

OptCodeTrans: Boost LLMs on Low-Resource Programming Language Translation

Jianbo Lin¹, Yi Shen¹, Chuanyi Li^{1*}, Changan Niu¹, Bin Luo¹

¹ National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

* Corresponding author

{linjianbo, yishen}@smail.nju.edu.cn, {lcy, luobin}@nju.edu.cn, niu.ca@outlook.com

Abstract—Program translation aims to translate source code from one programming language (PL) to another. Current research on code translation predominantly focuses on high-resource PLs like Python and Java, leaving low-resource languages insufficiently explored. Fortunately, the rapid advancement of Large Language Models (LLMs) has created new opportunities for research on low-resource PLs.

To mitigate this gap in the era of foundation models, we introduce OptCodeTrans, a two-phase post-training approach involving continued pre-training and instruction fine-tuning. We provide a high-quality dataset of three low-resource languages representing different programming paradigms, including Cangjie, Julia, and OCaml.

Our work provides valuable insights into effective post-training strategies for adapting LLMs to low-resource code translation tasks. Extensive experiments demonstrate the effectiveness of OptCodeTrans, achieving an average improvement of 10.28 in BLEU and 5.15 in functional equivalence across all translation tasks and backbone models.

Index Terms—Code Translation, Low-resource Programming Language, Post-training

I. INTRODUCTION

Code translation involves converting source code between high-level programming languages (e.g., Java $\rightarrow$ C++), meanwhile preserving its original functionality and semantics. Code translation is important in many scenarios and addresses several critical software engineering challenges [1, 2]. This includes migrating legacy software from outdated PLs to modern ones, supporting cross-platform development through translating code between platform-specific PLs and translating libraries from high-resource PLs to help ecosystems of low-resource PLs.

The emergence of foundation models has fundamentally transformed natural language processing (NLP), particularly in machine translation [3–6]. Similarly, code LLMs [7–10] have also introduced new opportunities for code translation, especially for low-resource PLs that were previously challenging to support.

Prior research in code translation has predominantly focused on a small number of high-resource PLs [11–14] and research on low-resource PLs is not comprehensive [15, 16]. Due to various limitations, researchers have not thoroughly explored code translation of low-resource PLs across programming paradigms, resulting in a substantial knowledge gap.

We study code translation capabilities of code LLMs across different programming paradigms, including imperative, object-oriented and functional, focusing on three low-resource programming languages: Cangjie¹, Julia² and OCaml³.

We selected these three languages for several key reasons. First, they represent programming paradigms that have been largely overlooked in current low-resource code translation research. Second, we were able to construct high-quality datasets. For Cangjie, we carefully curated the dataset manually, while for Julia and OCaml, we leveraged existing resources[17]. Finally, these three languages all have a significant impact: Cangjie is the primary language for application development in HarmonyOS, an ecosystem powering over 4,000 apps on 800 million devices; Julia combines Python-like accessibility with C-level performance for scientific computing; and OCaml is valued in industry for formal verification, compiler design, and financial systems.

In this study, we aim to explore how to efficiently post-train LLMs for low-resource code translation tasks. We propose a new post-training recipe called OptCodeTrans to help adapt LLMs to these tasks for low-resource PLs in the post-training stage. This recipe first undergoes continued pre-training on the target low-resource languages, followed by instruction fine-tuning on multiple translation directions. To make this post-training process possible, we collected a high-quality dataset across these three low-resource PLs: Cangjie, Julia, and OCaml. We conduct comprehensive experiments to evaluate our post-training recipe using both n-gram matching-based automated metric and test-based metric⁴.

Our contributions include:

(1) OptCodeTrans: An effective post-training recipe for adapting pre-trained LLMs to low-resource code translation tasks. Through detailed experiments, we demonstrate the effectiveness of OptCodeTrans and analyze the impact of the two stages of our recipe.

(2) Low-resource Code Translation Dataset: We constructed a high-quality dataset for the three PLs studied, including parallel corpora and monolingual corpora for each. This dataset facilitates future research on low-resource language

¹<https://cangjie-lang.cn/en>

²<https://julialang.org/>

³<https://ocaml.org/>

⁴Our code and dataset are available at <https://github.com/brick-pid/OptCodeTrans>

code translation.

