

# NL in the Middle: Code Translation with LLMs and Intermediate Representations

Chi-en Amy Tai\*, Pengyu Nie, Lukasz Golab, and Alexander Wong

University of Waterloo, Canada

Email: \*amy.tai@uwaterloo.ca

**Abstract**—Studies show that large language models (LLMs) produce buggy code translations. One promising avenue to improve translation accuracy is through intermediate representations, which provide structured guidance for the translation process. We investigate whether LLM-based code translation can benefit from intermediate representations, specifically in the form of natural language (NL) summaries and abstract syntax trees (ASTs). Since prompt engineering greatly affects LLM performance, we consider several ways to integrate these representations, from one-shot to chain-of-thought (CoT) prompting. Using Open GPT4 8X7B and specialized StarCoder and CodeGen models on popular code translation benchmarks (CodeNet and AVATAR), we find that CoT with an intermediate NL summary performs best, with an increase of 13.8% and 6.7%, respectively, in successful translations for the best-performing model (Open GPT4 8X7B) compared to the zero-shot prompt.

**Index Terms**—code translation, large language models, natural language, chain-of-thought prompting, abstract syntax trees

## I. INTRODUCTION

Code translation is the task of converting code from one programming language (e.g., Java) to another (e.g., Python) [1]. This task is important in software as organizations often need to migrate legacy systems to newer technologies, improve code interoperability, or leverage language-specific features and libraries [2]. However, manual code rewriting is a labor-intensive process that is inherently prone to human error, thereby impacting both efficiency and accuracy. While early statistical efforts [3] have recently given way to methods based on large language models (LLMs), studies show that LLMs still produce buggy translations when using a zero-shot prompt [2].

Prompt engineering is one way to improve LLM performance [4]. One such technique is chain-of-thought (CoT) prompting, shown to be effective for reasoning tasks [5]. CoT prompting uses intermediate reasoning steps to break down the problem at hand, which provides more information on the process that was used to arrive at a given answer.

For code translation specifically, improvements in code representation through natural language (NL) and abstract syntax trees (ASTs) have been reported for software engineering tasks [6], [7]. For example, summarizing code to NL leads to better code translation performance [8]. Similarly, employing AST in code summarization outperforms state-of-the-art models [7].

Motivated by these observations, we investigate whether LLM-driven code translation can benefit from NL and AST intermediate representations (IRs) when combined with suitable

prompt engineering. While there has been work on leveraging IRs for translation [8], [9] and on LLM-based translation [2], our work is the first to systematically combine LLMs with both forms of IRs for code translation.

We explore two approaches to incorporate IRs in code translation: (1) a two-step approach, where the LLM first translates the original code to IR and then translates this IR to the target language [8]; and (2) a CoT prompting approach, where the LLM is instructed to use IR to explain its reasoning during translation. We experiment with these two approaches using different permutations of IRs (NL, AST, or both), and compare with the simple zero-shot and one-shot prompt baselines. Experiments using Open GPT4 8X7B and specialized StarCoder and CodeGen models on code translation benchmarks (CodeNet and AVATAR) show that CoT with an intermediate NL representation produces the most successful translations, with an increase of 13.8% and 6.7% on CodeNet and AVATAR, respectively, compared to the zero-shot prompt.

Our main contributions in this work include:

- The first systematic study combining LLMs with natural language (NL) and abstract syntax trees (AST) as intermediate representations (IRs) for code translation.
- Evaluation across multiple LLMs (Open GPT4 8X7B, StarCoder, CodeGen), datasets (CodeNet, AVATAR), and prompting strategies (zero-shot, one-shot, two-step, and CoT).
- Our results show that CoT with NL summaries achieves the best performance, improving code translation accuracy by up to 13.8% compared to the zero-shot baseline.

Our code is available at:

<https://github.com/catai9/nl-in-middle/>

## II. RELATED WORK

We categorize related work into code translation with LLMs and code translation with intermediate steps. Combining these two directions, i.e., code translation with LLMs and intermediate representations, is the novelty of our work.

