How Far We've Come – A Characterization Study of Standalone WebAssembly Runtimes

Wenwen Wang

School of Computing
UNIVERSITY OF GEORGIA

What is WebAssembly?

A *low-level* and *portable* binary code format

Design Goal: facilitating the deployment of *non-web* applications on web platforms

Why WebAssembly?

Advantages:

- *Portability*
  - Independent on both programming languages and hardware platforms
- *Safety*
  - Sandboxed execution environment
- *Near-native speed*
  - Fast to compile to native binary code

WebAssembly is increasingly adopted beyond the web

- Trusted runtime environments for embedded systems
- Lightweight serverless frameworks for edge computing
- Software-fault isolation for serverless computing
- Resource accounting in remote computation
- …

“A new foundation for pervasive computing”

—— David Bryant (Mozilla Fellow)

Standalone WebAssembly Runtimes

The key enabler of WebAssembly outside the web

- Developed *specifically* to run WebAssembly binary code on a host physical machine without the need of browsers
- Just-In-Time (JIT) compilation or interpretation
- Much more lightweight than traditional language runtimes

However, there is very limited study about them

The characteristics of standalone WebAssembly runtimes are **not clear**

- How is the performance efficiency when running WebAssembly binaries?
- What amount of extra memory resource is consumed?
- Do they have an observable architectural impact?
- …

This work fills up this knowledge gap!

A characterization study of standalone WebAssembly runtimes

- Covering five popular standalone WebAssembly runtimes
- Constructing a benchmark suite, **WABench**
- Discovering many interesting findings

Methodology

Five standalone WebAssembly runtimes

- Actively developed and maintained
- Sufficiently mature to run a broad range of applications

	Language	LoC	Type	#Stars	History
Wasmtime	Rust	314K	JIT	7.6K	2 years
WAVM	C/C++	98K	JIT	2.2K	2 years
Wasmer	Rust	154K	JIT	12.4K	3 years
Wasm3	C	120K	Interpretation	5K	2 years
WAMR	C	145K	Interpretation	2.8K	2 years

WABench (50 Benchmarks)

A benchmark suite for standalone WebAssembly runtimes

- Existing benchmark suites
  - JetStream2, MiBench, and PolyBench
- Whole applications from a broad range of domains
  - bzip2, espeak, facedetection, gnuchess, mnist, snappy, and whitedb

Available at:

➔ https://github.com/wabench/wabench

Overall Performance

Baseline: execution time of native execution

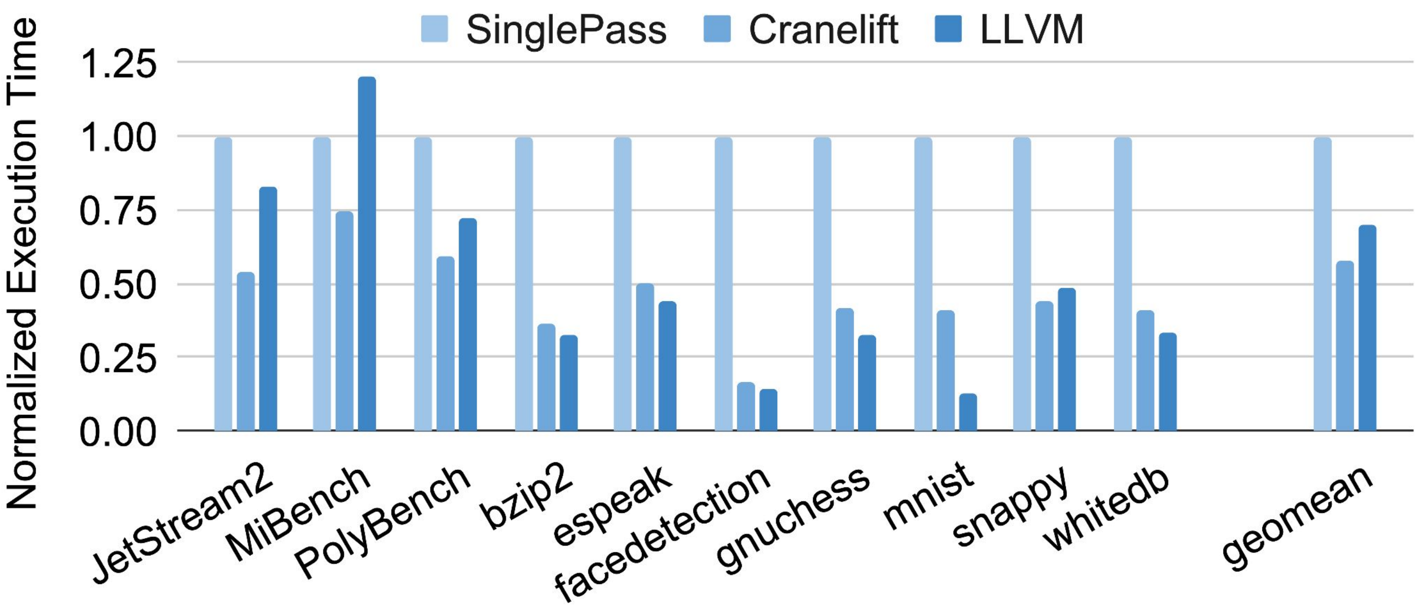
	Performance Slowdown
Wasmtime	1.67X
WAVM	3.54X
Wasmer	1.59X
Wasm3	6.99X
WAMR	9.57X

