The 32nd IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2025)

Distinguishability-guided Test Program Generation for WebAssembly Runtime Performance Testing

Shuyao Jiang[1], Ruiying Zeng[2], 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

ARISE Automated Reliable Intelligent Software Engineering

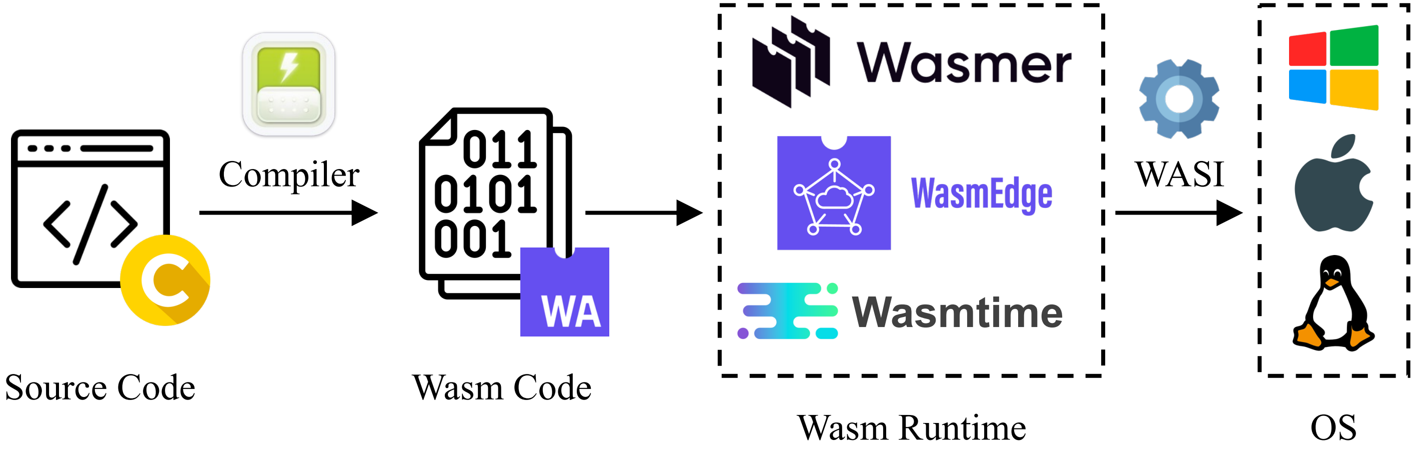
香港中文大學 The Chinese University of Hong Kong

復旦大學 FUDAN UNIVERSITY

WebAssembly (Wasm)

- A binary instruction format
  - Designed as a **compilation target** for programming languages
  - Fast, safe, portable, lightweight

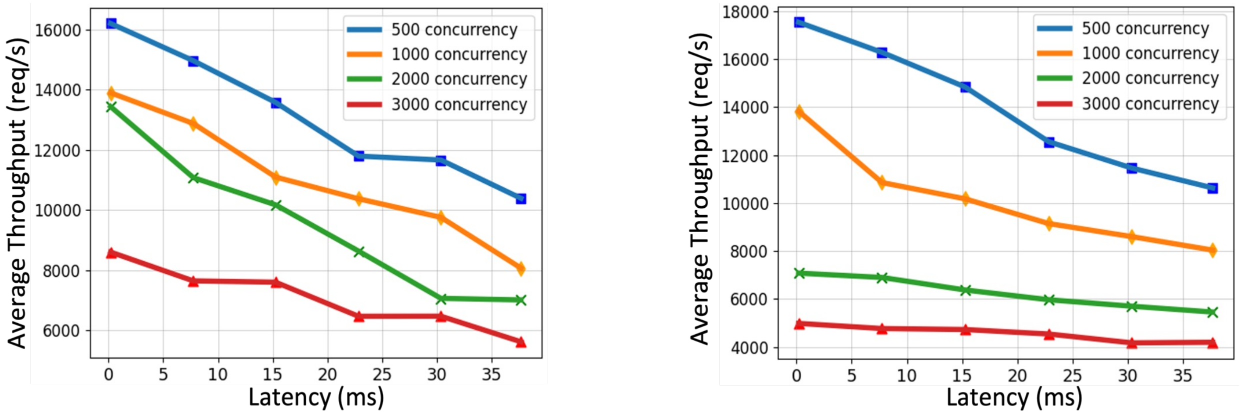
- Key component: Wasm runtime
  - Translate Wasm binary instructions to native machine code