### A. Code Translation with LLMs

Pan et al. [2] conducted a comprehensive study of LLM-based code translation across five languages (C, C++, Go, Java, and Python). They evaluated multiple models including specialized code LLMs (CodeGen [10], StarCoder [11], and CodeGeeX [12]), general-purpose LLMs, and proprietary

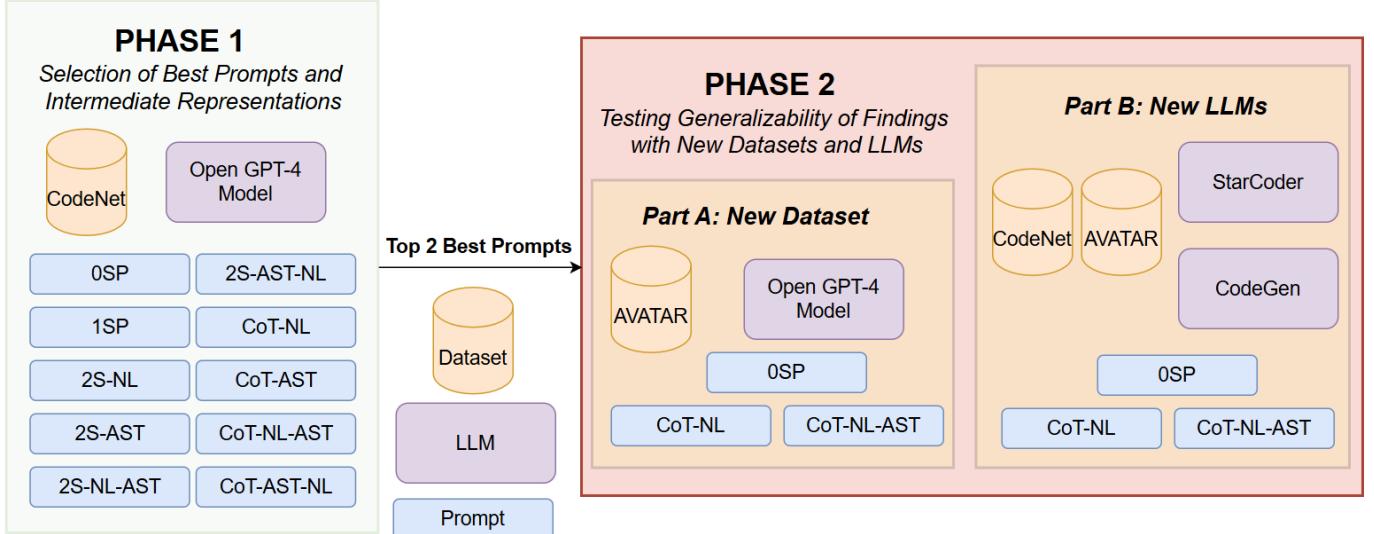


Fig. 1: Overview of our methodology, with Phase 1 exploring ten prompt types and Phase 2 leveraging the two best prompts from Phase 1 to explore the generalization on a new dataset and with new LLMs.

GPT-4 [13]. The authors also defined a metric to capture the percentage of successful translations, requiring the translated code to compile, pass runtime checks, and pass existing tests. They reported successful translation rates ranging from 2.1% to 47.3% across studied LLMs, with GPT-4 achieving the best performance. The authors also adjusted the LLM prompt in the case of unsuccessful translation by providing the previous translation in the prompt, stating that it was incorrect, and asking for a re-generation, which led to more successful translations. However, the use of intermediate representations was not considered.

Prior work has explored two main prompting strategies for LLM-based code translation. Few-shot learning improves model performance by providing custom examples to guide output style [14], [15]; beyond code translation, it has also been applied to other software engineering tasks such as program refactoring and code review [16], [17], [18]. Chain-of-thought (CoT) prompting, originally developed to be effective for math word problems [5], has been adapted for software engineering tasks including vulnerability discovery [19] and software architecture recovery [20]. However, manually creating effective CoT prompts for tasks such as code generation can be challenging [21], [22], which led to the development of three potential adaptions. The first approach, referred to as Structured CoTs, leverages program structures such as loops and branches to systematically create CoTs. This method organizes the reasoning process in a way that mirrors common programming practices, helping to structure and guide the thought process in a coherent manner [21]. The second, COTTON [22], uses lightweight language models for automatic CoT generation. The third, Tree of Thoughts (ToT) [23], explores multiple reasoning paths, but it tends to be slower and less effective for solving GitHub issues [24]. None of these extensions leverage intermediate representations, which

is the focus of this work.