(3) LowTransEval: We established a test-based benchmark for low-resource code translation based on existing code generation benchmark. For Cangjie, we manually translated both the prompts and test suites from HumanEval [18]. For the remaining two languages, we leveraged existing translations available [19].

II. BACKGROUND & RELATED WORK

A. Code Translation

Existing code translation datasets [11, 14, 15] primarily focus on commonly used PLs, overlooking low-resource PLs. Moreover, most data are collected from programming contest or tutorial sites [14, 15], neglecting the needs of real-world development scenarios.

Many methods have been proposed to improve the performance of code translation [12, 13, 15, 20]. However, these previous methods exhibit misalignment with current prevalent decoder-only architecture of LLMs. For example, TransCoder [12] adopts unsupervised masking and denoising training tasks.

Therefore, we need to rethink low-resource code translation in the era of foundation models. Recently, researchers investigate how to better work with code LLMs. Pan et al. [21] analyzed the limitations and capability boundaries of code LLMs in code translation tasks.

B. Post-training of Foundation Models

Research has centered on optimizing instruction fine-tuning to improve instruction following and task generalization [22–24], alongside alignment-focused methods for better matching models to human preferences [25, 26].

Some studies have shown that multi-task instruction fine-tuning can enhance the performance of original tasks through knowledge transfer [27]. Furthermore, research indicates that knowledge transfer among PLs is feasible [28, 29].

Beyond instruction fine-tuning and alignment, continued pre-training plays a pivotal role in post-training, especially for low-resource languages underrepresented in the pre-training process of LLMs. Continued pre-training involves training a pre-trained model on domain-specific unlabeled corpora before fine-tuning. Research has demonstrated its effectiveness in enhancing downstream task performance [30, 31], with recent work exploring more efficient strategies and hyperparameters to optimize this phase for LLMs [32, 33].

III. LOW-RESOURCE CODE TRANSLATION DATASET

In this section, we describe our low-resource PLs dataset, which includes: (1) A parallel corpus consisting of semantically equivalent code pairs between PLs; (2) A monolingual corpus containing large-scale function-level code for continued pre-training. The process is illustrated in Figure 1.

A. Cangjie Dataset

1) *Cangjie Parallel Corpus*: We construct a Java-Cangjie parallel corpus by mining open-source Java libraries that have been systematically ported to Cangjie by the community. Utilizing real-world projects, rather than synthetic code, ensures our dataset’s practical applicability.

Our parallel code pairs data collection pipeline consists of the following steps:

(1) *Parallel Repositories Identification*. For each available Cangjie repository, we check for explicit reference to the original Java library. In cases direct references are absent, we manually search on code hosting platforms (e.g., GitHub, Google Code) to find matched Java repositories.

(2) *Code Pair Extraction*. For each identified repository pair, we analyze the semantic correspondence between Cangjie and Java to find matched pairs. This analysis results in a collection of semantically equivalent Java-Cangjie code pairs.

(3) *Code Comment Collection*. We extracted comments that precede Cangjie function signatures as natural language descriptions for the parallel pairs. For functions lacking documentation, we use ChatGPT to generate descriptive comments.

(4) *Code Filtering*. To enhance data quality, we filter out semantically trivial functions. Specifically, we remove boilerplate code and trivial functions, such as getter/setter methods.

The resulting parallel dataset contains 2,234 Java-Cangjie code pairs sourced from 17 open-source projects, providing a balanced representation of real-world programming patterns.

2) *Cangjie Monolingual Corpus*: We collected all available Cangjie repositories as data source. After excluding repositories with limited utility (e.g., those related to character encoding), we obtained 418 repositories containing 11,764 source files. From these files, we extracted both the function implementations and their associated documentation (i.e. comments and docstrings). The resulting corpus contains 5.8 million tokens processed with CodeT5 tokenizer [7].

B. Julia and OCaml Dataset

For OCaml and Julia, we leverage existing community datasets to construct high-quality parallel code pairs with semantically equivalent program snippets. We utilize MultiPL-T [17], an high-quality semi-synthetic low-resource code generation dataset, to build our parallel corpus. MultiPL-T translates a Python dataset into Julia and OCaml, each instance with natural language documentation and a validated set of tests with high coverage. We pair these Julia and OCaml translations with the original Python code to create parallel corpus.