Performance Testing for Wasm Runtime

- Performance issues in Wasm runtimes
  - Abnormal latency caused by runtime design flaws

- Impact of performance issues on Wasm service throughput [1]
  - A 30ms-latency will result in up to 50% drop in service throughput



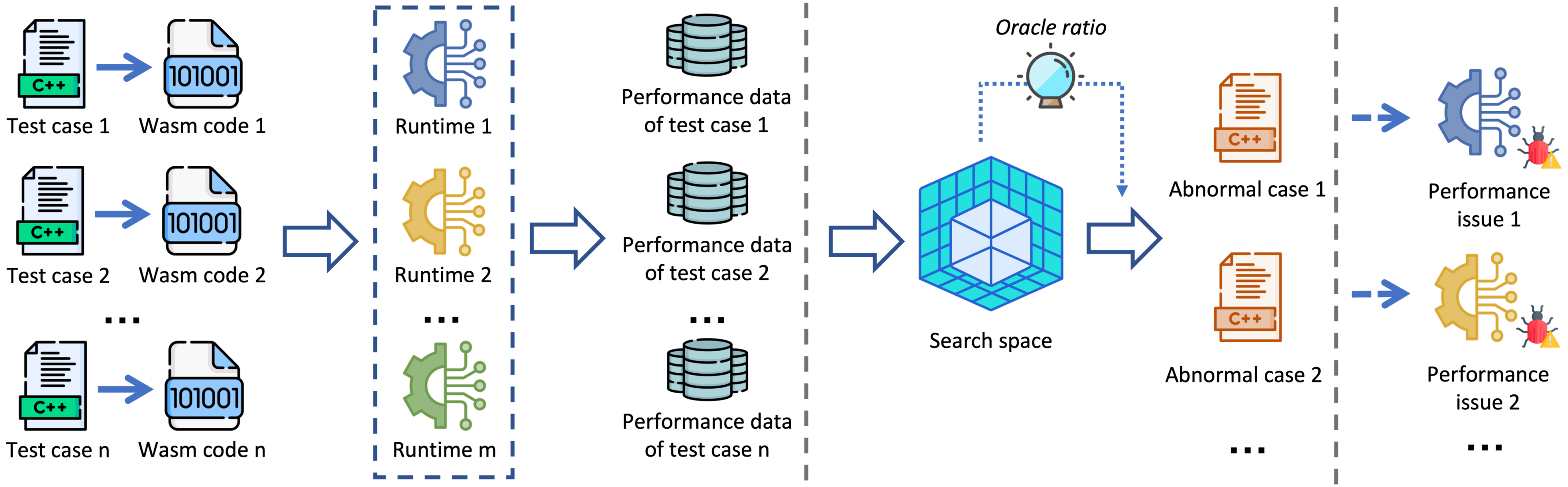
(a) 10,000 request

(b) 50,000 request

[1] "Revealing Performance Issues in Server-side WebAssembly Runtimes via Differential Testing." Shuyao Jiang, Ruiying Zeng, Zihao Rao, Jiazhen Gu, Yangfan Zhou, and Michael R. Lyu. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)*.

Existing Work: *WarpDiff* [1]

- A differential testing framework for identifying Wasm runtime performance issues
  - Key idea: The execution time of the same test case on different Wasm runtimes should follow a stable ratio (*i.e.*, *oracle ratio*)



- Limitation of *WarpDiff*: Insufficient high-quality test programs
  - Only use a small benchmark for testing

[1] "Revealing Performance Issues in Server-side WebAssembly Runtimes via Differential Testing." Shuyao Jiang, Ruiying Zeng, Zihao Rao, Jiazhen Gu, Yangfan Zhou, and Michael R. Lyu. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE 2023)*.

