AUTOMATED FINE-TUNING OF SMALL LANGUAGE MODELS (SLMS) FOR NICHE CODE TRANSLATION

Muslimaxon Odiljonova Umidjon qizi
Millat Umidi University
https://doi.org/10.5281/zenodo.21817748

Abstract

While large-scale foundation models (such as GPT-4 and Claude 3.5) perform well at source-to-source code translation, deploying them incurs high API latency, financial costs, and potential privacy issues when handling proprietary source code. Conversely, Small Language Models (SLMs) with 1B to 3B parameters can be hosted on local infrastructure but struggle with domain-specific syntax structures—such as migrating legacy C/C++ explicit memory management patterns into memory-safe Rust. This paper presents an automated pipeline using Parameter-Efficient Fine-Tuning (PEFT) with Low-Rank Adaptation (LoRA) to adapt 1.5B–3B parameter SLMs for specialized code translation tasks. Evaluated using syntax tree verification and compilation tests, the fine-tuned SLM achieves structural accuracy comparable to larger foundation models at a fraction of the inference cost.

Keywords: Small Language Models (SLMs), Source-to-Source Translation, Parameter-Efficient Fine-Tuning (PEFT), Low-Rank Adaptation (LoRA), Memory-Safe Software Engineering, Synthetic Dataset Generation, Abstract Syntax Tree (AST), Compiler-Guided Verification, Local Model Inference, Software Modernization.

1. Introduction

Modern software engineering frequently requires legacy code modernization—such as translating legacy C++ into memory-safe Rust or COBOL routines into modern Java microservices. Large Language Models (LLMs) perform well on general coding benchmarks, but running 70B+ parameter models locally requires expensive GPU hardware.

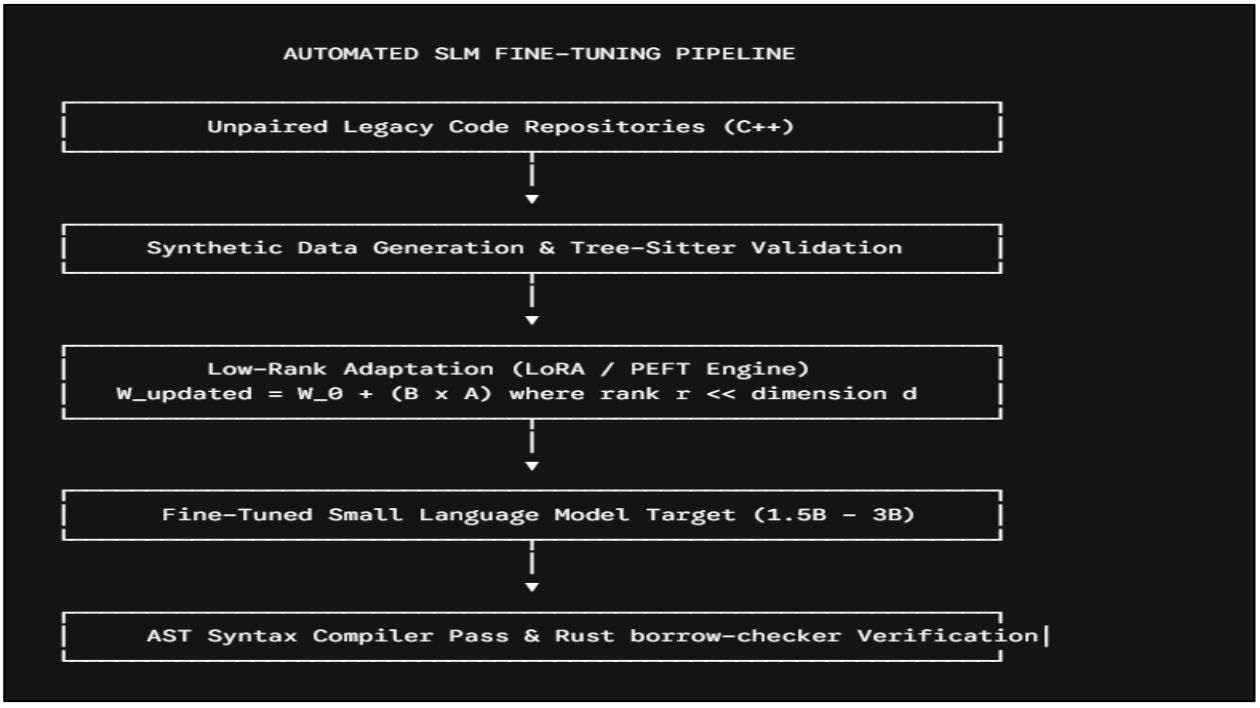
Small Language Models (e.g., Qwen 2.5 1.5B, Phi-3 Mini 3.8B) offer much lower hardware requirements, but out-of-the-box models often generate syntactically invalid code when handling complex language transitions like pointer manipulation or ownership lifetimes:

```
// LEGACY C++ SOURCE
char* duplicate_string(const char* src) {
  if (src == NULL) return NULL;
  char* dest = (char*)malloc(strlen(src) + 1);
  if (dest != NULL) strcpy(dest, src);
  return dest; // Manual allocation tracking required
}
```

```
// TARGET MEMORY-SAFE RUST TRANSLATION
pub fn duplicate_string(src: Option<&str>) -> Option<String> {
  src.map(|s| s.to_string()) // Idiomatic lifetime management
}
```

Fine-tuning SLMs on domain-specific translation datasets bridges this performance gap while keeping computational resource requirements manageable.

