

# LiReS IDS: A Lightweight, Real-time, and Secure Intrusion Detection System for RISC-V Edge Devices

Songxuan Liu  
School of Engineering  
Institute of Science Tokyo  
Tokyo, Japan  
liu.s.au@m.titech.ac.jp

Qingyu Zeng  
School of Engineering  
Institute of Science Tokyo  
Tokyo, Japan  
qyzeng@cad.ict.e.titech.ac.jp

Jan Tobias Mühlberg  
Ecole Polytechnique, BEAMS  
Université Libre de Bruxelles  
Brussels, Belgium  
jan.tobias.muehlberg@ulb.be

Yuko Hara  
School of Engineering  
Institute of Science Tokyo  
Tokyo, Japan  
hara@cad.ict.e.titech.ac.jp

**Abstract**—As edge devices take on greater responsibility for processing sensitive data, they also face escalating risks from sophisticated cyberattacks. Machine learning-based intrusion detection systems (IDSs) are a key defense mechanism, but deploying IDS on edge devices poses three challenges: models must remain lightweight, inference must operate in real-time, and the IDS itself must be secured against adversarial manipulation. In this paper, we present LiReS IDS, a lightweight, real-time, and secure IDS tailored for RISC-V edge devices. First, we introduce a genetic algorithm-driven structured pruning method that reduces the size of CNN-based IDS by up to 80% while preserving detection accuracy, enabling deployment on memory-constrained RISC-V edge platforms. Second, we design a TEE-secured IDS architecture using Keystone enclaves to ensure the confidentiality and integrity of the IDS. Finally, we implement and evaluate LiReS IDS on the SiFive HiFive Unmatched RISC-V board using multiple benchmark datasets (NSL-KDD, UNSW-NB15, and CICIDS-2018), achieving real-time detection performance with a latency of 50 ms, demonstrating its practical feasibility for next-generation RISC-V edge computing environments.

**Index Terms**—Intrusion Detection Systems, Trusted Execution Environments, RISC-V.

## I. INTRODUCTION

The rapid proliferation of Internet of Things (IoT) devices and edge computing platforms has transformed the way data is collected, processed, and analyzed [1]. Unlike traditional cloud-centric architectures, where raw data must be transmitted to distant servers for processing, edge computing enables computation to be performed closer to the data source. This paradigm shift provides several advantages: reduced communication latency, improved responsiveness for real-time applications, and enhanced privacy by keeping sensitive data local [2]. This shift is supported by the rise of RISC-V, an open architecture that enables researchers and industry practitioners to design customized processors with fine-grained control over performance, energy efficiency, and security features [3]. RISC-V-based edge devices have already been adopted across diverse domains, including servers, healthcare, and autonomous vehicles.

However, the ubiquity, connectivity, and ability of these devices to interact with sensitive data make them attractive

targets for cyber attackers. To address these risks, one promising approach is to deploy an intrusion detection system (IDS) directly on edge devices. IDS serves as a defense mechanism that monitors system activities and network traffic to identify malicious behavior. Broadly, IDSs can be classified into two categories: signature-based and anomaly-based. Signature-based IDS rely on predefined patterns of known attacks, offering efficient detection for well-characterized threats but limited capability against zero-day attacks. In contrast, anomaly-based IDS learn patterns of normal behavior and flag deviations as potential intrusions [4]. This latter approach, often driven by machine learning (ML) techniques, has proven to be more effective in recognizing previously unseen attacks.

Deploying an ML-based IDS on the edge has many technical challenges. First, it has to be lightweight enough to be deployed on resource-constrained edge devices [5]. Second, the protection of the IDS itself should be considered as it may become a new target of attack, including adversarial example injection, model extraction, and parameter tampering, which can degrade ML performance and leak sensitive information [6]. For protecting security-critical workloads in edge devices, Trusted Execution Environments (TEEs) offer a promising foundation. A TEE establishes an enclave that isolates sensitive applications from the untrusted operating system and applications in Rich Execution Environment (REE), ensuring confidentiality and integrity even in adversarial environments. However, the applicability of TEEs for intrusion detection on edge devices—specifically their ability to support lightweight, real-time, and secure IDS execution—remains an open question. This gap motivates our work.

