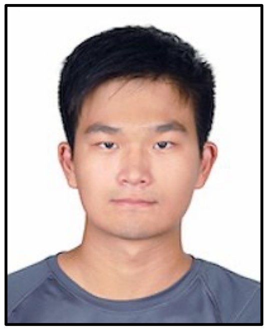
# Reusing Legacy Code in Wasm: Key Challenges of Compilation and Code Semantics Preservation

**Sara Baradaran** · Liyan Huang · Mukund Raghothaman · Weihang Wang

*University of Southern California, USA*



USC University of Southern California

## WebAssembly (Wasm)

- A statically typed binary instruction format widely used in browsers, IoT devices, and edge computing
- Serves as a compilation target for high-level programming languages, such as C/C++, Rust, and Go

```c
int increment(int x)
{
    return x + 1;
}
```

(a) C Source Code.

```
(func (param i32)
      (result i32)
    local.get 0
    i32.const 1
    i32.add
)
```

(b) WebAssembly text format.

```
[...] // header with
      // type info
20 00 // local.get 0
41 01 // i32.const 1
6a    // i32.add
0b    // end
```

(c) WebAssembly binary format.

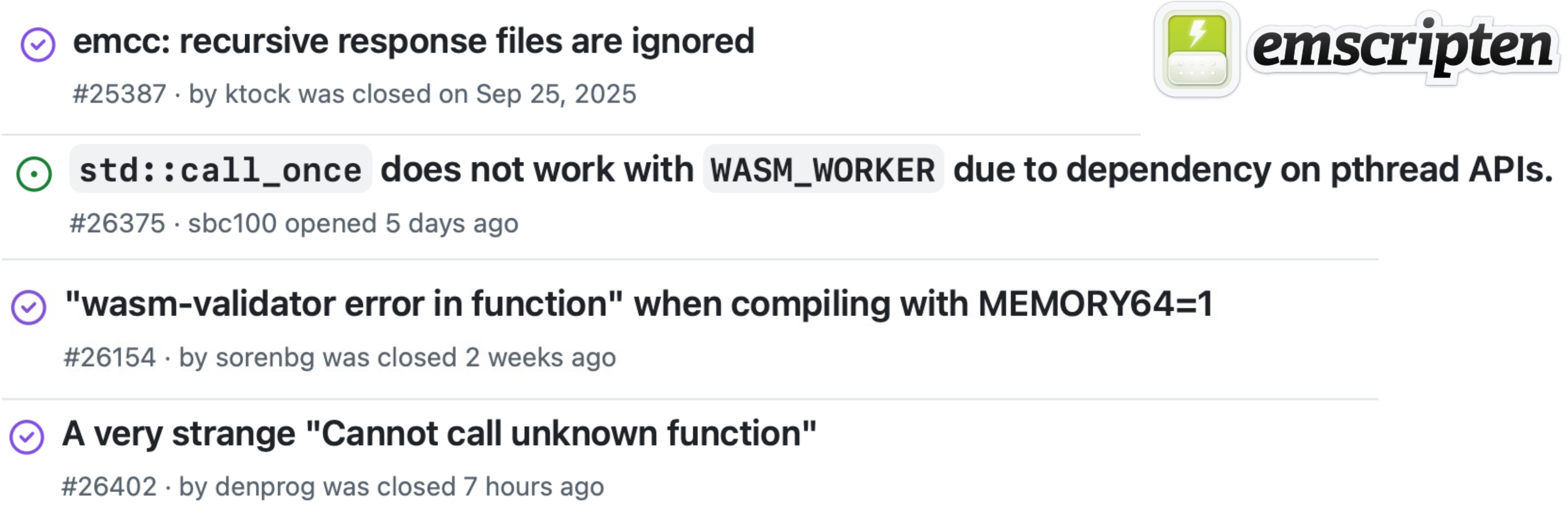
## Wasm Design Objectives

The ultimate goal of designers is to address the problem of safe, fast, portable code on the web

- Does it address security concerns?
  - ✓ Everything old is new again: Binary security of WebAssembly (Lehmann et al. USENIX 2020)
  - ✓ Security risks of porting C programs to Webassembly (Stiévenart et al. SAC 2022)
- Does it address speed concerns?
  - ✓ Understanding the performance of Webassembly applications (Yan et al. IMC 2021)
  - ✓ Not So Fast: Analyzing the performance of WebAssembly (Jangda et al. USENIX 2019)
- Does it address portability concerns?
  - ✓ We investigate it!

