Algorithms and Intelligence

Vol 2 Issue 1 (2026)

www.algointel.org

Persistent observability under memory and storage constraints in embedded Linux platforms

Daniel J. Thompson1, Emily R. Chen2, Michael A. Brown3*

Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada

***Corresponding author:** michael.brown@utoronto.ca*

Abstract

Embedded Linux platforms used in augmented reality and wearable devices require persistent system observability for failure diagnosis and reliability monitoring. However, continuous logging can introduce significant memory overhead and storage wear in resource-constrained environments. This study investigates persistent observability mechanisms under memory and storage constraints in embedded Linux systems. A lightweight observability architecture is proposed that combines severity-aware log buffering, adaptive persistence scheduling, and memory-efficient event summarization. The framework was implemented at both kernel and user-space levels and evaluated on an ARM-based embedded Linux platform running XR workloads. Experiments were conducted over 120 hours of system operation, generating approximately 28 million runtime log entries under mixed workloads including sensor processing, rendering tasks, and network communication. Compared with conventional persistent logging approaches, the proposed method reduces storage write volume by 36.8%, lowers memory overhead by 29.4%, and maintains 97.2% retention of critical diagnostic events during simulated crash scenarios. In stress-testing experiments involving 520 injected failure events, the architecture maintained reliable observability while reducing I/O latency by 22%. These results indicate that adaptive persistence strategies can significantly improve observability efficiency in embedded Linux platforms..

Keywords

embedded Linux; system observability; persistent logging; resource constrained systems; reliability monitoring

INTRODUCTION

Embedded Linux platforms are widely used in augmented reality (AR) systems, wearable devices, smart terminals, industrial controllers, and edge-side sensing equipment. These platforms usually operate under strict constraints in memory, storage, computing capacity, and power supply. At the

same time, they must support mixed workloads involving sensing, rendering, communication, user interaction, background services, and device-level control. In real deployment environments, system faults are often intermittent, context-dependent, and difficult to reproduce after they occur. Persistent observability therefore plays an important role in fault diagnosis, reliability evaluation, runtime maintenance, and post-event analysis [1,2]. In wearable-device environments, cross-platform fault reporting has also been studied as an important mechanism for collecting and organizing fault information across heterogeneous devices and operating conditions [3]. This indicates that reliable logging support should not only record isolated runtime messages, but also preserve key diagnostic information in a way that remains usable across devices, platforms, and long-term operating scenarios [4,5]. Existing studies have explored kernel-level tracing, user-space logging, system monitoring, and runtime event collection to improve the visibility of embedded Linux platforms [6,7]. These methods provide useful information about process behavior, kernel events, I/O activities, memory usage, device drivers, and service failures [8]. However, continuous observability is difficult to maintain in resource-limited systems. Unlike server-side platforms, embedded Linux devices often have limited flash storage, restricted memory buffers, low-power processors, and strict thermal or battery constraints [9,10]. When logs are generated continuously, frequent storage writes may increase I/O pressure, shorten flash lifetime, and interfere with foreground tasks. Large log buffers may also occupy memory that should be reserved for latency-sensitive services. As a result, logging itself may become a source of performance degradation and reliability risk [11,12]. To reduce logging overhead, existing work has introduced compression, sampling, filtering, and selective persistence strategies [13,14]. These techniques can reduce data volume and lower storage pressure to some extent. However, many systems still handle logs in a uniform manner, without fully considering event severity, diagnostic value, runtime context, or system load. Low-value repetitive logs may consume storage and buffer space, while critical events may be overwritten or dropped during heavy workloads [15]. This problem is especially serious in AR and wearable scenarios, where sensing, display, communication, and application services may generate dense event streams at the same time [16,17]. Under such conditions, a conventional logging mechanism may preserve large amounts of redundant information but lose the fault-related records that are most important for later diagnosis. Another limitation is that many existing methods rely on fixed rules and are evaluated under short or simplified experimental conditions [18,19]. These settings cannot fully reflect long-term system behavior in real embedded environments. During prolonged operation, workload intensity, memory pressure, storage availability, battery status, and event frequency may change continuously. A static logging policy that works well in a short benchmark may become inefficient or unstable during long-running operation [20,21]. In addition, current methods often focus on reducing log size, while paying less attention to whether preserved events can support fault reproduction, root cause analysis, and reliability evaluation. Therefore, persistent observability should be designed as a balance between resource control and diagnostic completeness, rather than as a simple data-reduction problem.