**Contributions.** The goal of this work is to design and implement a lightweight, real-time, and secure IDS, *LiReS IDS*, that can be practically deployed on resource-constrained RISC-V edge devices. To this end, we implement the *LiReS IDS* on a real board. Figure 1 shows our design and implementation overview. We make the following contributions:

- **Genetic Algorithm-driven Pruning.** We introduce a genetic algorithm-driven structured pruning technique specifically tailored for CNN-based IDS in edge com-

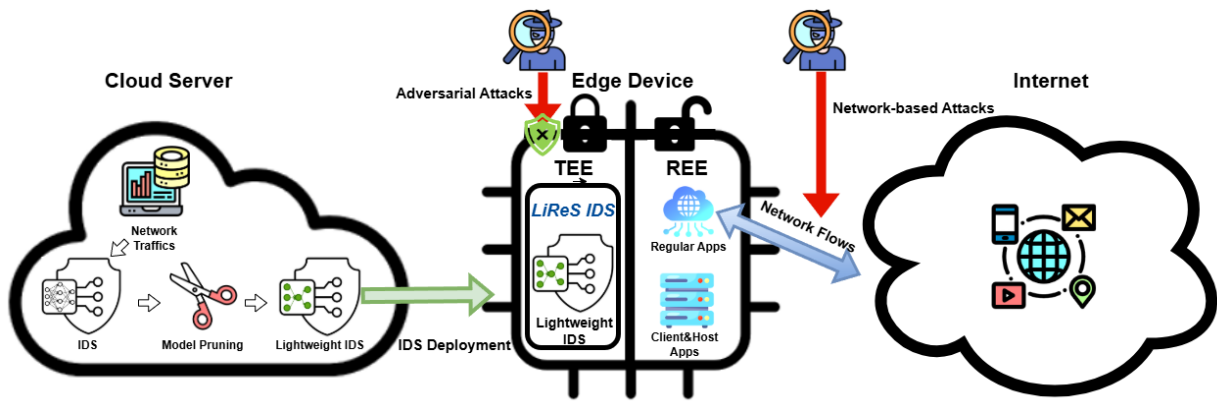


Fig. 1. Overview of the LiReS IDS workflow. The baseline CNN-based IDS is trained and pruned in the cloud to produce a lightweight model, which is then securely deployed on RISC-V edge devices. Within the edge device, the IDS runs inside Keystone TEEs to ensure confidentiality and integrity, while the REE hosts regular and client/host applications. The system protects against both network-based and adversarial attacks while processing real-time network flows.

puting environments. By searching for and removing redundant filters, our method compresses the IDS model by 80% while preserving detection accuracy.

- **TEE-Secured IDS Architecture.** We propose a novel TEE-secured IDS architecture. By isolating IDS and its inference processes within enclaves, the proposed IDS architecture provides confidentiality, integrity, and tamper-resistance against machine learning-specific threats.
- **Full-System Implementation on RISC-V Hardware.** We deploy and validate LiReS IDS on the SiFive HiFive Unmatched RISC-V development board. Our implementation demonstrates successful end-to-end integration of data processing, pruning, and achieving real-time detection performance on a physical edge computing platform.

## II. RELATED WORKS

With the rise of the Internet of Things (IoT) and edge computing, there is growing interest in deploying IDSs on edge devices. Recently, large language models (LLMs) have been explored for anomaly detection tasks [7]; however, their immense parameter size and computational demand make them unsuitable for resource-constrained edge devices. Machine learning and deep learning have been widely adopted for IDSs due to their ability to detect zero-day attacks beyond the capabilities of signature-based methods. Among these, deep convolutional neural networks (CNNs) have emerged as a particularly promising approach, as they can automatically learn and generalize from raw data features, thereby reducing the need for manual feature engineering. Despite these advantages, deep CNNs still demand substantial memory and computational resources, which presents significant challenges for deployment on IoT and edge platforms.

