Full-speed Fuzzing: Reducing Fuzzing Overhead through Coverage-guided Tracing

Stefan Nagy (snagy2@vt.edu), **Matthew Hicks** (mdhicks2@vt.edu)

Introduction

Of coverage-guided fuzzing's three main components:

(1) *test case generation,* (2) *code coverage tracing,* and (3) *crash triage*, code coverage tracing amounts to **over 90%** of total fuzzer runtime.

Current fuzzers identify coverage-increasing test cases by tracing *all of them*—even when **over 99.99% do not increase coverage and thus are discarded**.

We introduce **Coverage-guided Tracing**—an approach restricting tracing to test cases *guaranteed* to increase coverage while filtering-out the rest at *native speed*.

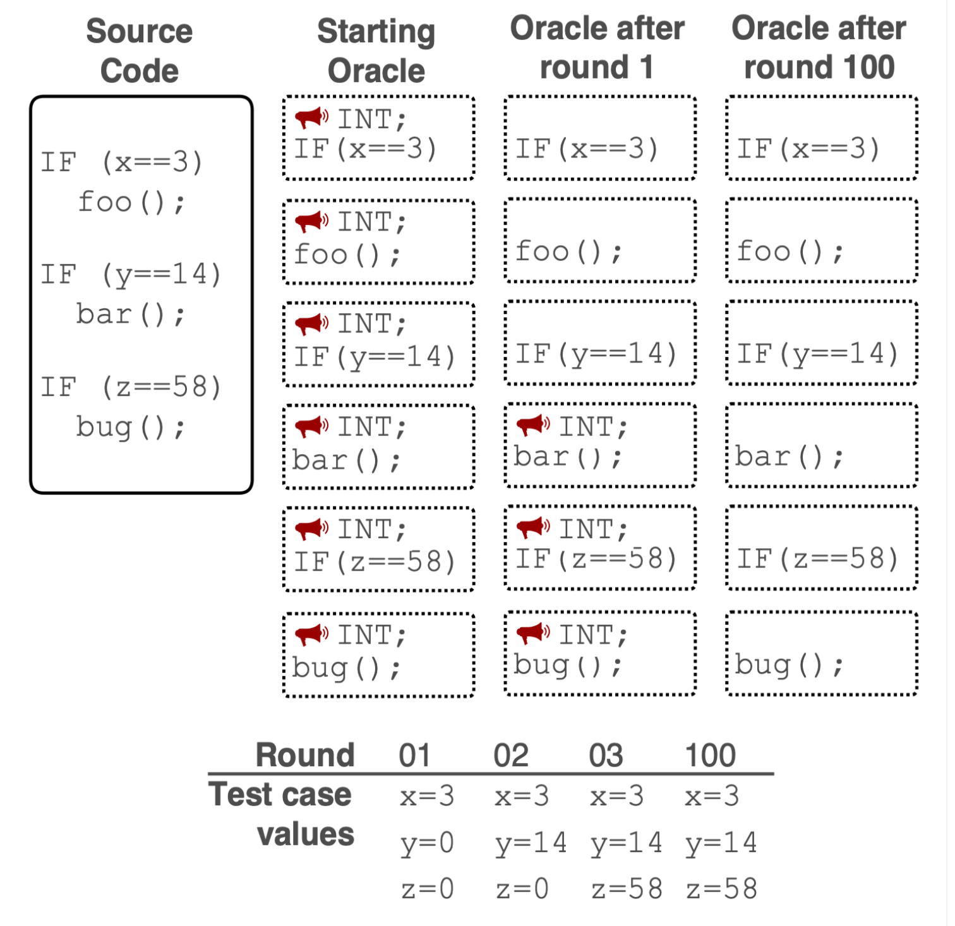
Coverage-guided Tracing

Driving observations:

(1) a small fraction of generated test cases increase coverage

(2) coverage-increasing test cases become less frequent over time

Coverage-guided tracing transforms the target binary so that it self-reports when a test case produces new coverage—without tracing. This restricts the expense of tracing to *only* coverage-increasing test cases.



