Case Study Report: Dynamic IoT Applications and Isomorphic IoT Systems Using WebAssembly

Uddhav P. Gautam, Ram Mude
upgautam@vt.edu, ramm@vt.edu

September 29, 2025

1 Summary

This case study analyzes the research paper [1], which presents innovative solutions to critical challenges in modern IoT system development. The study demonstrates how WebAssembly (Wasm) can enable rapid IoT device updates without system downtime and simplify multi-platform IoT development through isomorphic system architectures.

The research addresses two fundamental problems: (1) the need for rapid and frequent IoT device updates to meet changing user requirements, and (2) the complexity of developing and maintaining multi-layer IoT systems across different platforms and architectures. The proposed solutions leverage WebAssembly's portability and security features to create dynamic IoT applications and isomorphic IoT systems.

2 Problem Statement and Motivation

The rapid proliferation of IoT devices has created significant development challenges [2] that traditional approaches struggle to address:

1. **Rapid Update Requirements:** Developers need to respond quickly to diverse and rapidly changing user requirements, but traditional firmware updates require complete system restarts (10+ seconds), causing significant downtime.
2. **Multi-Platform Complexity:** IoT systems now span multiple layers (device, edge, cloud) with different platforms and architectures, leading to development efficiency and maintainability problems.
3. **Technology Fragmentation:** Different technologies across system layers create complexity and require developers to be proficient in various technologies.

3 Proposed Solutions

The research proposes two complementary solutions using WebAssembly:

3.1 Dynamic IoT Applications

The first solution enables on-demand partial updates of IoT device behavior without requiring system reboots. Unlike traditional firmware updates that require complete system restarts, this approach only updates the application logic (Wasm module) while the core system continues running. The fundamental concept is to separate the application logic into two parts: the core application and the Wasm module. The core application is responsible for stable functionality, while the Wasm module (because it is patched on-demand) is responsible for dynamic functionality. The architecture consists of:

- **Core Application (Elixir):** Handles system interfaces, hardware access, and stable functionality
- **Wasmtube Bridge:** Custom library enabling communication between Elixir and Wasm modules
- **Wasm Runtime:** Executes dynamic application logic in an isolated sandbox
- **Dynamic Wasm Binary:** Contains updatable application behavior (partial update patch)

The system uses the Linux inotify API to detect Wasm binary updates and automatically restarts only the Wasm runtime (1.4 seconds) instead of the entire system (10+ seconds), achieving 7x faster update times.

3.2 Wasmtube: The Universal Wasm Bridge

Wasmtube is a bridging library that enables communication between Elixir applications and WebAssembly (Wasm) modules [3]. It serves as the critical component that makes both dynamic IoT applications and isomorphic IoT systems possible.

3.2.1 Architecture and Functionality

Wasmtube [3] 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.

- **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

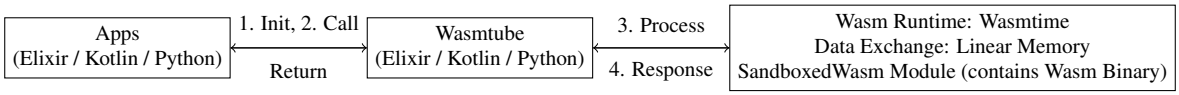

Figure 1: Wasmtube Architecture: Multi-Language Support

3.2.2 Multi-Language Support

The Wasmtube concept demonstrates the universal nature of WebAssembly bridges. While the current implementation is Elixir-specific, the same architectural pattern can be implemented in other languages:

- **Kotlin/Java:** Using Wasmtime Java bindings or JNI wrappers
- **Python:** Using wasmtime-py or similar Python bindings
- **C/C++:** Direct integration with Wasmtime C API
- **JavaScript/Node.js:** Using WebAssembly API

This multi-language support is what enables the isomorphic nature of IoT systems, where the same Wasm binary can run across different platforms and programming languages.

3.3 Isomorphic IoT Systems

The second solution enables the use of the same Wasm binary across different layers of IoT infrastructure, built from a common codebase. This approach provides:

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

Table 1: Isomorphic IoT Systems Architecture Across Different Platforms

- **Code Reuse:** Same Wasm binary runs on different architectures (ARM, x86/64, arm64)
- **Reduced Complexity:** Single codebase instead of multiple platform-specific versions
- **Consistent Behavior:** Identical logic across device, edge, and cloud layers
- **Simplified Deployment:** Single binary deployment process for all platforms

4 Implementation and Technologies

4.1 Core Technologies

WebAssembly (Wasm) [4]

- Binary format for stack-based virtual machines
- Originally developed for web browsers
- Platform-agnostic execution environment
- Sandboxed security model with linear memory

Apache TVM [5]

- Machine learning compiler framework
- Compiles ML models to various target formats
- Supports WebAssembly compilation
- Enables deployment of ML models as Wasm binaries