# Motivation

Developers frequently encounter difficulties when porting legacy code into WebAssembly



**emcc: recursive response files are ignored**
#25387 · by ktock was closed on Sep 25, 2025

**`std::call_once` does not work with `WASM_WORKER` due to dependency on pthread APIs.**
#26375 · sbc100 opened 5 days ago

**"wasm-validator error in function" when compiling with MEMORY64=1**
#26154 · by sorenbg was closed 2 weeks ago

**A very strange "Cannot call unknown function"**
#26402 · by denprog was closed 7 hours ago

- The code compilation fails
- The generated Wasm binaries behave unexpectedly

## Problem Statement

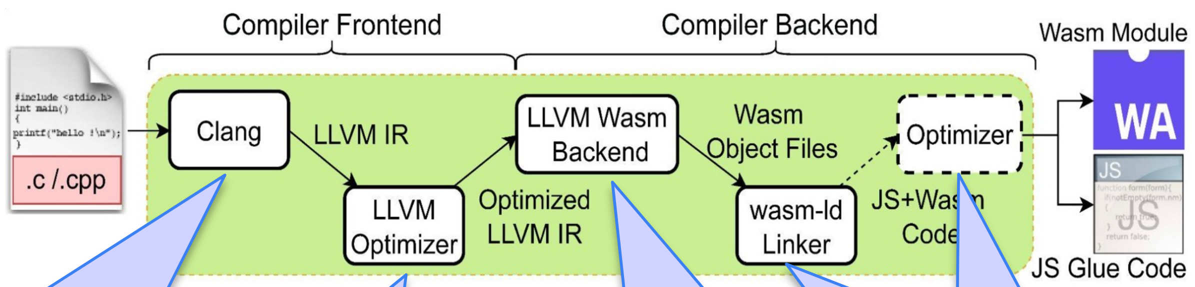
How practically can developers use legacy code in web using WebAssembly?

- **RQ1:** What challenges arise when cross-compiling high-level language code into WebAssembly?
- **RQ2:** How well do WebAssembly compilers preserve source code semantics in the resulting Wasm binaries?

USC University of Southern California

# Study Subjects

- **C/C++** account for ≈65% of source code in real-world Wasm binaries[1]
- **Emscripten** is the most widely-used WebAssembly compiler[2]



Analyzes syntax and semantics of code and translates it into Ll

Optimizes IR for size and efficiency

Translates and produc

Links mu libraries

Performs a multi-pass optimization process

uired ble

[1] Hilbig et al., “An Empirical Study of Real-World WebAssembly Binaries: Security, Languages, Use Cases”
[2] Romano et al., “An Empirical Study of Bugs in WebAssembly Compilers”

USC University of Southern California

## Data Collection

Collected a set of open-source C/C++ libraries for

- ✓ Document parsing (e.g., JSON, YAML, XML)
- ✓ Mathematical computation
- ✓ Data structure implementations
- ✓ Logging
- ✓ …