Goal & Challenges

- **Our goal:** Generate high-quality test programs for further testing
  - What are high-quality test programs?

  -- Tend to **trigger performance issues** in Wasm runtimes

Challenge 1: Lack of sufficient prior knowledge about Wasm runtime performance

Challenge 2: Difficult to verify the quality of generated test programs

Approach Design Insights

Insight 1: Historical issue-triggering test programs contain information that helps detect new issues.

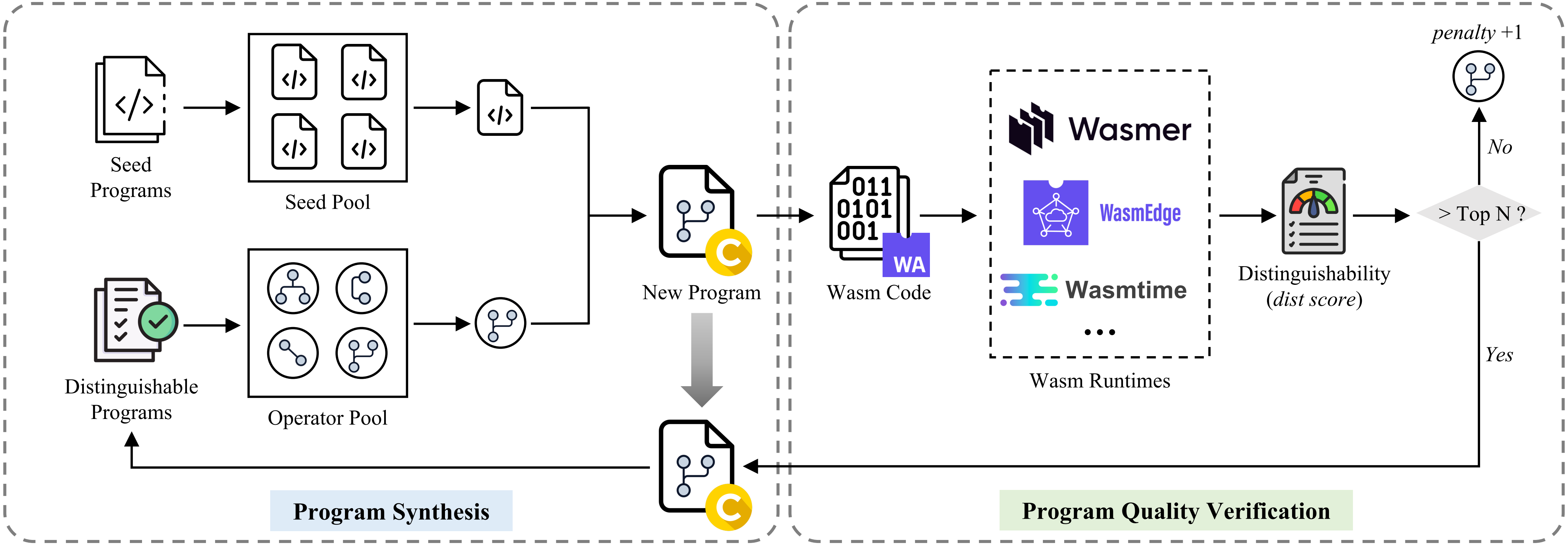
Practice: Extract code snippets from historical abnormal cases, then insert them in different contexts to generate new test programs.

Insight 2: The test oracle proposed by *WarpDiff* can inspire test program quality verification.

Practice: Propose an indicator *distinguishability* to guide test program generation process.

