The 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)

Revealing Performance Issues in Server-side WebAssembly Runtimes via Differential Testing

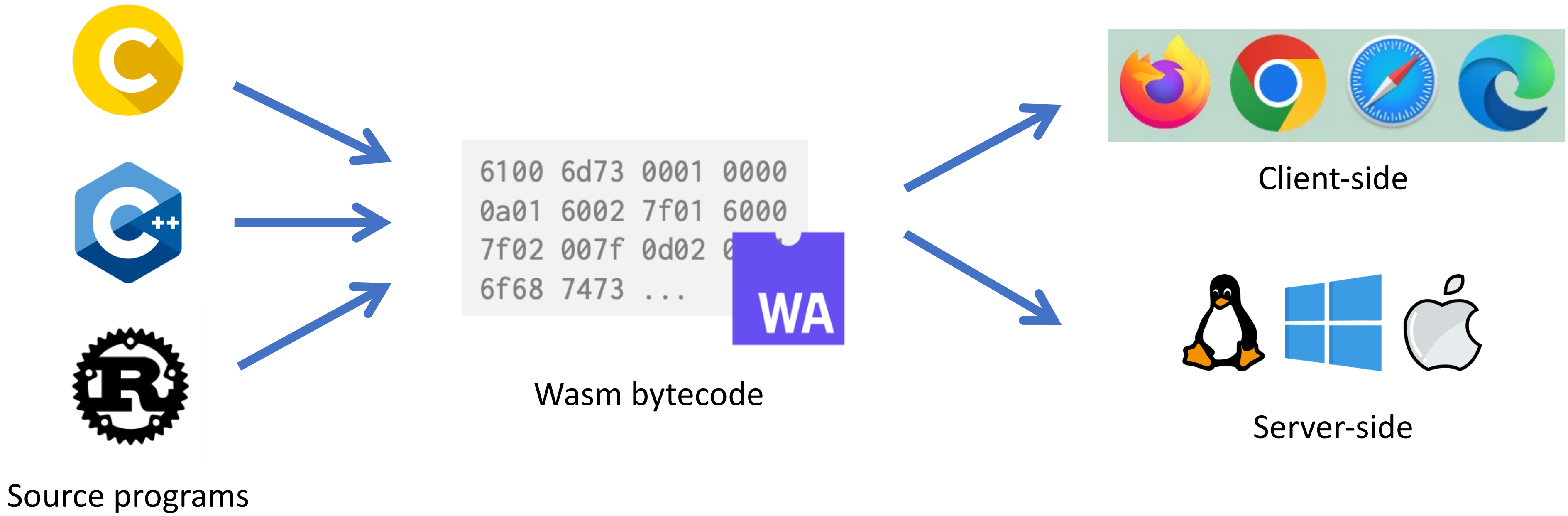
Shuyao Jiang[1], Ruiying Zeng[2], Zihao Rao[2], Jiazhen Gu[1], Yangfan Zhou[2], Michael R. Lyu[1]

1. Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China

2. School of Computer Science, Fudan University, Shanghai, China

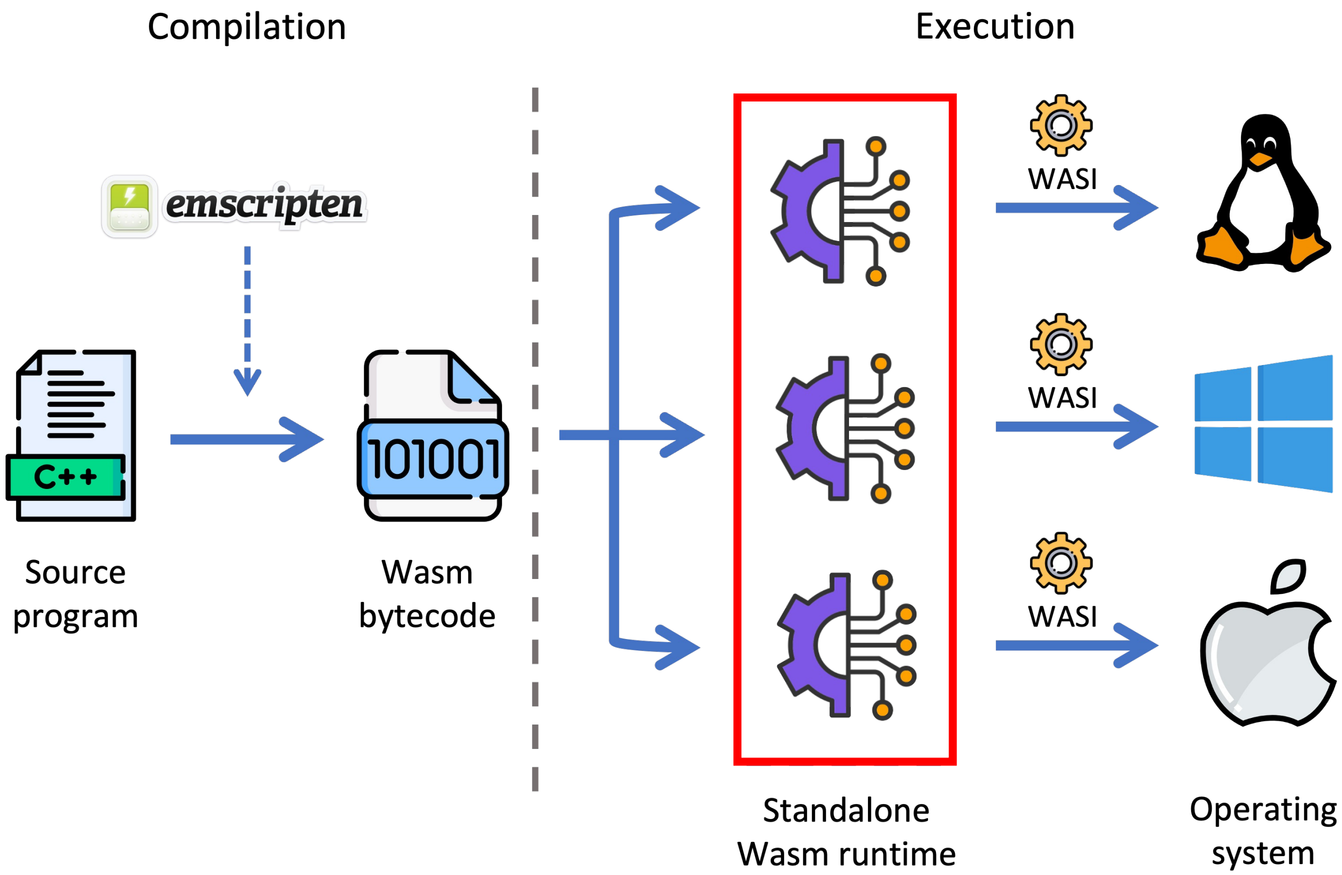
WebAssembly (Wasm)