Finding 1: All WebAssembly runtimes introduce extra performance overhead, compared to native execution.

Impact of JIT Compilers

Three JIT compilers of Wasmer

- SinglePass (baseline), Cranelift, and LLVM

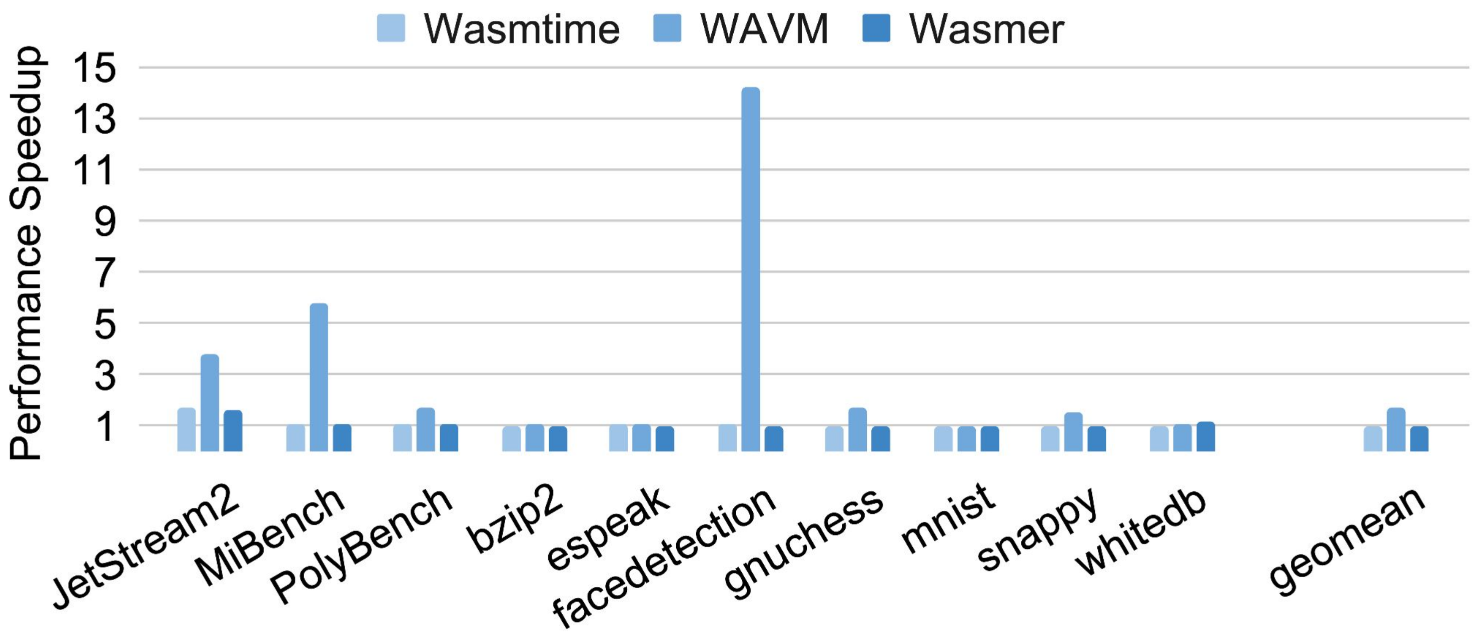


<u>Finding 2:</u> Cranelift and LLVM can achieve better performance than SinglePass.

