CCSW-325 Software Construction

Class Project Part 1

*Papers’ Summary*

# **Unraveling the Potential of Large Language Models in Code Translation: How Far Are We?**

This study explores the strengths and weaknesses of large language models (LLMs) in code translation, revealing their struggles due to limited exposure to parallel multilingual code. The authors introduce **PolyHumanEval**, a benchmark extending HumanEval to **14 programming languages**, and evaluate over **110,000 translations** using models like CodeLlama and StarCoder. Findings show that LLMs perform well when translating into Python but struggle with Python-to-other translations. The study proposes **intermediary translation** (using Go as a bridge language) and **self-training** (fine-tuning on self-generated parallel data), achieving an **11.7% accuracy improvement**. While these methods enhance performance, **challenges remain in handling complex** language-specific **features**, indicating room for further research.

# **Towards Translating Real-World Code with LLMs: A Study of Translating to Rust**

This paper investigates LLM-based translation of real-world code into Rust, a challenging task due to Rust’s strict safety rules. The authors develop **FLOURINE**, a tool that validates translations via **cross-language differential fuzzing**, eliminating the need for pre-existing test cases. They evaluate five LLMs (GPT-4, Claude 3, Gemini Pro, etc.) on **408 real-world code samples** from **seven open-source projects**, finding that the best model **successfully translates** **47% of cases**. The study also explores feedback mechanisms to refine translations but finds that counterexample-based feedback is less effective than simple re-prompting. Results highlight LLMs’ potential in real-world code migration but emphasize **the need for improved error handling** and semantic correctness.

# **TRANSAGENT: An LLM-Based Multi-Agent System for Code Translation**

by effectively fixing syntax and semantic errors. It consists of four agents: **Initial Code Translator**, **Syntax Error Fixer**, **Code Aligner**, and **Semantic Error Fixer**. The key innovation is **error localization**, which narrows down faulty code blocks, making fixes more efficient. The system is evaluated on a newly constructed benchmark with Python, Java, and C++ translation tasks, using **Computational Accuracy (CA)**, **CodeBLEU**, and **Mapping Accuracy** as metrics. Results show that TRANSAGENT **outperforms UniTrans and TransCoder**, achieving up to **33.3% higher accuracy** and **39.6% better code mapping** while being more efficient. It generalizes well across different LLMs, reducing translation errors with fewer iterations. Despite some computational costs, its fine-grained error handling makes it a powerful tool for **more accurate and reliable automated code translation**.

# **Exploring and Unleashing the Power of Large Language Models in Automated Code Translation**

The paper *Exploring and Unleashing the Power of Large Language Models in Automated Code Translation* proposes **UniTrans**, a framework to enhance code translation using LLMs like GPT-3.5 and LLaMA. The **idea** is to address LLMs' limitations (e.g., logic errors, I/O type mismatches) by leveraging auto-generated test cases for translation augmentation and iterative repair. The **method** involves three phases: generating test cases from source programs, augmenting translations with test case-guided prompts, and repairing errors using execution feedback. The study uses a **cleaned dataset** of 568 Python/Java/C++ functions from GeeksforGeeks, evaluating with **computational accuracy (CA)** and **exact match accuracy (EM Acc)**. **Results** show UniTrans boosts LLMs significantly (e.g., LLaMA-7B’s CA improves by 28.58%), outperforming state-of-the-art transpilers. **Reflection** highlights UniTrans’s effectiveness in mitigating LLM weaknesses but notes challenges in resolving subtle precision errors, suggesting future work on fine-grained discrepancy detection. The framework demonstrates practical utility but underscores the need for iterative repair balancing efficiency and accuracy.

# **Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code**

This study looks at how well large language models (LLMs) can translate code between different programming languages like C, C++, Go, Java, and Python. The authors tested **1,700 code samples** from various benchmarks and real-world projects to see how accurate the translations were. The **results showed that LLMs often make mistakes**, with **accuracy rates ranging from just 2.1% to 47.3%**. They found 15 different types of translation bugs, including incorrect data types, API mismatches, and missing dependencies. The study also compared LLMs to traditional code translation tools and found that while LLMs work better in some cases, older tools like C2Rust still create safer and more reliable translations. To help fix mistakes, the researchers suggest using better prompts that give the LLMs feedback on their errors, which improved accuracy by 5.5%.The findings indicate that while LLMs show promise, they require **better contextual understanding, structured feedback mechanisms, and hybrid approaches combining rule-based and AI-driven methods** for more reliable code translation.

# **Sources:**

[Unraveling the Potential of Large Language Models in Code Translation: How Far Are We?](https://arxiv.org/pdf/2410.09812)

[Towards Translating Real-World Code with LLMs: A Study of Translating to Rust](https://arxiv.org/pdf/2405.11514)

[TRANSAGENT: An LLM-Based Multi-Agent System for Code Translation](https://arxiv.org/pdf/2409.19894)

[Exploring and Unleashing the Power of Large Language Models in Automated Code Translation](https://dl.acm.org/doi/pdf/10.1145/3660778)

[Lost in Translation: A Study of Bugs Introduced by Large Language Models while Translating Code](https://dl.acm.org/doi/pdf/10.1145/3597503.3639226)