We partition the dataset to create parallel and monolingual corpora. To ensure fair comparative evaluation across PLs, we sample 2,234 code pairs (matching the size of the Cangjie parallel corpus), and construct a monolingual corpus of 5.8 million tokens from the remaining data.

IV. APPROACH

In this section, we detail OptCodeTrans, a post-training recipe for adapting code LLMs on low-resource code translation task. Our recipe consists of two stages: the first is

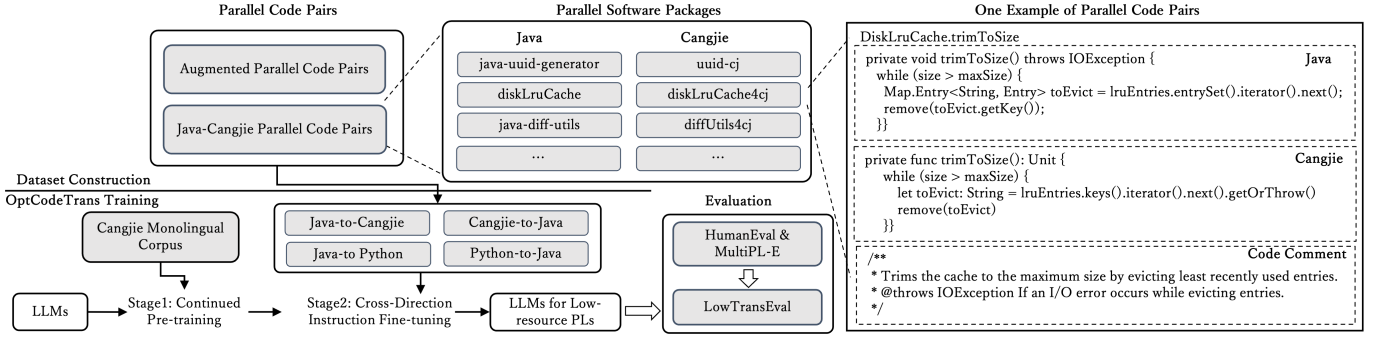


Fig. 1. Our framework consists of three components: (1) dataset construction combining parallel and monolingual corpora, (2) OptCodeTrans, a post-training recipe for low-resource programming language translation, and (3) LowTransEval, a test-based evaluation benchmark. Here we take Cangjie language as an example to describe our framework.

TABLE I
CROSS-DIRECTION INSTRUCT TUNING

Translation Direction	Augmented Direction
Java to Cangjie	Cangjie to Java Python to Java Java to Python
Python to Julia	Julia to Python Python to R R to Python
Python to OCaml	OCaml to Python Python to Racket Racket to Python

continued pre-training with monolingual function-level corpus; the second is cross-direction instruction fine-tuning with high quality parallel code pairs.

Continued Pre-training (CPT): We use continued pre-training with monolingual function-level corpora to improve model’s knowledge about the target low-resource PLs. Our CPT stage utilizes function-level corpora rather than complete file code, aiming to bridge the gap between the pre-trained model and the target function-level translation task. Existing research suggests that employing data that closely resembles the downstream task in continued pre-training stage can improve efficiency [34]. Drawing inspiration from this, our continued pre-training corpus comprises function-level data that mirrors our downstream translation task, facilitating more precise distribution adaptation.

Cross-direction Instruction Fine-tuning (Cross-IFT): After the CPT stage, we fine-tuned the model with high-quality parallel data, as described in Section III. Due to the challenges associated with collecting parallel corpora and considering the similarities and knowledge sharing across translation tasks, we augmented each translation direction with three additional translation directions, as shown in Table I. Through cross-direction instruction fine-tuning, the model can improve translation performance for low-resource PLs via knowledge transferring. All translation tasks were presented in the same instruction format as follows:

```
### Translate this from [src] to [tgt]
```

```
[src]: [function in src]
### Translate to [tgt]:
[ground truth function in tgt]
```

For the augmented direction, we leveraged existing parallel code translation datasets. For the Java-Python data, we extracted 2,758 parallel code pairs from AVATAR [14]. Similarly, for Python-R and Python-Racket, we sampled equal number of samples from MultiPL-T [17] for augmentation.

