Chain-of-Thought LLM-Based Code Translation

Using Chain-of-Thought to Improve LLM-Based Translation of C++ to Java

Blago Gunev, TU Delft

1. Introduction

Large Language Models (LLMs) have exploded in popularity due to their success in **text** and **code generation**. Many programmers are using them to learn syntax, yet no study has investigated the effects different prompting methods have on the results. Consequently, this research explores **Chain-of-Thought** prompting for LLM-based C++ to Java translation.

2. Research Questions

1. Does Chain-of-Thought prompting **improve the compilation success rate** of C++ programs translated to Java when compared to simple prompting?
2. Does Chain-of-Thought prompting result in a **higher rate of behaviour-preserving programs** than simple prompting when translating C++ source code to Java?
3. What is the **difference in code quality** between Chain-of-Thought and simple prompting of LLMs when translating C++ code to Java?

Contact Information

- Email: bgunev@tudelft.nl

Correctness metrics

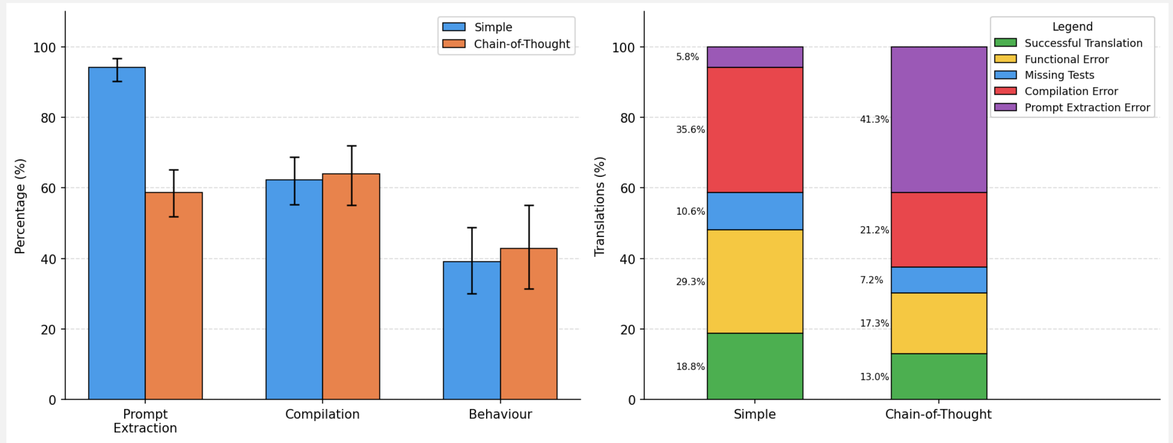


Figure: Results summary. The left chart presents the success rates at prompt extraction, compilation and behaviour verification steps. Data presented with 95% Wilson confidence intervals. The right chart presents the percentage-wise distribution of samples across prompt extraction, compilation and functional errors, missing tests and functionally correct translations.

3. Materials

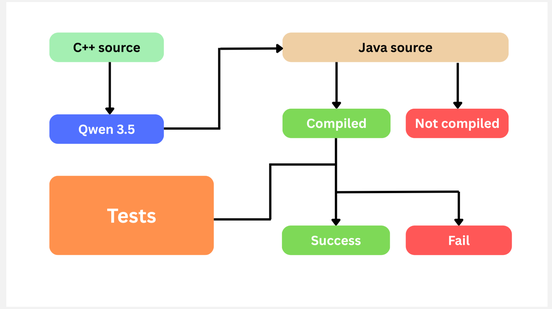
The LLM used for translating is **Qwen3.5-9B** [1], the choice of which was influenced by practical constraints. **C++** and **Java** were chosen as they are widespread; thus, keeping the focus on LLM behaviour.

The dataset used is **open-r1/codeforces** [2]. It includes **user submitted** C++ code samples and **input/output pairs** as test cases.

Prompt templates consist of **task statement**, **context** and **constraints**. Simple template provides **minimal information**, Chain-of-Thought template **expands on the context**.

4. Experimental setup

1. **LLM Part** – Both prompt **templates with C++ code** are supplied to the LLM to produce responses.
2. **Benchmark Part** – Responses are processed to **extract Java sources**. Then, those sources are tested for **syntax correctness**. Finally, compiled sources are evaluated for **functional correctness**.



Code metrics

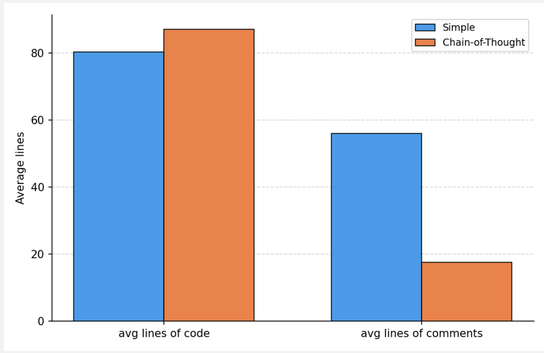


Figure: Code quality metrics. Average lines of code computed over all lines with functional code. Average lines of comments computed over lines that contain a single comment or are part of a comment block.

5. Conclusions

- Some evidence exists to show CoT prompting results in **more syntactically correct** and **behaviourally-preserved** code.
- When reasoning with CoT, LLMs might be capable of translating **more complex code**.
- Future work should address the faults unrelated to the code translation.

References

[1] Qwen Team, "Qwen3.5: Towards native multimodal agents," February 2026. https://qwen.ai/blog?id=qwen3.5.

[2] G. Penedo, A. Lozhkov, H. Kydlíček, L. B. Allal, E. Beeching, A. P. Lajarín, Q. Gallouédec, N. Habib, L. Tunstall, and L. von Werra, "Codeforces." https://huggingface.co/datasets/open-r1/codeforces, 2025.