CodeSynapse: Evaluating LLMs for Cross-Language Code Translation

CS540 Project Final Report

Manan Patel
Graduate Student
University of Tennessee
mpatel65@vols.utk.edu

Zachary Perry
Graduate Student
University of Tennessee
zperry4@vols.utk.edu

Shayana Shreshta
Graduate Student
University of Tennessee
sshres25@vols.utk.edu

Eric Vaughan
Graduate Student
University of Tennessee
evaugha3@vols.utk.edu

***Abstract*—BACKGROUND: Translating code between programming languages like C++, Java, and Python is essential for improving software reusability and interoperability. Large language models (LLMs) show promise in this area, but there's still limited research on how effectively they can translate code while maintaining accuracy and idiomatic usage.**

OBJECTIVE: This project introduces CodeSynapse, a tool that evaluates LLMs for translating code across languages. The project also includes a web app that allows users to choose programming languages and select the LLM that performs best for their chosen language pair.

METHODS: We used the XLCoST dataset, extracting subsets relevant to Python, C++, and Java, with 100 samples from each language. Three LLMs—Llama-3.2/3B, Deepseek-coder/6.7B, and Phi/2.7B—were evaluated on a shared set of code snippets translated between the three languages. The evaluation strategy includes CodeBleu, BLEU with different weights, and keyword matching. The web app enables users to submit code snippets with source and target languages (Python, C++, or Java) and receive translations using the best-performing LLM for that language pair.

RESULTS: Llama 3.2/3B emerged as the best overall LLM for code translation, outperforming the other models in all translation tasks except for C++ to Java. No clear correlation was found between model size and translation ability, with Llama 3.2/3B outperforming Deepseek-Coder consistently. Interestingly, smaller models, when well-tuned, were able to rival or outperform larger models in certain tasks. We found that metrics like CodeBLEU, which account for both syntax and semantics, were more effective than traditional BLEU for evaluating code translations.

CONCLUSION: CodeSynapse provides valuable insights into the strengths and weaknesses of current LLMs in cross-language code translation. Our findings suggest that while larger models like Llama generally perform better, well-tuned smaller models can be competitive. The web app offers an easy-to-use platform for exploring LLM-based code translation, and our evaluation highlights the need for more comprehensive metrics like CodeBLEU to better capture translation accuracy.

I. Introduction

The rise of Large Language Models (LLMs) have completely revolutionized the field of software development and research. With these new tools, developer productivity and efficiency have drastically increased and additional tools, such as specialized coding assistants, have only pushed this boundary forward. While LLMs provide many capabilities, this project focuses on their application in the domain of cross-language code translation, with a specific interest in the accuracy when translating across languages with different paradigms. Currently, this area lacks comprehensive research for LLM performance for this specific task and we aim to fill this gap and help determine how LLMs can be further used to enhance developer productivity. Our team for this project is composed of members with various backgrounds in software engineering and research. Having a team with experience in both of these domains is perfect for this project, as both will be required to complete it and the research itself is relevant to both fields.

II. Motivation & Research Questions

The ability to translate code between different programming languages is becoming increasingly important in modern software development environments. As organizations maintain diverse technology stacks and legacy systems, developers often need to port existing functionality across language boundaries. Manual translation is time-consuming, error-prone, and requires expertise in multiple programming paradigms. This creates a significant opportunity for automated translation tools powered by Large Language Models (LLMs).

Recent advancements in LLMs have demonstrated promising capabilities for code understanding and generation tasks [1], [4]. However, the effectiveness of smaller, more efficient models like Llama-3.2/3B, Deepseek-coder/6.7B, and Phi/2.7B for cross-language code translation remains underexplored, particularly when translating between languages with different programming paradigms.

Our work is motivated by several key considerations:

- **Accessibility**: Smaller, more efficient LLMs require fewer computational resources to run, making automated code translation more accessible to developers with limited hardware.
- **Paradigm Gaps**: Programming languages like Python, C++, and Java represent different programming styles and paradigms. Evaluating how well LLMs bridge these conceptual gaps provides insights into their understanding of programming concepts beyond syntax.

- **Practical Application**: Building a web interface for code translation creates a practical tool for developers while simultaneously enabling continuous evaluation and improvement of underlying models.
- **Benchmarking**: Using standardized datasets with ground truth translations enables quantitative assessment of translation quality and facilitates comparison between different models.

