Enhancing LLM-based Compiler Fuzzing with Error Detection and Correction

Iti Shree Karine Even-Mendoza Tomasz Radzik

Abstract

LLM-based compiler fuzzers generate diverse but often invalid test programs, limiting their effectiveness in testing compiler optimizations and backend components. Our framework systematically detects and fixes errors in these programs using a feedback loop powered by a local LLM.

Evaluated on black, grey, and white-box fuzzing approaches targeting GCC and LLVM/Clang, our tool improved test program validity from **12.3%-47.9%** to **55.9%-80.7%**, with processing overhead of only **11-13 seconds** per program. This led to significant coverage increases in critical compiler components, including a **20.2%** improvement in inlining coverage.

`ReFuzzer` Architecture

`ReFuzzer` employs a feedback loop with a local LLM to improve test program validity:

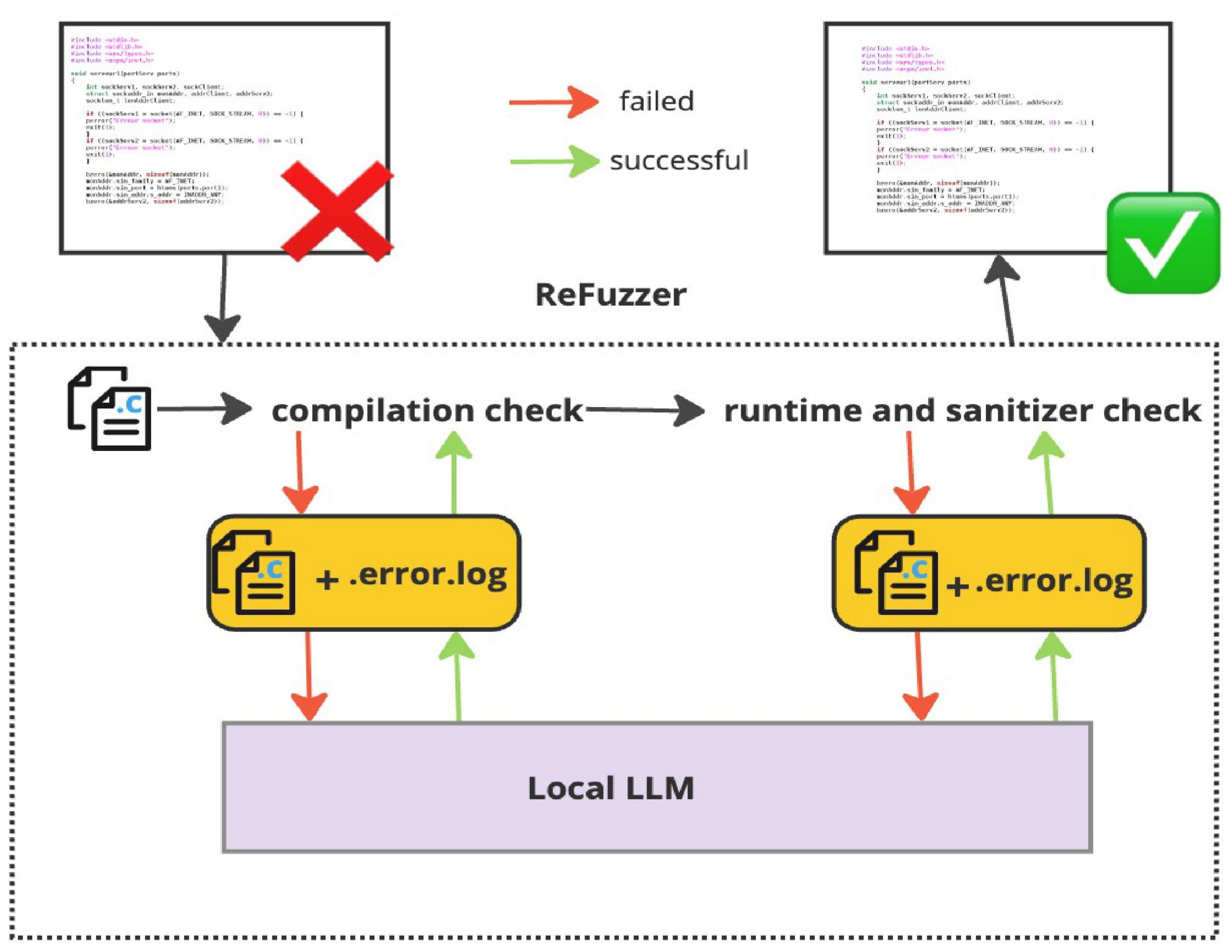

Figure 1. Framework overview: Detecting and correcting errors using an LLM-based feedback loop

The Problem

- Traditional compiler fuzzers generate valid but redundant programs
- LLM-based approaches offer diversity but produce many invalid programs
- Invalid programs fail to exercise important compiler optimizations
- Most fuzzing targets compiler front-ends, neglecting optimization passes

Our Solution

- Feedback loop: LLM + validation tools to correct errors while preserving diversity
- Static validation: Detect and fix compilation errors with structured feedback
- Dynamic validation: Identify and correct undefined behaviors before execution
- Targeted optimization testing: Focus on compiler middle and backend components

