CODETRANSENGINE: READY-TO-USE BACKEND FOR LLM-BASED CODE TRANSLATION

Marcos Macedo[1,†] **Yuan Tian**[1] **Bram Adams**[1]
[1]School of Computing, Queen's University

ABSTRACT

Code translation, the process of converting a program from one programming language to another, plays a significant role in modernizing legacy systems, ensuring cross-platform compatibility, and improving performance of programs. With the rise of Large Language Models (LLMs), new possibilities have emerged for automating this complex task. However, building effective LLM-based code translation pipelines involves numerous challenges, including prompt engineering, efficient inference, output control, test case validation, and resource optimization. To address these challenges, we introduce CODETRANSENGINE, a ready-to-use backend that simplifies LLM-based code translation. CODETRANSENGINE lowers the entry barrier for practitioners and reduces the effort required to leverage the power of LLMs for code translation, enabling them to focus on achieving their translation goals.

1 INTRODUCTION

Utilizing Large Language Models (LLMs) for code translation presents significant opportunities to enhance the accuracy and idiomatic quality of generated code compared to traditional transpilers (Pan et al., 2024; Yang et al., 2024). However, this method introduces new challenges, particularly regarding the increased operational complexity of managing LLM-driven code translation. Unlike traditional transpilers, which adhere to deterministic rules, LLMs necessitate a more advanced inference infrastructure, careful output control (ensuring that only source code, rather than extraneous text, is extracted or validated) (Macedo et al., 2024), and ongoing monitoring to maintain translation quality and efficiency. Furthermore, each phase, from prompt engineering and efficient inference to output control and evaluation, involves various design choices, escalating the time and effort required to develop a fully functional code translation pipeline.

For developers looking to leverage LLMs for code translation, the process often involves finding effective prompts, writing scripts for their chosen LLM and creating an evaluation suite to assess performance. This process is slow and labor-intensive, requiring technical expertise, which poses a significant barrier for many software developers looking to use LLMs without navigating the complexities of the underlying systems.

Meanwhile, researchers are continually developing innovative techniques to enhance the accuracy of LLMs for code translation. These advances necessitate the creation of detailed experiment scripts and the utilization of various technologies for inference and evaluation across research efforts, complicating direct comparisons between studies.

CODETRANSENGINE addresses these challenges by providing a ready-to-use solution. This offers advantages to practitioners aiming to leverage LLMs for code translation with minimal setup, as well as researchers who can build upon an existing pipeline to test and refine their algorithms. To the best of our knowledge, no current library or system delivers a comparable level of integration and usability for LLM-driven code translation.

[†]Correspondence to: `marcos.macedo@queensu.ca`

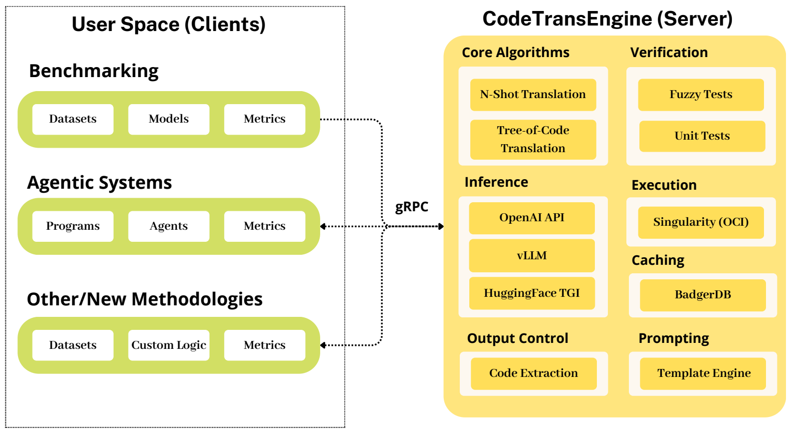

Figure 1: High-level description of CODETRANSENGINE functionality. Clients can leverage CODETRANSENGINE as the backend for various use cases.

2 SYSTEM DESCRIPTION

2.1 ARCHITECTURE

CODETRANSENGINE is designed to support a variety of user cases, ensuring seamless integration into a wide range of code translation workflows. This adaptability allows it to function both as a standalone tool and as a modular component within larger pipelines for automated code translation. For instance, it can serve as the backend for whole-repository translation initiatives, where tasks such as file listing, planning, and program slicing can be executed outside of the engine, or it can be used standalone to translate source code.

Fig. 1 presents a high-level overview of CODETRANSENGINE. At its core, the engine follows a client-server architecture, enabling efficient communication between different components. The engine itself operates as a server that exposes gRPC-based APIs, offering various functionalities that enable different translation strategies (algorithms) and validation mechanisms. To facilitate ease of use, a Python client is provided as a reference implementation, enabling researchers and developers to interact with the engine.