By systematically evaluating how these smaller LLMs handle cross-language translations, we aim to provide insights into their capabilities and limitations. The results will help inform both the development of more effective code translation tools and contribute to the broader understanding of how LLMs process and generate code across different programming paradigms.

III. Related Work

A. Traditional Approaches to Code Translation

Prior to the deep learning era, code translation primarily relied on rule-based systems and abstract syntax tree (AST) transformations. Lachaux et al. [2] note that traditional approaches like source-to-source compilers (transpilers) such as Java2C++. required extensive manual engineering of language-specific rules. These systems struggled with idiomatic expressions and language-specific design patterns, often producing technically correct but unnatural code.

B. Large Language Models for Code

The emergence of large language models has dramatically changed the code generation and translation landscape. Models like Codex [1], which powers GitHub Copilot, demonstrated impressive code generation capabilities across multiple languages. More recently, specialized code models like StarCoder, DeepSeek-Coder, and CodeLlama have been developed specifically for programming tasks. These models can handle translation between programming languages, but most research has focused on larger models with billions of parameters, requiring significant computational resources.

C. Smaller and More Efficient LLMs

While research on massive LLMs continues, there is growing interest in developing smaller, more efficient models. Models like Phi-1 and Phi-2 demonstrated that carefully curated training data can produce models with strong coding abilities despite having fewer parameters. The Llama series has also evolved to include smaller variants that maintain reasonable performance across various tasks. Evaluating these smaller models specifically for code translation tasks represents an important research direction with practical applications.

D. Cross-Paradigm Translation Challenges

Programming language paradigms impose different conceptual frameworks that complicate translation efforts. Ahmed et al. identified specific challenges when translating between object-oriented and functional programming languages, including handling state management and different abstraction mechanisms. Nguyen et al. [3] proposed specialized techniques for translating API usage patterns across language boundaries. Our work builds on these insights by specifically evaluating how modern LLMs handle these paradigmatic differences.

IV. Identification of Research Gaps

While significant progress has been made in the area of code translation using large language models (LLMs), several gaps remain. First, most existing studies focus on large-scale models with billions of parameters, which limits the accessibility and deployment of these solutions in resource-constrained environments. Although recent work highlights the potential of smaller models, systematic evaluations of their effectiveness in real-world translation tasks remain limited.

Second, evaluation metrics such as BLEU and CodeBLEU, though widely used, often fail to fully capture the semantic and functional equivalence of translated code. These metrics primarily rely on lexical overlap and static analysis, which may overlook subtle behavioral differences or logical inconsistencies in the output.

Lastly, there is a lack of publicly available, fine-grained benchmark datasets specifically designed to assess cross-language code translation with detailed annotations for semantic fidelity and functional correctness. This absence hinders the reproducibility and comparability of results across different models and methodologies.

Our work addresses these gaps by evaluating lightweight LLMs using syntax- and semantics-aware metrics and constructing a dataset that supports fine-grained evaluation of code translation quality.

V. Methodology

A. Research Objectives

The primary objective of this study is to evaluate the effectiveness of small-to-medium-sized large language models (LLMs) in performing cross-language code translation between Python, C++, and Java. Specifically, the study investigates:

- How accurately LLMs can translate code across languages with differing programming paradigms (e.g., object-oriented vs. imperative).
- The comparative performance of LLMs with varying parameter sizes in translation tasks.
- The feasibility of integrating the most effective model into a user-facing web application for real-time code translation.

B. Dataset Selection

We use the XLCoST dataset, a multilingual code dataset that contains parallel code samples across several programming languages. From this dataset, we extract subsets specifically relevant to Python, C++, and Java, which are representative of different programming paradigms. Each subset consists of 100 samples per language, ensuring a balanced and manageable evaluation set. This dataset is well-suited for the task as it contains semantically equivalent code snippets, enabling consistent comparison of translation fidelity across models.

C. Model Selection

Three open-source LLMs were selected for evaluation:

- **LLaMA 3.2/3B** – A recent and compact instruction-tuned model, known for strong general-purpose reasoning abilities.
- **Deepseek-coder/6.7B** – A model trained specifically on code, optimized for understanding and generating programming content.
- **Phi-2/2.7B** – A lightweight, performance-efficient model with strong few-shot capabilities.

These models were chosen to cover a spectrum of parameter sizes and training objectives. Evaluating models in the 2–7B range aligns with the research goal of identifying cost-effective alternatives to much larger LLMs, while still achieving competitive translation performance.

