AVATAR: A Parallel Corpus for Java-Python Program Translation

Wasi Uddin Ahmad[†], Md Golam Rahman Tushar[§]
Saikat Chakraborty[‡], Kai-Wei Chang[†]

[†]University of California, Los Angeles, [‡]Microsoft Research, [§]Independent Contributor
[†]{wasiahmad, kwchang}@cs.ucla.edu
[‡]saikatc@microsoft.com, [§]grtushar11@gmail.com

Abstract

Program translation refers to migrating source code from one programming language to another. It has tremendous practical value in software development, as porting software across languages is time-consuming and costly. Automating program translation is of paramount importance in software migration, and recently researchers explored unsupervised approaches due to the unavailability of parallel corpora. However, the availability of pre-trained language models for programming languages enables supervised fine-tuning with a small number of labeled examples. Therefore, we present AVATAR, a collection of 9,515 programming problems and their solutions written in two popular languages, Java and Python. AVATAR is collected from competitive programming sites, online platforms, and open-source repositories. Furthermore, AVATAR includes unit tests for 250 examples to facilitate functional correctness evaluation. We benchmark several pre-trained language models fine-tuned on AVATAR. Experiment results show that the models lack in generating functionally accurate code.

1 Introduction

Software developers and researchers often require to convert software codebases or research prototypes from one platform to another or rewrite them in the target programming languages. Manually rewriting software is time-consuming, expensive, and requires expertise in both the source and target languages. For example, the Commonwealth Bank of Australia spent around $750 million and 5 years translating its platform from COBOL to Java (Lachaux et al., 2020). A program translation system that converts the source code of a program written in a programming language to an equivalent program in a different programming language is known as a transcompiler, transpiler, or source-to-source compiler. Transcompilers have a prodigious practical value; they could help to reduce the translation efforts of developers and researchers by not requiring them to write code from scratch, instead, they can edit the translated code with less effort.

The conventional transcompilers are based on rule-based approaches; they first convert source code into an Abstract Syntax Tree (AST) and then apply handwritten rules to translate to the target language. Development and adaptation of transcompilers need advanced knowledge and therefore are available in a handful of programming languages. Undoubtedly, the automation of program translation would facilitate software development and research tremendously.

With the recent advancements in data-driven neural machine translation (NMT) approaches between natural languages, researchers have started investigating them for programming language translation. Lachaux et al. (2020) trained an NMT system in an unsupervised fashion using large-scale monolingual source code from GitHub that showed noteworthy success in source code translation between Java, Python, and C++ languages. Pre-trained language models (PLMs) of code have been shown to work well on Java-C# translation after fine-tuning on a small amount of parallel examples (Feng et al., 2020; Guo et al., 2021; Ahmad et al., 2021; Wang et al., 2021). Motivated by these favorable results, in this work, we propose a new parallel corpus of Java and Python programs.

We propose a corpus, AVATAR (jAVA-pyThon progrAm tRanslation) that consists of solutions written in Java and Python for 9,515 programming problems collected from competitive programming sites, online platforms, and open source repositories. AVATAR includes 250 examples with unit tests to facilitate functional correctness evaluation of program translation. We train several baselines, including models trained from scratch or pre-trained on large-scale source code collection and fine-tuned on AVATAR. The experiment results indicate that while the models perform considerably in terms of the lexical match, they lack Fur-

Findings of the Association for Computational Linguistics: ACL 2023, pages 2268–2281
July 9-14, 2023 ©2023 Association for Computational Linguistics

Source	#Prob.	Java		Python		Soln. / Prob.	Train	Valid / Test
		#Soln.	Avg_L	#Soln.	Avg_L			
AtCoder	871	3,990	276.5	4,344	180.3	[1 – 5]	14,604	36 / 195
Code Jam	120	508	390.9	460	266.5	[1 – 5]	1,586/7	7/ 19
Codeforces	2,193	6,790	246.2	10,383	123.8	[1 – 5]	24,754	102 / 436
GeeksforGeeks	5,019	5,019	194.8	5,019	138.4	1	3,754	269 / 996
LeetCode	107	107	140.0	107	97.4	1	82	7 / 18
Project Euler	162	162	227.3	162	139.4	1	110	11 / 41
AIZU	1,043	4,343	304.2	4,603	171.3	[1 – 5]	15,248	44 / 199
Total	9,515	20,919	254.5	25,078	147.9	-	60,138	476 / 1,906

Table 1: Statistics of the AVATAR dataset. Avg_L indicates the average program length (after parsing) written in Java and Python languages. We split the dataset into 75:5:20 to form training, validation, and test examples. To form parallel examples for training, we pair up solutions in Java and Python. For validation and test examples, we consider multiple solutions as ground truth.

thermore, AVATAR offers 3,391 parallel functions that we use to train models or fine-tune pre-trained language models and perform function translation evaluation on the dataset released by Lachaux et al. (2020). Our code and data are released at `https://github.com/wasiahmad/AVATAR`.