Approach: *WarpGen*

- A *distinguishability*-guided test program generation approach for Wasm runtime performance testing



Program Validity

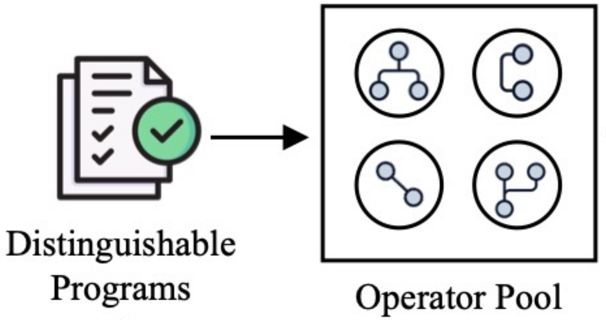
- A critical challenge in program synthesis
  - Ensure the **validity** of the synthesized program

Syntax Validity: The synthesized program should conform to the syntax rules and be able to pass compilation.

Insertion Validity: The inserted operator should affect the behavior of the seed program.

Data Preprocessing: Operator Extraction

- Sequential Operator
  - A code block containing sequential statements without branches and loops.
- Branching Operator
  - A code block containing conditional branching statements (i.e., `if`, `else`, and `else if`), including conditions and corresponding code to be executed.
- Looping Operator
  - A code block containing looping statements (i.e., `for`, `while`, and `do-while`), including loop conditions and loop bodies. It may contain nested loops.
- Mixed Operator
  - A code block containing a combination of the above three operators.



Implementation: Clang tool based on *LLVM Project*

Data Preprocessing: Operator Extraction

- Context recording: Pre-context & Post-context

Pre-context — Syntax Validity

All used variables (excluding variables declared and assigned in this block) and called functions (excluding standard library functions) in this block.

Post-context — Insertion Validity

All variables assigned in this block (excluding variables declared in this block).

Example

```
for( i = 1 ; i <= m-1 ; i++ )
{
  u = (double)i * x;
  w = u * u;
  s = s + w*(w*(w*(w*(w*(B6*w+B5)+B4)+B3)+B2)+B1)+one;
}
```

Pre-context:	Post-context:
{m, x, B6, B5, B4, B3, B2, B1, one}	{i, u, w, s}

Data Preprocessing: Seed Profiling

- Profiling information: Variable Usage & Code Coverage

Variable Usage

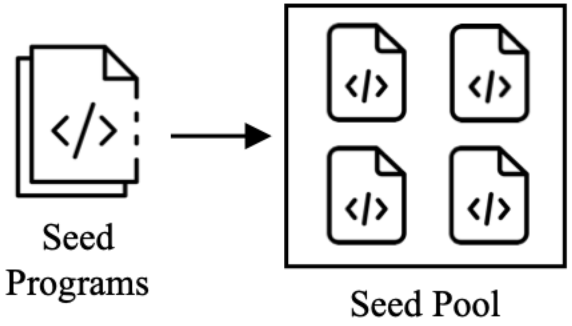
Syntax Validity

All the declared variables in the seed program
- Position of first declared
- Position of last used

Code Coverage

Insertion Validity

Covered code lines during seed program execution

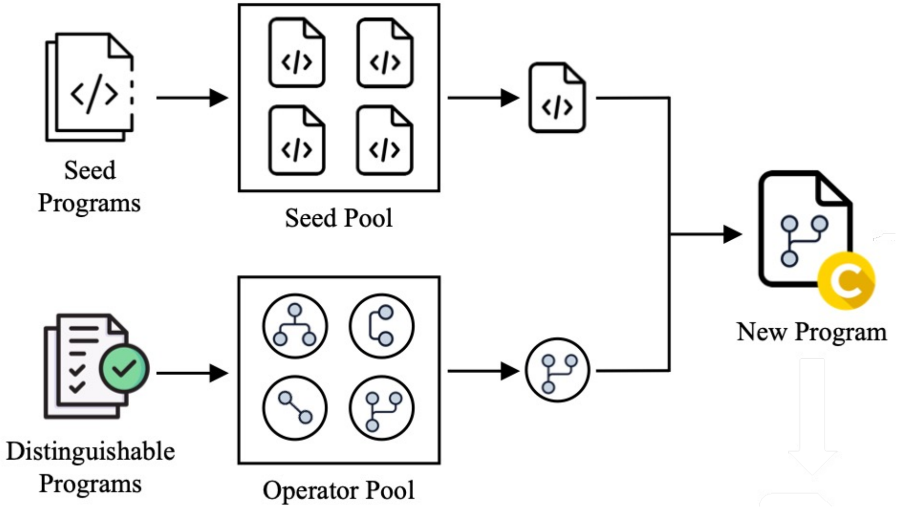


Program Synthesis

- Given an operator and a seed, synthesize a program by two steps

- Step 1: Insertion point selection
  - Traverse the code lines recorded in the **code coverage** of the seed
    - Collect the current context (local/global variables) of the seed
    - Read the pre/post contexts of the operator to check whether this line is a valid insertion point
  - Randomly select a valid insertion point