- A low-level bytecode format
- Fast, safe, portable
- Support in both browsers and server-side apps



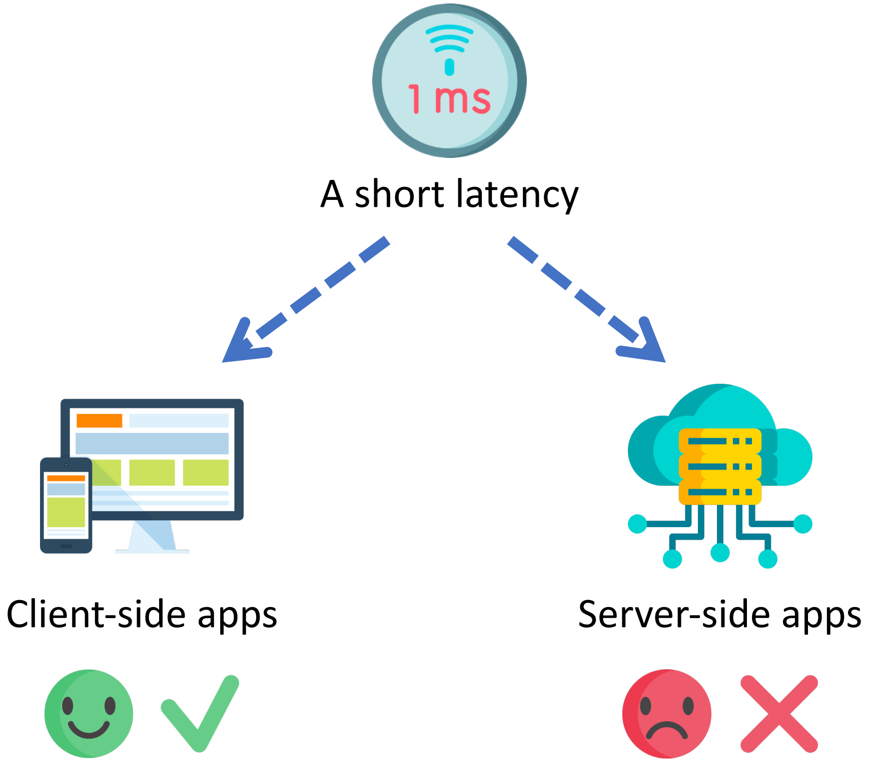
Server-side Wasm Workflow

- Key component: Standalone Wasm runtimes



Performance Issues in Server-side Wasm

- The impact of performance issues on the server side is usually greater than that on the client side.



- Standalone Wasm runtimes are still immature and more likely to cause performance issues.

