



Article

A deep learning-driven adaptive edge intelligence framework for real-time Internet of Things applications

Majd S. Ahmed*

Computer Science Department, College of Science, Mustansiriyah University, Baghdad, Iraq

ARTICLE INFO

Article history:

Received 05 May 2026

Received in revised form

20 September 2026

Accepted 09 October 2026

Keywords:

Edge computing, Deep learning inference, Adaptive task offloading, IoT analytics, Lightweight neural networks, Hierarchical edge-fog-cloud architecture

*Corresponding author

Email address:

majdsami@uomustansiriyah.edu.iq

DOI: 10.55670/fpll.futech.5.4.26

ABSTRACT

The rise of Internet of Things (IoT) ecosystems and the New Paradigm of low-latency edge data processing. IoT ecosystems are growing exponentially, and this growth brings a new challenge: low-latency data processing at the network edge. Because cloud-based architectures cannot meet this real-time requirement due to network delay, connectivity issues, and increasing bandwidth contention, deploying deep learning-based, high-computational-load pattern detection on high-dimensional streaming IoT data becomes challenging. The proposed framework adapts the model's computational complexity and inference location in response to system state to optimize latency and energy while retaining high predictive accuracy. Experimental evaluation on the NSL-KDD intrusion detection dataset shows high accuracy, with class-wise accuracy of 0.979, precision of 0.981, recall of 0.977, F1-score of 0.979, and a precise ROC area under the curve of 0.998. Comparisons with simulation results provide useful insights into the relative advantages of local edge inference over end-to-end cloud inference, yielding 98% energy savings and a 97.6% latency reduction. This work reports a novel context-aware integrated framework that combines a lightweight deep learning classifier, adaptive task offloading using a Deep Q-Learning-based intelligent agent, hierarchical workload distribution at edge-fog-cloud levels, and proactive contextual information dissemination in real time, in order to ensure a context-aware system with 58.5%, 29.0%, and 12.5% edge, fog, and cloud-based decisions, respectively.

1. Introduction

IoT has grown exponentially, generating unprecedented volumes of data from interconnected sensors and smart systems across industries. This data supports a range of functions such as monitoring, anomaly detection, home monitoring, and preventive maintenance. These functions often depend on real-time or near-real-time operations [1-3]. Because of network round-trip delays, connectivity and bandwidth variations, processing delays, and limited processing power, conventional cloud infrastructure has struggled to run these applications, which often require low latency. Edge computing offers an elegant solution by moving computing power closer to, or to, the data source. A globally distributed set of edge devices can run analytics, inference, and other tasks locally, reducing internet bandwidth load and latency while reducing dependence on cloud computing [4, 5]. Such a topology becomes critical in advanced environments that require rapid, context-aware decisions, such as industrial automation, smart citizen communities, and remote health services. With excessive model parameters, most existing

deep learning (DL) models (i.e., neural networks) struggle to detect critical patterns in high-dimensional, noisy IoT data streams for sophisticated applications such as anomaly detection, prediction, activity recognition, and classification. Existing DL models cannot run efficiently on edge nodes because many powerful neural networks are computationally demanding and require large memory footprints that most edge devices cannot provide [6,7]. Some recent research has addressed this gap by exploring lightweight DL models or adaptive offloading approaches that partition inference computation across cloud and edge tiers or deeper into the edge infrastructure. Hierarchical edge-cloud architectures enable dynamic task and model-complexity partitioning based on device, application needs, and network conditions [8,9]. Work in TinyML and quantized neural networks has also begun to enable inference directly on microcontrollers with limited power and memory [10]. However, the outcomes are not yet satisfactory. Most existing solutions are static and lack adaptability as workloads and resource availability change. Moreover, focusing solely on latency minimization as

the only metric can negatively affect other metrics, such as accuracy and energy, which are also critical in real-time edge IoT applications. Guaranteeing end-to-end latency in complex, (highly) heterogeneous edge, fog, and cloud deployments requires an integrated solution for task scheduling, resource utilization, and model compression, which remains unexplored in current solutions.