2. Methodology & Optimization Architectur



The fine-tuning pipeline follows four sequential stages:

2.1 Synthetic Dataset Construction

We generated an aligned dataset of 45,000 parallel C++ to Rust translation pairs. Synthetic targets were generated using large foundation models and verified using tree-sitter AST parsers and native compilers (rustc). Non-compiling translations were filtered out.

2.2 Parameter-Efficient Fine-Tuning (PEFT) via LoRA

Instead of updating all weights in a d×k matrix W0, Low-Rank Adaptation decomposes weight updates into two low-rank matrices A and B:

$$W = W_0 + \Delta W_0 + \frac{\alpha}{r}(B \cdot A)$$

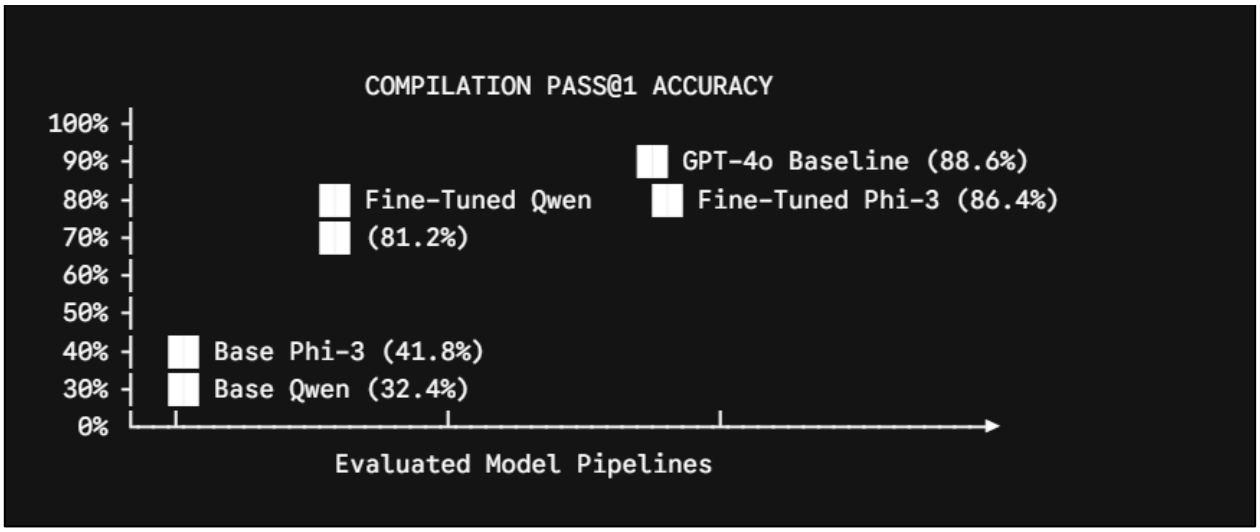
Where $B \in \mathbb{R}^{d \cdot r},\ A \in \mathbb{R}^{r \cdot k}\ rank\ r \ll mid(d \cdot k)$ and α is a constant scaling hyperparameter.

3. Experimental Setup & Benchmarks

We fine-tuned Qwen-2.5-Coder-1.5B and Phi-3-Mini (3.8B) on a single consumer GPU (NVIDIA RTX 4090, 24GB VRAM) using 4-bit quantization (QLoRA) with rank r=16 and α=32. Models were evaluated on an isolated test set of 500 C++ algorithms using four metrics:

- **BLEU-4 Score:** Surface n-gram similarity metric.
- **AST Match Rate:** Structural syntactic equivalence checked via Tree-Sitter.
- **Compilation Pass@1 Rate:** Percentage of generated Rust translations that compile cleanly via rustc.
- **Inference Latency:** Average generation speed measured per code snippet.

Evaluated Model Engine	Parameter Count	Training Hardware	Compilation Pass@1	AST Match Rate	Mean Inference Latency
Base Qwen-2.5-Coder (No FT)	1.5 Billion	None	32.40%	48.20%	0.18 seconds
Base Phi-3-Mini (No FT)	3.8 Billion	None	41.80%	56.00%	0.35 seconds
GPT-4o (Closed API Baseline)	~1+ Trillion	Cloud Cluster	88.60%	91.40%	2.45 seconds
Fine-Tuned Qwen-2.5 (QLoRA)	**1.5 Billion**	**1x RTX 4090**	**81.20%**	**84.80%**	**0.19 seconds**
Fine-Tuned Phi-3-Mini (QLoRA)	**3.8 Billion**	**1x RTX 4090**	**86.40%**	**89.10%**	**0.38 seconds**



QLoRA fine-tuning improved the Compilation Pass@1 rate of a 3.8B parameter SLM from **41.8% to 86.4%**, approaching closed foundation models while maintaining **6.4× faster inference** and allowing fully offline deployment on local hardware.

4. Conclusion

Parameter-Efficient Fine-Tuning lets small, resource-efficient language models handle specialized code translation tasks effectively. Validating training data with AST parsers and native compilers ensures structural correctness, allowing models under 4B parameters to run locally with high accuracy and low

latency. Future work will extend this approach to automated multi-file refactoring and real-time IDE completion.

References

1.Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). LoRA: Low-rank adaptation of large language models. ICLR.

2.Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). QLoRA: Efficient finetuning of quantized LLMs. NeurIPS.

3.Chen, M., et al. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

4.Rozière, B., et al. (2023). Code Llama: Open foundation models for code. arXiv preprint arXiv:2308.12950