Major browsers: Well-developed

BYTECODE ALLIANCE

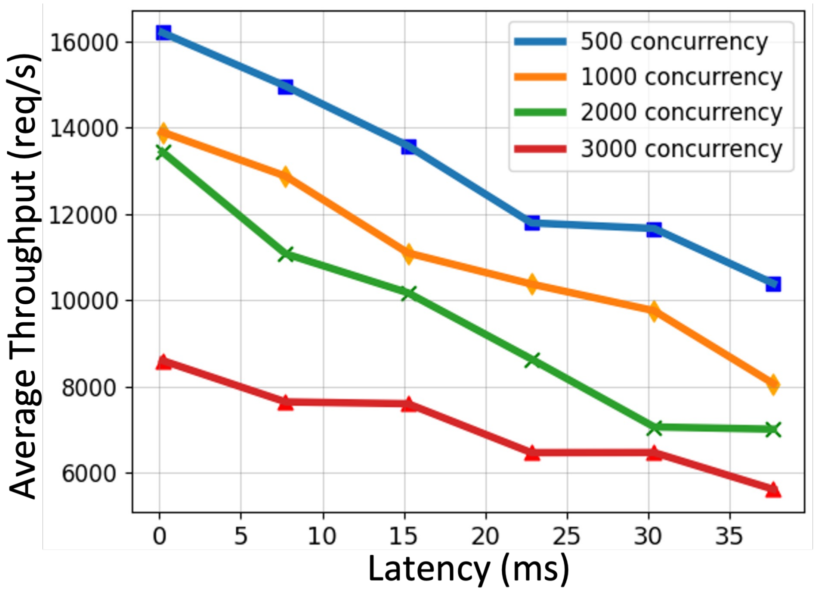
Wasmer

WasmEdge

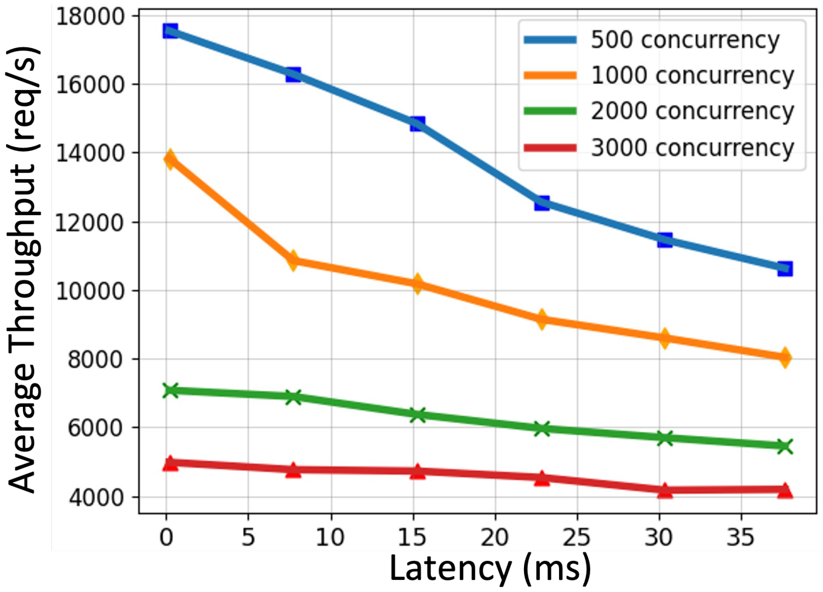
Standalone Wasm runtimes: Immature

Impact of Performance Issues: A Real Case

- Impact of WasmEdge runtime latency on service throughput
  - Service: [microservice-rust-mysql]



(a) 10,000 request



(a) 50,000 request

A 30ms-latency will result in a 20% to 50% drop in service throughput!

Challenges & Solutions

- **Our goal:** Revealing performance issues in standalone Wasm runtimes

Challenge: Hard to manually analyze each Wasm runtime

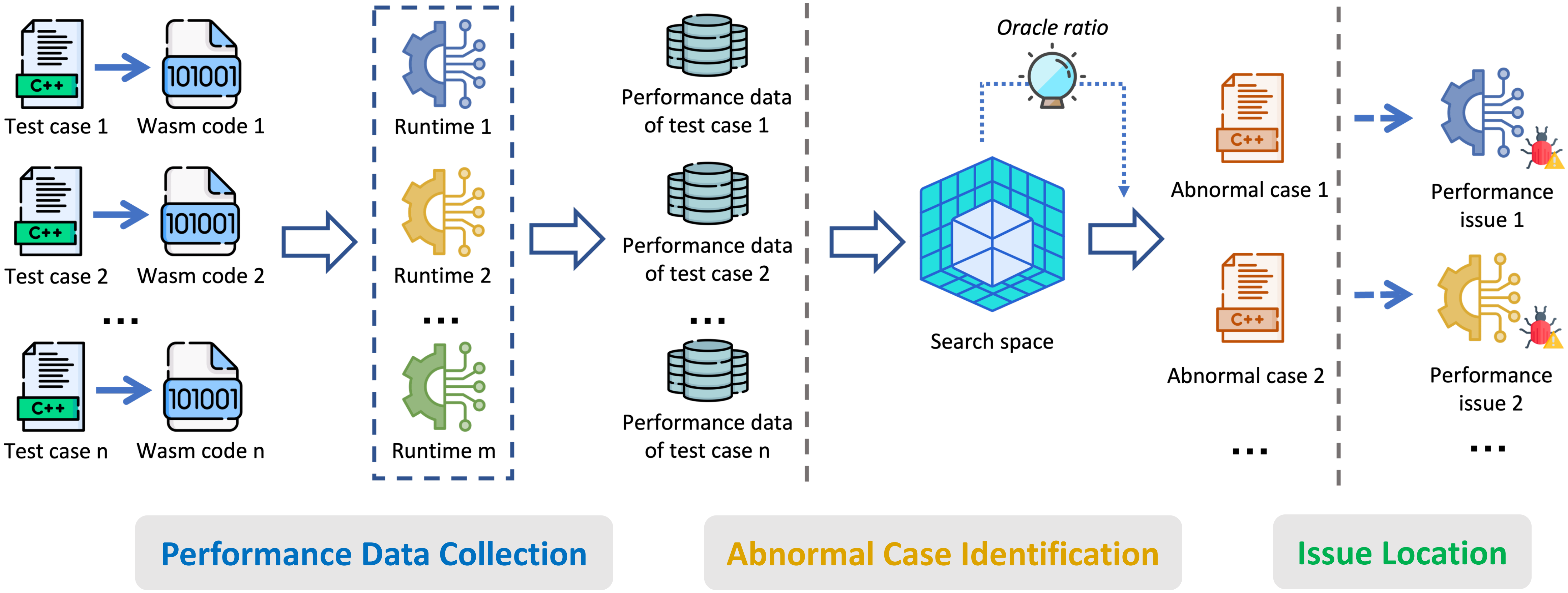
Solution: Adopt the idea of differential testing

