Towards Data Augmentation for Supervised Code Translation

Binger Chen
chen@tu-berlin.de
TU Berlin
Berlin, Germany

Jacek golebiowski
jacekgo@amazon.de
Amazon AWS
Berlin, Germany

Ziawasch Abedjan
abedjan@dbs.uni-hannover.de
Leibniz Universität Hannover & L3S Research Center
Hannover, Germany

ABSTRACT

Supervised learning is a robust strategy for data-driven program translation. This work addresses the challenge of insufficient parallel training data in code translation by exploring two innovative data augmentation methods: a rule-based approach specifically designed for code translation datasets and a retrieval-based method leveraging unorganized code repositories.

1 INTRODUCTION

Translating software across different programming languages is a practical application, such as the updates or migrations to modern platforms, a process that is both time-intensive and costly [11]. Traditional **rule-based** compilers are limited and demand significant human input. **Supervised methods** [5] show promise but heavily rely on parallel datasets. **Unsupervised methods**, despite being a potential solution, fall short in terms of accuracy compared to supervised ones [11, 12]. **Large Language Models** (LLMs) [6] offer new prospects but currently lack reliability in code translation [9]. The latter two challenges underscore the need for enhanced training data augmentation methods to improve supervised translation techniques. Current training data for code translation is mostly manually created. A promising approach to deal with training data scarcity is to generate training data via **data augmentation** (DA), which generates additional data from existing training data. There are already many approaches in the realm of supervised natural language translation [7, 13], which are not directly transferable to code translation due to the strict grammar of code, evolving vocabularies, and lack of mapping dictionaries across languages. LLMs is a potential tool for DA, but face challenges like API misuses and quality issues in generated code [8, 14]. There have been novel augmentation methods in the realm of supervised bug fixing [1, 10]. However, these augmentation techniques do not account for the fact that code is executable.

In this work, we focus on developing augmentation methods to enhance supervised code translation. We explored three strategies: applying rule-based and back-translation techniques from NLP directly to code, designing **tailored rule-based approach** for code translation, and a new **retrieval-based approach** using Big Code. The study found that rule-based methods lack diversification and

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0502-1/24/04
https://doi.org/10.1145/3639478.3643115

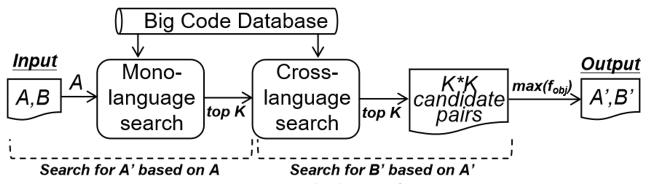

Figure 1: Workflow of CTAUG

back-translation depends on model quality, whereas retrieval-based augmentation with refined optimization functions, shows effectiveness in finding new parallel code pairs and outperforms other methods. While this abstract provides an overview of the challenges and the outcomes, a detailed description of methods and analysis can be found in the full publication [4]. All code and data can be found on our GitHub repository [1].

2 AUGMENTING VIA RULES AND RETRIEVAL

Training datasets for supervised code translation are so-called parallel datasets, where code with the same functionality in two or more languages has already been aligned. The main goal in augmenting training data for supervised code translation is to introduce unseen source-target pairs. Promising approaches rooted in literature include adapting rule-based methods from natural language translation to code and leveraging big code repositories for additional snippet pairs, aiming to enrich the training dataset with novel, yet relevant, relationships.

Rule-based DA. The main idea for rule-based data augmentation is to use code transformation rules that when applied to an existing code pair generate a new pair of functionally and semantically correct pieces. These rules, adapted from bug detection strategies [1, 10], focus on reversible changes and code structure modifications, ensuring logical consistency across languages. We create the following rules:

(1) **Reverse**: Adjusting logic in conditional statements and making corresponding changes in the target language code.
(2) **Merge**: Combining code at the control structure or program level, using logical operators or concatenation.
(3) **Split**: Separating combined conditional statements.
(4) **All**: A composite application of the above rules.