D. Translation and Evaluation Strategy

Each model is tasked with translating a shared set of 100 code snippets between every pair of the three selected languages (Python-C++, Python-Java, Java-C++). This results in six translation directions, allowing for a comprehensive cross-paradigm analysis.

Translation quality is measured using both general and code-specific metrics:

- **BLEU Score** – A traditional metric for evaluating n-gram overlap between predicted and reference translations.
- **CodeBLEU** – A specialized metric for code that extends BLEU by incorporating syntax tree structure, data flow, and keyword alignment. CodeBLEU is better aligned with semantic and structural correctness, making it highly suitable for this task.
- **Keyword Match Rate** – A custom metric capturing how well language-specific keywords are retained or translated appropriately.

To better assess real-world utility, weights may be assigned to each metric based on their relevance to end-user goals—e.g., favoring CodeBLEU for functional correctness.

E. Deployment as a Web Application

To demonstrate practical applicability, the best-performing model for each language pair is integrated into a web-based tool. The tool allows users to:

1) Input a code snippet in Python, C++, or Java.
2) Select the desired target language.
3) Receive a translated version of the code using the highest-scoring LLM for that translation direction.

This design bridges academic evaluation with end-user interaction, validating model choices in a real-world scenario and highlighting the practical potential of lightweight LLMs in code translation tasks.

F. Justification of Methodology

This methodology (Fig.1) is well-aligned with the research questions. Using XLCoST ensures high-quality data for fair comparison. The selected LLMs offer a balanced exploration of size vs. performance trade-offs. Applying both lexical and semantic evaluation metrics provides a multi-faceted view of translation quality. Finally, the deployment of a web application validates the feasibility and effectiveness of the approach in practical use cases.

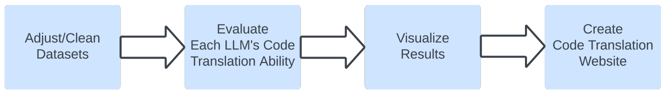
Adjust/Clean Datasets
Evaluate Each LLM's Code Translation Ability
Visualize Results
Create Code Translation Website

Fig. 1. Methodology

VI. Research Value

This research is intended for both researchers and practitioners in programming languages, software development, and artificial intelligence. Researchers aim to evaluate how well large language models (LLMs) translate code between languages with different programming paradigms, such as imperative and object-oriented. Practitioners, including software developers and educators, need reliable tools and best practices to translate code accurately, saving time and reducing errors in projects or teaching. The main challenge is the lack of comprehensive studies on LLM performance in cross-paradigm translation and the difficulty practitioners face in manually rewriting code or relying on existing automated tools. While solutions like OpenAI's Codex and rule-based translators exist, they often struggle with paradigm-specific differences, highlighting the need for a deeper understanding of how LLMs handle these challenges.

This research will provide both researchers and developers with practical insights and useful tools. Researchers will gain access to a benchmark dataset to evaluate how well large language models (LLMs) translate code across different programming paradigms, along with a detailed analysis of their strengths and weaknesses. This will help guide future studies and highlight areas where LLMs still struggle. For developers and educators, the research will offer clear guidelines on which LLMs work best for specific translation tasks and how to improve accuracy when converting code between languages like Python, C++, Java, etc. By reducing the need for manual effort and minimizing errors, these insights will make coding workflows more efficient. Ultimately, this research aims to close the gap between theory and real-world use, helping both researchers and practitioners get the most out of LLMs for code translation.

The success of this project will be measured through both quantitative and qualitative factors. For researchers, success means that the benchmark dataset is useful for evaluating LLMs in future studies and that the findings contribute to discussions on improving code translation. For developers and educators, success will be reflected in clear, practical insights on using LLMs for cross-paradigm translation, helping them reduce errors and improve efficiency. On a technical level, accuracy rates above 90% and error rates below 5% will indicate strong model performance. Additionally, feedback from classmates, instructors, or developers testing the results will help assess the quality and usability of the findings. Ultimately, this project aims to bridge the gap between research and real-world application, providing valuable insights for both academic and practical use.

VII. Results

For the purposes of this project, all three LLMs generated several code translations for all possible combinations of the starting and ending languages. The LLMs were then evaluated using three metrics: Bilingual Evaluation Understudy 4-gram (BLEU4) score, keyword match score, and Code Bilingual Evaluation Understudy (CodeBLEU) score.