Challenge: Determine the oracle of performance issues

Solution: Propose an *oracle ratio* that reflects the systematic performance gaps among different Wasm runtimes

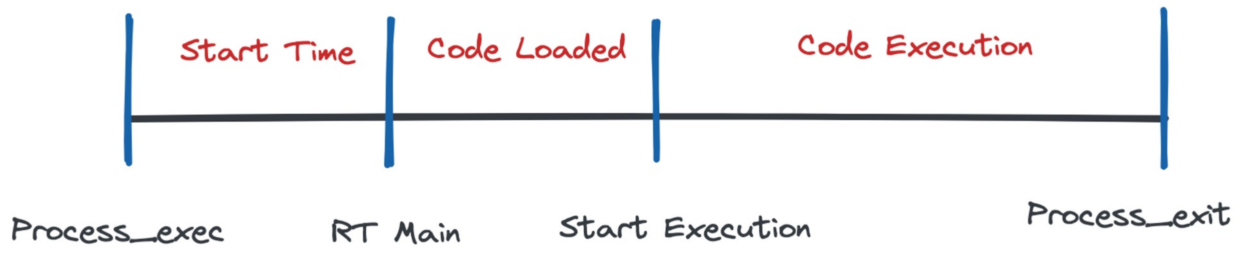
Approach: *WarpDiff*

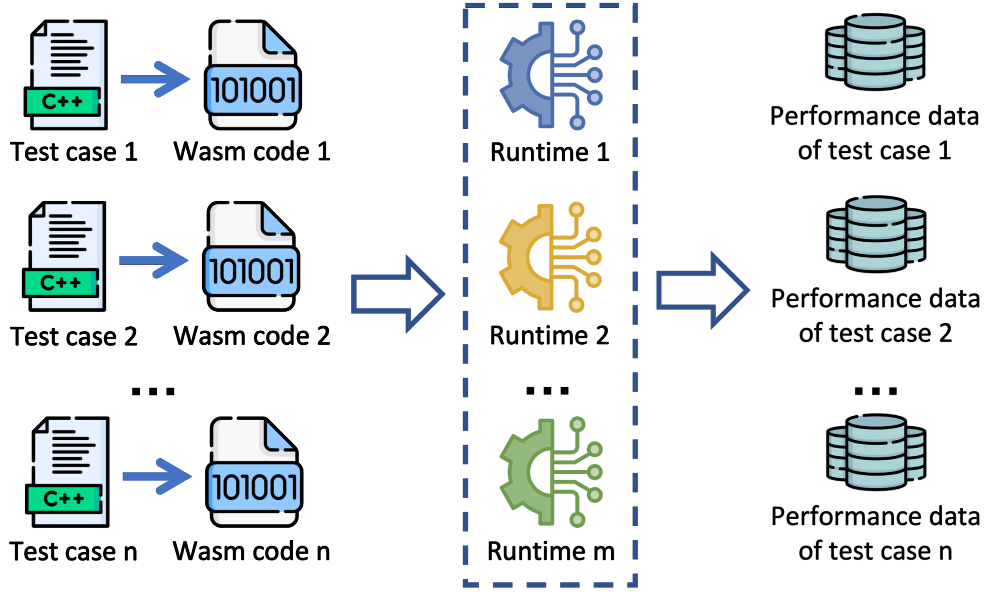
- **Wa**sm **R**untime **P**erformance **Diff**erential Testing



Phase 1: Performance Data Collection

- Test case selection
  - Well supported by standalone Wasm runtimes
  - More likely to trigger performance issues
- Wasm code execution
  - Compile to Wasm ➔ Execute on different runtimes
  - Ensure the correctness of the execution results
- Performance data recording
  - Three running stages