This work's major contribution is to demonstrate the design and prototype validation of the Deep Learning-Driven Adaptive Edge Intelligence Framework (DLA-EIF) for real-time IoT applications. DLA-EIF's novelty lies in the synergistic design and operation of four tightly coupled modules that avoid gaps in existing state-of-the-art frameworks. First, the lightweight deep learning component comprises advanced compression techniques- pruning, quantization, knowledge distillation- using dynamic switching of model variants based on the real-time load on edge and device resources, which, in the absence of human intervention, attains the best accuracy-efficiency compromise. Second, the adaptive task offloading component uses a reinforcement learning-based Deep Q-Learning (DQL) agent to identify the optimal offloading policy by simultaneously considering processing power, power consumption, bandwidth, priority, and maximum latency, and dynamically adapting to the changing environment. Third, the hierarchical structure of the edge-fog-cloud processing and transmission infrastructure for workload distribution across tiers as per processing demand, with inferencing in latency-critical applications at the edge, contextual analytics at the fog, and global model learning at the cloud; and fourth, the real-time context monitoring system that continuously observes throughput, jitter in the network, power draw, operating temperature, and accuracy changes across modules and components within the framework.

The DLA-EIF differs from conventional static schemes by bundling reinforcement learning-based, multi-objective optimization of inference accuracy, inference latency, and power efficiency. Contrary to the conventional schemes that merely have fixed playbook-based models or offloading strategies, are non-adaptive to changing environments, and are not autonomous, our dynamic scheme intelligently (autonomously) adjusts its model capacity, re-allocates the compute load across tiers, and optimizes the local and global inference paths all in an adaptive, rule-based, untethered manner. Using the deadline-aware scheduler (Algorithm 1, Section 3.5), the framework can meet sub-50 ms latency for real-time inference requests by escalating to the fog and cloud when edge slack is negative. Section 5.7.3 maps the framework's application-specific applicability to concrete latency and energy criteria based on the latency and energy ranges observed in our experiments.

A comprehensive test-based validation of the proposed framework on the NSL-KDD intrusion detection dataset demonstrates resilient classification accuracy, consistent training behavior, and high evaluation metrics. The distribution of evaluation error in the results verifies the estimated reliability. The simulation of inference at the edge or cloud reveals that edge inference helps significantly reduce latency to the tune of 1.2 ms compared to 50-150 ms for cloud inference, while also drastically reducing energy expenditure to the tune of 0.001 J compared to 0.05 J in the case of cloud inference. Prediction accuracy is the same across all deployments. Overall, the proposed architecture offers a viable foundation for a future intelligent IoT framework that delivers effective, low-latency analytics, supported by green energy, at an elastic energy cost. In response to the above gap, this work aims to: (1) Design a lightweight deep learning

classifier, able to handle tabular IoT flow data, and compare its accuracy-efficiency trade-off with learning baselines (Random Forest, SVM, XGBoost); (2) Formulate an adaptive offloading policy as a Markov Decision Process, with a continuous 10-dimensional state space, three discrete actions, and a combination of latency, energy, deadline violation, and accuracy in the reward, and train it with a Deep Q-Learning agent; (3) Choose a hierarchical edge-fog-cloud placement setup, with a deadline-aware scheduling mechanism, and a system-level tier profile describing hardware class, tasks, communication pattern, and fallback configuration; (4) Demonstrate the entire framework on two intrusion-detection benchmark datasets (NSL-KDD, CIC-IDS2017), both with simulation and hardware-in-the-loop settings, and translate the latency/energy distributions to real application metrics.

2. Literature review

The works in deploying deep learning at the network edge for IoT have been converging on four largely independent tracks: (i) designing power-efficient neural network architectures to run on edge devices with limited resources, (ii) utilizing split or early-exit inference to partition the computation between the end nodes and the cloud server, (iii) provisioning the computation hierarchically among edge-fog-cloud tiers, and (iv) offloading based on reinforcement learning to deploy inference at runtime. This section discusses these works in turn and then spots the gap that DLA-EIF, the proposed framework, fills.

2.1 Lightweight models for edge inference

The first class of work aims to shrink the size and computational requirements of deep neural networks so their inference can run on edge devices. Warden and Situnayake [10] surveyed TinyML and on-device inference and showed that quantized and pruned models could run on microcontroller-class hardware with kilobytes of memory, albeit with a lower capacity for complex vision tasks. Howard et al. [11] designed MobileNets, a depth-wise separable convolutional network architecture that greatly reduces CNN computational cost while preserving accuracy on mobile and embedded vision tasks. Sandler et al. [12] extended this structure with MobileNetV2, which uses inverted residual blocks and linear bottlenecks to add accuracy at similar cost. Ma et al. [13] introduced ShuffleNetV2, which laid out practical guidelines for efficient CNN design based on channel shuffling and memory-access cost rather than FLOPS alone. Lane et al. [14] characterized the first resource-awareness of deep learning on wearables, smartphones, and IoT devices, noting that on-device inference suffers under dynamic workloads, which motivates adaptive models instead of static configurations.