#### B. Code Translation with Intermediate Steps

Prior work has explored several kinds of intermediate representations for code translation. Ahmad et al. [8] added a natural language summary to a sequence-to-sequence model, where code is first summarized and then the summary is used to generate code in the target language. This approach achieved competitive results on GitHub and CodeNet datasets. Szafraniec et al. [25] used a compiler intermediate representation to improve translation performance. Huang et al. [9] used abstract syntax trees (ASTs) as intermediate code representations. ASTs have also proven to be a valuable intermediate step in other software engineering tasks, such as code evolution, code summarization, and cross-language program classification [26], [27], [28], [29], [30]. This utility was further demonstrated in experiments using the CodeXGLUE and TransCoder translation benchmarks, where Huang et al. [9] reported an average absolute improvement of 12.7%.

### III. METHODOLOGY

As illustrated in Figure 1, we compare the performance of various prompts with and without intermediate code representations (Phase 1). Following work by Pan et al. [2], we use the percentage of successful translations as the performance metric, where a translation is “successful” if the code compiles, passes runtime checks, and passes existing tests (recall Section II-A). We then test the generalizability of the two best prompts with an additional dataset (Phase 2A) and additional models (Phase 2B). Three NVIDIA RTX 6000 GPUs were used, with 51.5 GB memory each. All prompts were evaluated with a temperature of 0.2 to reduce randomness of the outputs. Postprocessing was also conducted on the LLM outputs to remove the initial first line if it starts with “Here is” or

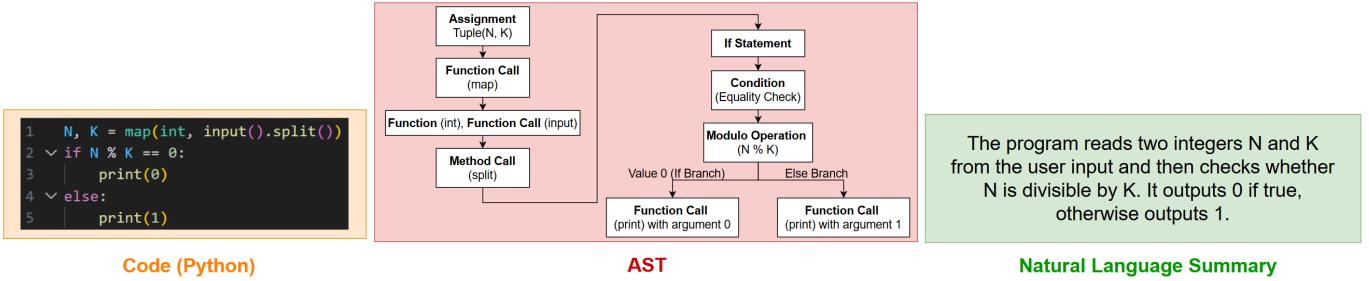


Fig. 2: Sample code snippet with its AST and NL summary.

“Here’s”, trim the first line of the file to remove any whitespace prefix, and delete any line that started with “End of Code”.

In both Phase 1 and 2, to enable comparisons with prior work, we use the same dataset as in Pan et al. [2], i.e., a sample of size 1,000 from the CodeNet dataset [31], a popular code translation benchmark. This sample contains 200 code snippets each of C++, C, Python, Java, and Go. We run the experiments across all combinations of these languages (i.e., translating the 200 C++ code snippets to C, C++ to Python, C++ to Java, C++ to Go, C to C++, C to Python, etc.), resulting in 4,000 combinations.

The CodeNet dataset was created by consolidating submissions from two programming contest web sites (AIZU Online Judge [32] and AtCoder [33]) and contains code from over 50 programming languages.

#### A. Phase 1: Selection of Best Prompts and Intermediate Representations

In Phase 1, we examine LLM-driven code translation through the use of various intermediate representations. Figure 2 shows a piece of code and its corresponding AST and NL representations. The AST shows the structure of the code snippet in a tree-style format and is derived by using the ‘ast’ method in Python. Notably, the AST must be flattened (converted to text) for input to an LLM. The NL summary describes the code in plain text.