Performance Data Collection

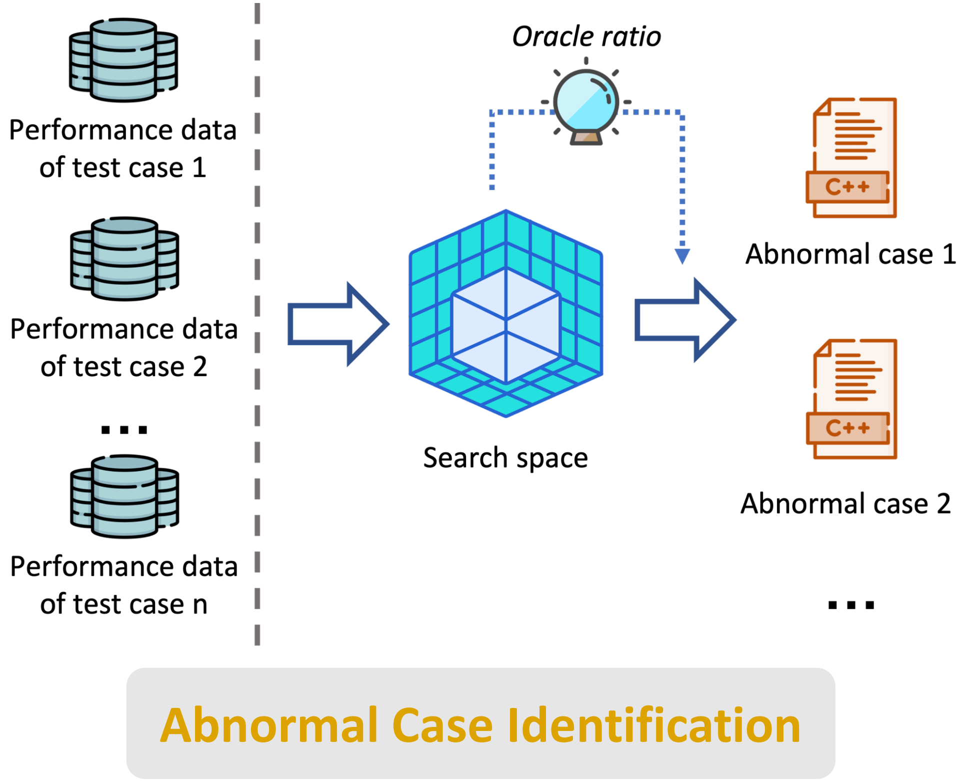
Phase 2: Abnormal Case Identification

- **Key insight:** The execution time of the same test case on different Wasm runtimes should follow a stable ratio (*i.e.*, *oracle ratio*) in normal cases.

- How to represent the execution time ratio?
  - Vectorization for each test case
  - *e.g.*, case x ran for 1s, 2s, 3s on three runtimes ➔ the vector of x is $[1,2,3]$ ➔ normalization

- How to determine the *oracle ratio*?
  - Take the center of all normalized vectors as the estimated *oracle ratio*

- Calculate the distance between a case vector and the estimated *oracle ratio*

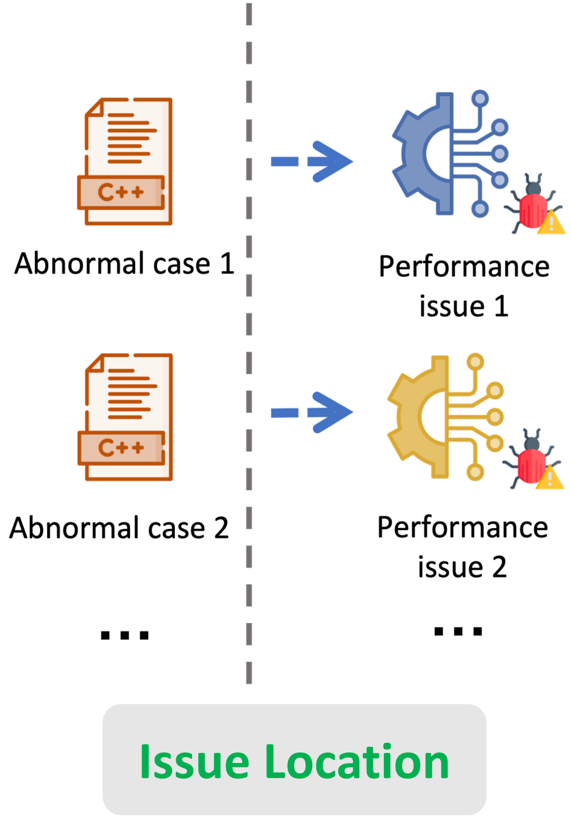


Phase 3: Performance Issue Location

- **Goal:** Locate the runtime in which the performance issue occurs

- Analyze the impact of each runtime on the abnormal case
    - For each dimension in the case vector, adjust its value to make the case vector closest to the estimated *oracle ratio*
    - Record the adjustment value as *deviation degree*

- Treat the runtime with the largest *deviation degree* as the issue-related runtime



Research Questions

RQ1: How does *WarpDiff* perform in identifying performance issues in real-world standalone Wasm runtimes?

RQ2: What are the causes of the identified performance issues, and how can we verify them?

RQ3: What is the computational overhead of differential testing in *WarpDiff* ?

Experiment Settings

- Test cases
  - 141 C/C++ programs from *LLVM test suite*
  - Valid results on 123 programs

- Wasm runtimes for testing
  - Five Wasm runtimes with top *popularity* and *activity* on GitHub

TABLE I
INFORMATION OF OUR TEST CASES FROM THE LLVM TEST SUITE.

Benchmark	#Program	#LOC*	Benchmark	#Program	#LOC*
Adobe-C++	6	1,615	Misc-C++	7	1,322
BenchmarkGame	8	486	Misc-C++-EH	1	16,817
CoyoteBench	4	1,471	Polybench	30	4,364
Dhrystone	2	642	Shootout	14	573
Linpack	1	693	Shootout-C++	25	783
McGill	4	956	SmallPT	1	96
Misc	27	5,052	Stanford	11	1,135
			Total	141	36,005

* LOC: lines of code.