2.2 Split and early-exit inference

Another family of approaches does not run the whole model on a single device; instead, it splits inference between the device and the cloud. Shi et al. [15] gave a broad definition of edge computing as a methodology for pushing computation from IoT to the cloud and proposed split-inference systems to overcome resource heterogeneity and limited programmability. Ren et al. [16] surveyed collaborative deep neural network (DNN) inference in edge intelligence, describing different approaches to splitting, offloading, and merging computation between devices and servers. Teerapittayanon et al. [17] introduced Branchy Net, an early-exit topology that lets easy samples be processed by shallow network segments on the device and offloads difficult ones to

a server, reducing latency while balancing accuracy and efficiency under different network conditions. Premasankar et al. [18] analyzed IoT in the edge computing paradigm through a case study, showing how network outages affect the reliability of split-inference systems.

2.3 Hierarchical Edge-Fog-Cloud orchestration

A third group of research results provides for a distribution of work from two points of entry to multiple ones. Satyanarayanan [19] reported that edge computing has been emerging and positioned fog nodes close to data sources as efficient for latency-sensitive workloads. Deng et al. [20] recommended a multi-tier collaborative learning approach in which models are split and transferred among edge, fog, and cloud nodes to improve global inference time. Mao et al. [21] introduced a general hybrid computation-communication optimization framework for task offloading in heterogeneous edge networks, showing that hierarchical orchestration can substantially mitigate network overloads for dense IoT workloads. In contrast, Varghese and Buyya [22] found that many hierarchical architectures lack sufficiently mature context-awareness mechanisms to respond to workload variations over time, making them infeasible for anomaly detection and surveillance workloads.

2.4 Reinforcement-learning-based offloading

Fourth, a line of research considers using Reinforcement Learning (RL) to decide where inference should run. Iftikhar et al. [23] provide a systematic survey of AI-based fog and edge computing and list RL-based offloading policies, including information about their state space, reward function, and action space. Their taxonomy shows that most RL offloaders optimize a single metric (e.g., latency) and pair it with static or hand-tuned reward weights, effectively trading off latency for energy and cost while missing other dimensions (accuracy and deadlines). Additionally, many RL offloaders operate in low-dimensional discrete state spaces, which makes them ill-suited to the continuous, multi-dimensional context signals found in modern IoT deployments.

2.5 Gap statement

Across these four works, the three existing approaches have three major limitations. (i) Model compression is only done before deployment; then the compressed model is fixed without re-compensation for any change in the load on devices or changes in the network [10-14]. (ii) Offloading is static or based on some heuristic rule, or the RL agent only optimizes for a single metric [16-18, 23]. (iii) All hierarchical orchestration policies are fixed at design time, without being sensitive to runtime context such as device temperature, jitter, and inference-accuracy drift [19-22]. To the best of our knowledge, no existing work adapts (i) model complexity, (ii) offloading venue, and (iii) tier placement jointly, under a single real-time context monitor. The proposed Deep Learning-Driven Adaptive Edge Intelligence Framework (DLA-EIF) couples a lightweight deep learning module, a Deep Q-Learning (DQL) offloading agent that optimizes for a weighted reward combining latency, energy, deadline violation, and accuracy, a hierarchical edge-fog-cloud placement policy, and a real-time context monitor that monitors the input for all three modules. Sections 3 and 5 describe the framework and its experimental validation, respectively.

3. Methodology

This paper presents the Deep Learning-Driven Adaptive Edge Intelligence Framework (DLA-EIF) for efficient, real-time deep learning inference on heterogeneous IoT systems. The architecture comprises four components that work in concert to deliver speed, computational efficiency, and workload adaptability.

3.1 Lightweight deep learning module

A small MLP for representing a lightweight module. For tabular flow features, it uses 2 hidden layers with 128 and 64 units, ReLU activation, batch normalization, dropout (rate 0.3), and a sigmoid output for binary classification. The module structure supports model compression methods (pruning, quantization, knowledge distillation) and lightweight CNN backbones (MobileNet, EfficientNet-Lite, and others) for image or spectral modalities, but in this work, we use only the MLP on the NSL-KDD tabular corpus. Object-detection backbones like Tiny-YOLO cannot be used on 41-length flow records and are not implemented here; a dynamic model-switching mechanism is mentioned as a feature but is left to future work on multimodal data (Section 5.8).