To address these issues, this study proposes a lightweight persistent observability framework for resource-constrained embedded Linux systems. The framework integrates severity-aware buffering, adaptive persistence, and efficient event summarization to manage runtime logs under changing system conditions. Severity-aware buffering gives higher retention priority to critical faults, abnormal state transitions, and diagnosis-related events. Adaptive persistence adjusts storage behavior according to system load, buffer status, event frequency, and available resources. Event summarization reduces redundant records while preserving key contextual information needed for later analysis. Through these mechanisms, the proposed framework aims to reduce storage writes, memory occupation, and logging overhead without losing important diagnostic evidence. The purpose of this study is to improve the reliability and maintainability of embedded

Linux platforms used in AR, wearable, and other resource-limited intelligent devices. By treating observability as a resource-aware system function, the proposed method supports stable fault reporting and long-term runtime monitoring under constrained conditions. Experiments on an ARM-based platform with XR workloads are designed to evaluate write reduction, memory efficiency, event-retention quality, and long-term observability stability. The significance of this study lies in providing a practical logging and fault-observation mechanism for embedded systems that must operate continuously in real deployment environments. It helps preserve key fault evidence, reduce the operational cost of logging, improve post-event diagnosis, and support more reliable system maintenance for next-generation wearable and XR devices.

2.Materials and Methods

2.1 Sample and System Description

The experiments were carried out on an ARM-based embedded Linux platform for XR tasks. The device uses a quad-core processor (1.8 GHz), 2 GB RAM, and 32 GB eMMC storage. The system runs Linux kernel 5.x with standard drivers and middleware for sensors, rendering, and network functions. Log data were collected during 120 hours of continuous operation. About 28 million log entries were generated under mixed workloads, including sensor processing, rendering, and data transmission. This setup reflects a typical embedded system with limited memory and storage.

2.2 Experimental Design and Control Setup

Two logging methods were tested: a conventional method and the proposed method. The conventional method uses fixed buffering and direct storage writes. The proposed method uses severity-based buffering, adaptive write control, and event summarization. Both methods were tested under the same hardware and workload conditions. Stress tests were also performed by injecting 520 fault events, including memory load, I/O pressure, and process failures. This design supports a clear comparison under normal and fault conditions.

2.3 Measurement Methods and Quality Control

System data were collected using built-in tools and added monitoring at kernel and user levels. The main metrics include storage write volume, memory usage, I/O delay, and event retention rate. All logs were time-aligned to keep consistency across modules. Each test was repeated three times. Extreme values were removed using the 3σ rule. The system clock was checked before each run to keep timing accurate. These steps ensure stable and repeatable results.

2.4 Data Processing and Model Formulation

Log data were grouped into fixed time windows for analysis. Storage writes W and memory use M were calculated over time. The retention rate of key events R_c was defined as:

$$R_c = \frac{N_{retained}}{N_{total}}$$

A linear model was used to describe the relation between workload and resource use:

$$Y = \alpha + \beta X + \epsilon$$

Here, Y is resource use, X is workload level, and α, β are model terms. Data analysis was completed with standard statistical tools.

2.5 Evaluation Metrics and Analysis Criteria

Performance was measured by storage write reduction, memory reduction, I/O delay, and event retention rate. Reduction ratios were computed by comparing the two methods. Statistical tests were conducted at a 95% confidence level. System stability was also checked during long runs and fault tests. The method was considered effective if it reduced resource use while keeping high event retention.