TABLE II
INFORMATION OF WASM RUNTIMES FOR TESTING.

Runtime	#GitHub Stars*	Test Version	Execution Mode
Wasmer	15.1k	3.2.0	AOT
Wasmtime	12.1k	cli 8.0.0	AOT
Wasm3	6k	v0.5.0	Interpreter
WasmEdge	5.9k	0.12.0	AOT
WAMR	3.7k	1.1.2	Interpreter/AOT

* Statistics of Github stars is by April 2023.

RQ1: Identifying Performance Issues

- Top 10 abnormal cases
  - Based on the descending order of the *deviation degree* of the issue-related runtime

TABLE III

Deviation degree OF EACH RUNTIME SETTING ON THE TOP 10 ABNORMAL CASES.

Case	Wasmer	Wasmtime	Wasm3	Wasm3_compile	WasmEdge	WAMR	WAMR_AOT
BenchmarkGame/fasta.c	0.702	0.113	-0.248	-0.244	0.082	-0.270	0.081
Shootout/methcall.c	-0.051	-0.028	-0.164	-0.164	0.502	0.044	-0.014
Shootout-C++/methcall.cpp	-0.036	-0.031	-0.126	-0.128	0.415	0.072	-0.009
Shootout/random.c	0.075	0.315	-0.060	-0.060	0.079	-0.026	0.101
Shootout-C++/random.cpp	0.096	0.309	-0.063	-0.063	0.098	-0.036	0.121
Polybench/2mm.c	-0.038	-0.039	-0.151	-0.149	-0.035	0.268	0.003
Polybench/gemm.c	-0.038	-0.041	-0.145	-0.153	-0.036	0.267	0.007
Polybench/3mm.c	-0.037	-0.040	-0.145	-0.140	-0.034	0.261	0.005
Misc/flops-8.c	-0.019	0.012	-0.142	-0.142	-0.009	0.251	0.015
Misc/flops-4.c	0.234	-0.003	-0.127	-0.127	-0.019	0.168	0.001

Performance issues are common in existing standalone Wasm runtimes.

RQ2: Case Analysis

- Abnormal stage location ➔ Fine-grained cause location ➔ Cause verification

TABLE IV
SUMMARY OF PERFORMANCE ISSUES RELATED TO THE 10 ABNORMAL CASES.

Case	Related Runtime	Issue ID	Cause of Performance Issue	Status
`BenchmarkGame/fasta.c`	Wasmer	#3784	Improper implementation of `fd_write`	Confirmed
`Misc/flops-4.c`	Wasmer	#3821	Version issue of the *Cranelift* code generator	Confirmed
`Shootout/methcall.c`	WasmEdge	#2444	Improper handling when invoking function pointer	Confirmed
`Shootout-C++/methcall.cpp`	WasmEdge	#2442	Improper handling of virtual function	Confirmed
`Shootout/random.c` `Shootout-C++/random.cpp`	Wasmtime Wasmtime	#6287	Insufficient optimization for division and modulo	Confirmed

`Polybench/2mm.c`
`Polybench/gemm.c`
`Polybench/3mm.c`
	WAMR
	WAMR
	WAMR
		#2175	Insufficient optimization for matrix multiplications	Confirmed

`Misc/flops-8.c`	WAMR	#2167	Insufficient optimization for complex arithmetic expressions	Confirmed

We summarize 7 performance issues for the 10 abnormal cases.