These rules are specifically designed for programming languages, but general enough to common imperative languages. Thus, they can be applied to both the input code and its translation, maintaining the logical semantics between source and target.

Retrieval-based DA. Our second approach searches for code pairs in Big Code repositories. Using Big Code, we hope to find unseen code pairs. In this process, an across-language retrieval method is used to find potential translations of a piece of given code. In theory,

[1]https://github.com/LUH-DBS/Binger/tree/main/CTAug

Table 1: Performance (PA) comparison with the baselines

Dataset	w/o Aug	CTAug	NLP Methods		Rule-based methods			
			WM	*BT*	*Reverse*	*Merge*	*Split*	*All*
Lucene	72.8%	**85.8%**	63.1%	70.2%	72.7%	72.0%	72.8%	75.9%
POI	72.2%	**86.1%**	62.4%	69.9%	73.6%	70.3%	72.5%	75.1%
IText	67.5%	**83.2%**	61.5%	66.3%	68.9%	66.9%	67.8%	72.8%
JGit	68.7%	**81.7%**	59.3%	65.4%	70.3%	70.2%	69.1%	72.8%
JTS	68.2%	**82.3%**	61.2%	67.7%	70.9%	69.8%	69.6%	73.3%
ANTLR	31.9%	**66.2%**	25.3%	31.9%	33.6%	36.7%	32.7%	40.1%

one could pick random pieces of code and search for their translations. However, this approach might convolute existing training data or lead to code pairs that are maximally dissimilar to the original training data. Ideally, we want the augmented data points to be explainable in terms of being close to data points that we already have but with previously unseen inter-language relationships. Thus we also use a mono-language retrieval technique to identify similar input pieces. Combining the two retrieval methods, one has to make a decision of which retrieval results to combine.

The simplest approach is greedy search, where the most similar piece of code per input value and its best translation are composed to a new data point. In our work, we propose an optimization method that ensures the code pieces are similar in several ways. Figure 1 shows the workflow, which retrieves top-K candidates using a mono-language code search tool, forming a candidate set. For each candidate, we find top-K translations using a cross-language tool, resulting in $K \times K$ candidate pairs. The value of K can be adjusted based on computational resources and needs.

Our selection process for the optimal translation pair from the $K \times K$ candidate space ($D_{candidate}$) involves matching each candidate A' with a corresponding B' that mirrors the similarities and differences it has with the original A and B. We use the similarity function S implemented in the code search tools [2, 3], which evaluates structural and textual aspects of code snippets. This function is employed in three distinct ways to define independent optimization criteria, ensuring each selected pair aligns closely with the original code's context and characteristics.

Considering the context $(A, B) \in D_0$ and a sample from a search space $(A', B') \in D_{candidate}$ we consider

(1) $f_1(A', B') = S(A', B')$ as a similarity measure within the new pair,
(2) $f_2(A', B') = S(A', A) + S(B', B)$ as a similarity measure between the new tuple and context and
(3) $f_3(A', B') = -|S(A', A) - S(B', B)|$ as an alternative similarity measure within the new pair. Intuitively, we aim to find examples where A is just as similar to A' as B is to B'.

Given the three objective functions that map the search space into $\mathbb{R}$, we can formalize our task as a multi-objective optimization problem. We rely on optimizing a scalarized objective $f_{obj} = \alpha f_1 + \beta f_2 + \gamma f_3$ where α, β and γ are set to 1 by default. The goal is to find a pair of $(A', B') \in D_{aug}$ that satisfies

$$\mathbf{max} f_{obj} = \alpha \mathbf{max} f_1(A', B') + \beta \mathbf{max} f_2(A', B') + \gamma \mathbf{max} f_3(A', B')$$