Existing approaches have sought to reduce IDS complexity by designing lightweight models using classical machine learning algorithms such as random forests, support vector machines, and k-nearest neighbors [5]. While these approaches achieve low overhead, they often sacrifice detection accuracy. Recent studies also have explored model compression tech-

niques for deep learning, including pruning, quantization, and knowledge distillation [8]. Among these, pruning is particularly appealing because it directly reduces the number of active parameters or filters, enabling faster inference and smaller memory footprints [9]. Nevertheless, existing pruning methods often depend on global pruning ratios and overlook not only the trade-off between accuracy and sparsity but also the need to maintain real-time inference performance.

Most current IDS deployments on edge devices primarily focus on reducing model size and computational cost, but they often overlook the confidentiality and integrity of the IDS itself. In typical edge IDS implementations, the trained ML-based IDS and its inference processes run entirely in the REE. Since the REE is untrusted and vulnerable to compromise, IDSs executed in this environment remains exposed to threats such as model extraction and parameter tampering. Although several approaches such as model encryption have been proposed to protect machine learning models, these techniques typically incur prohibitive overhead and runtime [10].

TEE provides a promising solution to this challenge. A TEE establishes a secure enclave within the processor, isolating sensitive computations from the untrusted REE and ensuring confidentiality and integrity even in the presence of a compromised operating system. Existing TEEs such as Intel SGX and ARM TrustZone are designed specifically for x86 and ARM architectures, and therefore cannot be directly applied to RISC-V platforms. Securing RISC-V edge devices has therefore become an urgent research need. In this context, Keystone, an open-source TEE for RISC-V, provides a suitable foundation for developing secure IDS on RISC-V edge devices.

## III. THREAT MODEL

We assume that a RISC-V-based edge device runs general-purpose applications inside the Rich Execution Environment (REE) and security-critical applications inside a Keystone Trusted Execution Environment (TEE). We assume the IDS is securely deployed after training in the cloud.

Our security objective is to protect the RISC-V-based edge device from network-based attacks, and to ensure the confidentiality and integrity of the deployed IDS model and its inference process. The IDS must process input network traffic securely inside the enclave, and generate alerts when malicious activity is detected. Even if the host operating system is compromised, the enclave must ensure that IDS model is not leaked or tampered with and that inference outputs reflect the true classification results.

We assume adversaries may gain full control over the REE by compromising the operating system, installing malicious software, or arbitrarily modifying any unprotected memory region. These adversaries can launch network-based attacks. To capture these behaviors comprehensively, we evaluate LiReS IDS using three widely adopted intrusion detection datasets that reflect real-world adversarial patterns:

- **NSL-KDD**: Includes DoS, Probe, R2L, and U2R.
- **UNSW-NB15**: Includes Fuzzers, Exploits, Generic, etc.
- **CICIDS-2018**: We select brute-force attack scenarios includes FTP\_BruteForce and SSH\_BruteForce.

#### IV. LIGHTWEIGHT MODEL DESIGN

This section first outlines the baseline IDS model design workflow in a cloud environment and then introduces a genetic algorithm-based structured pruning approach that creates a lightweight IDS model for edge deployment.

##### A. Baseline Model Design

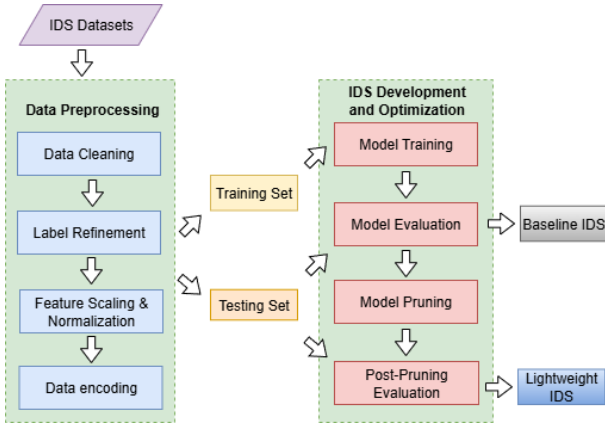


Fig. 2. IDS design workflow in the cloud environment. The process consists of two main stages: (1) data preprocessing, which includes data cleaning, label refinement, feature scaling, normalization, and encoding; and (2) IDS development and optimization, involving model training, evaluation, pruning, and post-pruning evaluation to produce the lightweight IDS for edge deployment.