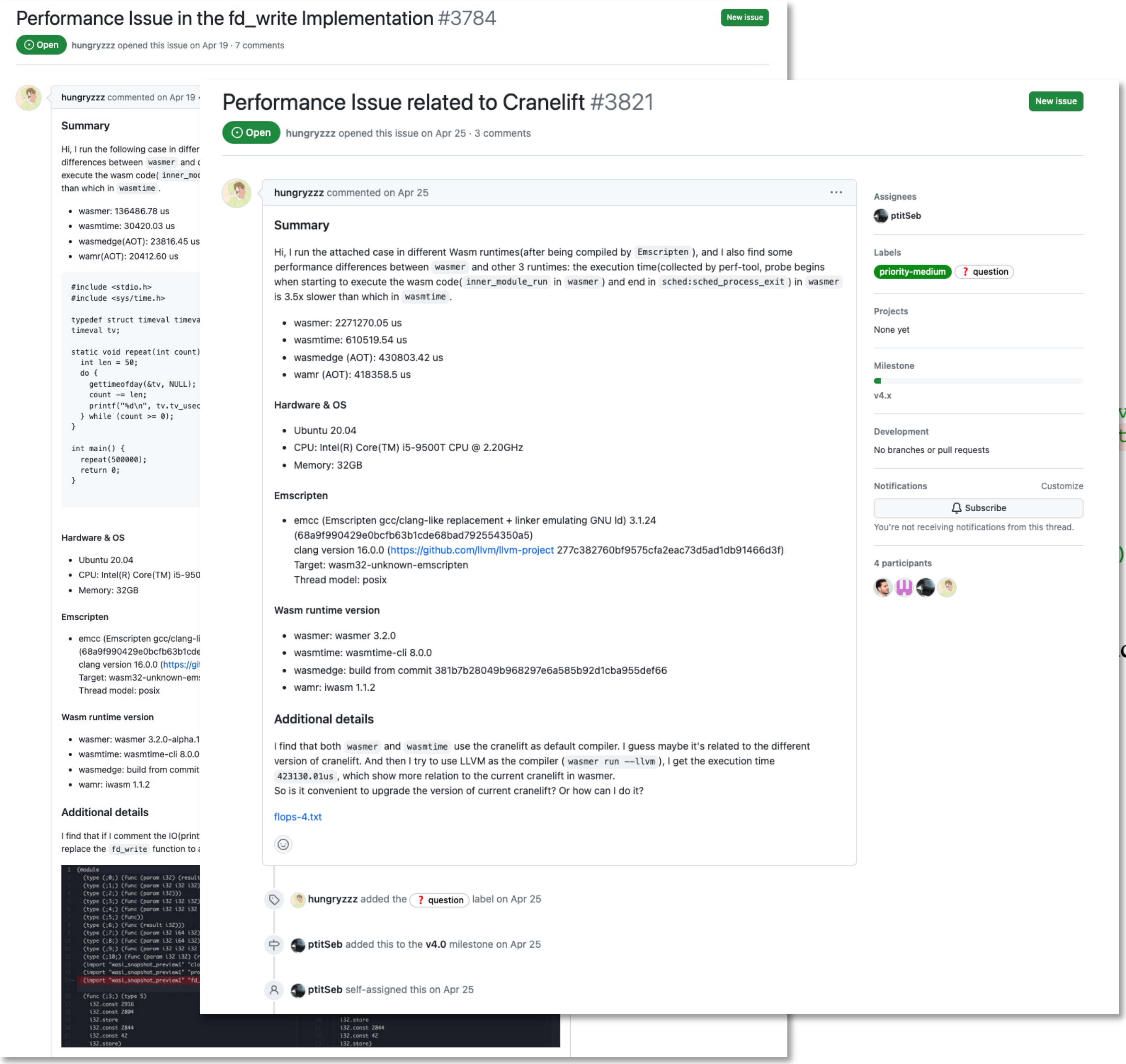
Case Analy

- Issue 1: Improper in

```
static void repeat_fasta (char
    size_t pos = 0;
    size_t len = strlen (s);
    char *s2 = malloc (len + WI
    memcpy (s2, s, len);
    memcpy (s2 + len, s, WIDTH)
    do {
        size_t line = MIN(WIDTH,
        fwrite (s2 + pos,1,line,
        putchar ('\n');
        pos += line;
        if (pos >= len) pos -= l
        count -= line;
    } while (count);
    free (s2);
}
```

(a) Issue-related code snipp

- Issue 2: Version issu



value of count
t the value of count

) with parameter 500000

ce Issue #3784.

Case Analysis: WasmEdge

- Issue 3: Improper handling when invoking function pointer (#2444)
- Issue 4: Improper handling of virtual function (#2442)

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Toggle {  // define a structure of Toggle
    char state;
    void (*activate)(struct Toggle);
} Toggle;

void toggle_activate(Toggle this) {  // activate the toggle
    this.state = !this.state;
}

int main() {
    int i, n = 1000000;
    Toggle tog;
    tog.state = 1;
    tog.activate = toggle_activate;

    for (i=0; i<n; i++) {
        tog.activate(tog);  // invoke the function by pointer
        // toggle_activate(tog);  // invoke the function directly
    }
    puts(tog.state ? "true\n" : "false\n");
    return 0;
}
```

Fig. 5. Simplified `methcall.c` related to Issue #2444 of WasmEdge.

Case Analysis: Wasmtime

- Issue 5: Insufficient optimization for division and modulo (#6287)

```c
inline double gen_random(double max) {  // generate a random number
  static long last = 42;

  last = (last * IA + IC) % IM; // compound operations of *, + and %
  return( max * last / IM ); // compound operations of * and /
}
```

(a) Issue-related code snippet of `random.c`.

```c
#include <stdio.h>

int main() {
     int N = 10000000, last = 42;
     while (N--) {
          last = (last + 33) % 13;   // compound operations of + and %
     }
     printf("%d\n", last);
     return(0);
}
```

(b) A new test case that can reproduce Issue #6287.

Case Analysis: WAMR

- Issue 6: Insufficient optimization for matrix multiplications (#2175)
- Issue 7: Insufficient optimization for complex arithmetic expressions (#2167)

```c
#pragma scop
  /* D := alpha*A*B*C + beta*D */
  for (i = 0; i < _PB_NI; i++)
    for (j = 0; j < _PB_NJ; j++)
    {
      tmp[i][j] = 0;
      for (k = 0; k < _PB_NK; ++k)
        tmp[i][j] += alpha * A[i][k] * B[k][j];  // alpha*A*B
    }
  for (i = 0; i < _PB_NI; i++)
    for (j = 0; j < _PB_NL; j++)
    {
      D[i][j] *= beta;                            // beta*D
      for (k = 0; k < _PB_NJ; ++k)
        D[i][j] += tmp[i][k] * C[k][j];           // alpha*A*B*C + beta*D
    }
#pragma endscop
```

Fig. 7. Issue-related code snippet of `2mm.c` in Issue #2175 of WAMR.

