



# **Evaluation of WebAssembly IoT Runtimes on a ESP32 Microcontroller**

Evaluation von WebAssembly IoT Runtimes auf einem ESP32  
Microcontroller



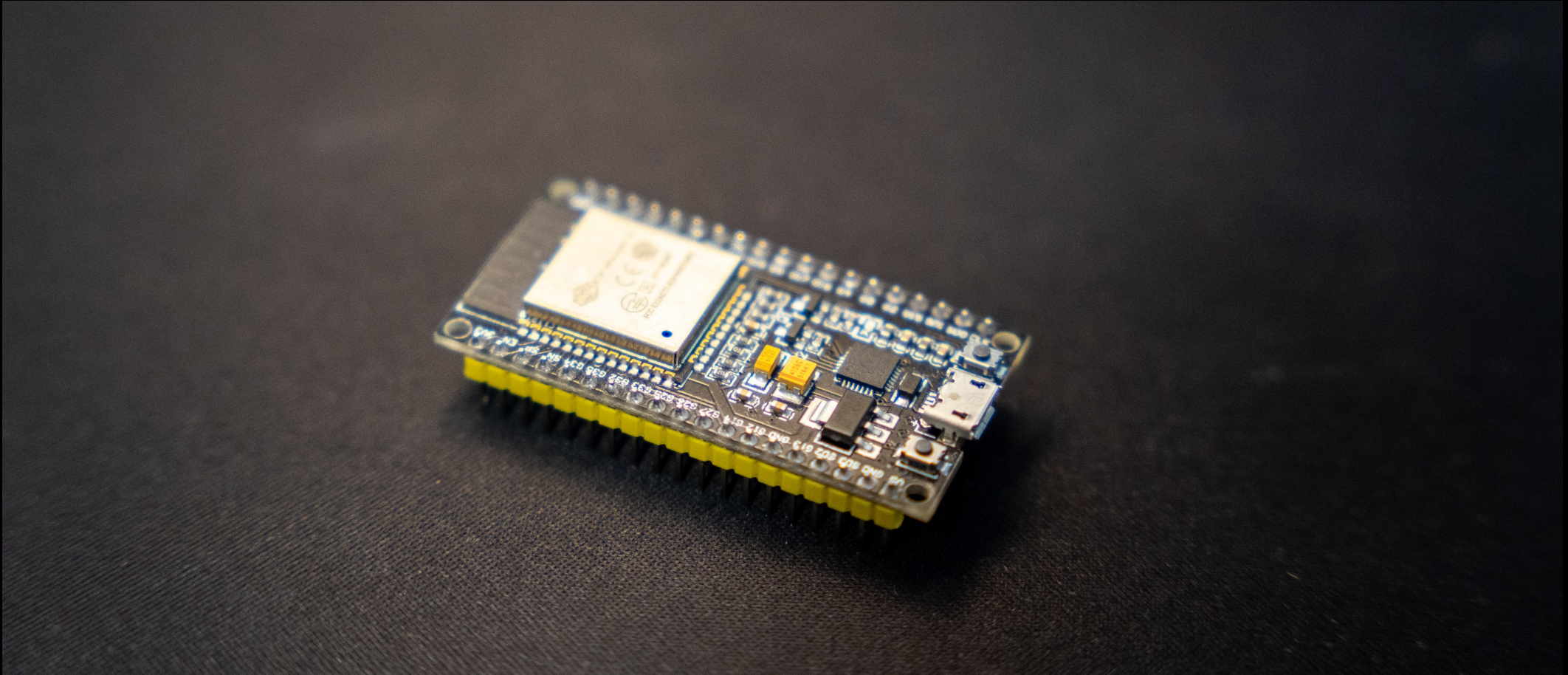
# Abstract

- Run WebAssembly on the ESP32
- Found runtime
- WASM code runs on the ESP32 with decreased performance

# Outline

- Background: ESP32 and WebAssembly
- Methodology: Runtime and tests
- Evaluation: Rest results and learnings

# The ESP32



# WebAssembly adoption

## WebAssembly - OTHER

Usage % of all users ?  
Global 90.93%

WebAssembly or "wasm" is a new portable, size- and load-time-efficient format suitable for compilation to the web.