Source Code	Starting Oracle	Oracle after round 1	Oracle after round 100

IF (x==3)
foo();
IF (y==14)
bar();
IF (z==58)
bug();
	INT;
	IF(x==3)
		IF(x==3)	IF(x==3)

	INT;
	foo();
		foo();	foo();

	INT;
	IF(y==14)
		IF(y==14)	IF(y==14)

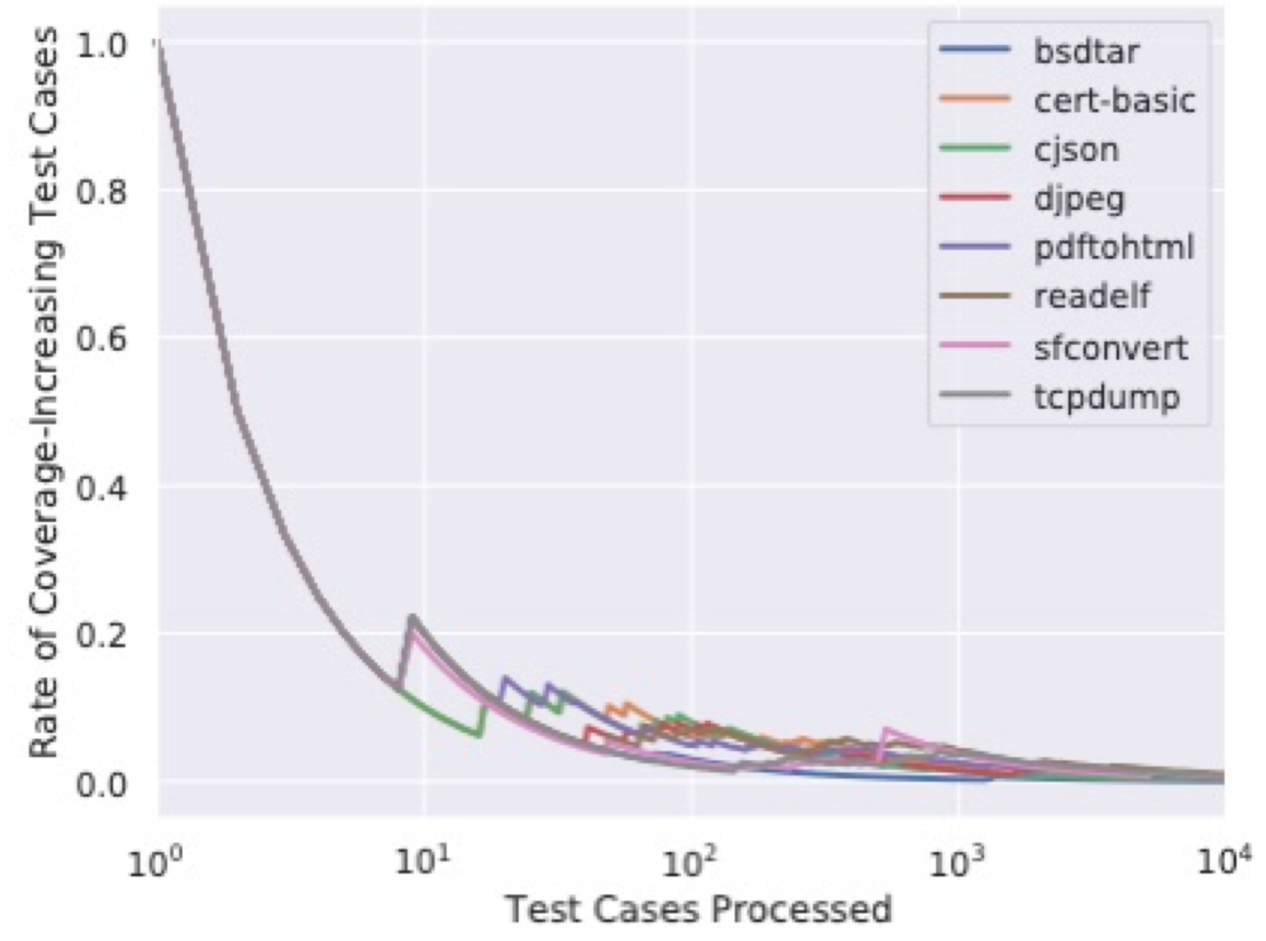
	INT;
	bar();
		INT;
		bar();
			bar();

