Dynamic IoT Applications and Isomorphic IoT Systems Using WebAssembly

A Case Study Analysis

Uddhav P. Gautam & Ram Mude

ECE 4984/5984 IoT System Design
Virginia Tech

Problem Statement

Critical Challenges in Modern IoT Development

Challenge 1: Slow Device Updates

- Traditional firmware updates require 10+ seconds downtime
- Full system reboot needed for every update
- Cannot respond quickly to changing user requirements (ML model updates, sensor calibration changes)

Challenge 2: Multi-Platform Complexity

- IoT systems span device, edge, and cloud layers (Raspberry Pi, Google Cloud, local machine)
- Different platforms require different technologies (ARM, x86/64, different OS)
- Multiple codebases increase maintenance burden (separate implementations for ARM, x86/64, arm64)

Research Question: How can WebAssembly solve these IoT development challenges?

Proposed Solutions

Two Innovative WebAssembly-Based Approaches

Solution 1: Dynamic IoT Applications

- Partial updates without system reboot
- Only restart Wasm runtime (1.4s vs 10s+)
- Enable rapid response to user requirements (ML model updates, bug fixes)

Solution 2: Isomorphic IoT Systems

- Same Wasm binary across all layers (device, edge, cloud)
- Common codebase for device, edge, cloud (single Rust/Elixir codebase)
- Improved development efficiency and maintainability (common codebase across platforms)

Key Innovation: WebAssembly's platform-agnostic execution enables both solutions

Same Wasm Binary Runs Across All Layers!

Wasmtube Bridge Enables Dynamic Updates

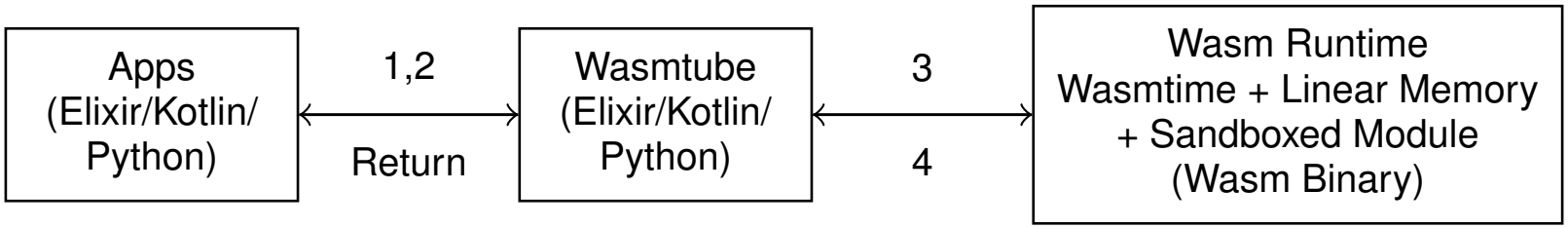
Wasm Runtime			
Wasmtube	**Wasmtube**	**Wasmtube**	**Wasmtube**
Node.js, Python	C/C++, Rust	Swift, Java, Kotlin, Flutter	C/C++, Rust, Elixir
Linux	Linux	iOS, Android OS	FreeRTOS, Linux
VM/Containers	RPi/Baremetal	iPhone, Android	ESP32, Raspberry Pi
Cloud	Edge/Gateway	Mobile	IoT Device

Key Benefits:

- Same Wasmtube (custom development) to bridge runtime for all languages
- Same binary works on IoT, Edge, Mobile, and Cloud
- Real-time application updates without downtime
- Platform-agnostic development and deployment

Wasmtube Architecture: Multi-Language Support

Detailed Communication Flow



Process Flow:

1. **Initialization:** Host app starts Wasmtube with Wasm binary
2. **Function Calls:** Host app calls Wasmtube with function name and data
3. **Processing:** Wasm function processes data in sandboxed environment
4. **Response:** Results returned through Wasmtube to host app

Architecture and Functionality

Wasmtube [**?**] provides a simple API to instantiate Wasm sandboxes individualized for specific Wasm binaries. The library supports structured values and images as arguments to be passed into Wasm functions, enabling complex data exchange between Elixir applications and Wasm modules.