2 AVATAR Construction

Data Collection We construct AVATAR based on solutions of computational problems written in Java and Python collected from open source programming contest sites: AtCoder, AIZU Online Judge, Google Code Jam, Codeforces, and online platforms: GeeksforGeeks, LeetCode, Project Euler. We crawl Codeforces and GeeksforGeeks sites to collect the problem statements and their solutions. We collect the AtCoder and AIZU data from Puri et al. (2021), Google Code Jam data from Nafi et al. (2019)[1], and LeetCode and Project Euler problem solutions from open source Github repositories.[2,3] We collect [1 – 20] accepted solutions for a single problem written in Java and Python.

Preprocessing & Filtering At first, we tokenize the solution code and remove docstrings and comments from them. We use the `javalang`[4] tokenizer for Java and the `tokenizer`[5] of the standard library for Python. After tokenization, we filter out solutions that are longer than a specified length threshold (= 464). In the initial data collection, there are [1 – 20] accepted solutions for each problem. We filter out solutions and only keep at most 5 solutions per problem. Our goal is to keep the solutions that are maximally different from others in order to increase diversity among solutions of the same problem. We use the open source library `difflib`[6] to compare all the solutions pairwise (individually in Java and Python) and select five solutions that differ most from others.

Data Statistics We split 9,515 problem statements into a 75:5:20 ratio to form 7,133 training, 476 validation, and 1,906 test examples. Table 1 summarizes the data statistics. Since we collect [1 – 5] accepted solutions for each problem statement in both languages, we form [1 – 25] parallel examples per problem for training. In evaluation, we use multiple ground truths and select the best performance according to the evaluation metrics.

Unit Tests AVATAR presents unit tests for 250 evaluation examples (out of 1,906) to perform functional accuracy evaluation of the translation models. The unit tests are collected from the publicly available test cases released by AtCoder.[7]

Parallel Functions AVATAR includes 3,391 parallel Java and Python functions.[8] The functions are extracted by parsing programs that include *only* one function. We use them for training models and evaluating using the dataset released by Lachaux et al. (2020).

[1] `https://github.com/Kawser-nerd/CLCDSA`
[2] `https://github.com/qiyuangong/leetcode`
[3] `https://github.com/nayuki/Project-Euler-solutions`
[4] `https://github.com/c2nes/javalang`
[5] `https://docs.python.org/3/library/tokenize.html`
[6] `https://docs.python.org/3/library/difflib.html`
[7] `https://atcoder.jp/posts/21`
[8] Deduplicated against the evaluation dataset released by Lachaux et al. (2020) using `https://github.com/microsoft/dpu-utils`.

3 Experiment & Results

3.1 Evaluation Metrics

BLEU computes the overlap between candidate and reference translations (Papineni et al., 2002).

Syntax Match (SM) represents the percentage of the sub-trees extracted from the candidate program's abstract syntax tree (AST) that match the sub-trees in reference programs' AST.

Dataflow Match (DM) is the ratio of the number of matched candidate data-flows and the total number of the reference data-flows (Ren et al., 2020).

CodeBLEU (CB) is the weighted average of the token level match, syntax level match (SM), and Dataflow match (DM) (Ren et al., 2020).

Execution Accuracy (EA) indicates the percentage of translated programs that are executable (results in no compilation or runtime errors).

Computational Accuracy (CA) Lachaux et al. (2020) proposed the metric to evaluate whether the candidate translation generates the same outputs as the reference when given the same inputs.

3.2 Models

We evaluate a variety of models on program and function translation using AVATAR and the evaluation dataset released by Lachaux et al. (2020).

Zero-shot This set of models is evaluated on AVATAR without any training or fine-tuning.

• **TransCoder** is pre-trained in an unsupervised fashion that can translate programs between Java, Python, and C++ languages (Lachaux et al., 2020).

• **DOBF** uses deobfuscation pretraining followed by unsupervised translation (anne Lachaux et al., 2021).

