Cross - Translation of Dynamically Typed and Interpreted Languages

Pranav H, Muppavarapu Sriharshini, Rachuri Tarun, Dept of computer science engineering, Amrita school of computing, Amrita Viswa Vidyapeetham, Bengaluru, India.
pranav.03.h@gmail.com, harshini.bannu2004@gmail.com tarunrachuri0303@gmail.com

I. Introduction

The field of programming is broad and dynamic with a vast array of programming languages accessible [1], each possessing distinct syntax and semantics, developers frequently have to collaborate across many languages[2]. This requirement may occur for a number of reasons, including the need to maintain old code, the need for a specific language for a given activity, or even just personal preference. But picking up and mastering several languages can be a difficult endeavour. This is where the idea of a converter and translator for programming languages is useful.

A tool called the Programming Language Translator and Converter [3,4] is intended to help create connections between various programming languages. Its objective is to transfer the logic and structure of the original source code written in one programming language is translated functionally to another programming language[5]. This tool can greatly increase productivity and efficiency by cutting down on the time and effort needed to rewrite code in a foreign language.

The goal of this project is to create a reliable and effective converter and translator for programming languages[6]. The major goal is to support translation popular languages such as Python, MATLAB, and, C/C++with plans for additional extension. In order to guarantee appropriate translation, the project also attempts to manage the subtleties and complexities of each language.

Hereafter, in Section 2 of the paper, the literature review surveys seven scholarly papers covering 10 different and modern methods of Language Translators. Section 3 of this Paper will be going through in detail of our proposed methodology.

II. LITERATURE SURVEY

Modern developments in summarization and source code translation are examined in this review of the literature. To ensure accurate code conversion, researchers have proposed unique and novel approaches that make use of various techniques explored further in this section.

Jana [7] introduces Co Tran, a technology that converts code while using feedback from compiler outputs together with symbolic execution for training massive language models. Through this method, the translated code is guaranteed to compile and have the same functionality as its source code thus being more effective than any other tool available since it is accurate in terms of compilation as well as functional equivalence.

Szafraniec [8] suggests using lower-level compiler intermediate representations like LLVM IR to beef up to better neural machine translation results in JavaScript source codes. Such advancements aimed at eliminating common semantic unoriginality during unsupervised programming translation significantly improve unsupervised programming translation including multiple languages.

Brauckmann [9] introduces ComPy-Learn as a flexible toolbox which is intended for the examination of divergent program code machine learning representations with the aim of optimizing compiler heuristics as well as other software engineering tasks. It allows conducting empirical research to uncover the best representations and models that combine higher-order syntax together with low-level compiler information.

Gourden [10] combines formally verified transformations integrated within CompCert, with a focus on Lazy Code Motion and Lazy Strength Reduction, to enhance performance optimizations provided by a compiler. The improvements result from introducing a Coq-verified validator that verifies correctness of these optimizations leading to their higher reliability.

Zugner [11] introduces a new model that leverages both the context and structure of the source code, utilizing language-agnostic features derived from the abstract syntax tree and the code itself, achieving exemplary results in code summarization across five unique monolingual programming languages but also pioneers a multilingual code summarization model demonstrates significant improvements, especially in low-resource languages, underscoring the advantage of integrating structure and context in learning representations of code.

Exploring advanced techniques for source code summarization using abstract syntax trees (ASTs) and transformers, Choi et al. [12] integrates graph convolution with transformers to capture both sequential and structural code features, enhancing summarization accuracy, Hou et al.

[13] developed a tree structure based transformer model, TreeXFMR, which utilizes hierarchical attention and bi-level positional encodings improving the representation and summarization of provided source code, showing significant performance improvements over existing methods.

Lachaux et al. [14] introduced an unsupervised neural translator that can translate among C++, Java as well as Python. It was trained by using monolingual source codes from GitHub. Liu [15] presents SDA-Trans, A model that is both Syntax and Domain Aware for unsupervised program translation, performing well on especially translation of functions between Python, Java, C++, especially with not specifically, using a smaller-scale corpus. Nguyen [16] proposes a novel refinement procedure for unsupervised machine translation models to focus on low-resource languages by disentangling them from high-resource languages in a multilingual environment, achieving state-of-the-art results in translating between multiple low-resource languages. Huang [17] developed Code Distillation (CoDist), model that uses a language-agnostic intermediate representation to overcome the lack of parallel corpora in program translation, improving performance on several program translation benchmarks.

Advancements in multi-language and multi-target compilers, focusing on efficiency and domain-specific needs have evolved as a field of considerable research in the recent years wherein Nandhini [18] discussed an online multi-language compiler system designed to streamline coding processes in educational settings, Boukham [19] explored a domain-specific language compiler for graph processing, emphasizing adaptable intermediate representations for various computing paradigms and Mullin [20] addresses the compilation of data parallel languages into an intermediate language, aiming for effective use across different multiprocessor topologies. Together, these studies underscore the evolving complexity and adaptability required in modern compiler design.

III. Proposed Methodology

There are three main ways by which the task of language translation can be tackled.