V. EVALUATION

A. Research Questions

In this study, we aim to evaluate whether the proposed post-training recipe is helpful. To do so, we formulate the following research questions:

RQ1: How does OptCodeTrans perform? We evaluate the performance of OptCodeTrans on different LLM backbones and across different PLs across paradigms. Our evaluation employs both automatic metric and test-based metric.

RQ2: How do the two stages contribute to OptCodeTrans? We conduct ablation studies by removing each stage to analyze their individual contributions.

B. Metrics

We employ BLEU [35] and functional equivalence (FE) [36] for evaluation, each metric focuses on different aspects of code translation quality.

(1) BLEU: It measures the similarity between translated and reference code by evaluating syntactic and lexical correspondence. However, BLEU exhibits limitations when applied to code translation tasks [37, 38].

(2) Functional Equivalence (FE): It measures the correctness and functionality of the translation. Given the same input, the source code and the translated code should produce the same output (i.e. pass the same test suite). FE effectively complements the limitations of BLEU.

C. LowTransEval Benchmark

To evaluate code translation models with test-based metrics (i.e. FE), we adapted the popular code generation benchmark HumanEval [18] for low-resource PLs. We translated original

TABLE II
RESULTS OF THE MAIN EXPERIMENT

Methods	Java to Cangjie		Python to Julia		Python to OCaml	
	BLEU	FE	BLEU	FE	BLEU	FE
<i>CodeT5+ 770M Backbone</i>						
SFT	63.59	20.73	49.79	7.55	47.65	1.94
CPT + SFT	71.14	31.09	52.50	7.55	49.46	2.58
Cross-IFT	68.25	35.37	66.63	8.80	54.45	2.58
OptCodeTrans	68.20	35.37	69.86	10.06	67.80	4.52
<i>StarCoder2-3B Backbone</i>						
SFT	67.06	39.63	49.55	27.55	54.49	13.90
CPT + SFT	70.15	42.07	47.37	25.46	55.06	13.61
Cross-IFT	69.60	44.51	53.95	27.78	57.08	15.19
OptCodeTrans	71.48	45.73	55.49	29.44	61.02	16.77

prompts and test suites into the target PLs, requiring the model to translate complete functions from the source PL to the target PL and then checking if it passes all test cases.

For Cangjie, we meticulously translated prompts and test suites, excluding infeasible tests due to language constraints, resulting in 155 instances. For OCaml and Julia, we leveraged and adapted existing data from MultiPL-E [19]. This process resulted in the development of LowTransEval, a benchmark for evaluating code translation tasks in low-resource PLs.

D. Backbone Models and Baselines

Our backbone models include mainstream code LLMs: CodeT5+ 770M [8], an effective code LLM of encoder-decoder architecture, and StarCoder2-3B [10], one of the most powerful open-source decoder-only code LLMs.

For comparison, we evaluate OptCodeTrans against the following baseline approaches:

- (1) Supervised Fine-tuning (SFT): Basic fine-tuning using only source-to-target parallel code pairs without additional data augmentation.
- (2) CPT + SFT: CPT stage followed by basic SFT, ablating the Cross-IFT stage of OptCodeTrans.
- (3) Cross-Direction Instruction Fine-tuning: Fine-tuning with extra cross-directional parallel code pairs, ablating the CPT stage of OptCodeTrans.
- (4) OptCodeTrans: Our complete post-training recipe that combines CPT and Cross-Direction Instruction Fine-tuning to leverage the benefits of both stages.

VI. RESULTS

A. RQ1: Evaluation of OptCodeTrans

Overall Performance of OptCodeTrans. We show our main results in Table II. Experimental results demonstrate that OptCodeTrans consistently outperforms all baselines across all translation directions. With the CodeT5+ backbone, our approach significantly enhances translation capabilities, particularly for out-of-distribution languages like Julia and OCaml where the baseline struggles. With the StarCoder2 backbone, we observe even more pronounced improvements, especially in FE metrics. Notably, decoder-only models exhibit stronger performance in FE metrics compared to BLEU scores, suggesting their superior code generation capabilities in producing functionally correct translations rather than structurally similar

ones. The StarCoder2-based implementation achieves best results in FE metrics across all translation directions.