In this phase, we evaluate the performance of the following options using the HuggingFace version of the open-source GPT-4 as the backbone (Open GPT4 8X7B [34]). The specific prompts that were used can also be found at <https://github.com/catai9/nl-in-middle/>.

- Zero-shot prompt (0SP), corresponding to previous work on code translation [2].
- One-shot prompt (1SP), with one example of source-to-target language translation for each source-target pair, randomly selected from the CodeTransOcean dataset [35].
- A zero-shot two-step approach of having the LLM translate the source language to either AST (2S-AST), NL (2S-NL), both AST and NL (2S-AST-NL), or both NL and AST (2S-NL-AST), followed by having the LLM translate the intermediate representation to the target language.

- A one-shot CoT prompt in which the LLM is instructed to use either AST (CoT-AST), NL (CoT-NL), both AST and NL (CoT-AST-NL), or both NL and AST (CoT-NL-AST) to explain its reasoning. The AST was obtained using the Python ‘ast’ method whereas the NL was obtained from the Rosetta Code website [36]. The same example used in the 1SP was also used here.

We test different combinations of AST and NL, given that prior work highlighted the importance of order in prompting [37].

#### B. Phase 2: Testing Generalizability with New Dataset and New LLMs

**New Dataset.** After comparing the prompts using Open GPT4 8X7B on the CodeNet dataset in Phase 1, we gauge the generalizability of our findings with another dataset in Phase 2A. We use the AVATAR dataset here [38] due to its large size and the availability of tests to allow for the computation of the percentage of successful translation metric. AVATAR contains 249 Java code snippets and 250 Python code snippets. We experiment with translating these snippets to C++, C, Go, Python (only for Java snippets), and Java (only for Python snippets) resulting in 1,996 total combinations. Other datasets such as CodeTransOcean [35] and CodeTrans from CodeXGLUE [39] do not include tests that could be used for judging whether the translation was successful and instead rely on metrics such as exact match, BLEU [40], and CodeBLEU [41], techniques that compare token n-grams in the generated text with the original reference. These metrics may be inappropriate for code translation: high n-gram similarity does not necessarily imply code correctness in the presence of subtle bugs and low n-gram similarity might be due to a different way to write a correct program [2].

**New Models.** Finally, in Phase 2B, we investigate how the two best prompts from Phase 1 compare against the baseline prompt (zero-shot) for specialized open-source LLMs. We run these three prompts (zero-shot, top two best prompts from Open GPT4 8X7B on CodeNet) for these LLMs (StarCoder, and CodeGen) on both the CodeNet [31] and AVATAR [38] datasets to investigate the transferability of the outcomes.

TABLE I: Performance of Open GPT4 8X7B on CodeNet compared to the results obtained by Pan et al. [2], with the two prompts achieving the best Open GPT4 8X7B results bolded; Success% = successful translation rate.

Prompt	Success%
<i>Open GPT4 8X7B</i>	
0SP	28.6%
1SP	33.5%
2S-NL	10.7%
2S-AST	2.6%
<b>CoT-NL</b>	<b>42.4%</b>
CoT-AST	32.2%
2S-NL-AST	9.0%
2S-AST-NL	11.2%
<b>CoT-NL-AST</b>	<b>39.6%</b>
CoT-AST-NL	37.9%
<i>Other LLMs from Pan et al. [2]</i>	
CodeGen	18.1%
CodeGeeX	8.4%
StarCoder	37.3%
Proprietary GPT-4	82.0%
Llama 2	13.2%
TB-Airoboros	9.3%
TB-Vicuna	2.0%

#### IV. RESULTS

##### A. Phase 1

Table I shows that our results for Open GPT4 8X7B, averaged across all source-target language pairs, differ from those reported by [2], which reported an average of 82.0% for the CodeNet samples for the same zero-shot prompt with the proprietary GPT-4 model. This difference could be attributed to the enhancements that are routinely added to GPT-4 and the general variability of LLM results. Notably, the zero-shot prompt results for Open GPT4 8X7B are still higher than those reported by the majority of the evaluated LLMs in [2]. Furthermore, the best prompt result from Open GPT4 8X7B (CoT-NL) outperforms all of the open-source reported models in [2].