Open-source C/C++ projects from GitHub with

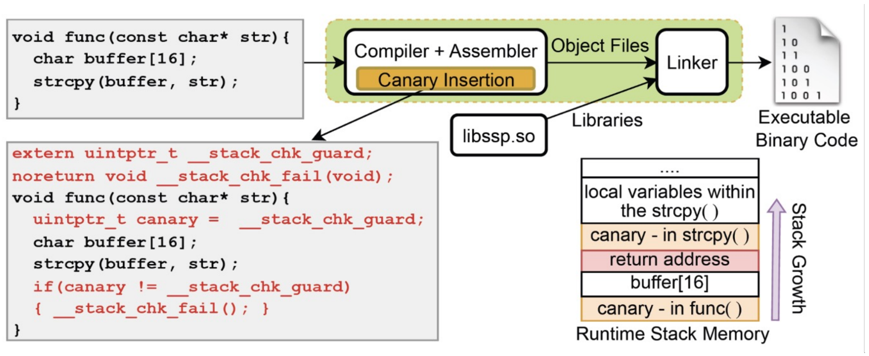
- CMake build script
- Test suite
- >= 50 stars

- Exploratory dataset
  - ✓ Including 115 codebases
- Validation dataset
  - ✓ Including 20 codebases

USC University of Southern California

# RQ1: Compilation Challenges

- Attempted to build all projects (source + tests) in Wasm
  - ✓ Could successfully build 81 of 115 projects (70%)
  - ✓ Failed on 34 projects (30%)
- Conducted a manual analysis and categorized build errors based on their root cases
  - ✓ Undefined symbols
  - ✓ Missing third-party libraries
  - ✓ Target-dependent warnings
  - ✓ Architecture- and platform-specific code
  - ✓ Incompatible compiler options
  - ✓ WebAssembly compiler bugs



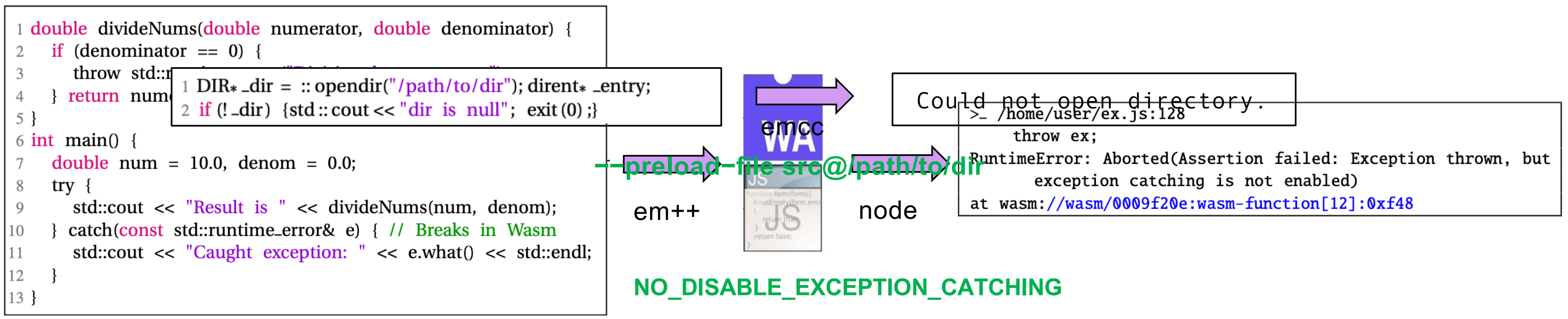
```
void func(const char* str){
  char buffer[16];
  strcpy(buffer, str);
}
```

```
extern uintptr_t __stack_chk_guard;
noreturn void __stack_chk_fail(void);
void func(const char* str){
  uintptr_t canary =  __stack_chk_guard;
  char buffer[16];
  strcpy(buffer, str);
  if(canary != __stack_chk_guard)
  { __stack_chk_fail(); }
}
```

# RQ2: Reliability of Wasm Code

- Executed tests of 81 projects in Wasm and x86-64
  - ✓ $\approx 9\%$ of executable tests showed inconsistent outcome
- A brief manual analysis shows that a large portion of test inconsistent occurs because of:
  - ✓ **Default-disabled compiler options**
  - ✓ **File access paradigms in C/C++ vs JS**
  - ✓ **Memory restrictions**