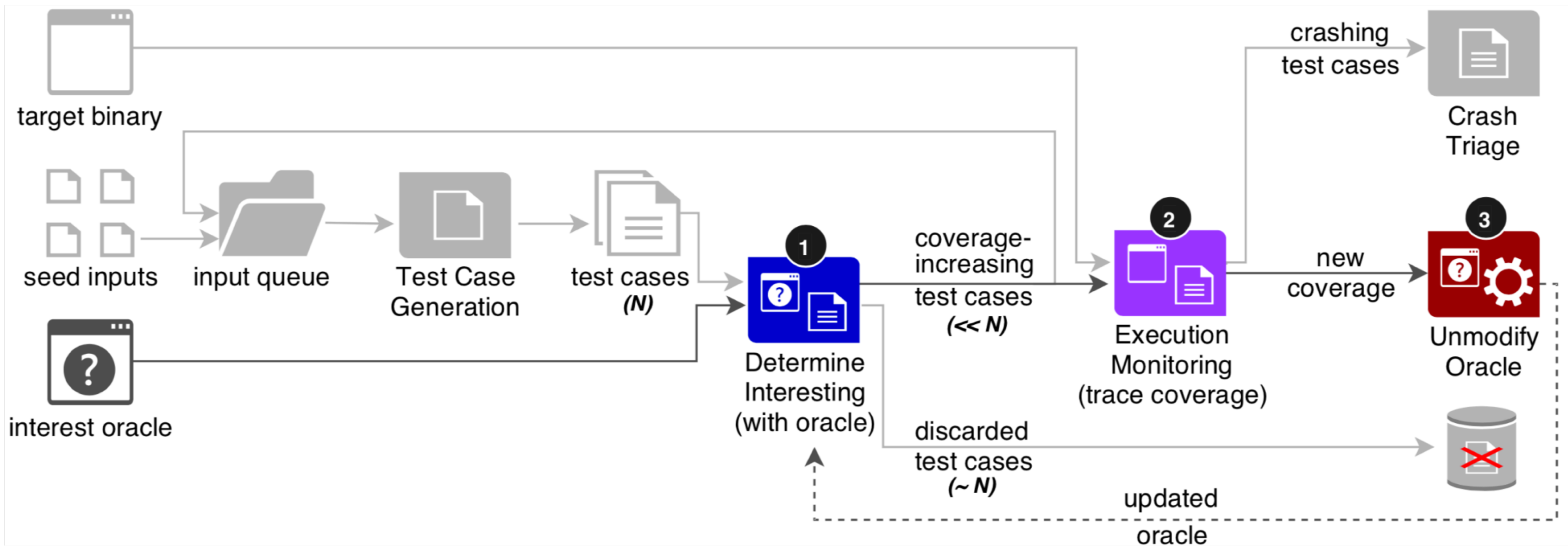
	INT;
	IF(z==58)
		INT;
		IF(z==58)
			IF(z==58)

	INT;
	bug();
		INT;
		bug();
			bug();