CoT-NL generally achieves the best performance, followed by CoT-NL-AST. Similar to prior works on LLM few-shot learning [37], we also found that order matters: NL-AST achieved better results than the order AST-NL in the prompt.

##### B. Phase 2A

Using the baseline zero-shot prompt and the top two prompts from Phase 1 (CoT-NL and CoT-NL-AST), we now compare performance on the AVATAR dataset [38] to gauge the generalizability of our findings. Table II shows that Open GPT4 8X7B also has the best performance using the CoT-NL prompt for the AVATAR dataset and CoT-NL-AST outperforms the zero-shot prompt by 4.6%.

##### C. Phase 2B

Experiments with other models – CodeGen and StarCoder – for both the CodeNet and AVATAR datasets show differing conclusions. First, we run the baseline zero-shot prompt from [2] and achieve similar performance on the zero-shot

TABLE II: Comparison of the top two best prompts against the zero-shot prompt for both the CodeNet and Avatar datasets for the three studied models, with the best result highlighted in blue

Model	Prompt	Success%	
		CodeNet	AVATAR
Open GPT 4 8X7B	0SP	28.6%	17.6%
	CoT-NL	<b>42.4%</b>	<b>24.3%</b>
	CoT-NL-AST	39.6%	22.2%
CodeGen	Pan et al. [2]	18.1%	5.9%
	0SP	18.4%	6.8%
	CoT-NL	4.9%	1.3%
StarCoder	CoT-NL-AST	2.9%	0.2%
	Pan et al. [2]	37.3%	13.1%
	0SP	36.3%	20.4%
StarCoder	CoT-NL	38.0%	20.6%
	CoT-NL-AST	31.2%	16.9%

prompt for both CodeGen and StarCoder on the CodeNet and AVATAR datasets. This is shown in Table II. Although the zero-shot performance with StarCoder on the AVATAR dataset was 7.3% higher than that reported by [2], the improvement exists for both Java and Python source languages and can be attributed to the inherent randomness in LLM outputs.

For the CodeGen model, Table II shows that the zero-shot prompt performs the best compared to CoT-NL and CoT-NL-AST. For the StarCoder model, CoT-NL generally achieves better percentages than the zero-shot prompt, but CoT-NL-AST performs worse than the zero-shot. However, Open GPT4 8X7B still obtains the highest percentage of successful translations compared to both CodeGen and StarCoder.

Open GPT4 8X7B has the fewest reported parameters of 7 billion, with a model size of 26.44 GB [34]. On the other hand, StarCoder has 15.5 billion parameters and a model size of 59.19 GB [11] whilst CodeGen has 16.0 billion parameters and a model size of 61.16 GB [10]. Although StarCoder initially obtained a higher percentage of successful translations for the zero-shot prompt compared to the Open GPT4 8X7B model, Table II shows that an enhanced prompt (CoT-NL) can result in better performance with a smaller model.

#### V. DISCUSSION

##### A. Limitations

One limitation of our work is the focus on open-source LLMs. According to Pan et al. [2], the proprietary GPT-4 model achieved significantly better results compared to the other approaches. However, we find our work to be more reflective of enterprise use cases for code translation as many companies have policies against using proprietary LLMs with sensitive data (such as company code) [42], [43], [44]. Our study shows that leveraging an open-source LLM could be beneficial for code translation and these open-source models could also operate locally, assuming the hardware requirements are met.

Another limitation may be due to our selection of datasets. Although we attempted to generalize the applicability of our findings by investigating two open-source datasets, it is

possible that these results are invalid on other datasets. On the other hand, these two datasets were the only ones with test cases. Similarly, we only experimented with three open-source LLMs, but other open-source LLMs could have better performance or showcase a different pattern for the prompt formats. That said, the selection of these open-source LLMs was based on prior research comparing the different open-source LLMs, so it is likely that these three are the best ones available.