• **TransCoder-ST** is developed by fine-tuning TransCoder on a parallel corpus created via an automated unit-testing system (Roziere et al., 2022).

Models trained from scratch These models are trained from scratch using AVATAR. We use the sentencepiece tokenizer and vocabulary from Ahmad et al. (2021) in these models.

• **Seq2Seq+Attn.** is an LSTM based sequence-to-sequence (Seq2Seq) model with attention mechanism (Bahdanau et al., 2015).

• **Transformer** is a self-attention based Seq2Seq model (Vaswani et al., 2017). We use the Transformer architecture studied in Ahmad et al. (2020).

Pre-trained Models We evaluated three types of pre-trained models (PLMs). First, we evaluate decoder-only PLMs (*e.g.,* CodeGPT) that generate auto-regressively. The second category of PLMs is encoder-only (*e.g.,* CodeBERT). We use a randomly initialized decoder to finetune such models in a Seq2Seq fashion. The third category of PLMs is Seq2Seq models (*e.g.,* PLBART), which we directly finetune on translation tasks.

• **CodeGPT and CodeGPT-adapted** are GPT-2 (Radford et al., 2019) style models pre-trained on CodeSearchNet (Lu et al., 2021). Note that CodeGPT-adapted starts from the GPT-2 checkpoint, while CodeGPT is pre-trained from scratch.

• **CodeBERT** is an encoder-only model that is pre-trained on unlabeled source code via masked language modeling (MLM) and replaced token detection objectives (Feng et al., 2020).

• **GraphCodeBERT** is pre-trained using MLM, data flow edge prediction, and variable-alignment between code and its' data flow (Guo et al., 2021).

• **PLBART** is a Transformer LM pre-trained via denoising autoencoding (Ahmad et al., 2021).

• **CodeT5** is a Transformer LM pre-trained via identifier-aware denoising (Wang et al., 2021).

In addition, we fine-tune TransCoder-ST, which is the best translation model in the literature.

3.3 Hyperparameters Details

We individually fine-tune the models for Java to Python and Python to Java program and function translation, respectively. We fine-tune the models for a maximum of 20 epochs using the Adam (Kingma and Ba, 2015) optimizer with a batch size of 32. We tune the learning rate in the range $[1e-4, 5e-5, 3e-5, 1e-5]$. The final models are selected based on the validation BLEU score. We use beam decoding with a beam size set to 10 for inference across all the models.

3.4 Results

Program Translation The performance comparison of all the experiment models is presented in Table 2. In general, all the models perform well in terms of match-based metrics, *e.g.,* BLEU and CodeBLEU. However, the computational accuracy (CA) clearly indicates that these models are far from perfect in generating functionally accurate translations. Overall, the best-performing model is PLBART, resulting in the highest execution accuracy (EA) and CA in Java to Python translation.

Models	Java to Python						Python to Java					
	BLEU	SM	DM	CB	EA	CA	BLEU	SM	DM	CB	EA	CA
TransCoder	38.7	31.6	38.2	36.4	77.3	0	45.2	39.3	20.1	32.4	0	0
DOBF	42.0	32.9	**42.9**	38.9	78.3	0	42.3	39.5	19.0	31.2	0	0
TransCoder-ST	41.7	33.1	42.6	39.3	85.8	0	42.5	37.4	20.4	30.7	0	0
Seq2Seq+Attn.	57.4	40.9	34.8	42.6	92.2	2.8	59.5	50.1	26.6	43.0	48.4	0.8
Transformer	39.6	35.0	33.5	34.8	92.3	0.4	43.5	44.9	25.2	35.6	63.8	0.4
CodeGPT	46.3	32.2	22.2	30.2	79.4	2.8	48.9	42.7	34.1	38.0	40.7	**2.0**
CodeGPT-adapted	44.3	31.6	20.4	29.3	80.2	2.4	48.0	43.0	28.3	36.7	46.7	0.8
CodeBERT	51.1	34.4	29.2	35.0	92.8	0.4	35.1	41.1	31.5	33.2	54.1	0
GraphCodeBERT	57.9	38.0	32.2	39.0	92.9	2.0	38.3	42.6	32.7	36.9	66.8	0
PLBART	**63.1**	**42.2**	37.9	**46.2**	**96.4**	**6.8**	**69.7**	54.2	30.9	48.8	**78.3**	0.8
CodeT5	62.7	41.7	37.9	**46.2**	91.8	6.0	60.8	**55.1**	**39.6**	**50.3**	68.7	1.6
TransCoder-ST	55.4	41.6	36.1	43.8	94.9	5.6	66.0	53.3	31.7	48.6	72.4	**2.0**

Table 2: Test set results using AVATAR for Java-Python program translation. SM, DM, CB, EA, and CA stand for Syntax Match, Dataflow Match, CodeBLEU, Execution Accuracy, and Computational Accuracy, respectively.

