

Toward Semantic Serializability in Agentic IoT: A Lane-Based Execution Model for Espressif Systems Platform 32-Bit Edge Nodes [†]

Hsien-Cheng Chou ^{*}, Jui-Ming Chang and Ching-I Ma 

Department of Information Engineering and Computer Science, Tatung University, Taipei 104327, Taiwan; unclfish000@gmail.com (J.-M.C.); catherine55688@gmail.com (C.-I.M.)

^{*} Correspondence: chchou@gm.ttu.edu.tw

[†] Presented at the 6th International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB 2026), New Taipei, Taiwan, 24–26 April 2026.

Abstract

The emergence of AI agents and open-domain automation frameworks such as OpenClaw has enabled large language model (LLM) reasoning to directly control microcontroller-based IoT devices. However, when agents issue concurrent asynchronous tool calls, conventional IoT systems may encounter race conditions, resulting in unpredictable state transitions, data loss, or configuration conflicts. In this study, the inconsistency introduced by agent-driven concurrency is addressed by proposing a semantically aware lane-based execution model (LEM). The model enforces deterministic and serializable state transitions on resource-constrained edge hardware by separating operations based on their semantic impact. State-modifying operations are executed in a single ordered main lane, while read-only operations are handled concurrently in side lanes. The model evaluation is conducted in two stages. First, software-based stress testing across representative agent scenarios shows that naive asynchronous execution achieves only 0–27% success rates, and LEM consistently maintains correctness. Second, hardware validation on an Espressif Systems Platform 32-bit, Reduced Instruction Set Computer-V Core (C3 variant) (ESP32-C3) running FreeRTOS indicates that LEM preserves correctness comparable to mutex-based synchronization, while reducing execution latency by up to 34% under read-dominant workloads. A semantic state hash mechanism is further introduced to verify state transitions, serving as a lightweight mechanism for execution verification and reproducibility. These results suggest that semantic-aware execution can serve as a practical and efficient approach for reliable AI-agent control in resource-constrained IoT environments.

Keywords: IoT; AI agents; concurrency control; embedded systems; edge computing; semantic serializability



Academic Editors: Teen-Hang Meen, Shu-Han Liao and Cheng-Fu Yang

Published: 21 October 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Recent advances in large language model (LLM)-based agents enable systems to autonomously invoke external tools and coordinate complex workflows. Frameworks such as OpenClaw exemplify this trend by enabling LLM-based agents to orchestrate tool execution across heterogeneous environments. However, such open-domain agent frameworks also amplify concurrency challenges when multiple tool calls interact with shared device state. In IoT environments, this capability allows edge devices to be controlled dynamically by agent-generated commands rather than static firmware logic.

However, this paradigm introduces concurrency problems. Unlike traditional thread-level concurrency, conflicts arise from semantic overlaps in agent-issued operations, where semantic refers to the effect of an operation on system state rather than low-level resource access. Such conflicts arise not from shared memory access, but from overlapping effects on logical system state. When multiple asynchronous tool calls access shared device state, uncontrolled interleavings may lead to inconsistent and non-deterministic outcomes. This causes a fundamental difference between agent-driven concurrency and conventional synchronization mechanisms, where conflicts are typically associated with shared resources rather than semantic dependencies.

Embedded systems commonly rely on synchronization primitives such as mutexes and semaphores provided by real-time operating systems (RTOS) [1]. Whereas these mechanisms ensure correctness, they often impose coarse-grained serialization and limit concurrency even when operations are logically independent. This requires solutions to ensure deterministic and correct execution in agent-driven IoT systems while preserving concurrency where operations are semantically independent.

To address this limitation, we introduce a lane-based execution model (LEM). The model separates operations according to their semantic roles: state-modifying operations are serialized in a main lane, while read-only operations are executed concurrently in side lanes. This approach preserves determinism while allowing concurrency where it is safe. In this study, a semantics-aware execution model for agent-driven IoT systems was developed with a lightweight implementation on Espressif Systems Platform 32-bit, Reduced Instruction Set Computer-V Core (C3 variant) (ESP32-C3) with Free Real-Time Operating System (FreeRTOS). A verification mechanism [semantic state hash (SSH)] for reproducible state transitions. Experimental validation results show both correctness and performance characteristics.