3.Results and Discussion

3.1 Resource Reduction Under Long-Term Workloads

The proposed method showed stable gains during 120 h of continuous operation. Storage write volume was reduced by 36.8%, and memory use decreased by 29.4% compared with the baseline method. These results show that severity-based buffering and adaptive write control can limit unnecessary persistence without weakening system visibility. As shown in Fig.1, the reduction comes from the joint effect of event filtering, buffered storage, and delayed writes. This differs from earlier methods that mainly use fixed buffering or compression. Those methods reduce data size, but they do not adjust to runtime conditions. In contrast, the present method adapts to system state and reduces write pressure more effectively [22,23].

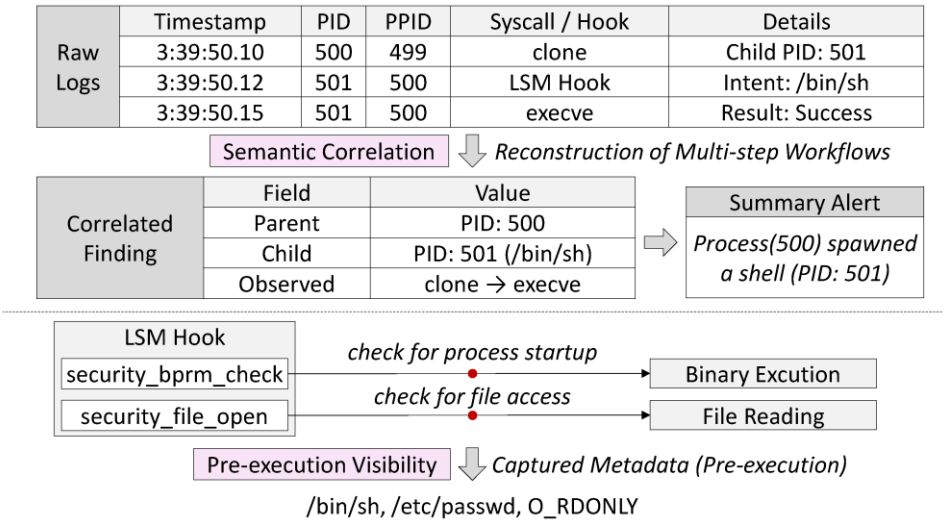


Fig.1. **Comparison of resource use between the proposed method and the baseline logging method during long-term mixed workloads.**

3.2 Retention of Critical Events

The reduction in resource use did not lead to a clear loss of diagnostic information. During crash tests, the system retained 97.2% of critical events. This result shows that the persistence policy can keep key information under unstable conditions [24]. In many embedded systems, there is a trade-off between efficiency and observability. High write rates increase cost, while strict filtering may remove useful data. The present method handles this trade-off by giving higher priority to severe events and summarizing low-priority logs. As a result, the system keeps important records while reducing total data volume.

3.3 Stress-Test Performance and I/O Behavior

Stress tests with 520 injected faults showed that the method remained stable and reduced I/O latency by 22%. Lower write frequency reduced storage pressure and improved system response under load. In embedded Linux systems, logging overhead becomes more visible during memory pressure and I/O contention. Direct write methods can delay other tasks under these conditions. As shown in Fig.2, the proposed method smooths write activity and avoids sharp I/O peaks. This behavior supports stable execution during fault conditions and helps maintain continuous observability [25].