- Step 2: Variable dependency handling
  - Replace the variable in the operator with another same-type variable in the seed
  - Define a new variable

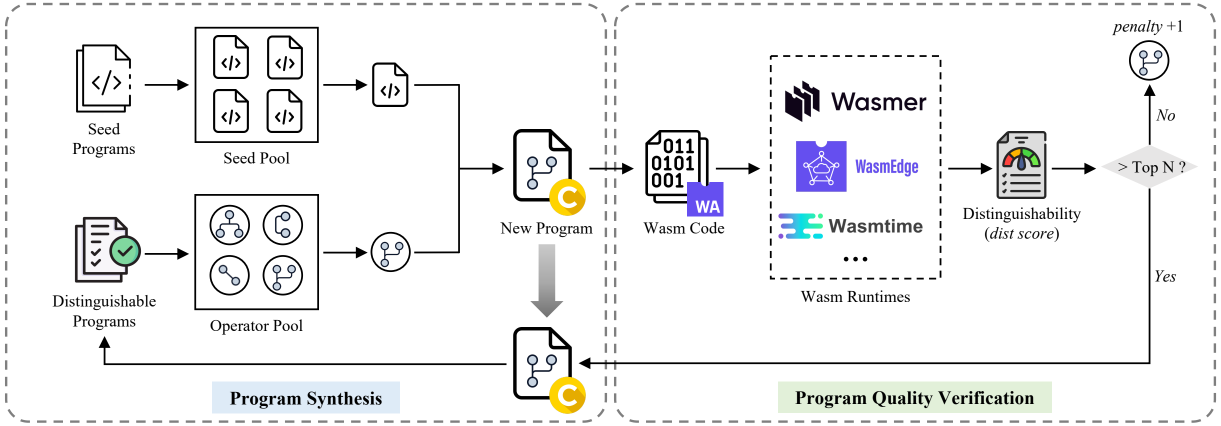


Iteration Process: Program Quality Indicator

- ***Distinguishability (dist score)***

The distinguishability (dist score) of a test program is the Euclidean distance between the normalized vector of its execution time ratio (on Wasm runtimes to be tested) and the normalized vector of the oracle ratio.

- Example
  - X = [1,2,3] —Normalize→ [0.17,0.33,0.5]
  - *Oracle ratio*: O = [0.2,0.2,0.4]
  - dist score(X) = EuDist(X,O) = 0.12



Goal of iteration: Identify top *N distinguishable programs* (*i.e.*, programs with top *N dist score*).

Iteration Process: Initial Iteration

- Initial operators: Extracted from abnormal cases reported by *WarpDiff*

- For each initial operator
  - Insert it into a random seed --> new program
  - Run the new program on several Wasm runtimes to calculate the *dist score*

- Collect the programs with top *N dist score* as *distinguishable programs*

- Extract operators from the *distinguishable programs*
  - Add the new operators to the operator pool