2. Related Work

Concurrency control has been extensively studied in database systems, where serializability ensures correctness under concurrent transactions [2,3]. These principles provide a theoretical foundation for ordered state transitions. In embedded environments, FreeRTOS offers synchronization primitives such as mutexes and semaphores for task coordination [1]. However, these mechanisms operate at the scheduling level and do not consider the semantic nature of operations.

Recent research on LLM-based agent systems, including automatic generation [4], focuses on multi-agent coordination and tool orchestration. These approaches typically assume cloud-scale execution environments and do not address concurrency control in resource-constrained devices. Edge computing research emphasizes efficiency and reliability under constrained resources [5], but semantic-aware concurrency control for agent-driven IoT systems remains relatively unexplored. LEM is used to introduce a lightweight execution model that incorporates semantic distinctions into concurrency management. However, existing approaches do not explicitly address semantic-level concurrency control in resource-constrained, agent-driven IoT environments.

3. LEM

3.1. System Model

Let S denote the system state. Each agent-issued operation A_n produces:

$$S_n = f(S_{\{n-1\}}, A_n) \quad (1)$$

Without ordering constraints, even if f is a deterministic state transition function, concurrent execution may lead to divergent state transitions.

3.2. Lane Architecture

Write operations are executed in the main lane to ensure serialization, and read operations are processed in side lanes to enable parallel execution. This design differs from conventional readers–writers synchronization by classifying operations based on their semantic effects rather than shared resource access (Figure 1).

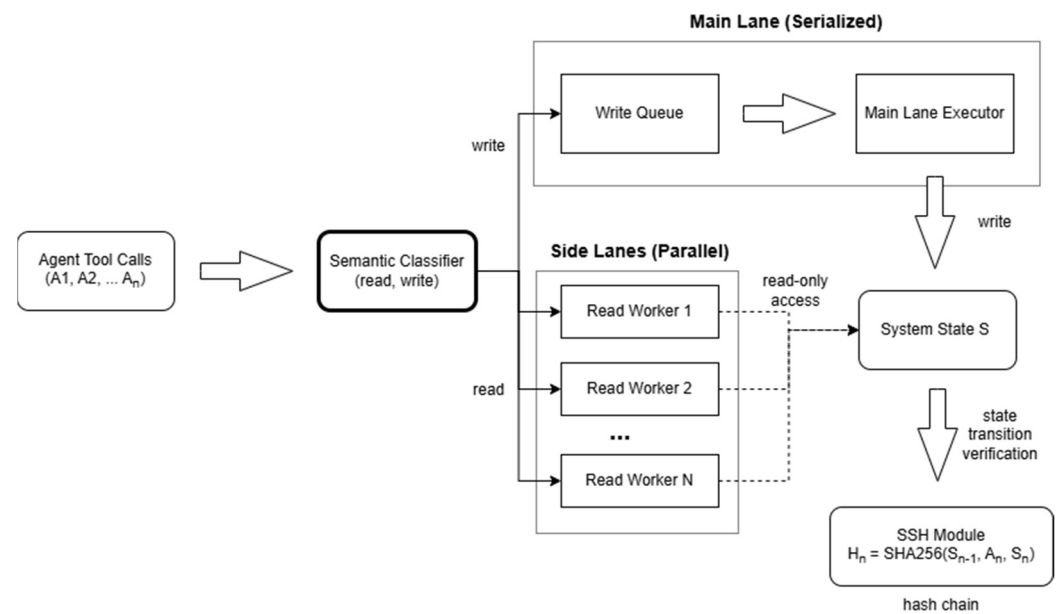


Figure 1. Overview of LEM. Operations are first classified based on their semantic effects. State-modifying operations are serialized in the main lane, while read-only operations are executed concurrently in side lanes. The SSH module verifies the consistency of state transitions.

3.3. SSH

Each state transition is associated with:

$$H_n = \text{SHA256}(S_{n-1} \| A_n \| S_n) \quad (2)$$

SSH is used to verify that state transitions form a consistent hash chain. Divergence across repeated executions indicates non-deterministic behavior, providing a lightweight mechanism for integrity and reproducibility checking. This mechanism forms a hash chain that enables tamper-evident verification of execution traces.