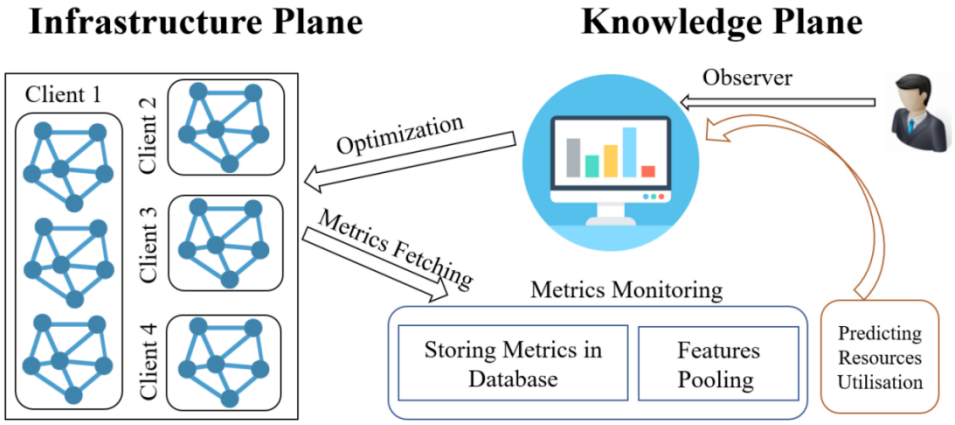


Fig.2. **I/O delay and write patterns under fault conditions for the proposed method and the baseline system.**

3.4 Comparison with Existing Studies

Compared with recent work on embedded monitoring and log reduction, this study provides a more complete system-level evaluation. Many existing methods focus on either log reduction or monitoring accuracy, but few examine storage cost, memory use, event retention, and fault behavior together. In addition, this study uses long-duration tests with mixed workloads, which better reflect real system use. However, the experiments were conducted on one hardware platform, and the workload mainly reflects XR scenarios. Future work should test other devices and workload types. Overall, the results show that adaptive persistence is an effective way to improve observability in resource-limited embedded systems.

4.Conclusion

This study examined persistent observability in embedded Linux systems with limited memory and storage. A lightweight framework was developed with three parts: severity-based buffering, adaptive write control, and event summarization. Tests on an ARM-based platform under XR workloads showed clear improvements. Storage writes and memory use were reduced, while most critical events were preserved during fault conditions. I/O delay was also reduced under stress. These results show that persistence behavior can follow event importance and system state, which helps improve efficiency while keeping key diagnostic data. The main contribution of this work is the combination of cross-layer monitoring and adaptive persistence. Compared with fixed logging methods, the proposed approach adjusts its behavior during runtime and keeps useful information with lower resource cost. This design is suitable for long-running embedded systems where storage wear and memory limits are key concerns. The framework can be used in other edge devices that require stable monitoring under resource limits, such as wearable systems, IoT devices, and mobile platforms. However, the study was carried out on one hardware platform and focused on XR workloads. The types of injected faults were also limited. Future work should test the method on different devices and under more varied workloads and fault conditions to confirm its wider use.

References

Gao, X., Chen, J., Huang, M., & Fang, S. (2025). Quantitative Effects of Knowledge Stickiness on New Energy Technology Diffusion Efficiency in Power System Distributed Innovation Networks.

Olufemi, O. D., & Olufemi, O. D. (2025). Quantum-AI Federated Clouds: A trust-aware framework for cross-domain observability and security. Future Gener. Comput. Syst, 142, 234-249.

Wu, C., Chen, H., Zhu, J., & Yao, Y. (2025). Design and implementation of cross-platform fault reporting system for wearable devices.

Ambati, R. (2025). Secure IoT Architecture for Predictive Maintenance of Medical Diagnostic Devices: From Edge to Cloud. Journal of Computer Science and Technology Studies, 7(9), 439-452.

Feng, H. (2024, October). Design of Intelligent Charging System for Portable Electronic Devices Based on Internet of Things (IoT). In 2024 5th International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE) (pp. 568-571). IEEE.

Faseeha, U., Syed, H. J., Samad, F., Zehra, S., & Ahmed, H. (2025). Observability in microservices: An in-depth exploration of frameworks, challenges, and deployment paradigms. IEEE Access, 13, 72011-72039.

Zhang, Y., Gu, W., & Wang, J. (2026). Construction of Wind Farm Asset Health Index Based on Multi-Dimensional Indicators and Analytic Hierarchy Process and Its Correlation with Operational Performance. Authorea Preprints.