Can be resolved by appropriately adjusting compiler settings



```
double divideNums(double numerator, double denominator) {
  if (denominator == 0) {
    throw std::r
  } return num
}
int main() {
  double num = 10.0, denom = 0.0;
  try {
    std::cout << "Result is " << divideNums(num, denom);
  } catch(const std::runtime_error& e) { // Breaks in Wasm
    std::cout << "Caught exception: " << e.what() << std::endl;
  }
}
```

```
DIR* _dir = ::opendir("/path/to/dir"); dirent* _entry;
if (! _dir) {std::cout << "dir is null"; exit(0);}
```

```
Could not open directory.
```

```
>_ /home/user/ex.js:128
     throw ex;
RuntimeError: Aborted(Assertion failed: Exception thrown, but
       exception catching is not enabled)
at wasm://wasm/0009f20e:wasm-function[12]:0xf48
```

emcc
--preload-file src@/path/to/dir
em++
node
NO_DISABLE_EXCEPTION_CATCHING

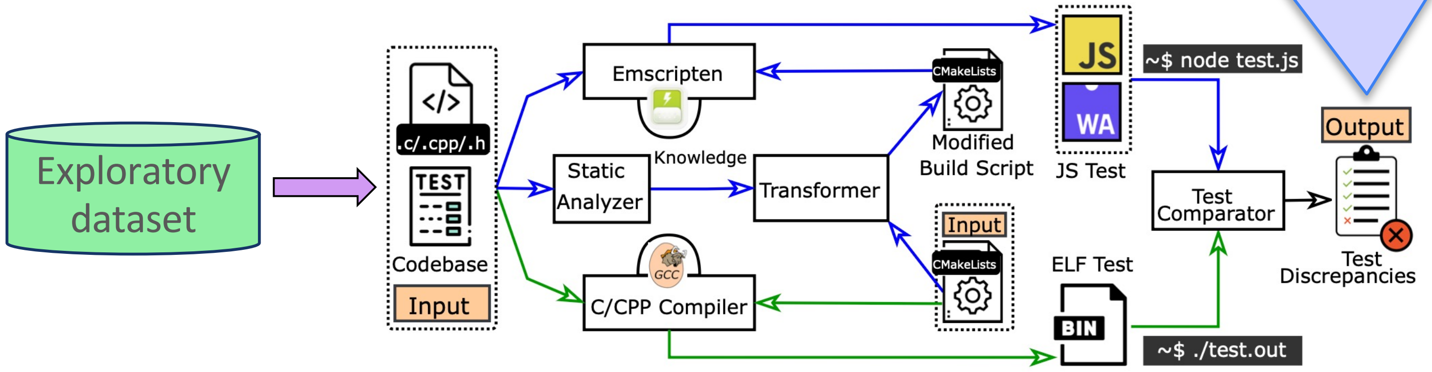
# WasmChecker

We propose a differential testing framework for end-to-end building and testing of C/C++ codebase in Wasm

- ✓ Static analyzer extracts code constructs using CodeQL
- ✓ Transformer appropriately adjusts compiler settings
- ✓ Builds the codebase in both Wasm and native binaries
- ✓ Executes the test in different binaries
- ✓ Reports potential test discrepancies

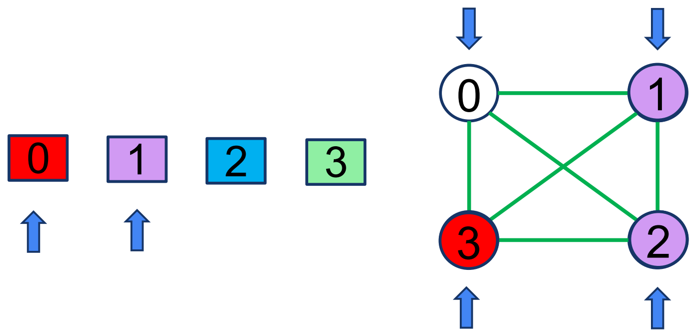
Of 115, could build 99 (86%) codebases in Wasm