Performance across Programming Languages. We analyze how different PLs and their paradigms influence OptCodeTrans’s performance. As shown in Table II, despite all three being low-resource PLs, the model exhibits notable performance variations. We attribute these differences to the backbone models’ varying familiarity with different programming paradigms. When a low-resource language shares paradigmatic similarities with commonly used languages (e.g., Cangjie’s similarity to Java), the model demonstrates better adaptation capability. Conversely, languages with paradigms less familiar to the backbone model (e.g. OCaml’s functional programming paradigm) present greater adaptation challenges.

Evaluation with BLEU and FE. BLEU and FE complement each other: BLEU provides continuous measurement of syntactic similarity, while FE offers discrete assessment of functional correctness through test case execution. Translation improvements may reflect in BLEU scores before meeting FE’s stricter requirement of full functionality. Therefore, a comprehensive evaluation considering both metrics is necessary to fully assess model performance.

B. RQ2: Impact of each Stage for OptCodeTrans

Impact of CPT Stage. The CPT stage contributes significantly to performance improvement. For OCaml, which represents a programming paradigm less familiar by CodeT5+ or StarCoder2, the ablation of CPT stage results in performance drop of 13 points for CodeT5+ backbone and 4 points for StarCoder2 backbone on BLEU. The importance of the CPT stage correlates with the extent of data distribution shift between the low-resource PL and the pre-training data of LLMs. Notably, stronger foundation models like StarCoder2 demonstrate better adaptation capabilities to new languages and different programming paradigms, likely due to their exposure to a wider variety of PLs during pre-training.

Impact of Cross-IFT Stage. The cross-direction instruction fine-tuning stage contribute to performance improvements in FE scores across all translation directions. However, we observe cases where cross-direction IFT decreases BLEU scores sometimes. This discrepancy suggests two insights: first, BLEU may not be the most reliable indicator of code translation correctness; second, cross-direction instruction fine-tuning might lead to translations that are functionally correct but structurally different from the ground truth.

VII. THREATS TO VALIDITY

Internal Validity. Manual dataset collection may introduce noise. To mitigate this, two annotators with multiple years of programming experience performed mutual validation to ensure data reliability.

External Validity. The language coverage of three PLs may limit the generalization of our findings. And we focuses on the post-training stage of foundation models, limiting comparisons with previous approaches. We plan more comprehensive research in the future study.

VIII. CONCLUSION AND FUTURE WORK

This paper introduces OptCodeTrans, a post-training approach for enhancing code translation in low-resource PLs. We also contribute a high-quality dataset and a evaluation framework, LowTransEval, to facilitate further research in this field. Future work will focus on extending OptCodeTrans to a wider range of low-resource languages and explore more effective post-training strategy for code translation of low-resource PLs.

ACKNOWLEDGMENTS

This work was supported by CCF-Huawei Populus Grove Fund. We want to thank Jiale Bu for his valuable assistance and contributions to this work. We also thank the reviewers for their helpful comments.