Current aligned	Usage relative	Date relative	Apply filters	Show all	?								
IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet
		2-46											
	12-14	<sup>1</sup> 47-51	4-50		10-37								
	<sup>3</sup> 15	<sup>4</sup> 52	<sup>2</sup> 51-56	3.1-10.1	<sup>2</sup> 38-43	3.2-10.3							4-6.4
6-10	16-79	53-73	57-79	11-12.1	44-65	11-13.2		2.1-4.4.4	12-12.1				7.2-10.1
11	80	74	80	13	66	13.3	all	80	46	80	68	12.12	11.1
		75-76	81-83	13.1-TP		13.4							



# WebAssembly

- Portable target for languages like C/C++ and Rust
- Shipped in self-contained modules
- Runs on stack machine
- No web dependencies
- Advantageous attributes for IoT
- Structured control flow



# Structured control flow

instr ::= . . .

| nop

| unreachable

| block resulttype instr\* end

| loop resulttype instr\* end

| if resulttype instr\* else instr\* end

| br labelidx

| **br\_if labelidx**

| br\_table vec(labelidx) labelidx

| **return**

| **call funcidx**

| call\_indirect typeidx

# Methodology



# Running WebAssembly

README.md



 **wasm** **v0.4.6** **issues** **17** **tests** **failing** **license** **MIT**

A high performance WebAssembly interpreter written in C.

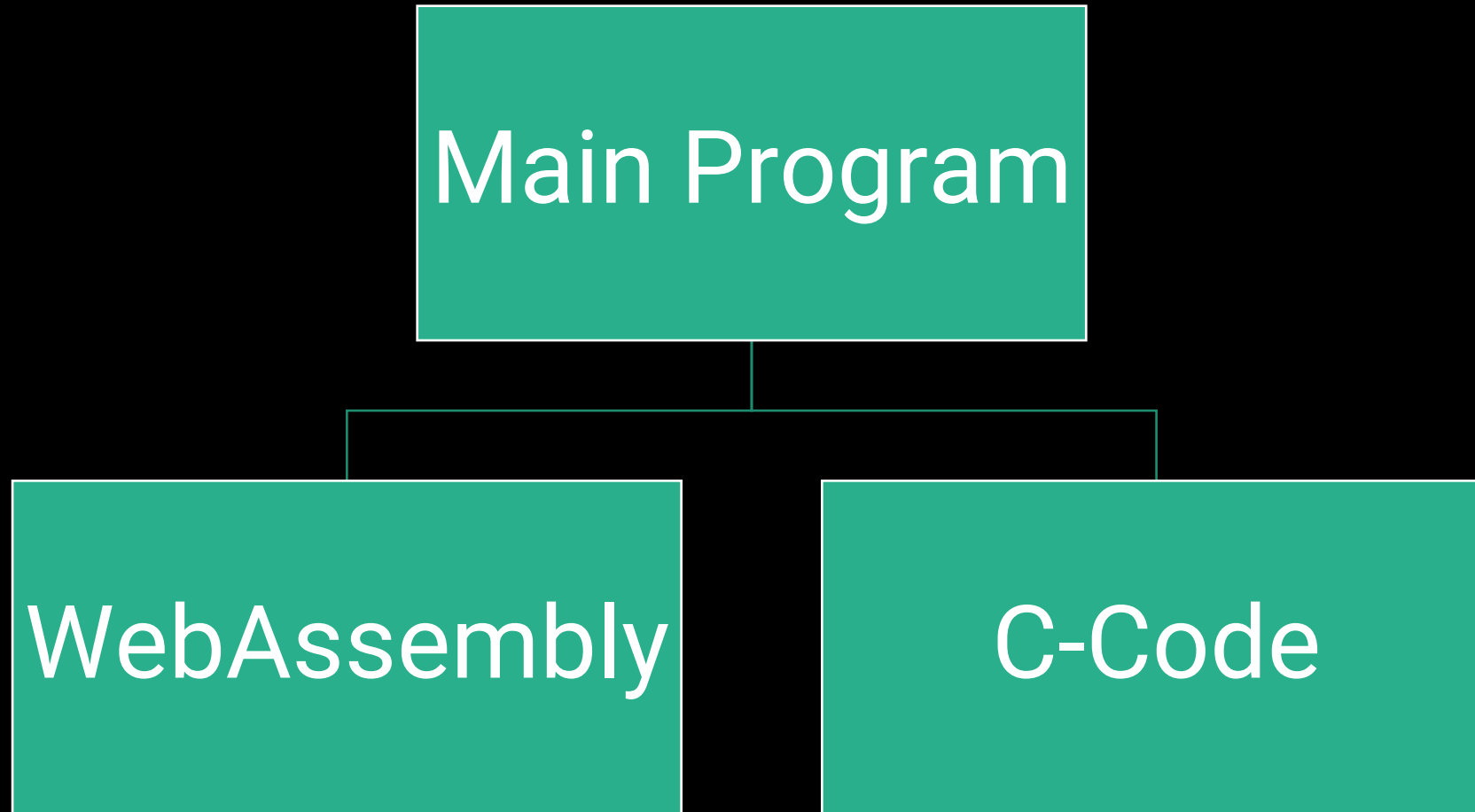
- ~ 8x faster than other known wasm interpreters
- ~ 4-5x slower than state of the art wasm **JIT** engines
- ~ 12x slower than native execution

```
Wasm3
Wasm3 v0.4.6 on iOS (arm64-v8a)
Build Feb  7 2020 21:48:53
Device: iPhone10,4

Loading WebAssembly...
Running fib(40) on WebAssembly...
Result: 102334155
Elapsed: 4844 ms

Running fib(40) on Native C...
Result: 102334155
Elapsed: 611 ms
```