Models	Java to Python						Python to Java					
	BLEU	SM	DM	CB	EA	CA	BLEU	SM	DM	CB	EA	CA
TransCoder	72.4	55.7	65.7	67.9	69.2	49.1	65.4	72.6	70.3	70.7	58.9	35.7
DOBF	72.2	56.6	63.7	67.5	73.1	52.2	67.7	72.8	69.4	71.2	63.5	44.4
TransCoder-ST	73.1	57.0	**66.3**	68.7	86.6	68.5	70.0	73.0	69.5	71.9	68.3	58.1
Seq2Seq+Attn.	50.9	53.6	55.2	56.6	51.5	28.9	29.5	44.0	13.5	29.3	18.0	1.5
Transformer	38.5	35.3	40.7	41.2	42.0	2.59	40.6	50.9	20.4	38.5	19.9	1.7
CodeGPT	64.9	53.2	52.7	59.3	65.9	41.8	49.2	54.9	48.5	51.3	47.3	31.1
CodeGPT-adapted	67.4	56.3	55.1	62.0	68.8	50.4	59.0	62.6	56.1	59.7	49.8	35.9
CodeBERT	52.0	45.6	41.5	48.9	45.5	10.4	45.4	54.9	32.6	45.0	25.7	4.2
GraphCodeBERT	58.6	49.6	46.9	54.5	46.8	18.3	51.9	58.9	37.4	50.4	27.0	10.0
PLBART	**79.9**	**64.9**	64.8	**73.2**	**88.4**	68.9	80.5	**78.6**	67.4	76.8	70.1	57.5
CodeT5	79.4	64.1	63.2	72.5	83.8	61.0	79.0	77.1	67.7	75.9	64.3	52.7
TransCoder-ST	79.3	64.2	64.7	72.9	87.5	**69.4**	**81.4**	**78.6**	**72.1**	**78.4**	**73.7**	**62.0**

Table 3: Evaluation results based on the data released by Lachaux et al. (2020) for Java-Python function translation. SM, DM, CB, EA, and CA stand for Syntax Match, Dataflow Match, CodeBLEU, Execution Accuracy, and Computational Accuracy, respectively.

Note that the zero EA score of TransCoder, DOBF, and TransCoder-ST in Python to Java translation is due to not generating a class correctly that fails execution of all translated programs.

Function Translation The performance comparison of all the experiment models is presented in Table 3. Apart from models trained from scratch, CodeBERT, and GraphCodeBERT, all the models perform well in terms of match-based metrics, execution, and computational accuracy. Overall, the best-performing model is fine-tuned TransCoder-ST, and PLBART is the closest competitor model.

3.5 Analysis

Execution-based Evaluation Breakdown We present the breakdown for the test-case-based evaluation in Table 4 (in the Appendix). We present the number of success, failure, and error counts. For program translation evaluation, AVATAR consists of 250 evaluation examples with unit tests. For function translation evaluation, we use the test examples released by (Lachaux et al., 2020). Among the examples, 464 Java to Python and 482 Python to Java examples have test cases. We further present the compilation and runtime error breakdown in Table 5 (in the Appendix).

To analyze program translation errors, we manually examine the errors made by PLBART. We observe that PLBART does not generate the import statements in Java properly, resulting in many failures to find symbols (*e.g.,* StringTokenizer, BufferedReader). Moreover, a quick look at the error made by all models reveals that *type mismatch* is one of the primary causes of compilation errors in all the models. We also notice that models fail to translate longer programs.

Qualitative Examples We demonstrate a couple of qualitative examples of Java to Python program translation by PLBART in Figure 1. We observe that PLBART correctly translates Java API `Math.pow()` to `pow()` in Python. We also observe that PLBART learns to translate a class with a function in Java to a function only in Python.

In Figure 2, we present an example of Python to Java program translation. We see PLBART fail to translate correctly. We notice PLBART unnecessarily generates `InputReader` class that uses `BufferedReader` to read from standard input. Furthermore, we observed another behavior: when translating from Python to Java, PLBART generates classes with the name either `Main` or `GFG`. This is presumably due to the generic class name used in many programming solutions and `GeeksforGeeks` examples.