2.2 INFERENCE

CODETRANSENGINE orchestrates the sequence of inference steps required to generate translations. The engine forwards requests to specialized inference backends, processes their responses, and ensures seamless integration into the translation workflow. This modular design allows users to experiment with different inference services without modifying the core translation system.

The engine supports inference services that adhere to the OpenAI-compatible API standard, making it interoperable with various LLM frameworks. It has been tested with vLLM (Kwon et al., 2023), an optimized inference system designed for efficient and high-throughput serving of large language models, and the OpenAI API, which provides access to proprietary models such as GPT-4. Additionally, it can be extended to support other backends, including Hugging Face Text Generation Inference (Wolf et al., 2020), a scalable solution for serving open-source models with optimized performance. By supporting these diverse inference engines, the system enables practitioners to select the most suitable approach based on their needs, whether prioritizing speed and cost-efficiency with vLLM, leveraging state-of-the-art proprietary models via the OpenAI API, or deploying open-source models through Hugging Face's infrastructure.

2.3 Evaluation

Benchmarking LLMs for code translation is essential for assessing their capabilities and limitations before integrating them into real-world software development workflows. Several well-established benchmark datasets, such as TransCoder (Roziere et al., 2021), HumanEval-X (Zheng et al., 2023), and CodeNet (Puri et al., 2021), are widely used in the research community. CodeTransEngine provides a flexible and standardized framework for evaluating LLMs, enabling researchers to seamlessly incorporate both existing and custom datasets. It supports two key evaluation strategies: *Fuzzy Test Cases*, which validate translations by ensuring that functionally equivalent programs in different languages produce the same output for identical inputs, and *Unit Test Cases*, which assess correctness by executing predefined unit tests. Test cases can be executed during translation, with the engine returning validation results alongside translated code, or after translation, by applying tests to pre-existing programs. To ensure a reliable evaluation process, output control is carefully implemented following best practices outlined in the literature (Macedo et al., 2024). To further support benchmarking, we maintain a curated repository of widely used benchmarks for seamless integration with CodeTransEngine, ensuring consistent and reproducible code translation assessments.

2.4 Execution Environment

CodeTransEngine provides a secure and built-in execution environment for evaluating the source code generated by LLMs using test cases. It is crucial to ensure that the evaluation occurs in a controlled environment, free from external factors that could influence the results. To facilitate this, we utilize Singularity containers Sylabs (n.d.), which we have found to be particularly well-suited for high-concurrency scenarios. Specifically, they allow one to rapidly deploy hundreds of containers for short durations, which is essential to efficiently run test cases. Moreover, Singularity enables the seamless conversion of Dockerfiles into Singularity images, allowing users to easily create customized environments to evaluate LLM-generated code. As part of the CodeTransEngine release, we provide pre-configured execution containers for popular programming languages, including Java, Python, Go, C#, C, C++, Rust, and JavaScript. These containers come with minimal dependencies, offering users a ready-to-use solution that can be further modified to meet specific evaluation needs.

2.5 Algorithm Support

CodeTransEngine comes equipped with popular code translation algorithms (core algorithms shown in Fig. 1), available out-of-the-box, designed to help users get started quickly. Among these are **N-Shot Translation** and **Tree of Code Translation (ToCT)** (Macedo et al., 2025). N-Shot translation is the simplest and most popular method that directly converts the code from the source to the target language utilizing prompts containing 0 to n translation examples. In contrast, ToCT is a more sophisticated algorithm that plans transitive intermediate translation sequences between a source and a target language. These sequences are validated in a specific order, and the algorithm has demonstrated state-of-the-art performance across three well-known benchmarks.

Beyond built-in algorithms, CodeTransEngine's greatest strength is its flexibility. Users can easily build and implement advanced techniques without needing to modify the core engine code. Thanks to the separation between the client and server, it is possible to introduce complex methodologies, such as multi-agent systems, the incorporation of compiler feedback, or any other innovative approaches, by simply leveraging the existing framework. This separation ensures that users can enhance the translation process and tailor it to their specific needs without altering the codebase.

2.6 Resource Usage

The tool is designed to adapt to the user's hardware, rather than requiring the user to adjust their system to meet its demands. This flexibility is achieved through several optimizations built into CodeTransEngine. One key optimization is the support for concurrent execution of inferences and test case validations. This approach ensures that resources are fully utilized, which is especially advantageous when operating on cloud instances billed by the hour, as it maximizes resource efficiency.

Conversely, when resources are limited and speed is not a primary concern, the tool can execute inferences and validations sequentially, thereby conserving computational resources. Additionally, the implemented ToCT algorithm offers tunable parameters, such as the list of intermediate languages and tree depth, that allow for a trade-off between accuracy and resource usage, enabling further customization based on available resources. The ToCT algorithm also supports early termination, which allows the process to be halted as soon as a satisfactory solution or result is found, without completing unnecessary computations.