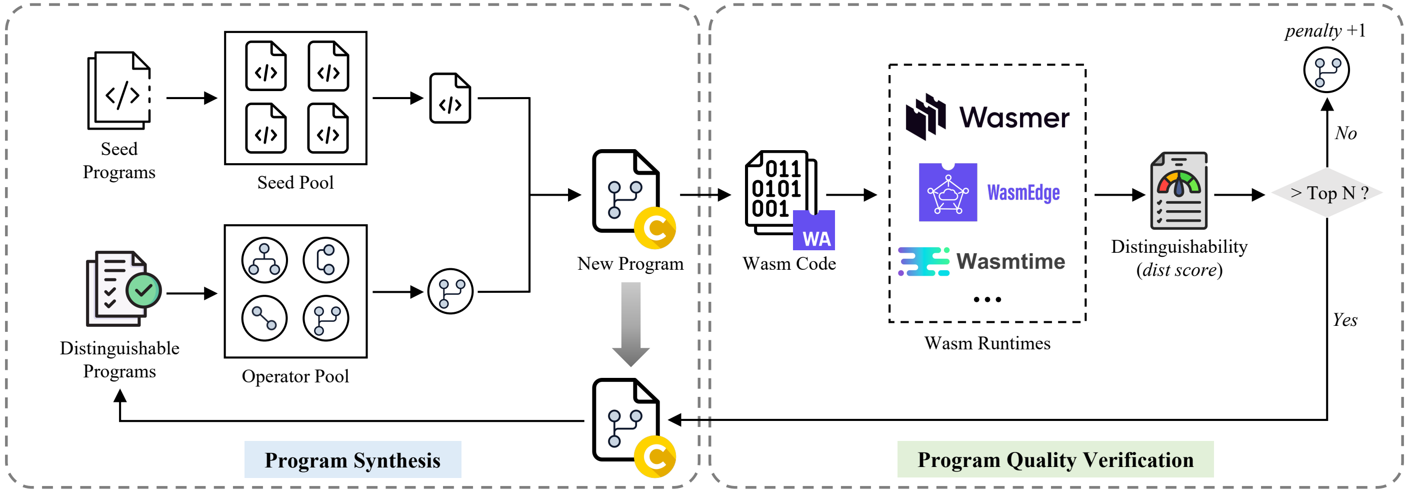
Iteration Process: Follow-up Iterations

- **Goal:** Find new test programs with higher *dist score* than the previous top N
- To improve the efficiency of iteration: Penalty mechanism
  - Assign a *penalty* (initial value = 0) for each operator
- Each time
  - Randomly select an operator and a seed --> new program
  - Calculate the *dist score* of the new program
    - If *dist score* > previous top N: Mark as a new *distinguishable program*
    - Else: *penalty* (of the selected operator) +1
- An operator with *penalty* > M will be removed

Iteration Process: The Whole Process

Algorithm 1: Iteration Process of *WarpGen*

```
Input   : seedPool, opPool, N, M, k
Output  : top N distinguishable programs
// Initial Iteration
foreach operator op in opPool do
    seed ← a random seed in seedPool;
    synProgram ← SynthesizeProgram(op, seed);
    distScore ← GetDistScore(synProgram);
topScoreSet ← top N distScore values;
topProgramSet ← programs with top N distScore values;
foreach program p in topProgramSet do
    opPool ← opPool ∪ ExtractOps(p);
foreach operator op in opPool do penalty_op ← 0 ;
// Follow-up Iterations
while the number of generated programs < k do
    op, seed ← a random value in opPool, seedPool;
    synProgram ← SynthesizeProgram(op, seed);
    distScore ← GetDistScore(synProgram);
    if distScore > Min(topScoreSet) then
        opPool ← opPool ∪ ExtractOps(synProgram);
        Update(topScoreSet, topProgramSet);
        penalty_op ← 0;
    else penalty_op ← penalty_op + 1;
    if penalty_op == M then opPool ← opPool \ {op};
```



Research Questions

RQ1: How efficient is *WarpGen* to generate high-quality test programs?

RQ2: How effective is the *distinguishability*-guided design in *WarpGen*?

RQ3: Can *WarpGen* detect new performance issues in Wasm runtimes?

Experiment Settings

- Wasm runtimes for testing
  - Wasmer, Wasmtime, WasmEdge, WAMR
- Initial Operators
  - 271 operators from 20 abnormal cases reported by *WarpDiff*
- Seed programs
  - 100 random C programs generated by Csmith
- Parameters
  - *Oracle ratio*: Based on the average execution time on the seed programs
  - $N = 20$, $M = 5$
- Compared Approaches
  - Csmith: Random approach
  - *WarpGen-base*: Do NOT update the operator pool and the top program set

TABLE I
WASM RUNTIMES AS TEST OBJECTS.