We first design and train a baseline CNN-based IDS model in a cloud environment. Figure 2 illustrates the overall workflow. Our model design encompasses two stages: data preprocessing, and IDS development and optimization. We use three open-source intrusion detection datasets (NSL-KDD, UNSW-NB15, and CICIDS-2018) to train and evaluate the baseline model. Each dataset is first parsed into tabular form and cleaned to remove missing or corrupted entries. We then

discard redundant or constant-value attributes to eliminate non-informative features. Numerical features are standardized using z-score normalization:

$$z_i = \frac{x_i - \mu_i}{\sigma_i}, \quad (1)$$

where  $x_i$  is the raw feature value, and  $\mu_i$  and  $\sigma_i$  denote the mean and standard deviation of feature  $i$  across the training set. Each dataset defines attack categories differently (e.g., DoS, Probe, U2R, R2L). We use one-hot encoding that transforms discrete variables into fixed-length binary vectors:

$$O(c_i) = [0, 0, \dots, 1, \dots, 0], \quad (2)$$

with a 1 placed at the index corresponding to the category  $c_i$ .

The baseline IDS models adopt a deep CNN architecture. We train the model using the Adam optimizer with categorical cross-entropy loss. The trained baseline models serve as the foundation for subsequent pruning and hardware deployment.

##### B. Genetic Algorithm-driven Pruning

Deep CNN-based IDS models typically require substantial storage capacity, which makes them unsuitable for deployment on resource-constrained RISC-V edge devices and for execution within an enclave. To address this, we adopt structured pruning as a model compression strategy, and treat pruning as a discrete optimization problem: learning binary masks that determine which filters or neurons to keep. This formulation allows us to search directly for lightweight subnets that achieve a balance between accuracy and sparsity. In contrast, most prior methods fail to achieve this balance—unstructured pruning reduces computation but yields minimal memory savings. In addition, existing structured pruning methods remove entire filters or channels and can reduce model size, but they typically rely on fixed heuristics or global pruning ratios. In contrast, our discrete optimization formulation with a genetic algorithm (GA) systematically explores a broader combinatorial space of pruning configurations, and GA’s evolutionary operators include selection, crossover, and mutation, enabling adaptive refinement of pruning decisions, making our approach particularly well suited for identifying compact yet accurate subnetworks for edge deployment. Figure 3 illustrates the GA-Pruning workflow.

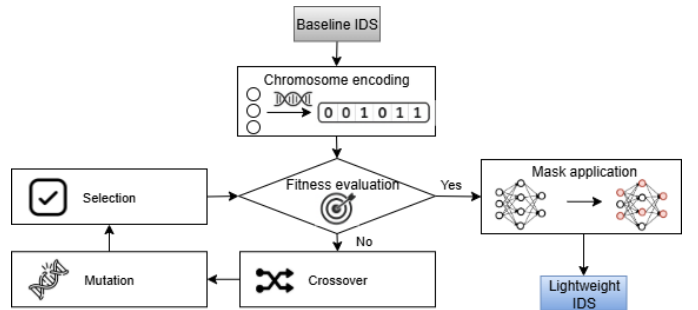


Fig. 3. GA-driven structured pruning tailored for CNN-based IDS.

*Chromosome encoding:* Each candidate solution (chromosome)  $\mathbf{c}$  is a tuple of binary masks

$$\mathbf{c} = (m^{(1)}, m^{(2)}, \dots, m^{(L)}),$$

where each  $m^{(\ell)} \in \{0, 1\}^{n_\ell}$  masks the  $n_\ell$  output filters or neurons of the  $\ell$ -th prunable layer in the network. A bit value 0 indicates pruning the corresponding unit; 1 keeps it active. We define the total number of prunable units as:

$$\gamma = \sum_{\ell=1}^L n_\ell$$

*Mask application:* Given a trained baseline model and chromosome  $\mathbf{c}$ , we clone the model and apply the masks  $\{m^{(\ell)}\}$  by zeroing (i) all kernel slices and bias entries of the pruned output channels in convolutional layers and (ii) the pruned output columns and biases in fully connected layers. The cloned model is compiled using categorical cross-entropy and Adam, and evaluated on a held-out validation set to obtain accuracy  $\text{acc}(\mathbf{c})$ .