Impact of Ahead-Of-Time (AOT) Compilation

Three JIT-based runtimes, baseline: w/o AOT

- Wasmtime, WAVM, and Wasmer

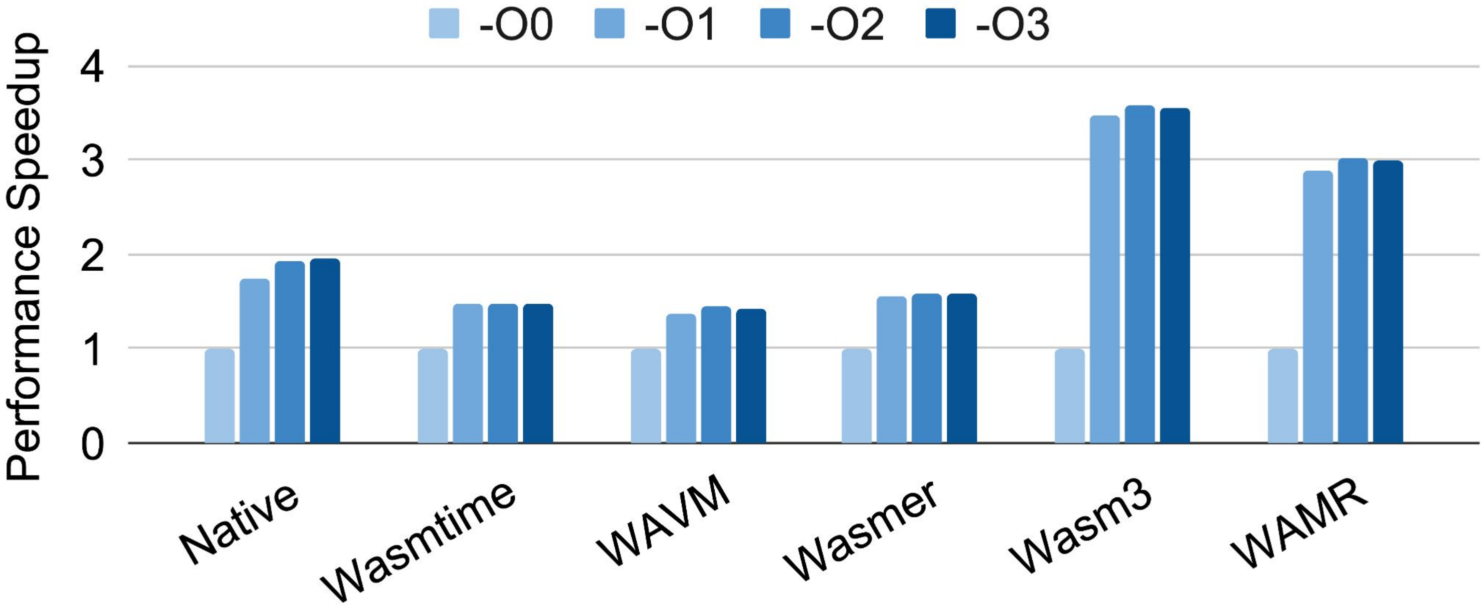


<u>Finding 3:</u> AOT has substantial performance impact on WAVM, but not Wasmtime and Wasmer.