Runtime	Language	#Stars	#Commits	Version
Wasmer	Rust	16.7k	16.3k	4.2.3
Wasmtime	Rust	13.5k	12.5k	cli 15.0.0
WasmEdge	C/C++	7.2k	2.9k	0.13.5
WAMR	C/C++	4.2k	1.5k	1.2.3

RQ1: Efficiency of *WarpGen*

- Top 20 *dist score* for each generated test program during the iteration process
  - Minimal: The threshold for updating *distinguishable programs*
  - Average: the average quality of *distinguishable programs*

TABLE II

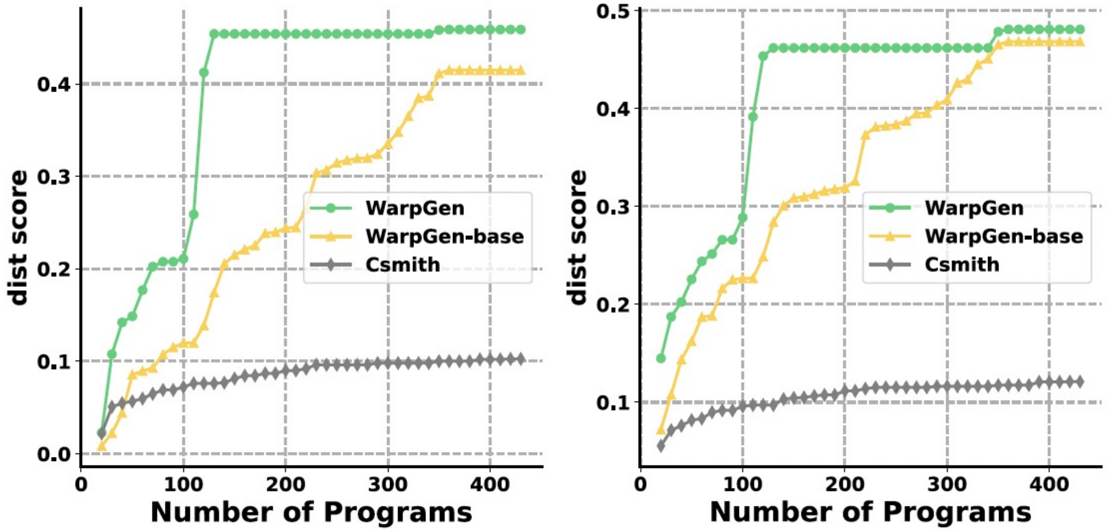
STATISTICS OF TOP 20 *dist score* WHEN *WarpGen* GENERATED DIFFERENT NUMBERS OF TEST PROGRAMS.

#Programs	20	50	80	110	140	170	200	230	260	290	320	350	380	410	436
Minimal	0.023	0.149	0.208	0.259	0.454	0.454	0.454	0.454	0.454	0.454	0.454	0.458	0.459	0.459	0.459
Average	0.145	0.226	0.266	0.392	0.462	0.462	0.462	0.462	0.462	0.462	0.462	0.479	0.481	0.481	0.481

WarpGen can generate high-quality test programs with high efficiency.

RQ2: Effectiveness of Guidance

- Comparison of top 20 *dist score* from different approaches
  - Csmith: Random approach
  - *WarpGen-base*: Do NOT update the operator pool and the top program set



(a) Minimal of top 20 (b) Average of top 20

WarpGen can generate test programs of *higher quality* and *more quickly* than the baseline approaches.