# Testing setup



# Recursive: C++

```
uint32_t run(uint32_t n) {  
    if (n < 2) {  
        return n;  
    }  
    return run(n - 1) + run(n - 2);  
}
```

# Recursive: WAT

```
(func $run (export "run") (type $t1) (param $p0 i32) (result i32)
  (block $B0
    (br_if $B0
      (i32.lt_u
        (local.get $p0)
        (i32.const 2)))
    (return
      (i32.add
        (call $run
          (i32.add
            (local.get $p0)
            (i32.const -1)))
        (call $run
          (i32.add
            (local.get $p0)
            (i32.const -2))))))
  (local.get $p0))
```

# Recursive: Comparison

```
uint32_t run(uint32_t n) {  
    if (n < 2) {  
        return n;  
    }  
    return run(n - 1) + run(n - 2);  
}
```

```
(block $B0  
  (br_if $B0  
    (i32.lt_u  
      (local.get $p0)  
      (i32.const 2)))  
  (return  
    (i32.add  
      (call $run  
        (i32.add  
          (local.get $p0)  
          (i32.const -1)))  
      (call $run  
        (i32.add  
          (local.get $p0)  
          (i32.const -2))))))  
  (local.get $p0)
```

# Recursive: Comparison

```
uint32_t run(uint32_t n) {  
    if (n < 2) {  
        return n;  
    }  
    return run(n - 1) + run(n - 2);  
}
```

```
(block $B0  
  (br_if $B0  
    (i32.lt_u  
      (local.get $p0)  
      (i32.const 2)))  
  (return  
    (i32.add  
      (call $run  
        (i32.add  
          (local.get $p0)  
          (i32.const -1)))  
      (call $run  
        (i32.add  
          (local.get $p0)  
          (i32.const -2))))))  
  (local.get $p0)
```

# Recursive: Comparison

```
uint32_t run(uint32_t n) {  
    if (n < 2) {  
        return n;  
    }  
    return run(n - 1) + run(n - 2);  
}
```

```
(block $B0  
  (br_if $B0  
    (i32.lt_u  
      (local.get $p0)  
      (i32.const 2)))  
  (return  
    (i32.add  
      (call $run  
        (i32.add  
          (local.get $p0)  
          (i32.const -1)))  
      (call $run  
        (i32.add  
          (local.get $p0)  
          (i32.const -2))))))  
  (local.get $p0)
```



# Testcases

Recursive

Switch

Memory

Matrix  
Multiply

Native  
Calls

TypeScript





# Evaluation

# Measurements

Test case	WASM Execution	Native Exectuion	Slowdown factor
Recursive Calls	41,766 $\mu$ s ( $\sigma=0.79$ )	1,000 $\mu$ s ( $\sigma=0.28$ )	42
Switch statement	1,021 $\mu$ s ( $\sigma=0.71$ )	0 $\mu$ s ( $\sigma=0.00$ )	
Memory Access	1,802 $\mu$ s ( $\sigma=0.74$ )	55 $\mu$ s ( $\sigma=0.26$ )	33
Matrix Multiplication	26,277 $\mu$ s ( $\sigma=0.60$ )	281 $\mu$ s ( $\sigma=0.58$ )	93
Native calls*	33 $\mu$ s ( $\sigma=5.15$ )	8 $\mu$ s ( $\sigma=0.64$ )	4
TypeScript execution	41,493 $\mu$ s ( $\sigma=0.85$ )	1,000 $\mu$ s ( $\sigma=0.24$ )	41



# Learnings

## **Drawbacks**

- Performance loss
- Significant limitations

## **Potential**

- Dynamic code
- Browser execution
- New languages



# Conclusion