Furthermore, our work is limited to the prompts that we tested. Although we attempted to create a representative prompt for each investigated technique, it is possible that better prompts may exist for each category. To mitigate this limitation, we also experimented with other formats and improvements such as using whitespace and custom examples for improving our prompts; however, we did not observe any performance improvements. Finally, we also focus on only five programming languages and it is possible that the behavior would differ across other programming languages.

### B. Future Work

In this paper, we assess the quality of the code translation using a binary metric that measures whether the translation was successful, but future work should explore more robust evaluation metrics that penalize more for significant functional errors rather than subtle syntactic errors (e.g., missing imports) that disproportionately affect the binary metric.

Future studies can also expand this work by considering additional languages beyond C++, C, Python, Java, and Go, including older languages such as COBOL, Pascal, and Fortran for legacy code translation. Alternatively, the study can be extended to focus specifically on test generation performance, particularly in creating effective unit and integration tests for production-level code. With the rising popularity of agents, future work can also examine using agentic artificial intelligence to improve code generation quality where the agent automatically corrects syntactic and minor errors that do not affect functional correctness.

Our work demonstrates that the general-purpose GPT-4 8x7B LLM outperformed larger, code-specialized LLMs; however, open-source models still lag behind the reported performance of proprietary models. Future research should investigate the reasons behind the performance gap between general-purpose and code-specialized LLMs, as well as between proprietary and open-source models, to better understand key architectural or training differences.

In addition, experimentation with few-shot learning and variations of CoT prompting (e.g., Structured CoTs [21], COTTON [22]) could also be conducted to determine if intermediate representations would also aid in increasing performance for code translation.

Other future work could be to improve the post-processing of the LLM output along with trying more specific intermediate code representations during code translation. In this work, we post-processed the LLM output to reduce the frequency of noisy non-code lines. However, we only addressed the most

common issues using a simple heuristic check, but there could be more specific post-processing to boost the percentage of successful translations. Another lens of future work would be to leverage more specific intermediate code representations. In our prompt formats, we focused on NL and a general AST but there has been progress on self-optimizing ASTs to incorporate specific language nuances [45]. Control flow graphs [46] are another option that could be used as an intermediate code representation.

## VI. CONCLUSIONS

We studied the effects of introducing an intermediate code representation step in code translation with LLMs. The motivation of this work stemmed from improvements in prompting techniques such as CoT and enhanced code representation frameworks such as ASTs. We experimented with different prompts such as two-step prompts and CoT prompts with natural language and AST intermediate code representations, and compared the performance using Open GPT4 8X7B, StarCoder, and CodeGen models on the CodeNet and AVATAR datasets. We found that CoT with natural language representation performed the best, with a 13.8% and 6.7% improvement on the CodeNet dataset and AVATAR dataset, respectively, compared to the initial zero-shot prompt with Open GPT4 8X7B. Our experiments with various models and datasets demonstrate the potential for generalizing our findings while also emphasizing the effectiveness of incorporating an intermediate code representation in code translation.

## REFERENCES

- [1] J. D. Weisz *et al.*, “Better together? an evaluation of ai-supported code translation,” in *Proceedings of the 27th International Conference on Intelligent User Interfaces*. New York, NY, USA: Association for Computing Machinery, 2022, pp. 369–391.
- [2] R. Pan *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*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 1–13.
- [3] Y. Oda *et al.*, “Learning to generate pseudo-code from source code using statistical machine translation,” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE. Los Alamitos, CA, USA: IEEE Computer Society, 2015, pp. 574–584.
- [4] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, “Unleashing the potential of prompt engineering in large language models: a comprehensive review,” 2024. [Online]. Available: <https://arxiv.org/abs/2310.14735>
- [5] J. Wei *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.
- [6] I. Neamtiu, J. S. Foster, and M. Hicks, “Understanding source code evolution using abstract syntax tree matching,” in *Proceedings of the 2005 international workshop on Mining software repositories*. New York, NY, USA: Association for Computing Machinery, 2005, pp. 1–5.
- [7] Z. Tang *et al.*, “Ast-transformer: Encoding abstract syntax trees efficiently for code summarization,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE. Los Alamitos, CA, USA: IEEE Computer Society, 2021, pp. 1193–1195.
- [8] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, “Summarize and generate to back-translate: Unsupervised translation of programming languages,” in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, A. Vlachos and I. Augenstein, Eds. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 1528–1542. [Online]. Available: <https://aclanthology.org/2023.eacl-main.112>