We present qualitative examples of Java to Python and Python to Java function translation by PLBART in Figure 3 and 4. Overall, we observe a pretty good quality of translations, although there are translations that do not pass all the unit tests, as demonstrated by the performance in terms of computational accuracy in the main result.

4 Related Works

Several works in the past have contributed to building a parallel corpus for source code translation. Nguyen et al. (2013) curated the first parallel corpus of Java and C# functions by developing a semi-automatic tool to search for similar class names and method signatures from two open source projects, `Lucene` and `Db4o`. Similarly, Karaivanov et al. (2014) built a mining tool that uses the Java and C# ANTLR grammar to search for similar methods from five open source projects - `Db4o`, `Lucene`, `Hibernate`, `Quartz`, and `Spring`. Subsequent works used libraries and transcompilers to construct parallel corpus. For example, Aggarwal et al. (2015) used `2to3`, a Python library[9] and Chen et al. (2018) used a transcompiler to create a parallel corpus between Python 2 – Python 3 and CoffeeScript – Javascript, respectively. Recently, Lachaux et al. (2020) collected programming problem solutions in Java, Python, and C++ (∼850 functions in each language) from `GeeksforGeeks` to evaluate their proposed translation model. Concurrent works (CodeGeeX, 2022; Athiwaratkun et al., 2023) present unit tests-based benchmarks to evaluate zero-shot translation capabilities of large language models. Different from these works, we propose a sizeable parallel corpus of Java and Python programs by collecting programming problem solutions from competitive programming sites, online platforms, and open-source repositories.

5 Conclusion

This work proposes a parallel corpus of Java and Python programs to contribute to the development of translation systems for programming languages that have a sizeable impact on software development. We evaluate several neural machine translation systems on the proposed dataset and perform analysis to reveal crucial factors that affect program translation accuracy. In our future work, we want to increase the size of the parallel corpus and support more programming languages.

Limitations

The proposed benchmark has a few limitations. First, AVATAR has a smaller training data size which limits training deep neural models from scratch. Second, the dataset covers only two programming languages. Third, AVATAR includes parallel examples of programs and functions that mostly focus on the use of data structures and algorithms. On the other hand, most software developers write programs as part of software projects that include API dependencies. Therefore, it is unknown whether AVATAR could facilitate program or function translation for such settings. Due to a lack of computational resources, we could not evaluate large language models (LLMs) (Nijkamp et al., 2023; Fried et al., 2023; CodeGeeX, 2022). Therefore, it is unknown how much AVATAR could bring value for LLMs. However, our code release would help to evaluate LLMs.

Ethics Statement

License The `LeetCode` examples we crawled from the GitHub repository are under an MIT license. On the other hand, `Project Euler` and `Code Jam` examples collected from GitHub do not have any license information. The `AtCoder` and `AIZU` examples are collected from `CodeNet`

[9] https://docs.python.org/2/library/2to3

which is under Apache-2.0 license. We crawl examples from `GeeksforGeeks` and `Codeforces` and release them under CC BY-NC-SA 4.0 license. To use the AVATAR benchmark, we are required to adhere to these licenses strictly.

Carbon Footprint We avoided fine-tuning large models due to computational limitations, resulting in a reduced impact on the environment. We fine-tuned nine models on program and function translation tasks and due to the smaller size of the training data, all jobs took a total of 1–2 days on RTX 2080 Ti GPUs. A total of 100 hours of training in a single RTX 2080 Ti GPU results in approximately 7.5kg of carbon emission into the environment.[10]

Sensitive Information AVATAR composed of parallel programs and functions that do not have any natural language (NL) comments or docstring. We remove them to get rid of any personally identifiable information or offensive content. However, there could still be such content in the form of *string* as we do not manually check each example.

Acknowledgements

We thank the anonymous reviewers for their insightful comments.

References

Karan Aggarwal, Mohammad Salameh, and Abram Hindle. 2015. Using machine translation for converting python 2 to python 3 code. Technical report, PeerJ PrePrints.

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4998–5007, Online. Association for Computational Linguistics.

Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified pre-training for program understanding and generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2668, Online. Association for Computational Linguistics.

Marie anne Lachaux, Baptiste Roziere, Marc Szafraniec, and Guillaume Lample. 2021. DOBF: A deobfuscation pre-training objective for programming languages. In *Advances in Neural Information Processing Systems*.

[10]Estimations were conducted using the MachineLearning Impact calculator presented in (Lacoste et al., 2019). We use Amazon Web Services as the provider.

Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng Li, Yuchen Tian, Ming Tan, Wasi Uddin Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Kumar Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, Arash Farahani, Siddhartha Jain, Robert Giaquinto, Haifeng Qian, Murali Krishna Ramanathan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, Sudipta Sengupta, Dan Roth, and Bing Xiang. 2023. Multi-lingual evaluation of code generation models. In *The Eleventh International Conference on Learning Representations*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree neural networks for program translation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

CodeGeeX. 2022. Codegeex: A multilingual code generation model. `http://keg.cs.tsinghua.edu.cn/codegeex`.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1536–1547, Online. Association for Computational Linguistics.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. In *The Eleventh International Conference on Learning Representations*.

Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Jian Yin, Daxin Jiang, et al. 2021. Graphcodebert: Pre-training code representations with data flow. In *International Conference on Learning Representations*.

Svetoslav Karaivanov, Veselin Raychev, and Martin Vechev. 2014. Phrase-based statistical translation of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, pages 173–184.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. In *Advances in Neural Information Processing Systems*, volume 33, pages 20601–20611. Curran Associates, Inc.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. 2019. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. 2021. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Kawser Wazed Nafi, Tonny Shekha Kar, Banani Roy, Chanchal K Roy, and Kevin A Schneider. 2019. Clcdsa: cross language code clone detection using syntactical features and api documentation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1026–1037. IEEE.

Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. 2013. Lexical statistical machine translation for language migration. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 651–654.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. 2021. Codenet: A large-scale AI for code dataset for learning a diversity of coding tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Baptiste Roziere, Jie Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. 2022. Leveraging automated unit tests for unsupervised code translation. In *International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Models	Java to Python					Python to Java				
	#Tests	Error	Failure	Timeout	Success	#Tests	Error	Failure	Timeout	Success
Program Translation										
TransCoder	250	53	197	0	0	250	250	0	0	0
DOBF	250	62	188	0	0	250	250	0	0	0
TransCoder-ST	250	55	195	0	0	250	250	0	0	0
Seq2Seq+Attn.	250	143	98	2	7	250	218	30	0	2
Transformer	250	156	92	1	1	250	246	3	0	1
CodeGPT	250	140	102	1	7	250	169	76	0	5
CodeGPT-adapted	250	119	121	4	6	250	245	3	0	2
CodeBERT	250	189	57	3	1	250	248	2	0	0
GraphCodeBERT	250	93	147	5	5	250	216	34	0	0
PLBART	250	102	124	7	17	250	241	6	1	2
CodeT5	250	111	119	5	15	250	226	20	0	4
TransCoder-ST	250	135	92	9	14	250	194	51	0	5
Function Translation										
TransCoder	464	143	89	4	228	482	198	106	6	172
DOBF	464	125	88	9	242	482	176	88	4	214
TransCoder-ST	464	62	79	5	318	482	153	48	1	280
Seq2Seq+Attn.	464	225	97	8	134	482	395	77	3	7
Transformer	464	269	170	13	12	482	386	83	5	8
CodeGPT	464	158	103	9	194	482	254	74	4	150
CodeGPT-adapted	464	145	78	7	234	482	242	64	3	173
CodeBERT	464	253	149	14	48	482	358	94	10	20
GraphCodeBERT	464	247	118	14	85	482	352	80	2	48
PLBART	464	54	91	4	315	482	144	58	3	277
CodeT5	464	75	97	9	283	482	172	51	5	254
TransCoder-ST	464	58	79	5	322	482	127	51	5	299

Table 4: Breakdown of the success, error, failure, and timeout in the execution based evaluation. #Tests indicates the number of evaluation examples with unit tests. While success indicates the number of examples passing all the unit tests, Failure indicates number of examples that did not at least one of the unit tests. The Error count indicates number of examples with compilation and runtime errors.

Models	Java to Python			Python to Java		
	#Tests	CE	RE	#Tests	CE	RE
TransCoder	464	0%	30.8%	482	31.3%	9.8%
DOBF	464	0%	26.9%	482	27.4%	9.1%
TransCoder-ST	464	0%	13.4%	482	24.9%	6.9%
Seq2Seq+Attn.	464	0%	48.5%	482	80.3%	1.5%
Transformer	464	0%	58.0%	482	78.0%	2.1%
CodeGPT	464	0%	34.1%	482	49.4%	2.7%
CodeGPT-adapted	464	0%	31.3%	482	46.7%	3.3%
CodeBERT	464	0%	54.5%	482	71.4%	2.7%
GraphCodeBERT	464	0%	53.2%	482	71.8%	1.2%
PLBART	464	0%	11.6%	482	25.3%	4.6%
CodeT5	464	0%	16.2%	482	32.4%	3.1%
TransCoder-ST	464	0%	12.5%	482	22.6%	3.7%

Table 5: Compilation error (CE) vs. runtime error (RE) percentage in function translation.