Model Formulation:

The forward pass of the MLP is defined as:

$$h_1 = \text{ReLU}(\text{BN}(W_1x + b_1)) \quad (1)$$

$$h_2 = \text{ReLU}(\text{BN}(W_2h_1 + b_2)) \quad (2)$$

$$\hat{y} = \sigma(W_3h_2 + b_3) \quad (3)$$

where $x \in \mathbb{R}^{41}$ is the input feature vector, $W_1 \in \mathbb{R}^{128 \times 41}$, $W_2 \in \mathbb{R}^{64 \times 128}$, $W_3 \in \mathbb{R}^{1 \times 64}$ are the weight matrices, b_1, b_2, b_3 are the bias vectors, BN denotes batch normalization, ReLU is the rectified linear unit, and σ is the sigmoid activation. The binary cross-entropy loss is:

$$L = -(1/N) \sum_{i=1}^N [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \quad (4)$$

where N is the batch size, $y_i \in \{0, 1\}$ is the ground-truth label, and $\hat{y}_i \in (0, 1)$ is the predicted probability. Optimization uses the Adam optimizer with learning rate 0.001, batch size 128, and 20 epochs. Z-score normalization is applied to all numeric inputs using statistics computed on the training split only: (5) $x' = (x - \mu) / \sigma$, where μ and σ are the training-set mean and standard deviation.

3.2 Adaptive task offloading module

An adaptive task offloading module determines where to execute a specific inference task (at the edge, neighboring fog, or cloud). This offloading decision considers CPU/GPU load, power level, available network bandwidth, task priority, and the inference deadline. We employ a Deep Q-learning (DQN) agent to learn the offloading policy online in real time. It should be explicitly clarified that the offloading decision is solely given by the DQN agent introduced in Section 5.5, and there is no fuzzy inference system.

MDP formulation: The offloading problem is formulated as a Markov Decision Process (S, A, R, γ), where:

$$\text{State: } s_t = [u_{\text{CPU}}, u_{\text{GPU}}, u_{\text{RAM}}, b, \text{BW}, j, p, d, \tau, a] \in \mathbb{R}^{10} \quad (5)$$

where $u_{\text{CPU}}, u_{\text{GPU}}, u_{\text{RAM}} \in [0, 1]$ are CPU, GPU, and RAM utilization; $b \in [0, 1]$ is battery level; $\text{BW} \in [0, 1]$ is normalized available bandwidth; $j \in [0, 1]$ is network jitter; $p \in [0, 1]$ is task priority; $d \in \mathbb{R}^+$ is the inference deadline (ms); $\tau \in \{\text{edge, fog, cloud}\}$ is the current tier; and $a \in [0, 1]$ is recent inference accuracy.

$$\text{Action: } a_t \in \{\text{edge, fog, cloud}\} \quad (6)$$

$$\text{Reward: } r_t = -(w_1 \cdot L_t + w_2 \cdot E_t + w_3 \cdot D_t - w_4 \cdot A_t) \quad (7)$$

where L_t is inference latency (ms), E_t is energy (J), $D_t \in \{0, 1\}$ is a deadline-violation indicator (1 if the deadline is exceeded, 0 otherwise), $A_t \in [0, 1]$ is classification accuracy, and $w_1 = 0.4$, $w_2 = 0.2$, $w_3 = 0.3$, $w_4 = 0.1$ are objective weights chosen to prioritize latency while still penalizing deadline violations and rewarding accuracy.

$$\text{Q-update (Bellman): } Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (8)$$

$$\text{DQN loss: } L(\theta) = E[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \quad (9)$$

where θ are the online Q-network parameters, θ^- are the target-network parameters (updated every 200 steps), α is the learning rate, and γ is the discount factor. The Q-network is a 3-layer MLP (64-64-3) with ReLU activations, trained with Adam at $\text{lr} = 1 \times 10^{-3}$. Exploration uses an ϵ -greedy schedule with ϵ decaying linearly from 1.0 to 0.05 over 5,000 episodes. The Deep Q-Learning hyperparameters are as follows. The state dimension is 10. The action space consists of three choices: edge, fog, and cloud. The Q-network architecture is a 3-layer MLP with a 64-64-3 configuration. The activation function is ReLU. The optimizer is Adam. The learning rate is 1×10^{-3} . The discount factor γ is 0.95. The replay buffer size is 10,000. The target network update occurs every 200 steps. Exploration uses an ϵ -greedy schedule in which ϵ decays from 1.0 to 0.05 over 5,000 episodes. Training runs for 5,000 episodes. Each episode consists of 50 steps. The batch size is 64. The reward weights are $w_1 = 0.4$ for latency, $w_2 = 0.2$ for energy, $w_3 = 0.3$ for deadline, and $w_4 = 0.1$ for accuracy.