BLEU4 evaluates n-gram precision for n-grams up to length four, meaning individual tokens and short sequences of tokens are examined. This allowed us to assess the accuracy and fluency of each LLM-generated translation. Figure 1 shows the average BLEU4 score for each LLM for each language translation. With the exception of the code translation from C++ to Java for which Deepseek Coder was, Llama had the best BLEU4 score for all code translations. The other key takeaway from this figure is that all LLMs performed the worst when translating code to Python from C++ and Java. This may indicate that LLMs struggle to translate code from mainly object-oriented languages to multi-paradigm languages.

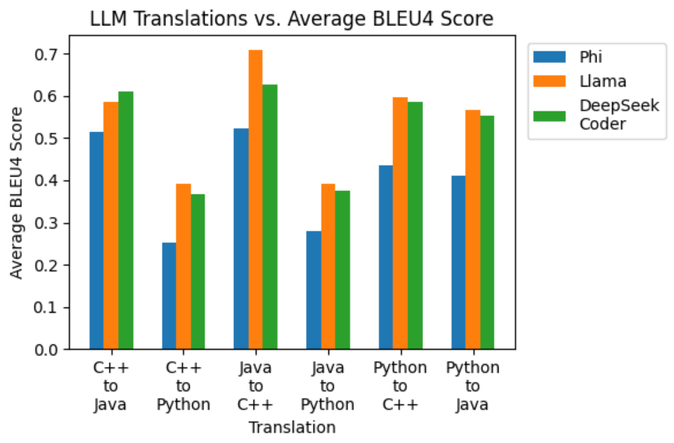

Fig. 2. LLM translations versus average BLEU4 score.

Next, we examined keyword matching. This metric is essentially the F1 score of the overlap between keywords, such as ”if” and ”while”, in the LLM-generated translation and the ground-truth translation. In Figure 2, we have visualized the average keyword match score for each LLM for each language translation. Once again, Llama has the best performance for translating between all languages, except for Java to C++ and from Python to C++. Additionally, the average keyword scores are somewhat similar across all translations. This means that, unlike with average BLEU4 score, the LLMs do not appear to struggle with any kind of translations based on average keyword match scores.

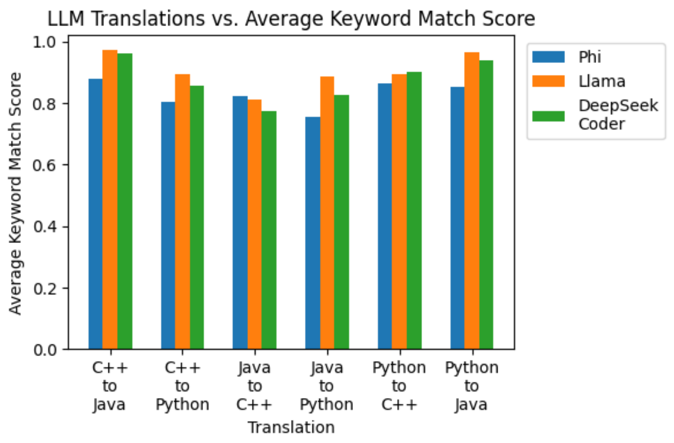

Fig. 3. LLM translations versus average keyword match score.

For our final metric, we calculated an average CodeBLEU score that incorporates the other two metrics and serves as an overall metric for translation accuracy. Specifically, CodeBLEU is the sum of 70% of the average BLEU4 score and 30% of the average keyword match score. Figure 3 depicts the average CodeBLEU score for each LLM for each language translation. Unsurprisingly, Llama had the best average CodeBLEU score for all code translations, except for C++ to Java for which DeepSeek Coder was the best.

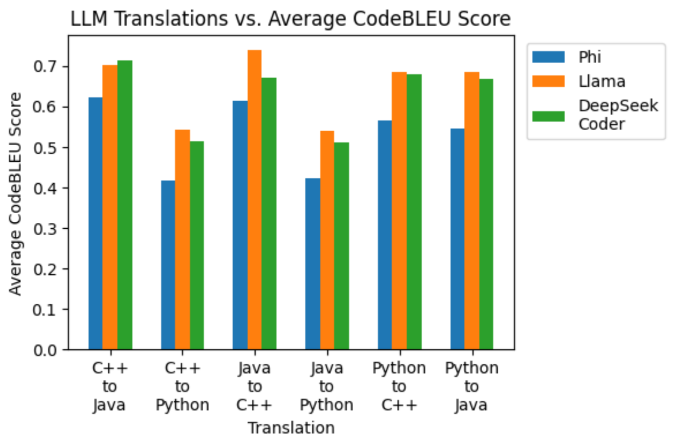

Fig. 4. LLM translations versus average CodeBLEU score.

