Scaling Code Translation

Introduction

Systems implemented in obsolete programming languages become increasingly difficult to maintain and evolve.

The scope of this challenge is significant. Analysis of SRDR data for 287 projects found that 22% used Ada as the primary programming language.[1]

Manual translation to newer programming languages is a slow, labor-intensive process (<5K SLOC per staff year) with a nontrivial risk of failure.

Large language models (LLMs) show promise for program translation at small scales (dozens of LOC) but break down as scale increases.

Methods

The SEI is creating a translation workflow that *incrementally translates Ada to C++*.

- Use the raw translation potential of LLMs.
- Generate context-sensitive prompts that reduce translation errors.
- Automatically generate glue code to incrementally integrate results.
- Limit, but not eliminate, developer involvement for quality control.

Our goal is to demonstrate **at least a 4x** improvement in translation speed.

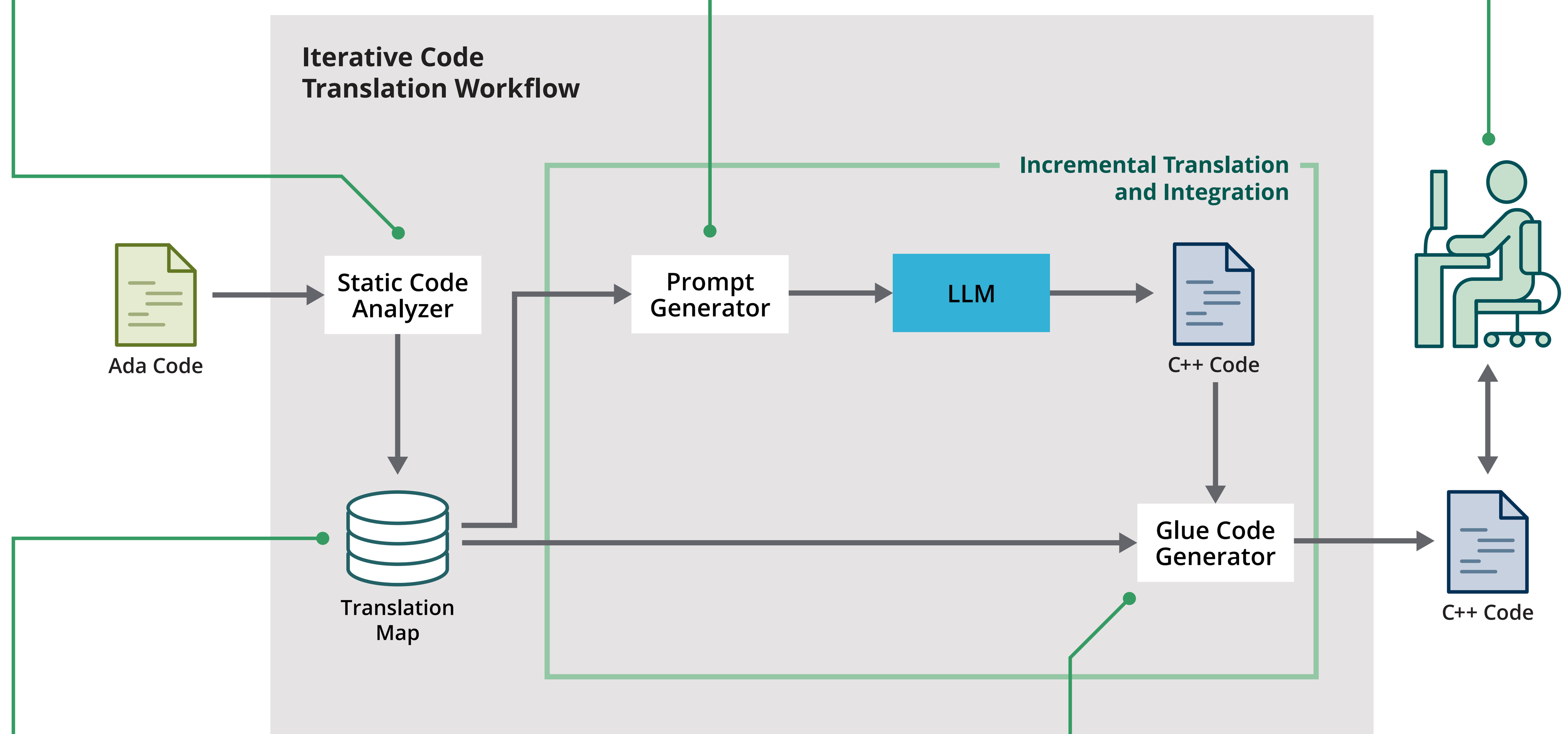
1 Clark, B.; Miller, C.; McCurley, J.; Zubrow, D.; Brown, R.; & Zuccher, M. *Department of Defense Software Factbook.* CMU/SEI-2017-TR-004. Software Engineering Institute. 2017.

Augmenting LLMs with new approaches allows us to **accelerate modernization** of systems using **dated programming languages**.

1 Static Code Analyzer gathers facts that LLMs struggle to infer, especially with limited context.

3 Prompt Generator provides translation-specific instructions and manages the LLM's context window to provide only data that is needed.

5 Developers remain in the loop to review, fix, and ultimately approve translations. Reducing the number of errors accelerates the overall modernization effort.