*Objective and fitness:* Let  $\Delta(\mathbf{c}) = \sum_{\ell=1}^L \|m^{(\ell)}\|_0$  be the number of kept units. We define the retention ratio

$$r(\mathbf{c}) = \frac{\Delta(\mathbf{c})}{\gamma} \in [0, 1],$$

and its complement  $1 - r(\mathbf{c})$  as sparsity (fraction pruned). The scalar fitness combines accuracy and sparsity:

$$\text{fit}(\mathbf{c}) = w(1 - r(\mathbf{c})) + (1 - w)\text{acc}(\mathbf{c}), \quad (3)$$

with  $w = 0.1$ . The GA maximizes Eq. (3).

*Initialization:* We sample an initial population of  $P = 100$  chromosomes by drawing each mask bit from  $\{0, 1\}$  with equal probability.

*Selection:* We use Roulette-wheel (fitness-proportional) selection to sample parents according to normalized fitness scores within the current population.

*Crossover:* We perform single-point crossover with probability  $p_\times = 0.6$  on one randomly chosen mask among  $\{m^{(1)}, m^{(2)}, \dots, m^{(L)}\}$ . Given parents, we uniformly choose a cut point  $p \in \{1, \dots, n_\ell - 1\}$  for that mask length  $n_\ell$  and swap the leading segments to form two children; other masks are unchanged.

*Mutation:* We flip each bit in each child independently with probability  $p_\mu = 0.05$ .

*Replacement and termination:* Children are generated in pairs until a new population of size  $P$  is formed. We keep track of the best chromosome  $\mathbf{c}^*$  and best fitness per generation. The algorithm stops after at most 100 generations or earlier if the absolute improvement of the best fitness across consecutive generations falls below  $10^{-6}$ .

*Outputs:* At termination, we report (i) the best chromosome  $\mathbf{c}^*$ , including the number of active units in each layer,

$$(\|m^{(1)}\|_0, \|m^{(2)}\|_0, \dots, \|m^{(L)}\|_0),$$

(ii) the final validation accuracy  $\text{acc}(\mathbf{c}^*)$ , and (iii) the retention ratio  $r(\mathbf{c}^*)$  (with sparsity  $1 - r(\mathbf{c}^*)$ ). This procedure can build

a pruned CNN-based IDS model while preserving accuracy under the trade-off in Eq. (3).

The genetic algorithm parameters, including population size  $P$ , crossover probability  $p_\times$ , mutation rate  $p_\mu$ , and the fitness weight  $w$  was determined empirically through a parameter sweep. Specifically, we performed a grid search over ranges ( $P \in \{50, 100, 150\}$ ,  $p_\times \in \{0.4, 0.6, 0.8\}$ ,  $p_\mu \in \{0.01, 0.05, 0.1\}$ ,  $w \in \{0.05, 0.1, 0.2\}$ ) to balance pruning ratio and accuracy. The final configuration ( $P = 100$ ,  $p_\times = 0.6$ ,  $p_\mu = 0.05$ ,  $w = 0.1$ ) achieved the most stable convergence and best trade-off between sparsity and accuracy.

## V. SYSTEM ARCHITECTURE

This section presents the TEE-secured IDS architecture, which consists of three primary components: a client application for data ingestion and preprocessing, a host application for enclave management and communication, and an enclave application implementing IDS inference functionality. Figure 4 shows our design overview.

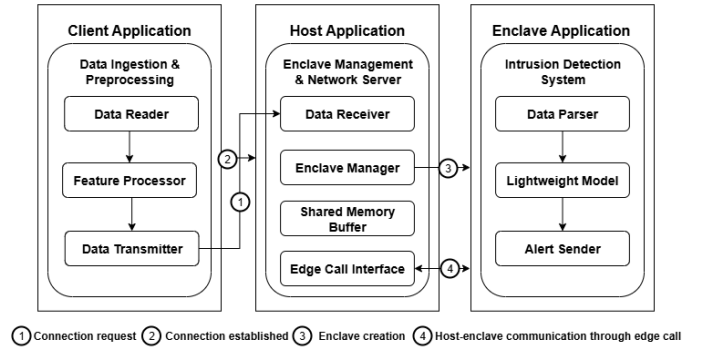


Fig. 4. Overview of the TEE-secured IDS architecture. The system consists of three main components: the client application for data ingestion and preprocessing, the host application for enclave management and communication, and the enclave application that performs IDS inference.