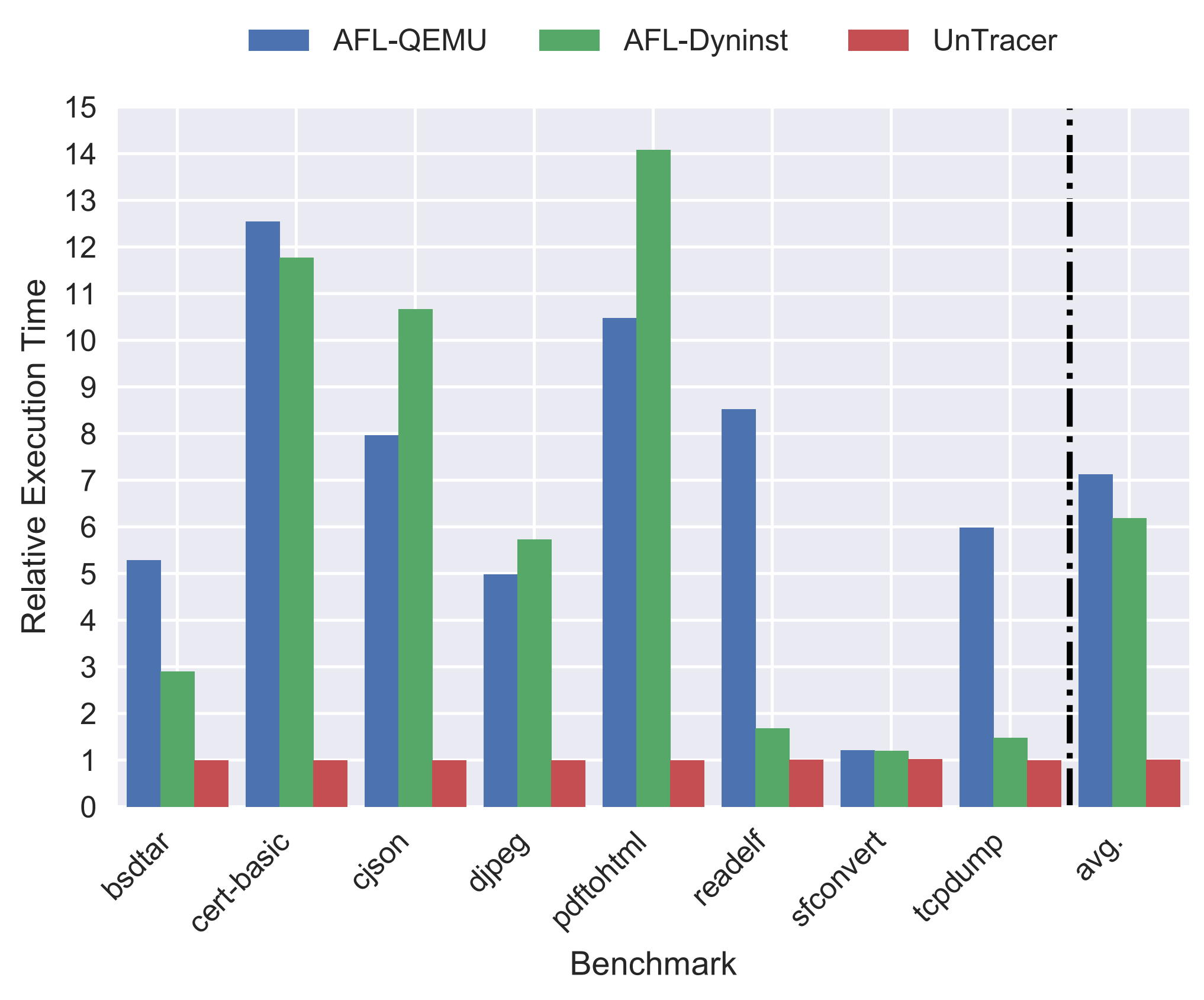
Round	01	02	03	100

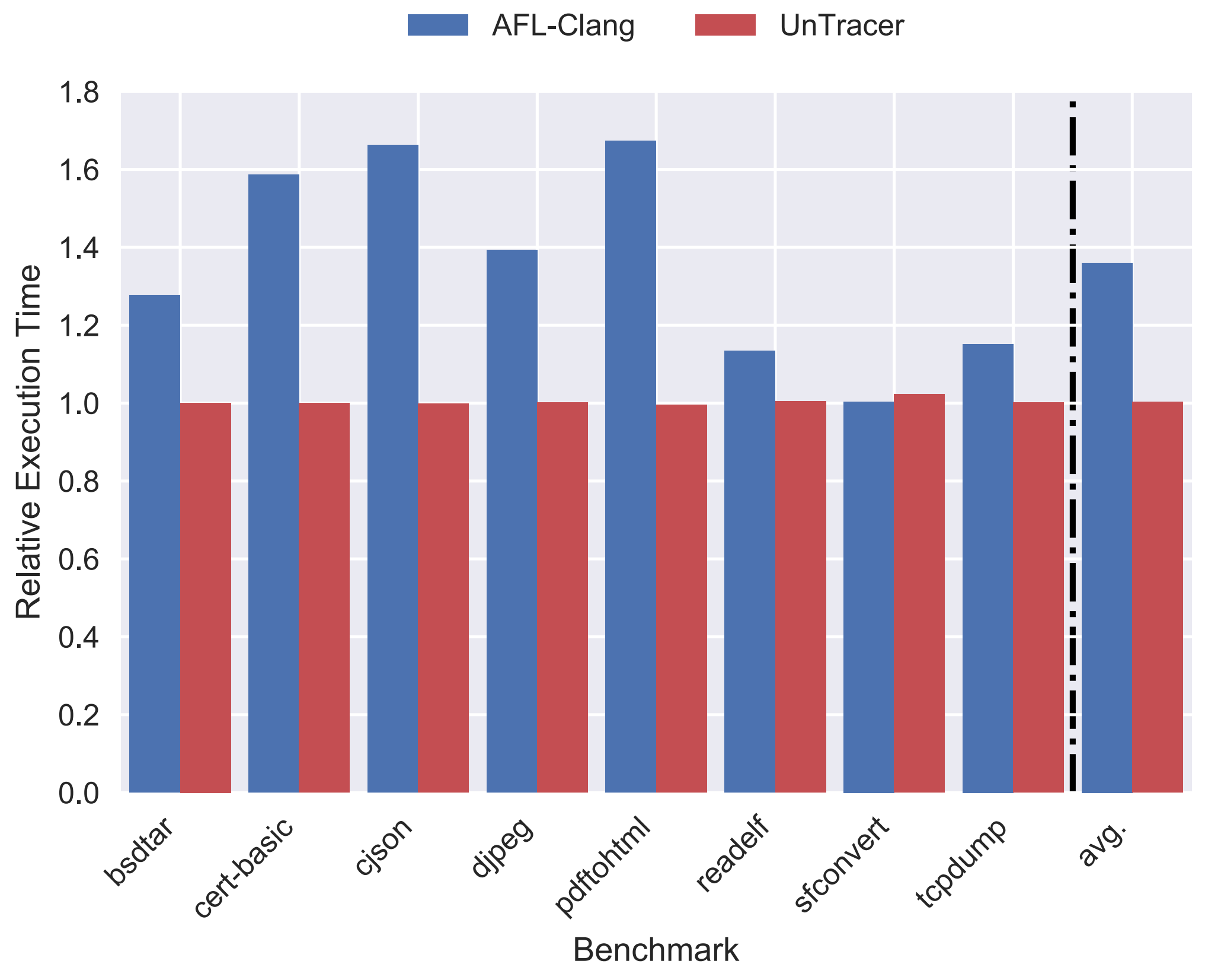
Test case values	x=3
	y=0
	z=0
		x=3
		y=14
		z=0
			x=3
			y=14
			z=58
				x=3
				y=14
				z=58