Key Results

Validity Improvement

- Static validity improved from 12.3%-47.9% to **55.9%-80.7%**
- Dynamic validity (programs without UBs) increased by **3.2×**
- Processing overhead: only **11-13 seconds** per program

Coverage Impact

- Inlining pass coverage: **+20.2%** (black-box), **+16.1%** (white-box)
- IR generation components: **+15.3%** average improvement
- Loop optimization passes: **+12.7%** coverage increase

Benefits & Applications

- **For compiler developers:** More effective testing of optimization passes with minimal pipeline changes
- **For testing teams:** Higher-quality test programs with 3-4× better validity rates
- **For research:** Bridges the gap between LLM diversity and traditional fuzzer validity
- **Extensibility:** Applicable to other programming languages and compilers

Future Work

- Support for additional programming languages and compiler infrastructures
- Specialized error correction for specific optimization passes
- Machine learning to predict high-coverage test programs
- Parallel processing to reduce feedback loop overhead

Take-home Message

`ReFuzzer` effectively addresses a critical gap in compiler testing by combining LLM-generated program diversity with systematic error detection and correction. The result: **4× more valid test programs** and **15-20% increased coverage** of critical compiler components with minimal overhead, along with security of using local LLM model.

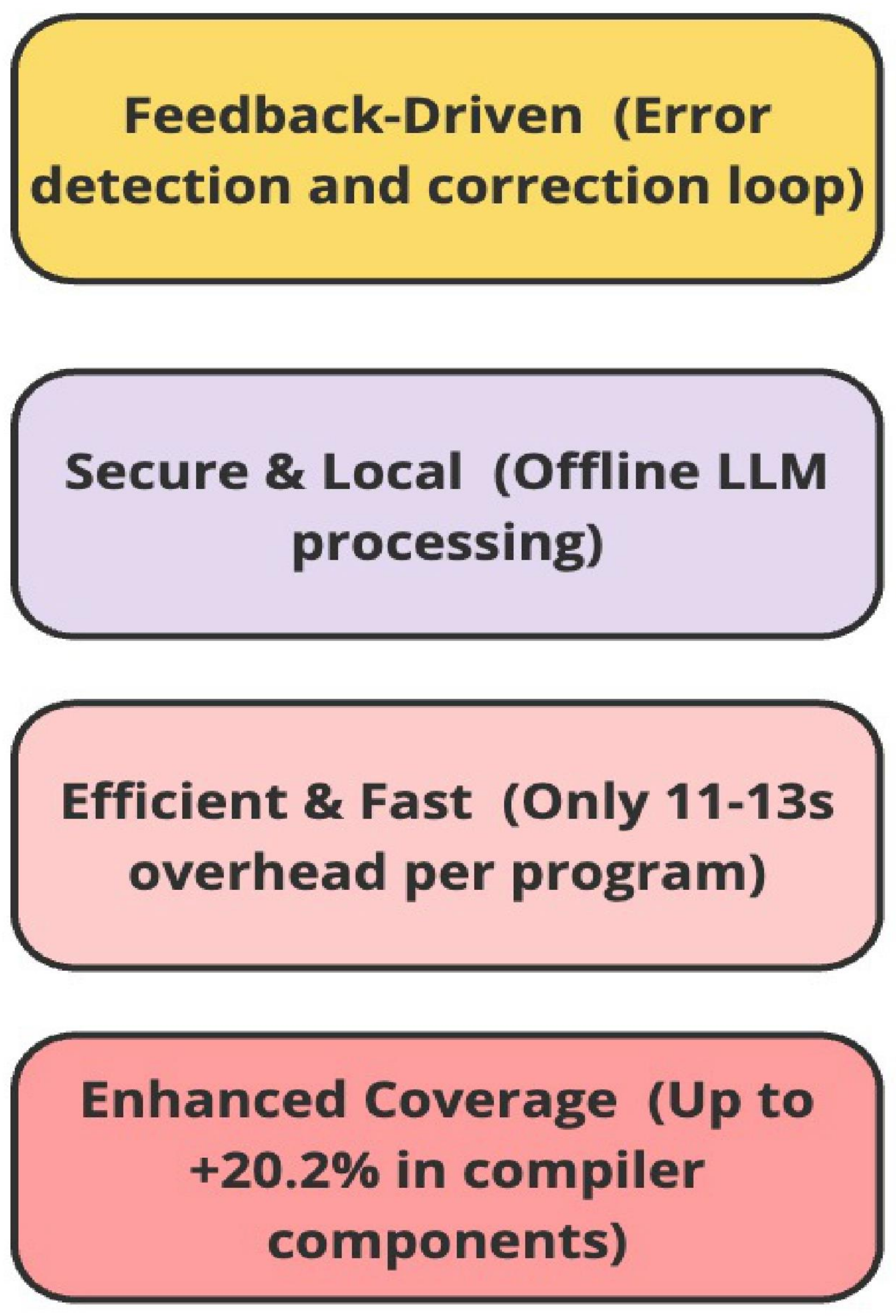
Feedback-Driven (Error detection and correction loop)
Secure & Local (Offline LLM processing)
Efficient & Fast (Only 11-13s overhead per program)
Enhanced Coverage (Up to +20.2% in compiler components)

Figure 2. Framework overview: Detecting and correcting errors using an LLM-based feedback loop