Elixir and Nerves Platform [6]

- Elixir: Functional programming language built on Erlang VM
- Nerves: IoT development platform for Elixir
- Provides reliable, fault-tolerant IoT device development
- Handles system interfaces and hardware access

4.2 Use Case Implementation

The research demonstrates the concepts through a practical machine learning use case:

1. **Model Compilation:** ResNet-50 and MobileNetV2 models compiled to Wasm binaries using Apache TVM
2. **Cross-Platform Deployment:** Identical binaries deployed on Raspberry Pi 4, Google Cloud instances, and local machines
3. **Inference Processing:** Image recognition and classification using the same Wasm binary across all layers

5 Performance Evaluation

5.1 Overhead Analysis

The study measured the overhead introduced by Wasm function calls:

- **Native Elixir execution:** 31.01 μs average
- **Wasm function call:** 252.48 μs average
- **Overhead:** $\sim$221 μs (approximately 200 μs as reported)

5.2 Performance Results

The overhead is acceptable because:

1. **Constant Overhead:** The 200μs overhead is nearly constant regardless of function complexity
2. **Negligible Impact:** For complex operations like ML inference, overhead becomes negligible (0.037% for MobileNetV2)
3. **Practical Performance:** MobileNetV2 runs in approximately 0.6 seconds on Raspberry Pi 4

6 Design Tradeoffs

6.1 Performance vs. Flexibility

Trade-off: Wasm introduces overhead but enables dynamic updates and cross-platform deployment.

Justification: The 200μs overhead is negligible for complex IoT operations while providing significant architectural benefits.

6.2 Portability vs. Hardware Acceleration

Trade-off: Wasm provides portability but currently lacks direct hardware acceleration support.

Current State: CPU-only execution limits performance for large models like ResNet-50 on resource-constrained devices.

Future Prospects: The wasi-nn specification development may enable GPU acceleration in the future.

7 Security and Privacy Considerations

7.1 Security Benefits

- **Sandboxed Execution:** Wasm provides an isolated execution environment
- **Memory Safety:** Linear memory model prevents buffer overflows
- **Code Integrity:** Wasm binaries can be verified before execution
- **Controlled Access:** No direct system access, controlled by host application

7.2 Security Challenges

- **Core Application Security:** Core application still handles sensitive system operations
- **Binary Distribution:** Need for secure channels for Wasm binary distribution
- **Runtime Security:** Security depends on Wasm runtime implementation
- **Malicious Binaries:** Potential for malicious Wasm binaries if not properly verified

8 Outcomes and Future Directions

8.1 Current Success

The study demonstrates significant practical benefits:

- **7x faster update times** compared to traditional firmware updates
- **Successful cross-platform deployment** of ML models
- **Practical performance** with MobileNetV2 achieving 0.6s inference on Raspberry Pi 4
- **Reduced development complexity** through code reuse

8.2 Future Research Directions

1. **MLOps Integration:** Construction of MLOps flows using isomorphic IoT systems
2. **Workload-Aware Offloading:** Processing offloading across layers using common codebase
3. **Federated Learning:** Application to federated learning scenarios
4. **Hardware Acceleration:** Integration with wasi-nn for GPU acceleration

9 Conclusion

This case study demonstrates that WebAssembly-based IoT systems offer a promising solution to critical challenges in modern IoT development. The research successfully addresses the need for rapid device updates and multi-platform development while maintaining practical performance characteristics.

The proposed dynamic IoT applications and isomorphic IoT systems provide significant benefits in terms of development efficiency, maintainability, and deployment flexibility. While current limitations exist regarding hardware acceleration, the foundation is established for future improvements that could further enhance the effectiveness of these approaches.

The study's success in addressing real-world IoT development challenges while maintaining practical performance characteristics makes it a valuable contribution to the field and a model for future research in IoT system architecture.

References

[1] K. Kuribayashi, Y. Miyake, K. Rikitake, K. Tanaka, and Y. Shinoda, "Dynamic iot applications and isomorphic iot systems using webassembly," in *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*, pp. 1–8, 2023.

[2] S. Bansal and D. Kumar, "Iot ecosystem: A survey on devices, gateways, operating systems, middleware and communication," *International Journal of Wireless Information Networks*, vol. 27, 09 2020.

[3] K. Kuribayashi, "Wasmtube: A bridging library which allows you to communicate between elixir and wasm." `https://github.com/kentaro/wasmtube`, 2023. Accessed 2023.

[4] WebAssembly Community Group, "Webassembly specification." `https://webassembly.org/`, 2023. Accessed 2023.

[5] Apache Software Foundation, "Apache tvm: An end to end machine learning compiler stack." `https://tvm.apache.org/`, 2023. Accessed 2023.

[6] The Nerves Project, "Nerves: Build bulletproof, embedded software in elixir." `https://nerves-project.org/`, 2023. Accessed 2023.