- [9] Y. Huang *et al.*, “Program translation via code distillation,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 10903–10914. [Online]. Available: <https://aclanthology.org/2023.emnlp-p-main.672>
- [10] E. Nijkamp *et al.*, “Codegen: An open large language model for code with multi-turn program synthesis,” in *Eleventh International Conference on Learning Representations*. Kigali, Rwanda: ICLR, 2022.
- [11] R. Li *et al.*, “Starcoder: may the source be with you!” 2023. [Online]. Available: <https://arxiv.org/abs/2305.06161>
- [12] Q. Zheng *et al.*, “Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x,” 2024. [Online]. Available: <https://arxiv.org/abs/2303.17568>
- [13] OpenAI *et al.*, “Gpt-4 technical report,” 2024. [Online]. Available: <https://arxiv.org/abs/2303.08774>
- [14] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1126–1135. [Online]. Available: <https://proceedings.mlr.press/v70/finn17a.html>
- [15] J. Snell, K. Swersky, and R. Zemel, “Prototypical networks for few-shot learning,” in *Advances in Neural Information Processing Systems*, I. Guyon *et al.*, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/cb8da6767461f2812ae4290eac7cbc42-Paper.pdf)
- [16] A. Shirafuji, Y. Oda, J. Suzuki, M. Morishita, and Y. Watanobe, “Refactoring programs using large language models with few-shot examples,” in *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, 2023, pp. 151–160.
- [17] C. Pornprasit and C. Tantithamthavorn, “Fine-tuning and prompt engineering for large language models-based code review automation,” *Information and Software Technology*, vol. 175, p. 107523, 2024.
- [18] X. Li, S. Yuan, X. Gu, Y. Chen, and B. Shen, “Few-shot code translation via task-adapted prompt learning,” *Journal of Systems and Software*, vol. 212, p. 112002, 2024. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121224000451>
- [19] Y. Nong *et al.*, “Chain-of-thought prompting of large language models for discovering and fixing software vulnerabilities,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.17230>
- [20] S. A. Rukmono, L. Ochoa, and M. Chaudron, “Deductive software architecture recovery via chain-of-thought prompting,” in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 92–96. [Online]. Available: <https://doi.org/10.1145/3639476.3639776>
- [21] J. Li, G. Li, Y. Li, and Z. Jin, “Structured chain-of-thought prompting for code generation,” *ACM Trans. Softw. Eng. Methodol.*, Aug. 2024, just Accepted. [Online]. Available: <https://doi.org/10.1145/3690635>
- [22] G. Yang, Y. Zhou, X. Chen, X. Zhang, T. Y. Zhuo, and T. Chen, “Chain-of-thought in neural code generation: From and for lightweight language models,” *IEEE Transactions on Software Engineering*, vol. 50, no. 9, pp. 2437–2457, 2024.
- [23] S. Yao *et al.*, “Tree of thoughts: Deliberate problem solving with large language models,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [24] R. L. Rosa, C. Hulse, and B. Liu, “Can github issues be solved with tree of thoughts?” 2024. [Online]. Available: <https://arxiv.org/abs/2405.13057>
- [25] M. Szafraniec, B. Roziere, H. L. F. Charton, P. Labatut, and G. Synnaeve, “Code translation with compiler representations,” *ICLR*, 2023.
- [26] I. Neamtiu, J. S. Foster, and M. Hicks, “Understanding source code evolution using abstract syntax tree matching,” in *Proceedings of the 2005 International Workshop on Mining Software Repositories*, ser. MSR ’05. New York, NY, USA: Association for Computing Machinery, 2005, p. 1–5. [Online]. Available: <https://doi.org/10.1145/1083142.1083143>
- [27] D. Ziegner, T. Kirschstein, M. Catasta, J. Leskovec, and S. Günnemann, “Language-agnostic representation learning of source code from structure and context,” 2021. [Online]. Available: <https://arxiv.org/abs/2103.11318>