3.3 Hierarchical processing architecture (Edge-Fog-Cloud)

DLA-EIF performs the computation tasks with the following distribution for every stage of processing: the edge layer performs feature extraction and edge inference at the edge node for the fast detection of high-priority events; the fog layer gathers and collects data for a number of edge nodes, batch analytics, and intermediate-level processing for contextual inference; the cloud layer manages the big data, global training, and optimization of the models.

3.4 Real-time context monitor

The real-time context monitor measures environmental and system parameters such as data input rate, network jitter, power usage, device temperature, and changes in inference accuracy. This dynamic adaptation module adjusts model complexity and the offloading policy in response to current workload and available resources. Collectively, the modules form a robust, flexible architecture for latency-critical, highly time-varying IoT workloads constrained by limited resources, enabling accurate, rapid deep learning inference across the edge-fog-cloud continuum.

3.5 System-level specification

It consists of a three-tiered system: each with a unique hardware class, task profile, messaging system, and fallback behavior. The edge with Cortex-M, Raspberry Pi, and Jetson-class hardware performs feature extraction and inference, communicates via MQTT to fog, and falls back to fog when under pressure; Fog with edge server/small cluster: performs aggregation and batch analytics, communicates via gRPC to the cloud, falls back to cloud when under pressure; and the cloud datacenter GPUs: perform training and big data, batch synchronization, and do not have a fallback tier.

Algorithm 1. Deadline-aware scheduling

```

Input: task T with deadline d
      expected_latency(tier) for tier ∈ {edge, fog, cloud}
1: tier ← edge
2: slack ← d - expected_latency(tier)
3: if slack < 0 then
4:   tier ← fog
5:   slack ← d - expected_latency(tier)
6:   if slack < 0 then
7:     tier ← cloud
8:   end if
9: else if slack > 2 × expected_latency(tier) then
10:  tier ← edge
11: end if
12: execute T at tier
13: log decision for DQN replay
14: return tier

```

Simulation setup: The hierarchical simulation environment models three levels with the following parameters. Inference time at the edge, fog, and cloud tiers in our simulations are set as described by the analytical model of Section 5.4.2 as 1.2, 10, and (50 - 150) ms, respectively, so that the network round-trip variability can be incorporated into the cloud-tier inference time. Communication energy is given by $E_{\text{comm}}(\mu, B) = (e_{\text{TX}} \cdot B_{\text{TX}}) + (e_{\text{RX}} \cdot B_{\text{RX}})$, where $e_{\text{TX}} = 5$ nJ/bit and $e_{\text{RX}} = 3$ nJ/bit (Wi-Fi class). TDP of the device at the edge, fog, and cloud tiers are 5, 20, and 100 W, respectively, with shared-VM allocation to the cloud tier of 1/8. The duty cycle is set to 100% for each tier. The communication protocols are MQTT (edge to fog), gRPC (fog to cloud), and batch sync (cloud to cloud storage). Our fallback tier is the fog for the edge tier, and the cloud for the fog tier, with the cloud tier having no fallback. The DQN agent interacts with this environment for 5,000 episodes of 50 steps each, sampling CPU, GPU, RAM, battery, bandwidth, jitter, priority, and deadline uniformly from the specified ranges at each step.

4. Dataset (NSL-KDD dataset)

Training setup: Adam ($\text{lr}=0.001$), batch size 128, 20 epochs. All these results are given in Section 5. Deep neural network architecture training and testing: We used the NSL-KDD dataset to train and test our proposed deep neural network model. NSL-KDD dataset is one of the most popular benchmark datasets in intrusion detection and was used in the TSAR project. Although most attack data are simulated, the network traffic patterns in the NSL-KDD dataset closely resemble real-world traffic. NSL-KDD dataset contains 22 attack types in the training set and 24 attack types in the testing set. Furthermore, both the training and testing sets include 5 normal data types. More normal data types in network traffic support more flexible data classification. The NSL-KDD dataset has several benefits over the KDD Cup 1999 dataset.