INPUT PROGRAM IN JAVA

```
class GFG {
  static int cntSquares ( int n ) {
    int squares = 0 ;
    for ( int i = 1 ; i <= n ; i ++ ) {
      squares += Math . pow ( i , 2 ) ;
    }
    return squares ;
  }
  public static void main ( String args [ ] ) {
    int n = 4 ;
    System . out . print ( cntSquares ( 4 ) ) ;
  }
}
```

JAVA TO PYTHON TRANSLATION

```
def cntSquares ( n ) :
  squares = 0
  for i in range ( 1 , n + 1 ) :
    squares += pow ( i , 2 )
  return squares

n = 4
print ( cntSquares ( 4 ) )
```

INPUT PROGRAM IN JAVA

```
import java . util . Scanner ;

public class Main {
  public static void main ( String args [ ] ) throws Exception {
    Scanner sc = new Scanner ( System . in ) ;
    String s = sc . next ( ) ;
    int L = s . length ( ) ;
    int same = 0 ;
    if ( s . charAt ( 0 ) == s . charAt ( L - 1 ) ) {
      same = 1 ;
    }
    if ( ( L + same ) % 2 == 0 ) {
      System . out . println ( "Second" ) ;
    }
    else {
      System . out . println ( "First" ) ;
    }
  }
}
```

JAVA TO PYTHON TRANSLATION

```
s = input ( )
if s [ 0 ] == s [ - 1 ] :
  print ( "First" )
elif s [ 0 ] == s [ - 1 ] :
  print ( "Second" )
else :
  print ( "First" )
```

Figure 1: Qualitative examples of program translation from Java to Python by PLBART.

INPUT PROGRAM IN PYTHON

```
S = input ( )
if S [ 0 ] == S [ - 1 ] :
  if len ( S ) % 2 == 0 :
    print ( "First" )
  else :
    print ( "Second" )
else :
  if len ( S ) % 2 == 1 :
    print ( "First" )
  else :
    print ( "Second" )
```

PYTHON TO JAVA TRANSLATION

```
import java . io . OutputStream ;
import java . io . IOException ;
import java . io . InputStream ;
import java . io . PrintWriter ;
import java . util . InputMismatchException ;
import java . io . IOException ;
import java . io . InputStream ;
import java . io . InputStream ;

public class Main {
  public static void main ( String [ ] args ) {
    InputStream inputStream = System . in ;
    OutputStream outputStream = System . out ;
    InputReader in = new InputReader ( inputStream ) ;
    PrintWriter out = new PrintWriter ( outputStream ) ;
    Task solver = new Task ( ) ;
    solver . solve ( 1 , in , out ) ;
    out . close ( ) ;
  }
  static class Task {
    public void solve ( int testNumber , InputReader in , PrintWriter out ) {
      String s = in . next ( ) ;
      if ( s . length ( ) % 2 == 0 ) {
        out . println ( "First" ) ;
      }
      else {
        out . println ( "Second" ) ;
      }
    }
  }
  static class InputReader {
    public BufferedReader reader ;
    public StringTokenizer tokenizer ;
    public InputReader ( InputStream stream ) {
      reader = new BufferedReader ( new InputStreamReader ( stream ) , 32768 ) ;
      tokenizer = null ;
    }
    public String next ( ) {
      while ( tokenizer == null || ! tokenizer . hasMoreTokens ( ) ) {
        try {
          tokenizer = new StringTokenizer ( reader . readLine ( ) ) ;
        }
        catch ( IOException e ) {
          throw new InputMismatchException ( ) ;
        }
      }
      return tokenizer . nextToken ( ) ;
    }
    public int nextInt ( ) {
      return Integer . parseInt ( next ( ) ) ;
    }
  }
}
```

Figure 2: A qualitative examples of program translation from Python to Java by PLBART.