Impact of Compiler Optimizations

Four optimization levels, baseline: -O0

- -O0, -O1, -O2, and -O3

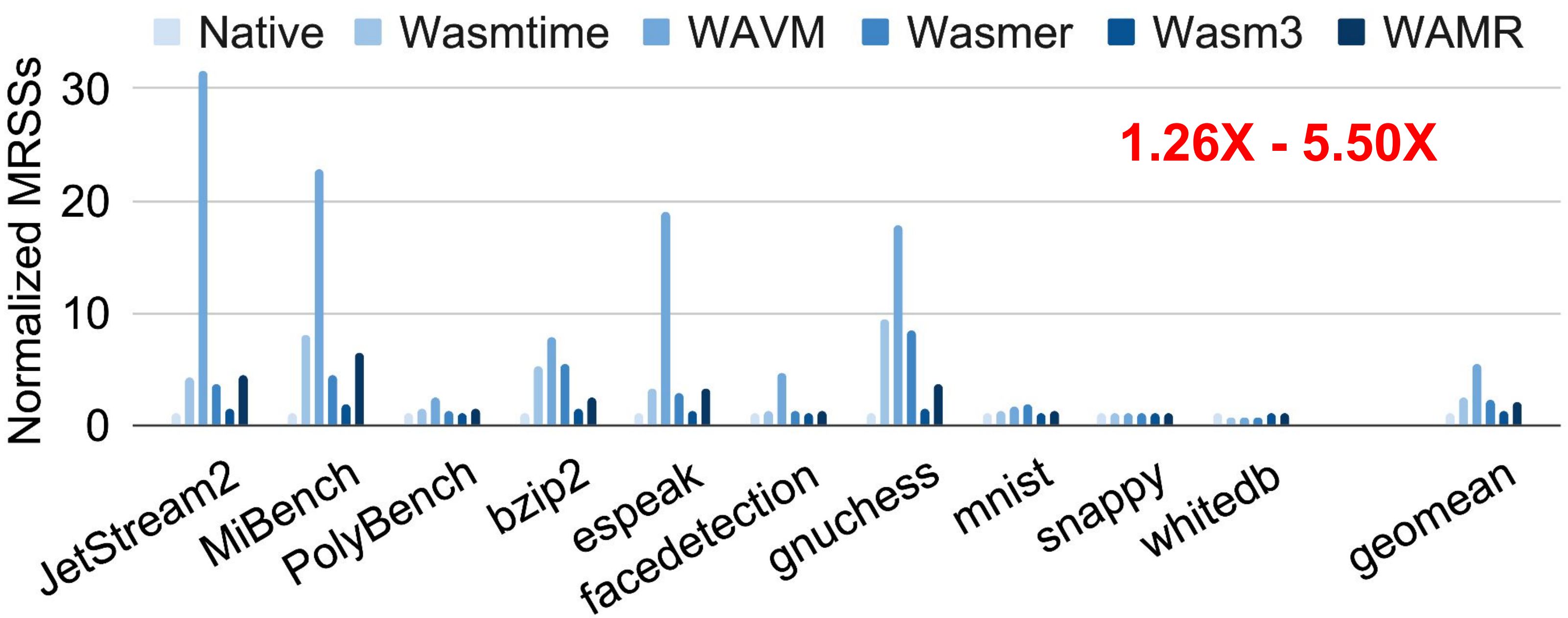


<u>Finding 4:</u> WebAssembly binaries compiled with higher optimization levels can achieve better performance.

Memory Overhead

Maximum resident set sizes (MRSSs)

