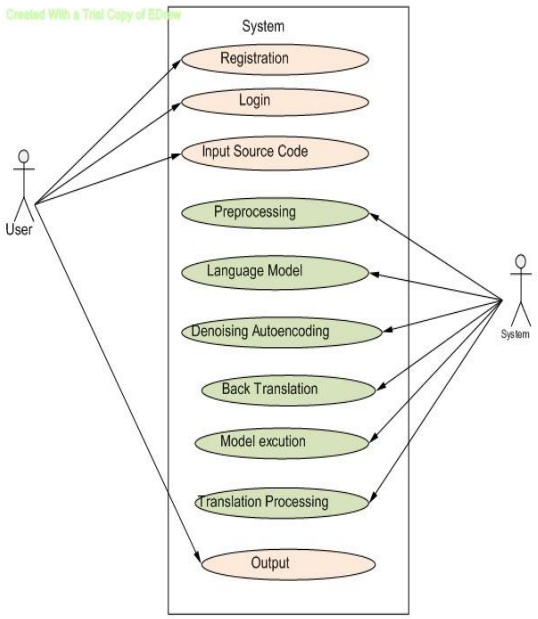
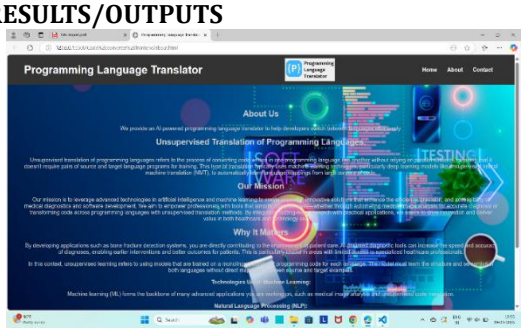


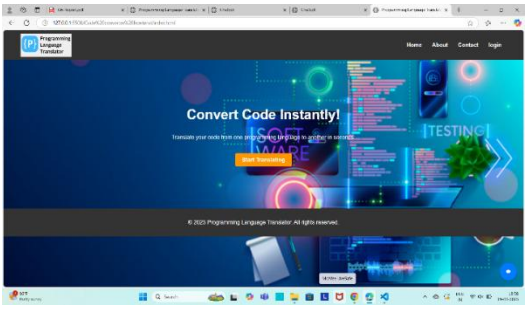
Fig 2. Use-case Diagram

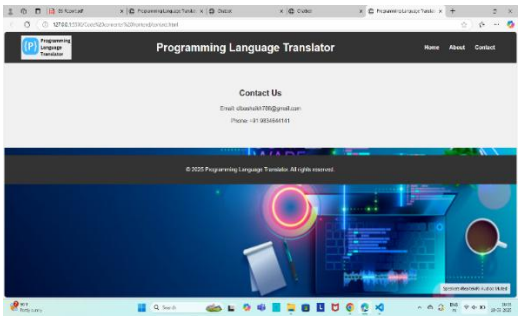
RESULT DISCUSSION
The results of a programming language translator using machine learning show both promising outcomes and notable challenges. When the model is trained on high-quality, diverse datasets, it can accurately translate code between

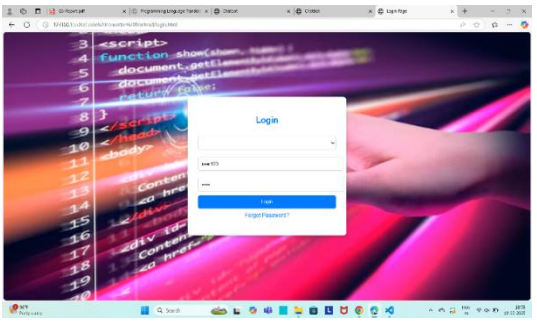
commonly used programming languages, achieving high correctness in terms of both syntax and semantics. However, performance tends to decline when translating complex or language-specific constructs, as some programming languages have unique features—such as Python's dynamic typing or Java's memory management—that don't easily map to others. This can result in the loss of functionality or errors in the translated code. While the model excels at translating simple constructs like loops and conditionals, it struggles with more advanced topics like concurrency, multi-threading, or language-specific idioms. Additionally, maintaining the readability and efficiency of the translated code across different languages can be difficult, as each language has its own conventions and optimizations. The speed of the translation also varies, with larger codebases taking longer to process. The model's generalization across languages depends on the variety of training data, which works well for widely-used languages but may falter with less common ones. Overall, while machine learning-based programming language translators offer significant potential for educational purposes and simple code conversion tasks, they still require refinement, continuous training, and hybrid approaches to handle edge cases and ensure production-level accuracy.