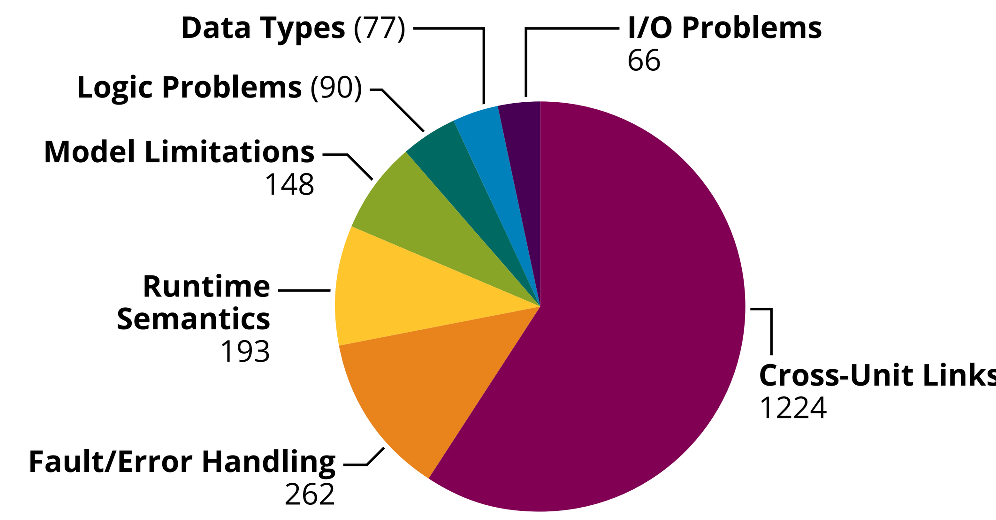


2 Translation Map preserves this information for use across increments and keeps track of the current state of translation.

4 Glue Code Generator puts the pieces together as translation progresses, allowing results to be validated incrementally instead of at the end of a long process.

Baseline LLM Performance

During translation from Ada to C++, LLMs inject ~140 errors per KSLOC.*



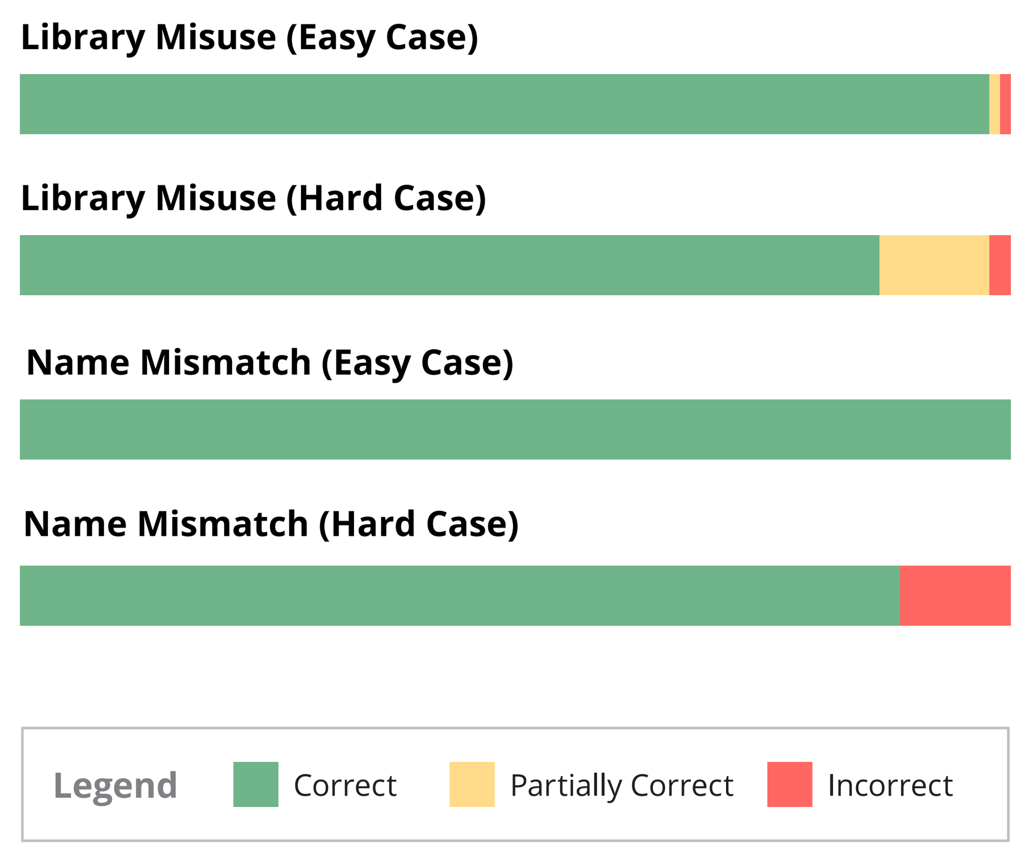
* Based on analysis of three translations of ~5K SLOC of Ada code using OpenAI models

Improving on the Baseline

Our approach generates prompts that inject missing context (e.g., type inference) and provide guidance to avoid recurring errors.

We piloted this approach on the two most common categories of cross-unit link errors (accounting for 41.5% of all errors).

Success ranges from 86.7% to 100%.



We estimate that current and planned improvements can **lower the error injection rate to <40 errors/KSLOC**.

James Ivers | jivers@sei.cmu.edu
Tapajit Dey, Chris Seifried, Mena Kostial

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM25-1391