```c
x = piref / ( three * (double)m );        /********************/
s = 0.0;                                   /*  Loop 9.         */
v = 0.0;                                   /********************/

for( i = 1 ; i <= m-1 ; i++ )
{
  u = (double)i * x;
  w = u * u;
  v = w*(w*(w*(w*(w*(B6*w+B5)+B4)+B3)+B2)+B1)+one;
  s = s + v*v*u*((((((A6*w+A5)*w+A4)*w+A3)*w+A2)*w+A1)*w+one);
}
```

Fig. 8. Issue-related code snippet of `flops-8.c` in Issue #2167 of WAMR.

RQ3: Computational Overhead

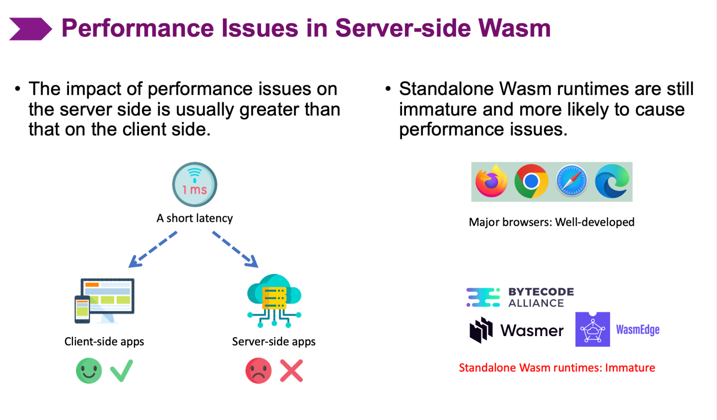
- Running time of the differential testing part in *WarpDiff*
  - With different numbers of runtime settings

TABLE V
COMPUTATIONAL OVERHEAD OF DIFFERENTIAL TESTING UNDER DIFFERENT NUMBERS OF RUNTIME SETTINGS.

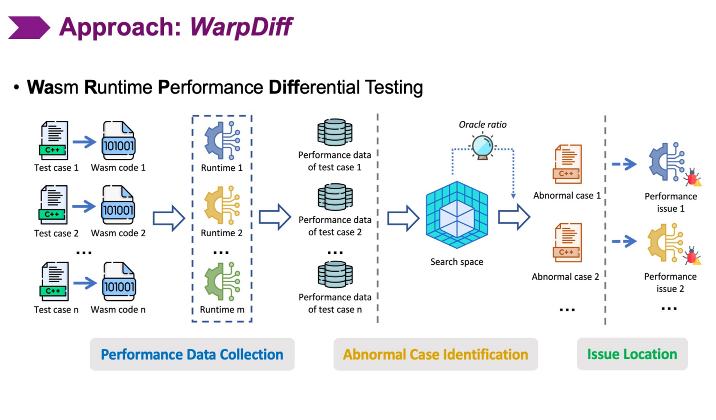
#Runtime	2	3	4	5	6	7
Avg. Overhead (s)	0.330	0.476	0.604	0.735	0.845	0.966
Std. Deviation	0.026	0.039	0.047	0.058	0.044	0.037

The computational overhead of differential testing only accounts for less than 0.01% of the whole process.

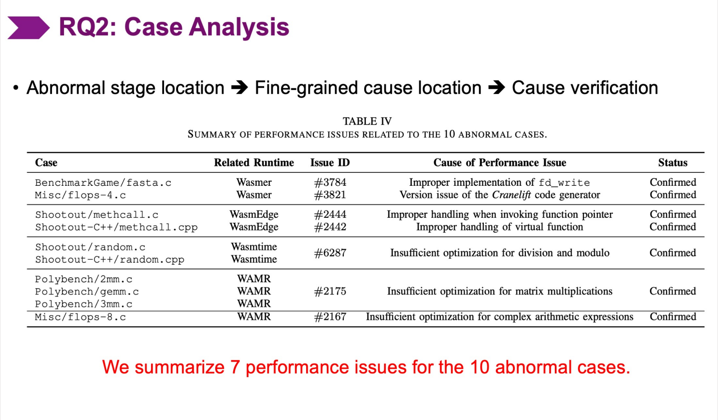
Conclusion



Significance



Approach



Case	Related Runtime	Issue ID	Cause of Performance Issue	Status
`BenchmarkGame/fasta.c`	Wasmer	#3784	Improper implementation of `fd_write`	Confirmed
`Misc/flops-4.c`	Wasmer	#3821	Version issue of the *Cranelift* code generator	Confirmed
`Shootout/methcall.c`	WasmEdge	#2444	Improper handling when invoking function pointer	Confirmed
`Shootout-C++/methcall.cpp`	WasmEdge	#2442	Improper handling of virtual function	Confirmed
`Shootout/random.c` `Shootout-C++/random.cpp`	Wasmtime Wasmtime	#6287	Insufficient optimization for division and modulo	Confirmed

`Polybench/2mm.c`
`Polybench/gemm.c`
`Polybench/3mm.c`
	WAMR
	WAMR
	WAMR
		#2175	Insufficient optimization for matrix multiplications	Confirmed

`Misc/flops-8.c`	WAMR	#2167	Insufficient optimization for complex arithmetic expressions	Confirmed

Results

Presenter: Shuyao Jiang

Email: syjiang21@cse.cuhk.edu.hk

Pre-print

Artifacts