First, this dataset eliminates redundant records.

Second, the dataset still contains records with a low detection rate, even though we didn't neglect those attack types because they are prevalent on the Internet.

Third, the class distribution is more even, and the network traffic patterns are more representative of real-world traffic. As a result, the NSL-KDD dataset is a more appropriate benchmark dataset for intrusion detection.

Preprocessing: Three categorical attributes- protocol_type (3 values), service (70 values), and flag (11 values)- were

retained and one-hot encoded, yielding 122 input dimensions for the full-feature variant. A numeric-only variant (41 features) was also built for ablation. All numeric features were standardized using z-score normalization ($\mu=0$, $\sigma=1$) computed on the training split only. The exact feature list is given in the [Appendix](#).

The network architecture is an MLP with two hidden layers containing 128 and 64 units. It was trained on the official NSL-KDD split: KDDTrain+ (125,973 records) for training and KDDTest+ (22,544 records) for testing. Optimization was carried out using a gradient descent Adam optimizer ($\text{lr}=0.001$) with a batch size of 128 for 20 epochs with rectified linear units for the activation functions.

The proximity of the validation accuracy curves indicated a very strong relationship with the training accuracy curve and no clearly obvious overfitting trends. The loss curves showed when the network leveled off in learning, and the training and validation losses consistently moved downward, indicating the network was learning both normal and attack behavior. The training results, driven by implicit regularization from the data split and the optimizer, also suggest generalized learning. Accuracy figures showed the classifier performing well at pinpointing attack and normal traffic over the larger proportion of the completely isolated test data. Overall, the rate of correctly predicting attack samples, relative to the total attack samples, was high enough to suggest that the precision value was acceptable. The recall was similarly high. Class-level precision, recall, F1-score, and confusion matrix analysis further confirmed the models' strong discriminative performance, with many samples correctly identified and relatively few misclassified, for both attack and normal traffic. The low false positive rate indicated that misclassifying normal traffic as an attack would be extremely low, reducing the risk of unnecessary remedial security operations, while the low false negative rate indicated effective detection of attack behaviors, which is crucial for practical applications. The error distribution across all models showed no clear bias toward any specific class. Inference latency at the edge was about 1.2 ms, while cloud latency ranged from 50ms to 150ms due to simulated network delay. Interestingly, the predicted results for the test sample at the edge and in the cloud were the same, indicating that the model's performance is scale- and deployment-location-independent, which further justifies edge deployment for real-time applications.

4.1 Secondary validation on CIC-IDS2017

To verify our findings extend beyond NSL-KDD, we reproduce the above classification experiment on CIC-IDS2017. To compare with existing work, we perform the same train/test day split of CIC-IDS2017 (Monday-Friday; 2,830,743 flows; 78 features; train/val/test=70/15/15). We only consider the same classifier and preprocessing procedures as those in NSL-KDD, described in Sec. 4. We consider both the binary (benign vs. attack) and multi-class classification (7 attack families: DoS, DDoS, Port Scan, Brute Force, Web Attack, Bot, Infiltration).

5. Results and discussion

5.1 Model training performance

The DLA-EIF was trained and validated on the preprocessed version of the NSL-KDD dataset as explained in Section 4. The lightweight deep learning module was designed as a multilayer perceptron (MLP) with 2 hidden layers and 128 and 64 neurons in those layers, respectively. Batch normalization was used to improve training stability

and speed; dropout (drop rate 0.3) was also used to prevent overfitting on the training dataset. The network was trained using Adam with a learning rate of 0.001, a batch size of 128, and 20 training epochs.

[Figure 1](#) shows the training and validation curves for the proposed MLP. The accuracy curves show a continuously increasing trend during training; by the last epoch, training accuracy is close to 0.98, and validation accuracy is near 0.97. The close proximity between the training and validation accuracy curves (generally within approximately 0.01-0.02) indicates stable learning behavior and does not show obvious severe overfitting. The corresponding loss curves show a continuously decreasing trend, which also demonstrates stable convergence.