1. Using Large Language Models
2. Using Compilers and Trans – Compilers
3. Using Unsupervised Learning on a Large Corpus of Data

We will be exploring all three approaches herein, to identify which works the best.

References

[1] Sharma, Mamillapaly Raghavender. “A Short Communication on Computer Programming Languages in Modern Era.” *International Journal of Computer Science and Mobile Computing* (2020): n. pag.

[2] Sahakyan, Davit and Gurgen Karapetyan. “The Language of Translation.” *Translation Studies: Theory and Practice* (2022): n. pag.

[3] Tonchev, Ognyan and Mohammed Elhafiz Ramadan Salih. “High-level programming languages translator.”

[4] Raj, Utkarsh, Navneet Kaur and Babita Rawat. “English Algorithm Translation to C Program using Syntax Directed Translation Schema.” *2022 International Conference on Cyber Resilience (ICCR)* (2022): 1-3.

[5] Mariano, Benjamin, Yanju Chen, Yu Feng, Greg Durrett and Işıl Dillig. “Automated transpilation of imperative to functional code using neural-guided program synthesis.” *Proceedings of the ACM on Programming Languages* 6 (2022): 1 - 27.

[6] Giannini, Paola and Albert Shaqiri. “A Provably Correct Compilation of Functional Languages into Scripting Languages.” *Sci. Ann. Comput. Sci.* 27 (2017): 19-76.

[7] Jana, Prithwish, Piyush Jha, Haoyang Ju, Gautham Kishore, Aryan Mahajan and Vijay Ganesh. “CoTran: An LLM-based Code Translator using Reinforcement Learning with Feedback from Compiler and Symbolic Execution.” (2023).

[8] Szafraniec M, Roziere B, Leather H, Charton F, Labatut P, Synnaeve G. Code translation with compiler representations. arXiv preprint arXiv:2207.03578. 2022 Jun 30.

[9] Brauckmann, Alexander, Andrés Goens and Jerónimo Castrillón. “ComPy-Learn: A toolbox for exploring machine learning representations for compilers.” *2020 Forum for Specification and Design Languages (FDL)* (2020): 1-4.

[10] Gourdin, Léo. “Lazy Code Transformations in a Formally Verified Compiler.” *Proceedings of the 18th ACM International Workshop on Implementation, Compilation, Optimization of OO Languages, Programs and Systems* (2023): n. pag.

[11] Zugner, Daniel, Tobias Kirschstein, Michele Catasta, Jure Leskovec and Stephan Gunnemann. “Language-Agnostic Representation Learning of Source Code from Structure and Context.” ArXiv abs/2103.11318 (2021): n. pag.

[12] Choi, YunSeok, Jinyeong Bak, CheolWon Na and Jee-Hyong Lee. “Learning Sequential and Structural

Information for Source Code Summarization." Findings (2021).

[13] Hou, Shifu, Lingwei Chen and Yanfang Ye. "Summarizing Source Code from Structure and Context." 2022 International Joint Conference on Neural Networks (IJCNN) (2022): 1-8.

[14] Lachaux, Marie-Anne, Baptiste Rozière, Lowik Chanussot and Guillaume Lample. "Unsupervised Translation of Programming Languages." ArXiv abs/2006.03511 (2020): n. pag.

[15] Liu, Fang, Jia Li and Li Zhang. "Syntax and Domain Aware Model for Unsupervised Program Translation." 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) (2023): 755-767.

[16] Nguyen, Xuan-Phi, Shafiq R. Joty, Wu Kui and Ai Ti Aw. "Refining Low-Resource Unsupervised Translation by Language Disentanglement of Multilingual Model." ArXiv abs/2205.15544 (2022): n. pag.

[17] Huang, Yufan, Mengnan Qi, Yongqiang Yao, Maoquan Wang, Bin Gu, Colin B. Clement and Neel Sundaresan. "Program Translation via Code Distillation." Conference on Empirical Methods in Natural Language Processing (2023).

[18] Nandhini, N., M. G. M. Prassanna, D. Tamilarasi, R. Varthini and S. Sadesh. "Implementation Of Multi Language Compiler For College Using Smart Lab System." (2019).

[19] Boukham, Houda, Guido Wachsmuth, Martijn Dwars and Dalila Chiadmi. "A Multi-target, Multi-paradigm DSL Compiler for Algorithmic Graph Processing." *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering* (2022): n. pag.

[20] Mullin, Lenore M. Restifo, C. Chang, S. Huang, Mikaël Mayer, Nader A. Nemer and C. Ramakrishna. "Intermediate Code Generation for Portable Scalable, Compilers. Intermediate Code Generation for Portable Scalable, Compilers. Architecture Independent Data Parallelism: The Preliminaries Architecture Independent Data Parallelism: The Preliminaries." (2020).

Progress Report.docx

ORIGINALITY REPORT

3%	2%	3%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Shifu Hou, Lingwei Chen, Yanfang Ye. "Summarizing Source Code from Structure and Context", 2022 International Joint Conference on Neural Networks (IJCNN), 2022 Publication	1%
2	export.arxiv.org Internet Source	1%
3	kth.diva-portal.org Internet Source	1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		