RQ3: New Performance Issues

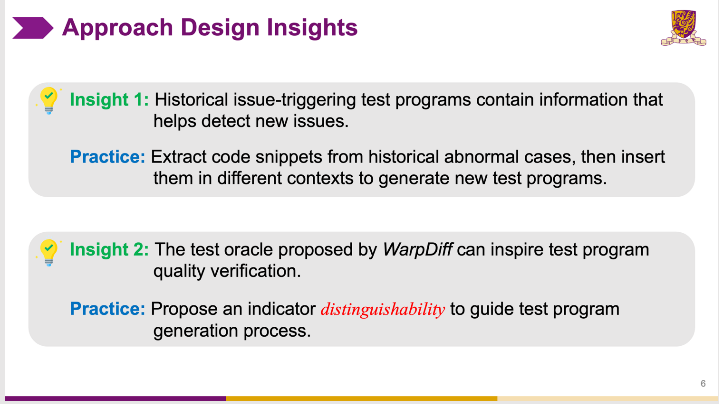
- Identify issues from the final top 20 *distinguishable programs*

TABLE III
PERFORMANCE ISSUES IDENTIFIED BY *WarpGen*.

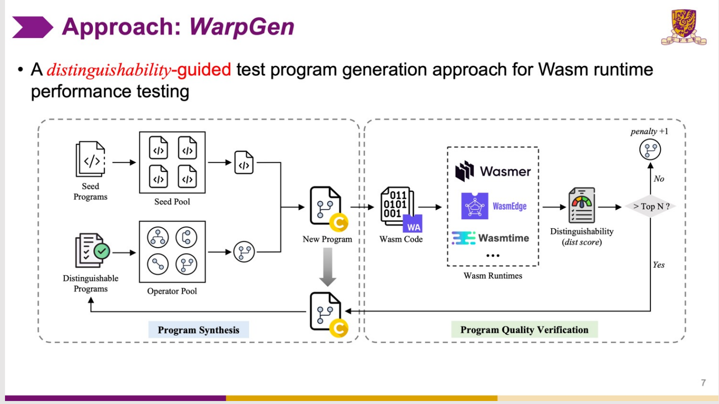
ID	Runtime	Scenario	Status
#7731	Wasmtime	Floating-point (FP) arithmetic	Fixed
#7732	Wasmtime	Access of pointers to constant	Confirmed
#7733	Wasmtime	Increment operation in nested loops	Confirmed
#4378	Wasmer	Operations on FP arrays	Fixed
#4379	Wasmer	Call of standard output functions	Fixed
#4380	Wasmer	Access of variable addresses	Fixed
#2938	WAMR	FP arithmetic	Confirmed

WarpGen is effective to detect new performance issues in Wasm runtimes.

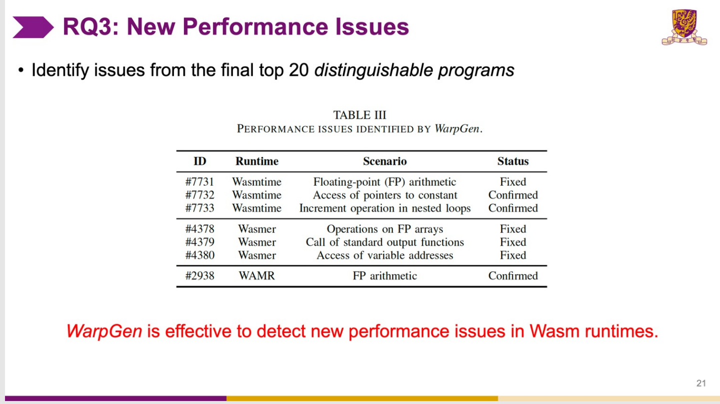
Conclusion



Insights



Approach



ID	Runtime	Scenario	Status
#7731	Wasmtime	Floating-point (FP) arithmetic	Fixed
#7732	Wasmtime	Access of pointers to constant	Confirmed
#7733	Wasmtime	Increment operation in nested loops	Confirmed
#4378	Wasmer	Operations on FP arrays	Fixed
#4379	Wasmer	Call of standard output functions	Fixed
#4380	Wasmer	Access of variable addresses	Fixed
#2938	WAMR	FP arithmetic	Confirmed

Evaluation

Presenter: Shuyao Jiang

Affiliation: The Chinese University of Hong Kong

Email: syjiang21@cse.cuhk.edu.hk

Pre-print

Homepage