Figure 3 also shows us that model size does not appear to have a clear relationship with performance. The models in this figure are in order from smallest to largest, with Phi (in blue) having roughly 2.7 billion parameters, Llama (in orange) having about 3.2 billion parameters, and DeepSeek Coder (in green) having approximately 6.7 billion parameters. Despite DeepSeek Coder having over twice as many parameters as Llama, DeepSeek Coder received a lower average CodeBLEU score for almost all code translations.

Resource Availability

All data and tools required for the project are open-source and readily available, minimizing dependencies on external resources. This accessibility allows us to focus on development and integration without delays related to resource acquisition. The use of open-source tools also ensures that our project remains cost-effective and scalable.

VIII. Limitations

Our research encountered several significant limitations that affected both our LLM analysis and the development of our web application. The primary limitations were computational constraints and costs. During the LLM evaluation, we frequently ran out of GPU time and RAM allocations on Google Colab. Additionally, attempts to run and evaluate these models locally created additional complications, with some team members experiencing hardware issues due to insufficient computing resources.

For the web application, we determined that local model deployment, while not ideal, represented the most cost-effective approach. When running the application locally, we had to host both Phi/2.7B and Deepseek-Coder/6.7B locally through Ollama, as neither was available through a hosted service, and deploying it ourselves on a platform such as GCP was too expensive. The resulting configuration created some noticeable performance issues, affecting both the hardware running the models and the application's responsiveness when translating using one of the locally hosted models.

Furthermore, we also could not benchmark any larger, more popular models such as GPT4o due to the significant pricing constraints on their API. Although the web application supports translations with GPT4o, benchmarking and evaluating it would have been too expensive.

Finally, due to computational limitations, we were required to reduce the size of the dataset we evaluated the models with. The reduction in size could have potentially affected the validity of our findings, as some of the models may have shown better performance when evaluated with more data.

IX. Future Work

Looking ahead, there are several promising directions for future research and work. Evaluating more models across a wider range of programming languages would give us a more complete view of LLM code translation performance. Additionally, it would also provide better insights into which types of models perform best for certain translations, leading to further research into the differences of these models and how they affect code translation. We're also interested in studying how different LLMs perform when handling translations between different programming paradigms, such as translating between functional languages and object-oriented languages. Another valuable direction would be exploring prompt engineering techniques to discover if certain prompting styles and formats significantly influence translation outcomes. Lastly, expanding and improving the web application by adding support for more programming languages, implementing cloud-based model deployment to address performance issues, and potentially creating a browser extension version would make this tool more accessible and valuable for developers in real-world coding scenarios.

X. Conclusion

The CodeSynapse project provides a thorough evaluation of large language models (LLMs) for translating code between Python, C++, and Java. Our results show that Llama-3.2/3B is the best overall model for code translation, outperforming others in all language pairs except for C++ to Java. This suggests that fine-tuning and model specialization are important for improving performance. We also found that there is no clear link between model size and translation ability, and smaller models, when properly tuned, can perform just as well or even better than larger models for certain tasks.

We used evaluation metrics like CodeBLEU, which consider both syntax and meaning, and found that these metrics work better than traditional BLEU scores for assessing code translations. This highlights the need for better metrics that capture the complexities of programming languages.

The CodeSynapse web app offers an easy way for users to experiment with different models and see which performs best for their translation needs. This tool not only helps developers and researchers but also gives important insights into where current models succeed and where they need improvement.

Our findings suggest that while LLMs like Llama are currently leading in performance, there is still room for improvement. Future research should focus on improving model accuracy, fine-tuning methods, and developing better evaluation tools to capture the full complexity of code translation.

References

[1] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021.

[2] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages, 2020.

[3] Qiushi Sun, Nuo Chen, Jianing Wang, Xiang Li, and Ming Gao. Transcoder: Towards unified transferable code representation learning inspired by human skills, 2024.

[4] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent J. Hellendoorn. A systematic evaluation of large language models of code, 2022.