### A. Client Application

The client application acts as the entry point for the LiReS IDS. The client application is responsible for ingesting raw or pre-collected network traffic, applying preprocessing and feature extraction, and securely transmitting processed inputs to the host application. In the real deployment scenario, the client can either read traffic data from live network interfaces (e.g., packet capture tools) or load pre-recorded datasets.

To ensure compatibility with the CNN model deployed inside the enclave, the client applies the same preprocessing schema defined during model training. This alignment guarantees that the feature vectors generated during inference match those used during training, preserving the consistency and accuracy of the model.

Once preprocessing is completed, the client establishes a secure TCP connection to the host application on a designated port. We implement a two-phase communication protocol: the client first sends the size of the feature vector as a string, followed by the vector itself. This approach enables the host

to dynamically allocate buffers and efficiently handle inputs of varying size.

### B. Host Application

The host application acts as the intermediary between the untrusted client and the trusted enclave, coordinating enclave lifecycle management and data exchange. The host listens on the client port, accepts client connections, and processes requests.

The host application is also responsible for enclave lifecycle management. When a new session is initiated, the host creates a Keystone enclave with memory settings tailored for intrusion detection workloads.

The host bridges communication between the client and the enclave through Keystone’s edge-call interface. Incoming data is copied to a shared memory buffer accessible to the enclave, while using Keystone’s OCALL mechanisms for communication, allowing the enclave to request additional resources or return inference results.

### C. Enclave Application

The enclave application is the core of LiReS IDS, implementing CNN-based IDS within the Keystone trusted execution environment. This design ensures confidentiality and integrity for the model parameters, architectures, and inference operations, protecting them from ML-specific threats.

Upon receiving input from the host, the enclave executes a feature parsing routine, performing bounds checking and input validation to mitigate risks of buffer overflows or malformed data. The IDS performs model inference to determine whether a sample represents benign or malicious activity. Results are then securely transmitted back to the host using Keystone’s edge call interface. The enclave design enforces the minimal trusted computing base (TCB) principles, confining only critical functionality — data parsing, model inference, and result generation — inside the enclave.

### D. Integration of Enclave Cache

Keystone implements integrity protection when launching an enclave: the Security Monitor (SM) computes a cryptographic hash over the entire statically linked enclave binary for essential attestation purposes. This process significantly increases the enclave startup latency, particularly for large binaries such as ML models. In practice, the startup time dominates the total latency when enclaves are created per request, making real-time IDS deployment infeasible.

To address this, we adopt the enclave application cache (ecache) mechanism [11], which securely stores previously measured enclave binaries in a PMP-protected memory region. On a cache hit, Keystone can launch a new enclave instance by copying the cached image without repeating the expensive measurement step. This design preserves the security guarantees of attestation while reducing startup latency. The integration of the enclave cache enables LiReS IDS to achieve real-time intrusion detection.

Component	Specification
SoC	SiFive Freedom U740
CPU	4× U74 cores 1× S7 core
ISA	RV64GC (U74) RV64IMAC (S7)
Frequency	1.2 GHz
Memory	16 GB DDR4

(a) HiFive Unmatched Hardware Specifications



(b) HiFive Unmatched Board

Fig. 5. HiFive Unmatched hardware.

TABLE I  
COMPARISON BEFORE AND AFTER GA-BASED PRUNING.

Dataset	Model Size (KB)	Reduction	Accuracy (%)
NSL-KDD	354.7 → 71.2	-80.0 %	99.4 → 98.1
UNSW-NB15	365.7 → 74.0	-79.8 %	94.1 → 93.0
CICIDS-2018	443.5 → 89.1	-80.0 %	97.8 → 96.5

## VI. EXPERIMENTAL EVALUATION

This section first defines the experimental setup and metrics. We then present results on detection accuracy, model size, and inference efficiency on RISC-V hardware. We implemented and evaluated the proposed LiReS IDS on the SiFive HiFive Unmatched RISC-V board. Figure 5 summarizes its key specifications. This board can be applied in real-world contexts such as edge computing prototypes and operating system development.