Key Features:

- **Language-Agnostic Design:** Same Wasm binary works all langs.
- **Dynamic Updates:** Wasmtube can handle updates of Wasm binaries on-the-fly, allowing hot-swappable Wasm modules without system restart
- **Structured Data Support:** Supports JSON-encoded structured values and binary-encoded images for complex data processing
- **Linear Memory Management:** Efficient data exchange between host applications and Wasm modules through linear memory

Use Case: Machine Learning Inference

Isomorphic ML System Across IoT Layers

Deployment Architecture:

- **Device Layer:** Raspberry Pi 4 (ARMv8-A)
- **Edge Layer:** Google Cloud (x86/64)
- **Local Layer:** iMac (arm64)

ML Models Tested:

- **ResNet-50:** 25M parameters, GPU-optimized (image classification)
- **MobileNetV2:** Mobile-optimized, efficient (mobile inference)

Process: ONNX models $\rightarrow$ Apache TVM $\rightarrow$ Wasm binary $\rightarrow$ Deploy everywhere

Performance Results

Quantitative Evaluation Results

Update Performance:

- Traditional firmware: 10+ seconds
- Wasm-based updates: 1.4 seconds
- **Improvement:** 7x faster updates

Function Call Overhead:

- Native Elixir: 31.01 μs
- Wasm function call: 252.48 μs
- **Overhead:** $\sim$200 μs (acceptable for complex tasks)

ML Inference Performance:

- MobileNetV2 on Pi 4: 588.69 ms
- MobileNetV2 on Cloud: 168.55 ms
- MobileNetV2 on Local: 58.41 ms

Design Tradeoffs

Technical Decisions and Tradeoffs

Tradeoff 1: Performance vs. Flexibility

- **Cost:** 200μs overhead per Wasm call
- **Benefit:** Dynamic updates without reboot
- **Decision:** Acceptable for complex processing tasks

Tradeoff 2: Portability vs. Hardware Acceleration

- **Cost:** CPU-only execution (no GPU acceleration)
- **Benefit:** Same binary across all platforms
- **Future:** wasi-nn specification for GPU access

Tradeoff 3: Simplicity vs. Functionality

- **Cost:** Limited to Wasm-supported operations
- **Benefit:** Simplified deployment and maintenance

Security and Privacy Considerations

Security Implications of Wasm-Based IoT Systems

Security Benefits:

- **Sandboxed Execution:** Wasm provides isolated execution environment
- **Memory Safety:** Linear memory model prevents buffer overflows
- **Code Integrity:** Binary verification before execution

Security Challenges:

- **System Interface Access:** Core application handles sensitive operations
- **Update Security:** Need secure channels for Wasm binary distribution
- **Runtime Security:** Wasm runtime must be kept updated

Privacy Considerations:

- **Data Processing:** ML models process sensitive image data
- **Edge Processing:** Local inference reduces data transmission
- **Compliance:** GDPR/privacy regulations for data handling

Outcomes and Impact

Success Factors and Future Directions

Success Metrics:

- ✓ **7x faster updates** compared to traditional methods
- ✓ **Successful ML inference** across heterogeneous platforms
- ✓ **Reduced development complexity** through code reuse
- ✓ **Practical performance** suitable for real-world deployment

Future Research Directions:

1. **MLOps Integration:** Streamlined ML model deployment
2. **Workload Offloading:** Dynamic processing distribution
3. **Federated Learning:** Distributed ML training using Wasm

Industry Impact: Enables rapid IoT development with reduced complexity

Summary

Key Takeaways

Problem Solved:

- Rapid IoT device updates without downtime
- Simplified multi-platform IoT development

Solution Delivered:

- Dynamic IoT applications using WebAssembly
- Isomorphic IoT systems with common codebase

Results Achieved:

- 7x faster update times
- Successful cross-platform ML deployment
- Practical performance characteristics

Impact: Demonstrates WebAssembly's potential to revolutionize IoT system development

References

Sources and Additional Reading

Primary Reference: K. Kuribayashi, Y. Miyake, K. Rikitake, K. Tanaka, and Y. Shinoda, "Dynamic IoT Applications and Isomorphic IoT Systems Using WebAssembly," 2023 IEEE 9th World Forum on Internet of Things (WF-IoT), 2023.

Key Technologies:

- WebAssembly: `https://webassembly.org/`
- Apache TVM: `https://tvm.apache.org/`
- Nerves Platform: `https://nerves-project.org/`
- Wasmtime Runtime: `https://wasmtime.dev/`

Related Work: S. Bansal and D. Kumar, "IoT ecosystem: A survey on devices, gateways, operating systems, middleware and communication," Int. J. Wireless Inf. Networks, 2020.

Questions?