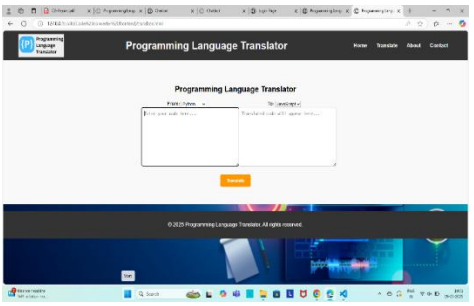
RESULTS/OUTPUTS

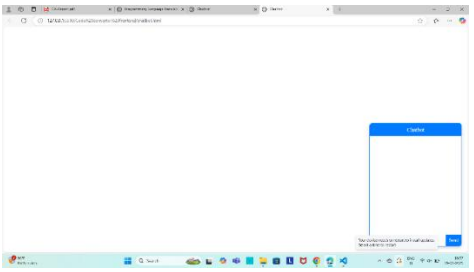












CONCLUSION

In this paper, we show that approaches of unsupervised machine translation can be applied to source code to create a transcompiler in a fully unsupervised way. TransCoder can easily be generalized to any programming language, does not require any expert knowledge, and outperforms commercial solutions by a large margin. Our results suggest that a lot of mistakes made by the model could easily be fixed by adding simple constraints to the decoder to ensure that the generated functions are syntactically correct, or by using dedicated architectures . Leveraging the compiler output or other approaches such as iterative error correction could also boost the performance.

References

Alon, Uri, et al. "code2seq: Generating sequences from structured representations of code." *arXiv preprint arXiv:1808.01400* (2018).

Alon, Uri, et al. "Structural language models of code." *International conference on machine learning*. PMLR, 2020.

Nalawade, V. S., Jadhav, O. D., Jadhav, R. M., Kargal, S. R., & Panhalkar, N. S. (2023). A Survey On Creating Digital Health Ecosystem with Lifewellness Portal Including Hospital and Insurance Company with Cloud Computing and Artificial Intelligence.

Artetxe, Mikel, Gorka Labaka, and Eneko Agirre. "Unsupervised statistical machine translation." *arXiv preprint arXiv:1809.01272* (2018).

Kang, Sungmin, and Shin Yoo. "Language models can prioritize patches for practical program patching." *Proceedings of the Third International Workshop on Automated Program Repair*. 2022.

Nalawade, V. S., Jagtap, T. G., Jamdar, P. B., Kadam, S. I., & Kenjale, R. S. (2023). Voice-Enabled Traffic Sign Recognition and Alert System using ML: A Review.

Devlin, J. "Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. NAACL-HLT (1) 2019: 4171-4186." *Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding* (2018).

Feng, Zhangyin, et al. "Codebert: A pre-trained model for programming and natural languages." *arXiv preprint arXiv:2002.08155* (2020).

Fu, Cheng, et al. "Coda: An end-to-end neural program decompiler." *Advances in Neural Information Processing Systems* 32 (2019).

Guzmán, Francisco, et al. "The flores evaluation datasets for low-resource machine translation: Nepali-english and sinhala-english." *arXiv preprint arXiv:1902.01382* (2019).

Nalawade, V. S., Aoute, Y. P., Dharurkar, A. S., & Gunavare, R. D. (2023). A Survey on Revolutionizing Document Security: A Comprehensive Deep Learning Approach For Signature Detection and Verification.

Hu, Xing, et al. "Deep code comment generation." *Proceedings of the 26th conference on program comprehension*. 2018.

Katz, Omer, et al. "Towards neural decompilation." *arXiv preprint arXiv:1905.08325* (2019).