REFERENCES

- 1 Zhong, H., Thummalapenta, S., Xie, T., Zhang, L., and Wang, Q., “Mining api mapping for language migration,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, New York, NY, USA, 2010. [Online]. Available: <https://doi.org/10.1145/1806799.1806831>
- 2 Nguyen, A. T., Nguyen, T. T., and Nguyen, T. N., “Lexical statistical machine translation for language migration,” in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, New York, NY, USA, 2013. [Online]. Available: <https://doi.org/10.1145/2491411.2494584>
- 3 Lyu, C., Du, Z., Xu, J. *et al.*, “A paradigm shift: The future of machine translation lies with large language models,” in *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation*, 2024. [Online]. Available: <https://aclanthology.org/2024.lrec-main.120>
- 4 Zhang, B., Haddow, B., and Birch, A., “Prompting large language model for machine translation: a case study,” in *Proceedings of the 40th International Conference on Machine Learning*, 2023. [Online]. Available: <https://arxiv.org/abs/2301.07069>
- 5 Zhu, W., Liu, H., Dong, Q., Xu, J. *et al.*, “Multilingual machine translation with large language models: Empirical results and analysis,” in *Findings of the Association for Computational Linguistics: NAACL 2024*, 2024. [Online]. Available: <https://aclanthology.org/2024.findings-naacl.176>
- 6 Hendy, A., Abdelrehim, M. *et al.*, “How good are gpt models at machine translation? a comprehensive evaluation,” *arXiv preprint arXiv:2302.09210*, 2023. [Online]. Available: <https://arxiv.org/abs/2302.09210>
- 7 Wang, Y., Wang, W., Joty, S. R., and Hoi, S. C. H., “CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation,” in *Proceedings of the 26th Conference on Empirical Methods in Natural Language Processing*. Virtual Event / Punta Cana, Dominican Republic: Association for Computational Linguistics, 2021. [Online]. Available: <https://aclanthology.org/2021.emnlp-main.685>
- 8 Wang, Y., Le, H., Gotmare, A., Bui, N., Li, J., and Hoi, S., “CodeT5+: Open Code Large Language Models for Code Understanding and Generation,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. [Online]. Available: <https://aclanthology.org/2023.emnlp-main.68>
- 9 Li, R., allal, L. B., Zi, Y., Muennighoff, N. *et al.*, “Starcoder: may the source be with you!” *Transactions on Machine Learning Research*, 2023. [Online]. Available: <https://openreview.net/forum?id=KoFOg41haE>
- 10 Lozhkov, A., Li, R., Allal, L. B., Cassano, F. *et al.*, “Starcoder 2 and the stack v2: The next generation,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.19173>
- 11 Lu, S., Guo, D., Ren, S., Huang, J. *et al.*, “Codexglue: A machine learning benchmark dataset for code understanding and generation,” in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. [Online]. Available: <https://arxiv.org/abs/2102.04664>
- 12 Roziere, B., Lachaux, M.-A., Chausson, L., and Lample, G., “Unsupervised translation of programming languages,” in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2020. [Online]. Available: <https://dl.acm.org/doi/10.5555/3495724.3497454>
- 13 Roziere, B., Zhang, J., Charton, F., Harman, M., Synnaeve, G., and Lample, G., “Leveraging automated unit tests for unsupervised code translation,” in *International Conference on Learning Representations*, 2022. [Online]. Available: <https://openreview.net/forum?id=cmt-6KtR4c4>
- 14 Ahmad, W. U., Tushar, M. G. R., Chakraborty, S., and Chang, K.-W., “AVATAR: A Parallel Corpus for Java-Python Program Translation,” in *Findings of the Association for Computational Linguistics*, 2023. [Online]. Available: <https://aclanthology.org/2023.findings-acl.143>
- 15 Zhu, M., Suresh, K., and Reddy, C. K., “Multilingual Code Snippets Training for Program Translation,” *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022. [Online]. Available: <https://doi.org/10.1609/aaai.v36i10.21434>
- 16 Yan, W., Tian, Y., Li, Y., Chen, Q., and Wang, W., “CodeTransOcean: A comprehensive multilingual benchmark for code translation,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. [Online]. Available: <https://aclanthology.org/2023.findings-emnlp.337>
- 17 Cassano, F., Gouwar, J., Lucchetti, F. *et al.*, “Knowledge transfer from high-resource to low-resource programming languages for code llms,” *OOPSLA*, vol. 8, 2024. [Online]. Available: <https://doi.org/10.1145/3689735>
- 18 Chen, M., Tworek, J., Jun, H. *et al.*, “Evaluating large language models trained on code,” *arXiv preprint arXiv:2107.03374*, 2021. [Online]. Available: <https://arxiv.org/abs/2107.03374>