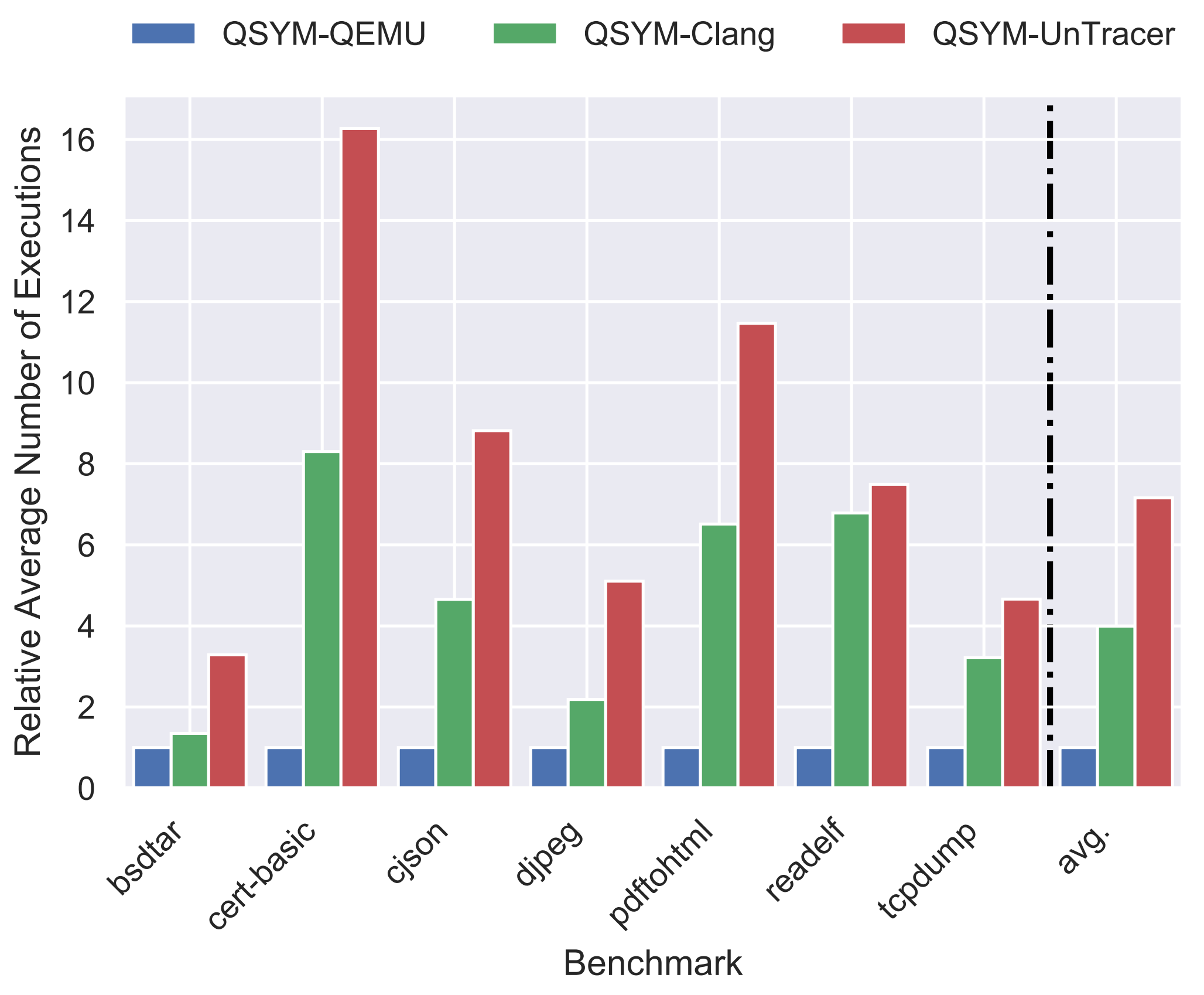




Tracing-only Overhead and Hybrid Fuzzing Evaluations







UnTracer: coverage-guided tracer based on the Dyninst black-box binary static rewriter

