Toward Scalable Software Engineering Systems

Dongjae Lee

2025.11.16

Abstract

Recent advances in AI are driven by **scaling laws**: models improve as we add data, parameters, and compute. This idea is simple but powerful in a world that already has massive data centers and datasets. However, key software engineering techniques such as static analysis, fuzzing, and synthesis are still largely disconnected from this scaling perspective. In this essay, I point out this gap and propose software engineering (SE) systems that exploit abundant compute.

In his "Bitter Lesson"[1], 2025 Turing Award winner Richard Sutton argues that human-designed heuristics help when resources are scarce but hinder progress when resources become abundant. Methods that win at small scale often lose badly at large scale. The transformer architecture and scaling laws have made this pattern hard to ignore.

Yet core SE techniques built on classical algorithms—fuzzing, synthesis, and static analysis—have not fully absorbed these lessons. They rarely assume access to truly large-scale compute. For example, a typical fuzzer restarts once coverage saturates, discarding most of its prior work; more compute just means more random independent fuzzers. Program synthesis faces exploding search spaces as program size grows, and rule-based pruning is not enough to satisfy complex specifications. Static analysis is hard to parallelize, and increasing precision often slows fixed-point convergence.

If we can systematically harness large-scale compute to amplify them, we may unlock a new path to safer software. Today, these techniques are mostly tuned for CI/CD settings, where they must run under strict time and resource budgets. Most research therefore optimizes who finds more bugs faster under fixed limits. In parallel, however, we could run heavyweight, background-scale analysis and testing on critical software, aiming for much stronger safety guarantees. This calls for new methodologies to build systems that can compete—and win—in an "unlimited weight class" of compute.

[1] `https://www.cs.utexas.edu/~eunsol/courses/data/bitter_lesson.pdf`