- 19 Cassano, F., Gouwar, J., Nguyen, D. *et al.*, “Multipl-e: A scalable and polyglot approach to benchmarking neural code generation,” *IEEE Trans. Softw. Eng.*, 2023. [Online]. Available: <https://doi.org/10.1109/TSE.2023.3267446>
- 20 Szafraniec, M., Roziere, B., Charton, H. L. F., Labatut, P., and Synnaeve, G., “Code translation with compiler representations,” in *International Conference on Learning Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=XomEU3eNeSQ>
- 21 Pan, R., Ibrahimzada, A. R., Krishna, R., Sankar, D. *et al.*, “Lost in translation: A study of bugs introduced by large language models while translating code,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024. [Online]. Available: <http://dx.doi.org/10.1145/3597503.3639226>
- 22 Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V., “Finetuned language models are zero-shot learners,” in *International Conference on Learning Representations*, 2022. [Online]. Available: <https://arxiv.org/abs/2109.01652>
- 23 Sanh, V., Webson, A., Raffel, C. *et al.*, “Multitask prompted training enables zero-shot task generalization,” *arXiv preprint arXiv:2110.08207*, 2021. [Online]. Available: <https://arxiv.org/abs/2110.08207>
- 24 Iyer, S., Lin, X. V., Pasunuru, R., Mihaylov, T., Simig, D., Yu, P., Shuster, K., Wang, T., Liu, Q., Koura, P. S. *et al.*, “Opt-impl: Scaling language model instruction meta learning through the lens of generalization,” *arXiv preprint arXiv:2212.12017*, 2022. [Online]. Available: <https://arxiv.org/abs/2212.12017>
- 25 Ouyang, L., Wu, J., Jiang, X. *et al.*, “Training language models to follow instructions with human feedback,” in *Advances in Neural Information Processing Systems*, 2022. [Online]. Available: <https://arxiv.org/abs/2203.02155>
- 26 Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C., “Direct preference optimization: Your language model is secretly a reward model,” in *Advances in Neural Information Processing Systems*, 2023. [Online]. Available: <https://arxiv.org/abs/2305.18290>
- 27 McCann, B., Keskar, N. S., Xiong, C., and Socher, R., “The natural language decathlon: Multitask learning as question answering,” 2019. [Online]. Available: <https://arxiv.org/abs/1806.08730>
- 28 Orlanski, G., Xiao, K., Garcia, X. *et al.*, “Measuring the impact of programming language distribution,” in *International Conference on Machine Learning*, 2023. [Online]. Available: <https://arxiv.org/abs/2302.01973>
- 29 Zan, D., Yu, A., Shen, B. *et al.*, “Can programming languages boost each other via instruction tuning?” *arXiv preprint arXiv:2308.16824*, 2023. [Online]. Available: <https://arxiv.org/abs/2308.16824>
- 30 Ke, Z., Liu, B., Ma, N., Xu, H., and Shu, L., “Achieving forgetting prevention and knowledge transfer in continual learning,” *Advances in Neural Information Processing Systems*, 2021. [Online]. Available: <https://arxiv.org/abs/2112.02706>
- 31 Ke, Z., Shao, Y., Lin, H., Konishi, T., Kim, G., and Liu, B., “Continual pre-training of language models,” in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: <https://arxiv.org/abs/2302.03241>
- 32 Ibrahim, A., Thérien, B., Gupta, K., Richter, M. L. *et al.*, “Simple and scalable strategies to continually pre-train large language models,” *arXiv preprint arXiv:2403.08763*, 2024. [Online]. Available: <https://arxiv.org/abs/2403.08763>
- 33 Gupta, K., Thérien, B., Ibrahim, A., Richter, M. L., Anthony, Q., Belilovsky, E., Rish, I., and Lesort, T., “Continual pre-training of large language models: How to (re) warm your model?” *arXiv preprint arXiv:2308.04014*, 2023. [Online]. Available: <https://arxiv.org/abs/2308.04014>
- 34 Xie, Y., Aggarwal, K., and Ahmad, A., “Efficient continual pre-training for building domain specific large language models,” *arXiv preprint arXiv:2311.08545*, 2023. [Online]. Available: <https://aclanthology.org/2024.findings-acl.606>
- 35 Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J., “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002. [Online]. Available: <https://doi.org/10.3115/1073083.1073135>
- 36 Xue, M., Andrzejak, A., and Leuther, M., “An interpretable error correction method for enhancing code-to-code translation,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=fVxIEHGnVT>
- 37 Tran, N., Tran, H., Nguyen, S., Nguyen, H., and Nguyen, T., “Does bleu score work for code migration?” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019. [Online]. Available: <http://dx.doi.org/10.1109/ICPC.2019.00034>
- 38 Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S. *et al.*, “Codebleu: a method for automatic evaluation of code synthesis,” 2020. [Online]. Available: <https://arxiv.org/abs/2009.10297>