4. Implementation

The system was implemented on an ESP32-C3 microcontroller (Espressif Systems (Shanghai) Co., Ltd., Shanghai, China) using ESP-IDF (version 5.3; Espressif Systems, Shanghai, China) and its integrated FreeRTOS implementation (kernel version 10.5.1). Operations are classified using predefined semantic annotations associated with each tool call, distinguishing between state-modifying and read-only behaviors. These annotations are manually defined at the interface level, where each operation explicitly specifies whether it alters persistent state, ensuring deterministic classification. Whereas predefined annotations are adopted in this work for deterministic evaluation, the classification mechanism is extended to dynamic approaches, such as LLM-based inference or metadata-driven tool descriptions. This allows the model to scale toward open-domain agent frameworks where tool semantics are not predefined.

The system contains several key components, including an operation classifier, a main lane scheduler, side lane workers, and an SSH verification module. The classifier determines the semantic role of each operation, while the main lane ensures serialized execution of state-modifying tasks. Side lane workers handle read-only operations in parallel, enabling improved concurrency. The implementation maintains a lightweight footprint suitable for resource-constrained devices. The total memory usage is approximately 12 KB of flash and 4 KB of random access memory, demonstrating the feasibility of deploying LEM on microcontroller platforms such as the ESP32-C3.

The SSH mechanism also serves as a lightweight verification layer for execution consistency. By forming a hash chain over state transitions, the system enables detection of non-deterministic behavior across repeated executions, providing an additional guarantee of reproducibility in agent-driven environments.

5. Experimental Evaluation

5.1. Method

Three execution models are evaluated in this study: naive execution without synchronization, mutex-based synchronization, and the proposed LEM approach. The evaluation considers two workload scenarios, namely write-heavy (70% write operations) and read-heavy (70% read operations), to capture different concurrency characteristics. In each experiment, 200 trials are conducted. In each trial, six concurrent tasks are created, and each task performs six operations. Operation types are sampled according to the specified workload ratio. To emulate realistic execution conditions, randomized scheduling yields are introduced within read-modify-write sections to simulate preemption. Latency values are averaged across trials, with variability reported using minimum and maximum values. Failure is defined as divergence in the final SSH across repeated executions under identical input conditions, indicating non-deterministic state transitions.

5.2. Results

Table 1 shows the success rates and average latency across different execution models under both write-heavy and read-heavy workloads. The results show a fundamental trade-off between serialization and parallelism. While strict serialization ensures correctness, it may unnecessarily limit concurrency when operations are semantically independent.

Table 1. Experimental results.

Workload	Mode	Success (%)	Avg. Latency (ms)
Write-heavy	Naive	0	45.5
	Mutex	100	89.1
	LEM	100	94.2
Read-heavy	Naive	0	15.6
	Mutex	100	124.8
	LEM	100	82.4

Although naive execution exhibits lower latency, this is achieved at the cost of correctness, as the absence of synchronization allows faster but non-deterministic execution. Due to randomized scheduling, even minor variations in execution order can lead to divergent state transitions, resulting in inconsistent final states. In contrast, the performance advantage of LEM becomes evident under read-heavy workloads. By allowing read-only operations to execute concurrently in side lanes, LEM reduces unnecessary serialization and improves overall system throughput. This demonstrates that exploiting semantic independence can significantly enhance execution efficiency.

However, under write-heavy conditions, the benefits of LEM diminish as most operations must be executed in the main lane. This contrast between read-heavy and write-heavy scenarios indicates that the effectiveness of LEM is closely tied to the proportion of semantically independent operations.

As shown in Figure 2, LEM achieves lower latency than mutex-based synchronization under read-heavy workloads, primarily due to the parallel execution of read-only operations. Specifically, latency is reduced from 124.8 ms (mutex) to 82.4 ms, corresponding to an improvement of approximately 34%. Under write-heavy workloads, the benefits of parallelism diminish, as state-modifying operations must be serialized to ensure semantic consistency.