### A. Detection Accuracy

As summarized in Table I, the proposed GA-based structured pruning compresses the CNN-based IDS by 80% on all three benchmarks while preserving high detection accuracy. While some deep learning-based methods achieve very high accuracy on cloud server or high-performance processor setups, such solutions depend on heavyweight models and long runtimes that are impractical for resource-constrained RISC-V edge devices. Our comparisons focus on IDSs designed for IoT devices, as summarized in Table II. LiReS IDS achieves higher detection accuracy than other IDS designed for IoT devices. On NSL-KDD, LiReS IDS achieves 98.1% accuracy, closely matching the prior method by Marir et al. [12] who leveraged deep feature extraction and multi-layer ensemble support vector machines which are unsuitable for deployment on edge devices, and is significantly higher than other lightweight approaches such as Kunang et al. [13]. On UNSW-NB15, LiReS IDS achieves the highest accuracy at 93.0%, which is better than Zhang et al. [14] and Peng et al. [15] by 4.1% and 10.7%, respectively. For the CICIDS-2018 dataset, LiReS IDS achieves 96.5% accuracy, outperforming other lightweight IDS models including Lin et al. [16] and Kunang et al. [13].

TABLE II  
ACCURACY (%) COMPARISON.

Dataset	Method	Accuracy (%)
NSL-KDD	Marir et al. [12]	<b>98.6</b>
	Kunang et al. [13]	86.0
	<b>LiReS IDS</b>	98.1
UNSW-NB15	Zhang et al. [14]	88.9
	Peng et al. [15]	82.3
	<b>LiReS IDS</b>	<b>93.0</b>
CICIDS-2018	Lin et al. [16]	96.2
	Kunang et al. [13]	95.4
	<b>LiReS IDS</b>	<b>96.5</b>

TABLE III  
INFERENCE TIME COMPARISON.

Author	Platform	Latency
Musthafa et al. [17]	Raspberry Pi 3B+	226.2 ms
Kong et al. [18]	NVIDIA Jetson AGX Xavier	82 ms
Chithra Rani et al. [19]	NVIDIA Jetson Nano	568 ms
<b>LiReS IDS</b>	RISC-V HiFive Unmatched	<b>50 ms</b>

### B. Inference Efficiency

We further evaluated the inference efficiency of the LiReS IDS to demonstrate its real-time detection capability. The experiment measures the latency of the TEE-secured IDS under continuous client–enclave communication. Each batch of 20 records is sent from the client to the host and then to the enclave for classification. We first measured the average latency per traffic sample, which was 19 ms. To emulate a stable edge monitoring scenario and maintain synchronization between client data streams and enclave inference cycles, the system was configured to process one batch per second. This setting balances throughput and response time, allowing the system to operate without congestion. As a result, the end-to-end detection latency is 50 ms, corresponding to real-time detection performance in edge environments.

Table III compares the inference performance of LiReS IDS with other real-time IDS deployed on different edge platforms. Musthafa et al. [17] applied model pruning and quantization to reduce the memory footprint of a deep learning–based IDS, reporting a latency of 226.2 ms on a Raspberry Pi 3B+. Kong et al. [18] proposed a lightweight IDS optimized for edge devices, achieving 82 ms latency on the NVIDIA Jetson AGX Xavier. Chithra Rani et al. [19] proposed a deep learning–based IDS, obtaining 568 ms latency on the NVIDIA Jetson Nano. LiReS IDS achieves an inference latency of 50 ms on the HiFive Unmatched RISC-V board, demonstrating that strong security guarantees can be provided without sacrificing real-time performance.