3 EXPERIMENTS AND CONCLUSION

To pitch the promise of our approach, we present one experiment on two datasets: one from previous work [5], comprising Java and C# code pairs from open-source projects; another is a big code database generated from GitHub repositories for code retrieval. The evaluation metric is Program Accuracy (PA), which measures the percentage of predicted translations matching the ground truth, focusing on semantic equivalence. We evaluate rule-based approaches and the retrieval-based method **CTAug** (Data **Aug**mentation for **C**ode **T**ranslation Model) for augmenting code translating training data against NLP baselines (word masking and back-translation). The augmentation ratio is 1 : 1. The results shown in Table 1 reveal that while NLP techniques are less effective for code, CTAug significantly improves translation accuracy. Rule-based methods show only marginal gains, with some causing overfitting. Overall, our language-agnostic approach CTAug excels in program accuracy. Detailed results including more metrics and mini-benchmarks can be found in the longer version of this paper [4].

Conclusion. We proposed two data augmentation methods for supervised code translation: rule-based and retrieval-based strategies. The more effective retrieval-based approach leverages mono- and cross-language code retrieval techniques to find translation pairs that enhance data diversity while preserving distribution. Though the rule-based method performs poorer, it surpasses basic NLP augmentation adaptations. **Future work** could explore combining these strategies and applying constrained retrieval in other supervised code-learning tasks like code stylization or summarization.

Acknowledgements. This work was funded by the German Federal Ministry of Education and Research under the grant BIFOLD24B.

REFERENCES

[1] Miltiadis Allamanis, Henry Jackson-Flux, and Marc Brockschmidt. 2021. Self-Supervised Bug Detection and Repair. In *Annual Conference on Neural Information Processing Systems 2021 (NeurIPS).*

[2] Binger Chen and Ziawasch Abedjan. 2021. Interactive Cross-language Code Retrieval with Auto-Encoders. In *36th International Conference on Automated Software Engineering (ASE).*

[3] Binger Chen and Ziawasch Abedjan. 2021. RPT: Effective and Efficient Retrieval of Program Translations from Big Code. In *43rd International Conference on Software Engineering: Companion Proceedings (ICSE).*

[4] Binger Chen, Jacek golebiowski, and Ziawasch Abedjan. 2024. Data Augmentation for Supervised Code Translation Learning. In *21th International Conference on Mining Software Repositories (MSR).*

[5] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree neural networks for program translation. In *Advances in Neural Information Processing Systems (NeurIPS).*

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT).*

[7] Sufeng Duan, Hai Zhao, Dongdong Zhang, and Rui Wang. 2020. Syntax-aware Data Augmentation for Neural Machine Translation. *CoRR* abs/2004.14200 (2020).

[8] Kevin Jesse, Toufique Ahmed, Premkumar T. Devanbu, and Emily Morgan. 2023. Large Language Models and Simple, Stupid Bugs. In *20th IEEE/ACM International Conference on Mining Software Repositories (MSR).*

[9] Shuai Lu, Daya Guo, and Shuo Ren et al. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1.*

[10] Pedro Orvalho, Mikolás Janota, and Vasco M. Manquinho. 2022. MultIPAs: applying program transformations to introductory programming assignments for data augmentation. In *The 30th Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).*

[11] Baptiste Rozière, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised Translation of Programming Languages. In *Annual Conference on Neural Information Processing Systems (NeurIPS).*

[12] Baptiste Rozière, Jie Zhang, and François Charton et al. 2022. Leveraging Automated Unit Tests for Unsupervised Code Translation. In *The Tenth International Conference on Learning Representations (ICLR).*

[13] Xinyi Wang, Hieu Pham, and Zihang Dai et al. 2018. SwitchOut: an Efficient Data Augmentation Algorithm for Neural Machine Translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[14] Li Zhong and Zilong Wang. 2023. A Study on Robustness and Reliability of Large Language Model Code Generation. *CoRR* abs/2308.10335 (2023).