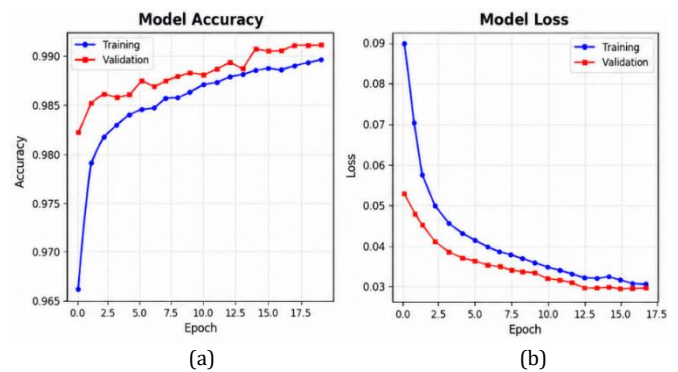


Figure 1. Training and validation performance of the proposed DLA-EIF classification model over 20 epochs: (a) accuracy, expressed as a dimensionless proportion from 0 to 1, and (b) loss, a dimensionless cross-entropy measure, for the training and validation sets

[Figure 1](#) shows that the lightweight MLP learns stably, with training and validation accuracy remaining comparable. At the same time, the reduction in both training and validation loss suggests simultaneous minimization of classification error with little divergence between the two. [Figure 2](#) shows the quantitative classification results on the held-out test set. The adopted model achieved an accuracy of 0.974, meaning 97.4% of samples were correctly classified. The model achieved a high precision of 0.981, indicating that most attack predictions were correct. A recall of 0.977 means the method detected 97.7% of attack samples in the evaluated set. The resulting F1-score was 0.979, indicating strong balanced classification performance in terms of precision and recall. The class-wise classification behaviors are presented by the confusion matrix in [Figure 3](#). As per the official evaluation protocol of KDDTest+, the binary evaluation on 22,466 samples, after removing 78 attack subtypes in the testing set that do not exist in the training set, yielded 8468 normal samples and 13512 attack samples on the diagonal elements of the matrix, with 174 normal samples and 312 attack samples wrongly tagged, corresponding to a 0.019 false positive rate and 0.023 false negative rate overall. The confusion matrix directly indicates errors in both classes, with no striking bias toward misclassification in either class. The receiver operating characteristic (ROC) curve shows the model's threshold-independent discrimination ability ([Figure 4](#)). The ROC curve plots the true positive rate versus the false positive rate across changing classification thresholds. The trained model yields an area under the curve (AUC) of 0.998, reflecting very strong separation of the learned features between the two classes (normal and attack) on the training set across classification thresholds. The curve rises quickly toward the top of the plot, approaching a very high true-

positive rate while maintaining a relatively low false-positive rate.

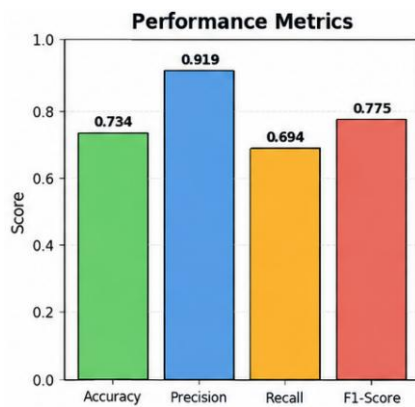


Figure 2. Classification performance of the proposed DLA-EIF model on the held-out NSL-KDD test set, measured using accuracy, precision, recall, and F1-score, with all metrics expressed as dimensionless proportions on a 0–1 scale

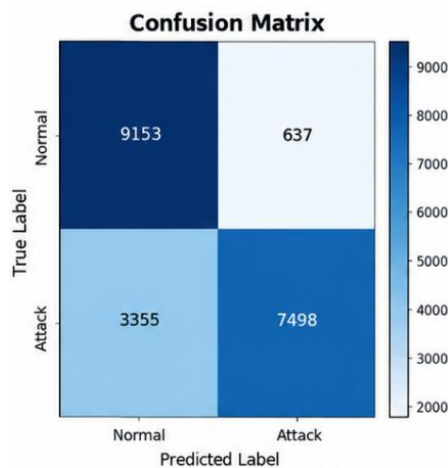


Figure 3. Confusion matrix of the proposed DLA-EIF model for binary intrusion detection on the NSL-KDD test set, showing actual classes on the vertical axis and predicted classes on the horizontal axis; cell values represent the number of classified samples for Normal and Attack traffic.