## VII. CONCLUSION

In this paper, we introduce LiReS IDS, a lightweight, real-time, and secure intrusion detection system designed for RISC-V edge devices. Our pruning method reduces the CNN-based IDS size by up to 80% with minimal accuracy loss. We designed a novel TEE-secured IDS architecture to ensure the confidentiality and integrity of the IDS. The implementation and evaluation on the HiFive Unmatched RISC-V board

demonstrated the real-time detection capability and practical feasibility of LiReS IDS. In future work, we plan to extend LiReS IDS to support federated learning–based collaborative intrusion detection and dynamic model updates within TEE.

## ACKNOWLEDGMENTS

This work is supported by JSPS KAKENHI Grant Numbers JP23H00464, JP24KK0184 and JP25K03091, and MEXT X-NICS Grant Number JPJ011438, Japan, as well as CyberExcellence, Walloon Region, Belgium. The first author is also supported by the Institute of Science Tokyo TSUBAME Scholarship.

## REFERENCES

- [1] Z. Zhou *et al.*, “Edge intelligence: Paving the last mile of artificial intelligence with edge computing,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.
- [2] W. Shi *et al.*, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [3] K. Asanović and D. A. Patterson, “Instruction sets should be free: The case for risc-v,” *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.
- [4] M. Eskandari *et al.*, “Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices,” *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6882–6897, 2020.
- [5] Q. Zeng and Y. Hara-Azumi, “Hardware/software codesign of real-time intrusion detection system for internet of things devices,” *IEEE Internet of Things Journal*, vol. 11, no. 12, pp. 22351–22363, 2024.
- [6] J. Wang *et al.*, “Def-ids: An ensemble defense mechanism against adversarial attacks for deep learning-based network intrusion detection,” in *International Conference on Computer Communications and Networks*, pp. 1–9, 2021.
- [7] H. Breniaux and D. Mouheb, “Towards context-aware log anomaly detection using fine-tuned large language models,” in *European Symposium on Research in Computer Security*, pp. 83–102, 2025.
- [8] M. Lei *et al.*, “P-dnn: An effective intrusion detection method based on pruning deep neural network,” in *International Joint Conference on Neural Networks*, pp. 1–9, 2020.
- [9] J. Tmamna *et al.*, “Pruning deep neural networks for green energy-efficient models: a survey,” *Cognitive Computation*, vol. 16, pp. 2931–2952, 2024.
- [10] Z. Sun *et al.*, “Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps,” in *USENIX Security Symposium*, pp. 1955–1972, 2021.
- [11] T. Umezawa, A. Saiki, and K. Kimura, “Enclave Application Cache for RISC-V Keystone,” in *European Symposium on Security and Privacy Workshops*, pp. 422–428, 2025.
- [12] N. Marir *et al.*, “Distributed abnormal behavior detection approach based on deep belief network and ensemble svm using spark,” *IEEE Access*, vol. 6, pp. 59657–59671, 2018.
- [13] Y. N. Kunang *et al.*, “Attack classification of an intrusion detection system using deep learning and hyperparameter optimization,” *Journal of Information Security and Applications*, vol. 58, p. 102804, 2021.
- [14] H. Zhang *et al.*, “Multi-dimensional feature fusion and stacking ensemble mechanism for network intrusion detection,” *Future Generation Computer Systems*, vol. 122, pp. 130–143, 2021.
- [15] P. Wei *et al.*, “An optimization method for intrusion detection classification model based on deep belief network,” *IEEE Access*, vol. 7, pp. 87593–87605, 2019.
- [16] P. Lin, K. Ye, and C.-Z. Xu, “Dynamic network anomaly detection system by using deep learning techniques,” in *Cloud Computing*, pp. 161–176, Springer, 2019.
- [17] M. B. Musthafa *et al.*, “Optimized ensemble deep learning for real-time intrusion detection on resource-constrained raspberry pi devices,” *IEEE Access*, vol. 13, pp. 113544–113556, 2025.
- [18] X. Kong *et al.*, “idetector: A novel real-time intrusion detection solution for iot networks,” *IEEE Internet of Things Journal*, vol. 11, no. 19, pp. 31153–31166, 2024.
- [19] P. R. Chithra Rani and K. Baalaji, “Deep learning-based ensemble stacking for enhanced intrusion detection in iot-edge platforms,” *Discover Applied Sciences*, vol. 7, p. 928, 2025.