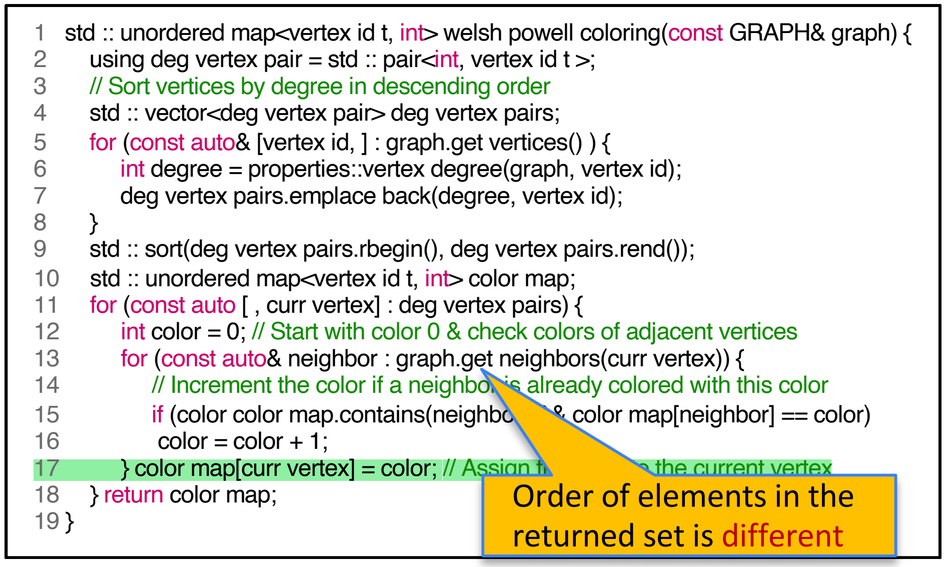
< 1% of the tests show inconsistent behavior



# RQ2: Semantic Divergence Among Binaries

- Executed tests of 119 (99 + 20) projects in Wasm and x86-64
  - ✓ $< 4\%$ of executable tests showed inconsistent outcome
- Manual analysis shows that Emscripten sometimes inevitably fails to preserve semantics of t he source code due to:
  - ✓ **Different standard libraries**
  - ✓ **Unsupported system calls/APIs**
  - ✓ **Wasm language features**
  - ✓ **WebAssembly compiler bugs**





```
std :: unordered map<vertex id t, int> welsh powell coloring(const GRAPH& graph) {
  using deg vertex pair = std :: pair<int, vertex id t >;
  // Sort vertices by degree in descending order
  std :: vector<deg vertex pair> deg vertex pairs;
  for (const auto& [vertex id, ] : graph.get vertices() ) {
      int degree = properties::vertex degree(graph, vertex id);
      deg vertex pairs.emplace back(degree, vertex id);
  }
  std :: sort(deg vertex pairs.rbegin(), deg vertex pairs.rend());
  std :: unordered map<vertex id t, int> color map;
  for (const auto [ , curr vertex] : deg vertex pairs) {
      int color = 0; // Start with color 0 & check colors of adjacent vertices
      for (const auto& neighbor : graph.get neighbors(curr vertex)) {
          // Increment the color if a neighbo  is already colored with this color
          if (color color map.contains(neighbo    & color map[neighbor] == color)
           color = color + 1;
      } color map[curr vertex] = color; // Assign t          the current vertex
  } return color map;
}
```

Order of elements in the returned set is different

## Summary

- We identified the key challenges which affect
  - ✓ Compilation of high-level language code into Wasm binaries
  - ✓ The reliability of WebAssembly code
- We presented WasmChecker, a differential testing tool for checking semantic equivalence between x86-64 binaries of C/C++ code and their Wasm counterparts
- We evaluated WasmChecker on 135 open-source projects and showed the gaps in code semantics translation between WebAssembly compilers and C/C++ compilers
- We identified 11 new bugs within the Emscripten compiler.
- Our collected dataset of open-source codebases and the source code of WasmChecker is available at …

USC University of Southern California

USC Viterbi School of Engineering
*Department of Aerospace and Mechanical Engineering*

Thank you!
Q&A