- [28] Z. Tang, C. Li, J. Ge, X. Shen, Z. Zhu, and B. Luo, “Ast-transformer: encoding abstract syntax trees efficiently for code summarization,” in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE ’21. Los Alamitos, CA, USA: IEEE Press, 2022, p. 1193–1195. [Online]. Available: <https://doi.org/10.1109/ASE51524.2021.9678882>
- [29] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, “A novel neural source code representation based on abstract syntax tree,” in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE ’19. Los Alamitos, CA, USA: IEEE Press, 2019, p. 783–794. [Online]. Available: <https://doi.org/10.1109/ICSE.2019.00086>
- [30] K. Wang, M. Yan, H. Zhang, and H. Hu, “Unified abstract syntax tree representation learning for cross-language program classification,” in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, ser. ICPC ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 390–400. [Online]. Available: <https://doi.org/10.1145/3524610.3527915>
- [31] R. Puri *et al.*, “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)*. San Diego CA: NeurIPS, 2021.
- [32] University of Aizu, “Aizu online judge,” University of Aizu, 2025. [Online]. Available: [https://onlinejudge.u-aizu.ac.jp/about\\_us](https://onlinejudge.u-aizu.ac.jp/about_us)
- [33] AtCoder Inc., “Atcoder,” AtCoder Inc., 2025. [Online]. Available: <https://atcoder.jp/home>
- [34] TheBloke, “Thebloke/open\_gpt4\_8x7b-gguf,” Hugging Face, 2024. [Online]. Available: [https://huggingface.co/TheBloke/Open\\\_\\\_Gpt4\\\_8x7B-GGUF](https://huggingface.co/TheBloke/Open\_\_Gpt4\_8x7B-GGUF)
- [35] W. Yan, Y. Tian, Y. Li, Q. Chen, and W. Wang, “CodeTransOcean: A comprehensive multilingual benchmark for code translation,” in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 5067–5089. [Online]. Available: <https://aclanthology.org/2023.findings-emnlp.337>
- [36] Rosetta Code, “Rosetta code,” Rosetta Code, 2024. [Online]. Available: [https://rosettacode.org/wiki/Rosetta\\_Code](https://rosettacode.org/wiki/Rosetta_Code)
- [37] S. Kumar and P. Talukdar, “Reordering examples helps during priming-based few-shot learning,” *arXiv preprint arXiv:2106.01751*, 2021.
- [38] W. U. Ahmad, M. G. R. Tushar, S. Chakraborty, and K.-W. Chang, “Avatar: A parallel corpus for java-python program translation,” 2023. [Online]. Available: <https://arxiv.org/abs/2108.11590>
- [39] S. Lu *et al.*, “Codexglue: A machine learning benchmark dataset for code understanding and generation,” 2021. [Online]. Available: <https://arxiv.org/abs/2102.04664>
- [40] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: a method for automatic evaluation of machine translation,” in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.
- [41] S. Ren *et al.*, “Codebleu: a method for automatic evaluation of code synthesis,” *arXiv preprint arXiv:2009.10297*, 2020.
- [42] S. Ray, “Apple joins a growing list of companies cracking down on use of chatgpt by staffers—here’s why,” 2023, [Accessed 19-12-2024]. [Online]. Available: <https://www.forbes.com/sites/siladityaray/2023/05/19/apple-joins-a-growing-list-of-companies-cracking-down-on-use-of-chatgpt-by-staffers-heres-why>
- [43] G. Drennan, “75% of companies are banning the use of chatgpt: What happened?” 2023, [Accessed 19-12-2024]. [Online]. Available: <https://hackernoon.com/75percent-of-companies-are-banning-the-use-of-chatgpt-what-happened>
- [44] N. Behbahani, “Employers must ban employees from pasting code into chatgpt,” 2023, [Accessed 19-12-2024]. [Online]. Available: <https://www.linkedin.com/pulse/employers-must-ban-employees-from-pasting-code-nicolas-behbahani>
- [45] T. Würthinger *et al.*, “Self-optimizing ast interpreters,” in *Proceedings of the 8th Symposium on Dynamic Languages*, 2012, pp. 73–82.
- [46] T. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.