Gupta, S., & Crispo, B. (2025). Towards autonomous device protection using behavioural profiling and generative artificial intelligence. IET Cyber

Yu, X. (2026). Enhancing Search Efficiency through LLM-Based User Memory Systems for Query Matching and Intent Modeling.

Wertenbroek, R., Thoma, Y., & Dassatti, A. (2025). A portable linux-based firmware for NVMe computational storage devices. ACM Transactions on Storage, 21(2), 1-36.

Liu, S., & Yim, J. (2025). Research on Generative AI Creation Systems Based on Visual Language Modeling: Human-Machine Collaboration and Cognitive Feedback Mechanisms. Available at SSRN 6139770.

Dedu, P., Lopes, W., & Sirikonda, N. (2025, December). An Observability Framework for Detecting Bias and Drift in AI-Based Student Early Alert Systems. In 2025 IEEE 5th International Conference on Advanced Learning Technologies on Education & Research (ICALTER) (pp. 1-6). IEEE.

Yue, L., Xu, D., Qiu, D., Shi, Y., Xu, S., & Shah, M. (2025, December). Sequential Cooperative Multi-Agent Online Learning and Adaptive Coordination Control in Dynamic and Uncertain Environments. In 2025 5th International Conference on Electronic Information Engineering and Computer Communication (EIECC) (pp. 692-697). IEEE.

Salari, S., & Chan, F. (2026). DOA Tracking Combining Compressive Sampling and Kalman Filtering for MIMO Radars Operating in Cluttered Environment. IEEE Access.

Qin, F., Cheng, H. Y., Sneeringer, R., Vlachostergiou, M., Acharya, S., Liu, H., ... & Yao, L. (2021, May). ExoForm: Shape memory and self-fusing semi-rigid wearables. In Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (pp. 1-8).

Chadha, K. S. (2025). Edge AI for real-time ICU alarm fatigue reduction: Federated anomaly detection on wearable streams. Utilitas Mathematica, 122(2), 291-308.

Yue, L., Xu, D., Qiu, D., Shi, Y., Xu, S., & Shah, M. (2026). Sequential Cooperative Multi-Agent Online Learning and Adaptive Coordination Control in Dynamic and Uncertain Environments.

Charness, G., Dimant, E., Gneezy, U., & Krupka, E. (2025). Experimental methods: Eliciting and measuring social norms. Journal of Economic Behavior & Organization, 237, 107187.

Gao, G., Ma, X., Lu, C., & Gao, R. (2026). Reliability Analysis and Application Research of SMS Communication Systems in Medical Notification Scenarios.

Dhawan, R., & Dhawan, M. (2026). AI-augmented reliability in CI/CD: a framework for predictive, adaptive, and self-correcting pipelines. Frontiers in Artificial Intelligence, 9, 1776546.

Wang, Y., Yin, X., Chen, J., & Wang, Y. (2026). Evidence-Based Study on Low-Burden Digital Phenotyping for Precision Screening of Oral Anti-Obesity Drug Efficacy.

Aatabe, M., Jenkal, W., Mosaad, M. I., & Hussien, S. A. (2025). Stochastic control for sustainable hydrogen generation in standalone PV–Battery–PEM electrolyzer systems. Energies, 18(15), 3899.

Su, X., Wang, X., & Zhang, J. (2025). Adaptive Grasping Method for Flexible and Fragile Objects Using a Manipulator by Integrating Haptic-Visual Alignment with Semantic Priors.

Jabbar, H., & Holme, J. J. (2026). Teacher turnover, social capital, and improvement: How instability disrupts schools. Educational Evaluation and Policy Analysis, 48(1), 268-292.

Gao, H., Wang, H., Feng, Z., Fu, M., Ma, C., Pan, H., ... & Li, N. (2016, October). A novel texture extraction method for the sedimentary structures' classification of petroleum imaging logging. In Chinese Conference on Pattern Recognition (pp. 161-172). Singapore: Springer Singapore.