Other optimizations include configurable caching mechanisms at the inference, test execution, and request levels. These caching strategies are particularly beneficial for conserving resources when LLMs generate identical outputs or when the same request is received multiple times. By intelligently reusing previously computed results, the system minimizes redundant computations and reduces resource consumption.

2.7 CONFIGURATION

A key objective of the CODETRANSENGINE is to streamline the process of conducting code translation experiments by minimizing the amount of code the user must write. To achieve this, the server is designed to be highly configurable via a .yml file. This configuration file allows users to specify various parameters, such as prompt templates and inference settings, including top-p, top-k, temperature, seed, and others. Additionally, the file accommodates settings for caching and other relevant configurations, providing users with a flexible and efficient way to tailor the tool to their needs without needing to write code.

3 USE CASES

3.1 CODE TRANSLATION APPLICATIONS

CODETRANSENGINE can be integrated as a module within more complex processes of code translation. For instance, whole-repository code translation spans several stages, including analyzing the repository's structure, formulating a translation plan, and slicing files as needed. In this context, CODETRANSENGINE can be leveraged as an external module responsible for executing the translations and performing verifications using test cases. This integration helps to reduce both the effort and time required to implement such systems, streamlining the overall process.

3.2 ALGORITHM RESEARCH

Research in code translation can benefit from CODETRANSENGINE by accelerating the pace of research and improving the rigor of the studies. Recent research has shown that even minor discrepancies in output format can lead to misleading results during the LLM evaluation (Macedo et al., 2024). Consequently, employing a consistent pipeline ensures that the steps are more replicable and comparable across different studies in code translation.

3.3 LARGE SCALE EMPIRICAL STUDIES

CODETRANSENGINE is suitable for large-scale empirical studies that involve the translation of thousands of programs across multiple datasets and LLMs. This is because of the optimizations presented previously. Additionally, the engine is optimized for batched workloads, which is common in these kinds of studies.

4 CONCLUSION

We presented CODETRANSENGINE, a versatile and extensible backend for conducting research and experiments on LLM-based code translation. This engine provides a standardized infrastructure that enables both industry practitioners and academic researchers to develop, evaluate, and compare code translation methodologies efficiently. By supporting multiple inference services, various execution-based evaluation strategies, careful output control, built-in benchmarks, and a modular client-server architecture, CODETRANSENGINE reduces the barrier to adopting LLM for code translation.

To facilitate its adoption, CodeTransEngine is accompanied by comprehensive documentation[0] covering installation, usage, and extensibility. Furthermore, it is open-sourced on GitHub[1], providing a framework for LLM-driven code translation techniques.

5 Acknowledgmenets

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and Mitacs [funding reference number: ALLRP 597628 - 24].

[0] `https://codetransengine.github.io/guides/`
[1] `https://github.com/CodeTransEngine/CodeTransEngine`

References

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Marcos Macedo, Yuan Tian, Filipe Cogo, and Bram Adams. Exploring the impact of the output format on the evaluation of large language models for code translation. In *Proceedings of the 2024 IEEE/ACM First International Conference on AI Foundation Models and Software Engineering*, pp. 57–68, 2024.

Marcos Macedo, Yuan Tian, Pengyu Nie, Filipe R. Cogo, and Bram Adams. InterTrans: Leveraging Transitive Intermediate Translations to Enhance LLM-based Code Translation. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 772–772, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. doi: 10.1109/ICSE55347.2025.00236. URL `https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00236`.

Rangeet Pan, Ali Reza Ibrahimzada, Rahul Krishna, Divya Sankar, Lambert Pouguem Wassi, Michele Merler, Boris Sobolev, Raju Pavuluri, Saurabh Sinha, and Reyhaneh Jabbarvand. 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*, pp. 1–13, 2024.

Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

Baptiste Roziere, Jie M Zhang, Francois Charton, Mark Harman, Gabriel Synnaeve, and Guillaume Lample. Leveraging automated unit tests for unsupervised code translation. *arXiv preprint arXiv:2110.06773*, 2021.

Sylabs. Singularity containers, n.d. URL `https://sylabs.io/singularity/`. Accessed: 2024-07-22.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. HuggingFace's Transformers: State-of-the-art Natural Language Processing, July 2020. URL `http://arxiv.org/abs/1910.03771`. arXiv:1910.03771 [cs].

Zhen Yang, Fang Liu, Zhongxing Yu, Jacky Wai Keung, Jia Li, Shuo Liu, Yifan Hong, Xiaoxue Ma, Zhi Jin, and Ge Li. Exploring and unleashing the power of large language models in automated code translation. *Proceedings of the ACM on Software Engineering*, 1(FSE):1585–1608, 2024.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, et al. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *arXiv preprint arXiv:2303.17568*, 2023.