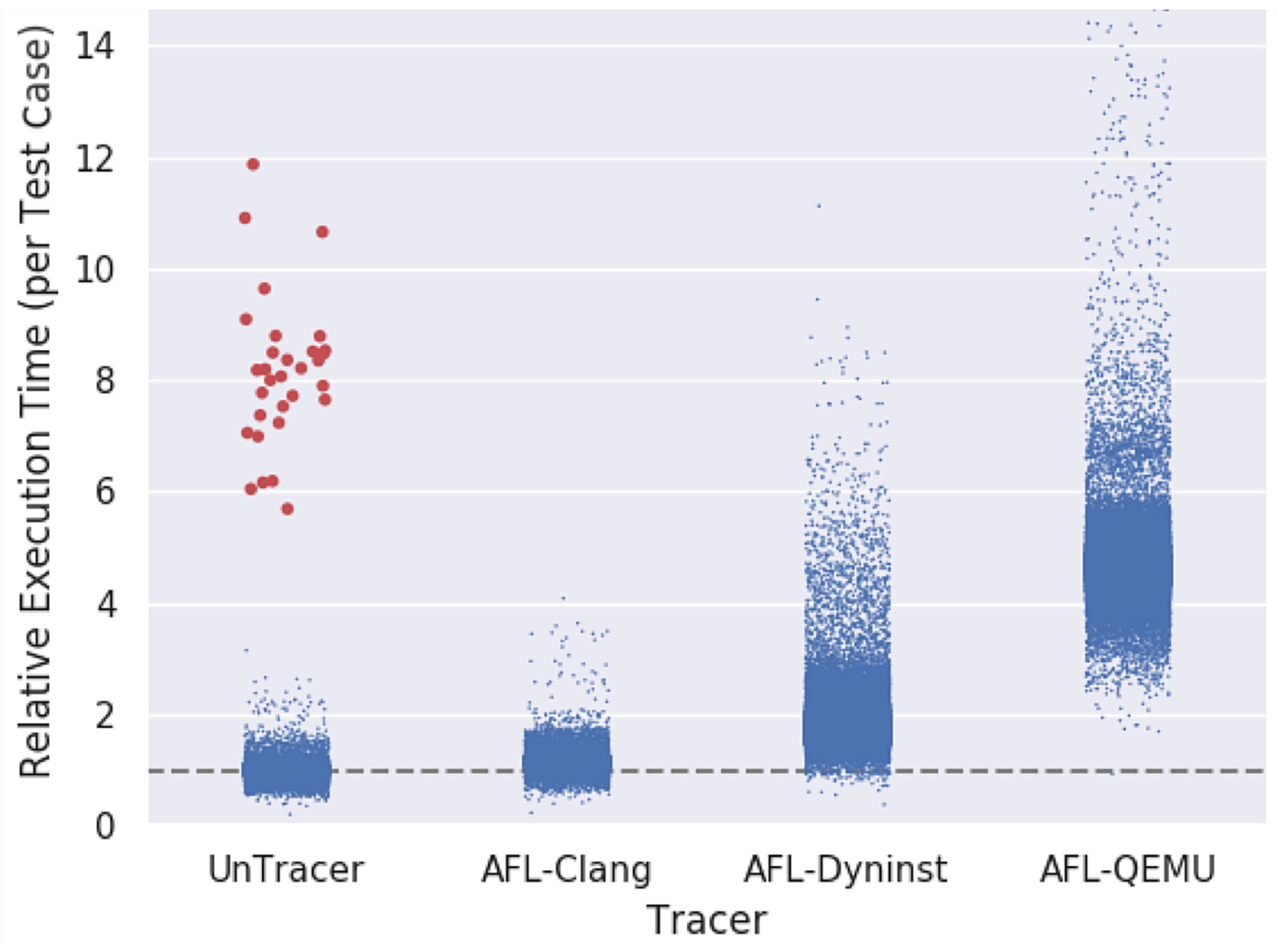
Tracing-only overhead evaluation:

- **36%** faster than AFL-Clang.
- **518%** faster than AFL-Dyninst.
- **612%** faster than AFL-QEMU.

Hybrid fuzzing throughput evaluation:

- **79%** more test cases QSYM-Clang.
- **618%** more test cases than QSYM-QEMU.

Discussion



- Orthogonal to other improvements in fuzzing.
- Increased overhead for coverage-increasing test cases (<<*N*) amortized by near-native speed for the rest (~*N*).
- Fully black-box (binary-only) approaches are feasible.
- Edge coverage an open challenge.

Conclusions & Future Work

Coverage-guided tracing leverages the fact that coverage-increasing test cases are the overwhelmingly uncommon case in fuzzing by modifying target binaries so that they self-report when a test case produces new coverage.

We report overhead reductions of as much as 1300% and 70% for black- and white-box tracing, respectively, and 616% and 79% more test case executions than black- and white-box tracing-based hybrid fuzzing.

Our current work focuses on improving the performance and precision of black-box binary fuzzing via static rewriting.

Where to find our software?

www.github.com/FoRTE-Research

☐	**UnTracer-AFL**	:	UnTracer integrated with AFL
☐	**afl-fid**	:	AFL for fixed-dataset experiments
☐	**FoRTE-FuzzBench**	:	our eight fuzzing benchmarks

Scan me

COMPUTER SCIENCE
VIRGINIA TECH