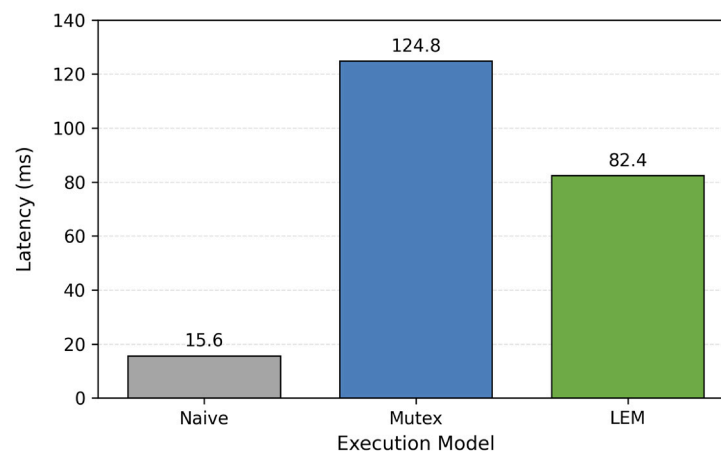


Figure 2. Latency comparison under read-heavy workload (70% read).

5.3. Analysis

LEM achieves correctness comparable to mutex-based synchronization. Under write-heavy workloads, it exhibits a slight increase in latency due to classification and scheduling overhead. The observed 0% success rate in naive execution reflects inherent sensitivity to race conditions under randomized scheduling, rather than an artificially constructed workload. This overhead is interpreted as a trade-off that enables parallel execution of read operations. When read operations dominate, LEM reduces latency by avoiding unnecessary serialization. Without ordering guarantees, execution-order variations can lead to divergent state outcomes. This suggests that semantic-aware scheduling can effectively decouple correctness guarantees from unnecessary synchronization overhead.

6. Conclusions

To address the problem of concurrency control in agent-driven IoT systems, where conflicts arise from semantic overlaps rather than traditional resource contention, LEM is used to enforce semantic serializability by separating operations according to their effects on system state. Experimental results show that LEM achieves deterministic and correct execution comparable to mutex-based synchronization, while improving performance under read-heavy workloads through parallel execution of semantically independent operations. These findings confirm that incorporating semantic awareness into execution scheduling enables a more efficient balance between correctness and concurrency. In addition, the proposed Semantic SSH mechanism enables a lightweight verification method for ensuring consistency and reproducibility of state transitions, making it suitable for resource-constrained edge environments such as ESP32-based systems.

The results of this study show that semantic-aware execution models serve as a practical foundation for reliable and scalable agent-driven IoT systems, bridging the gap between

AI-driven automation and embedded concurrency control. The developed method applies to other agent-driven or distributed systems where semantic-level conflicts dominate over resource-level contention.

Author Contributions: Conceptualization, J.-M.C.; software, J.-M.C.; formal analysis, H.-C.C., J.-M.C. and C.-I.M.; writing—original draft preparation, H.-C.C., J.-M.C. and C.-I.M.; writing—review and editing, H.-C.C., J.-M.C. and C.-I.M.; supervision, H.-C.C.; project administration, H.-C.C., J.-M.C. and C.-I.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Tatung University Fundamental Research Grant (Grant No. B114-I01-005).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The experimental data and source code generated and used in this study are not publicly available.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Barry, R. *Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide*; Real Time Engineers Ltd.: Bristol, UK, 2016. Available online: https://www.freertos.org/Documentation/RTOS_book.html (accessed on 2 April 2026).
2. Bernstein, P.A.; Hadzilacos, V.; Goodman, N. *Concurrency Control and Recovery in Database Systems*; Addison-Wesley: Boston, MA, USA, 1987.
3. Gray, J.; Reuter, A. *Transaction Processing: Concepts and Techniques*; Morgan Kaufmann: San Francisco, CA, USA, 1992.
4. Wu, Q.; Bansal, G.; Zhang, J.; Wu, Y.; Li, B.; Zhu, E.; Jiang, L.; Zhang, X.; Zhang, S.; Liu, J.; et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv* **2023**, arXiv:2308.08155. [[CrossRef](#)]
5. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.