INPUT FUNCTION IN JAVA

```java
static int smallest ( int x , int y , int z ) {
  if ( ( y / x ) != 1 ) return ( ( y / z ) != 1 ) ? y : z ;
  return ( ( x / z ) != 1 ) ? x : z ;
}
```

TRANSLATION IN PYTHON BY PLBART

```python
def smallest ( x , y , z ) :
  if ( ( y // x ) != 1 ) :
    return y
  if ( ( y // z ) != 1 ) :
    return x
  return z
```

INPUT FUNCTION IN JAVA

```java
static int maxvolume ( int s ) {
  int length = s / 3 ; s -= length ;
  int breadth = s / 2 ;
  int height = s - breadth ;
  return length * breadth * height ;
}
```

TRANSLATION IN PYTHON BY PLBART

```python
def maxvolume ( s ) :
  length = s // 3
  s -= length
  breadth = s // 2
  height = s - breadth
  return length * breadth * height
```

INPUT FUNCTION IN JAVA

```java
static void printClosest ( int arr [ ] , int n , int x ) {
  int res_l = 0 , res_r = 0 ;
  int l = 0 , r = n - 1 , diff = Integer . MAX_VALUE ;
  while ( r > l ) {
    if ( Math . abs ( arr [ l ] + arr [ r ] - x ) < diff ) {
      res_l = l ;
      res_r = r ;
      diff = Math . abs ( arr [ l ] + arr [ r ] - x ) ;
    }
    if ( arr [ l ] + arr [ r ] > x ) r -- ;
    else l ++ ;
  }
  System . out . println ( " The closest pair is " + arr [ res_l ] + " and " + arr
    [ res_r ] ) ;
}
```

TRANSLATION IN PYTHON BY PLBART

```python
def printClosest ( arr , n , x ) :
  res_l = 0
  res_r = 0
  l = 0
  r = n - 1
  diff = sys . maxsize NEW_LINE while ( r > l ) :
    if ( abs ( arr [ l ] + arr [ r ] - x ) < diff ) :
      res_l = l
      res_r = r
      diff = abs ( arr [ l ] + arr [ r ] - x )
    if ( arr [ l ] + arr [ r ] > x ) :
      r -= 1
    else :
      l += 1
  print ( " The closest pair is " , arr [ res_l ] , " and " , arr [ res_r ] )
```

Figure 3: Qualitative examples of function translation from Java to Python by PLBART.

INPUT FUNCTION IN PYTHON

```python
def minimum_cost ( a , n ) :
  mn = sys . maxsize
  sum = 0
  for i in range ( n ) :
    mn = min ( a [ i ] , mn )
    sum += a [ i ]
  return mn * ( sum - mn )
```

TRANSLATION IN JAVA BY PLBART

```java
static int minimum_cost ( int a [ ] , int n ) {
  int mn = Integer . MAX_VALUE ;
  int sum = 0 ;
  for ( int i = 0 ; i < n ; i ++ ) {
    mn = Math . min ( a [ i ] , mn ) ;
    sum += a [ i ] ;
  }
  return mn * ( sum - mn ) ;
}
```

INPUT FUNCTION IN PYTHON

```python
def nextPowerOf2 ( n ) :
  p = 1
  if ( n and not ( n & ( n - 1 ) ) ) :
    return n
  while ( p < n ) :
    p <<= 1
  return p
```

TRANSLATION IN JAVA BY PLBART

```java
static int nextPowerOf2 ( int n ) {
  int p = 1 ;
  if ( n && ! ( n & ( n - 1 ) ) )
    return n ;
  while ( p < n )
    p <<= 1 ;
  return p ;
}
```

INPUT FUNCTION IN PYTHON

```python
def isPerfectSquare ( n ) :
  i = 1
  the_sum = 0
  while the_sum < n :
    the_sum += i
    if the_sum == n :
      return True
    i += 2
  return False
```

TRANSLATION IN JAVA BY PLBART

```java
static boolean isPerfectSquare ( int n ) {
  int i = 1 , the_sum = 0 ;
  while ( the_sum < n ) {
    the_sum += i ;
    if ( the_sum == n )
      return true ;
    i += 2 ;
  }
  return false ;
}
```

Figure 4: Qualitative examples of function translation from Python to Java by PLBART.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Section after conclusion

☑ A2. Did you discuss any potential risks of your work?
In the limitations section.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and introduction.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 2

☑ B1. Did you cite the creators of artifacts you used?
Section 2

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Ethics Statement

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Ethics Statement

☑ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Ethics Statement

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Section 2

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 2

C ☑ Did you run computational experiments?

Section 3 and Appendix

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Ethics Statement

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
We will release the source code.

☒ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
We will release the source code.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.