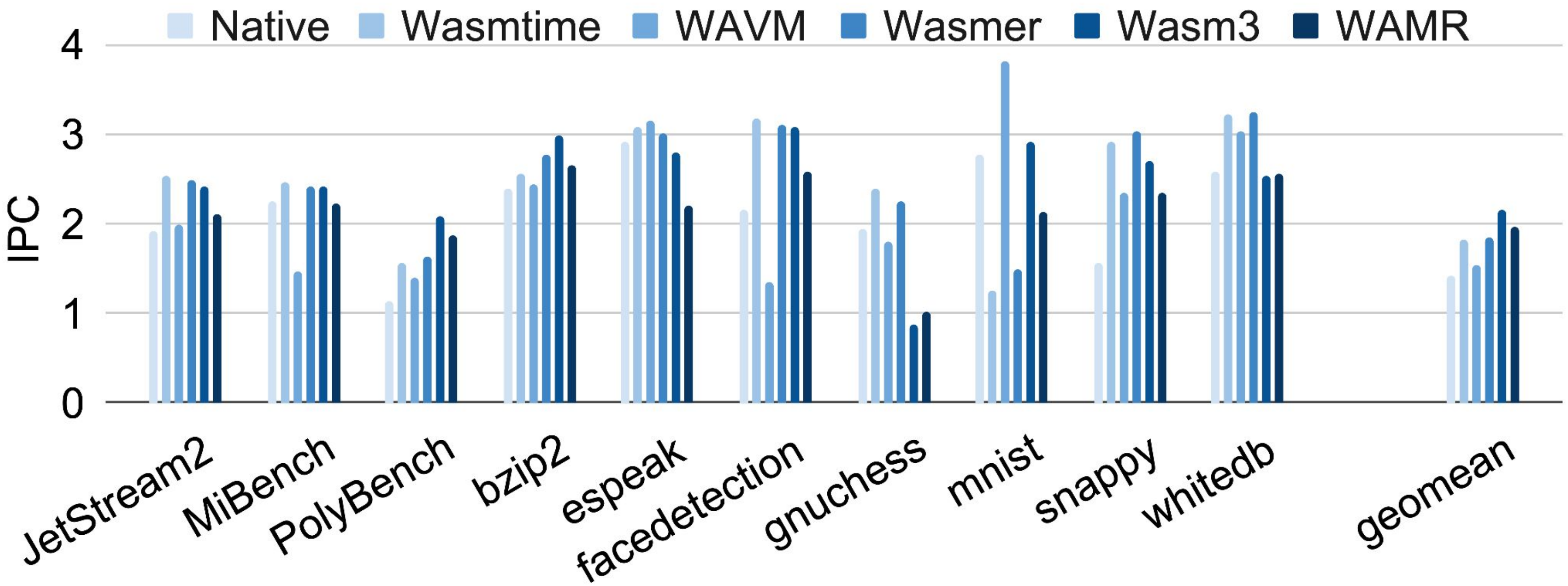
- Baseline: native execution



<u>Finding 5:</u> WebAssembly runtimes consume extra memory resources, compared to native execution.

Instructions Per Cycle (IPC)

Using hardware performance counters to measure IPC



<u>Finding 6:</u> WebAssembly runtimes exhibit higher IPC values than native execution.

Branch Prediction Misses

	Normalized Branch Prediction Misses	Branch Prediction Miss Ratios
Native	1X	1.01%
Wasmtime	1.52X	0.77%
WAVM	8.99X	1.69%
Wasmer	1.56X	0.92%
Wasm3	12.64X	0.76%
WAMR	8.14X	0.53%

Finding 7: WebAssembly runtimes have more branch misses, but the branch miss ratios are very close to native execution.

Cache Misses

	Normalized Cache Misses	Cache Miss Ratios
Native	1X	11.13%
Wasmtime	1.91X	12.98%
WAVM	4.60X	5.57%
Wasmer	1.73X	13.26%
Wasm3	1.39X	7.97%
WAMR	1.60X	8.99%

Finding 8: WebAssembly runtimes have more cache misses, but the cache miss ratios are very close to native execution.

Discussion

Insight 1: WebAssembly users need to select an *appropriate* standalone WebAssembly runtime.

Insight 2: It is necessary for standalone WebAssembly runtime developers to carefully *tune* the performance.

Insight 3: Developing *WebAssembly-specific* optimizations may further improve the performance.

Summary

- This work makes an attempt towards characterizing **standalone WebAssembly runtimes**
  - Covering five popular standalone WebAssembly runtimes
  - Constructing a benchmark suite, namely **WABench**
  - Discovering many interesting findings
- More research work is required to further understand the rationales behind the characteristics

WABench is available at

➔ https://github.com/wabench/wabench

Thank you!