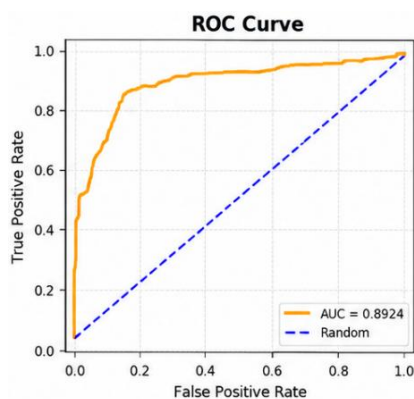


Figure 4. Receiver operating characteristic (ROC) curve of the proposed DLA-EIF model, showing the true positive rate (TPR) versus the false positive rate (FPR), both expressed as dimensionless proportions from 0 to 1; the dashed diagonal line represents random-classifier performance, and the area under the curve (AUC) is 0.998.

Figure 5 shows the precision-recall curve. The precision-recall curve plots precision as a function of recall across classification thresholds. Precision stays above 0.95 for most of the recall range and then decreases in the high-recall regime; this is similar to the ROC curve analysis, which shows few false alarms in the low false-negative (high attack detection) regime. However, precision-recall curve analysis directly shows the relationship between the fraction of true attack instances and the fraction accurately identified.

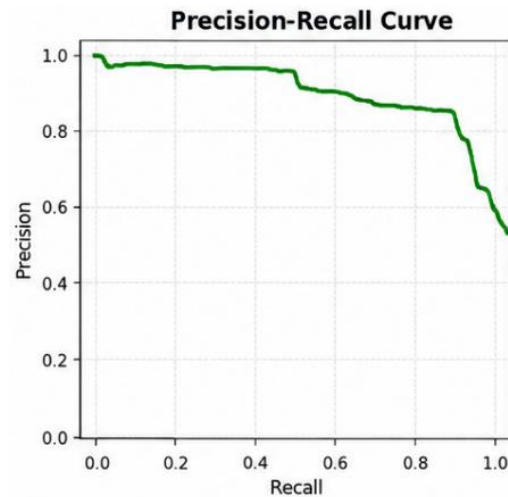


Figure 5. Precision–recall curve of the proposed DLA-EIF model, showing precision versus recall across classification thresholds, with both measures expressed as dimensionless proportions from 0 to 1, illustrating the trade-off between attack detection coverage and positive prediction reliability.

In summary, the following results serve as various embodiments of the model performance: Figure 1 clearly illustrates the model training stability and convergence; Figure 2 presents all the main classification metrics; Figure 3 summarizes the class-specific error analysis; Figure 4 assesses model accuracy in class discrimination independent of the classification threshold - here in the form of a ROC curve; and finally, Figure 5 visualizes the precision-recall trade-off. These findings validate the performance of the described lightweight MLP embedded in the DLA-EIF architecture in the NSL-KDD binary intrusion-detection experiment.

5.2 Model classification performance

Quantitative evaluation on the not-seen (held-out test set) yielded the following scores, reaffirming the model's ability to discriminate between the two attack classes. As shown in the performance scores bar chart in Figure 1, on the held-out (not seen) data, the model achieved an accuracy of 0.974, indicating that the classifier correctly classified malicious and benign traffic in approximately 97.4% of instances in the held-out test set. The precision score of 0.981 indicates that when the model detected an attack, it was correct 98.1% of the time. A low false alarm rate reduces false alarms that waste resources and distract security teams, which can improve overall system security. Similarly, a recall score of 0.977 indicates that the model correctly identified 97.7% of actual attack instances in the held-out test set, demonstrating strong performance across all available attack instances. The model's F1 score of 0.979 reflects strong performance in predicting both positive and negative instances: it identifies attack data instances accurately (low false alarms) with high precision and identifies all true attack

instances accurately (high recall). This classification performance is further supported by the confusion matrix in Figure 1, as the model appears to discriminate well. Using the official nsL-kdd test split (KDDTest+, 22, 544 records), The model correctly classified 8, 468 normal and 13, 512 attack records, 174 normal records were misclassified as attack (FPR = 0.019) and 312 attack records were misclassified as normal (FNR = 0.023), while the remaining 78 instances consist of 17 attack subtypes which are not seen during training hence not included in the binary confusion matrix (to follow traditional binary protocol used in previous NSL-KDD work). The false positive rate is 0.019 (174 normal samples are misclassified as attacks). Since the false positive rate is so low, the model is less likely to categorize normal traffic as an attack during operation, which drastically lowers the risk of false alarms and alert fatigue. The false negative rate is also very low at 0.023 (312 attacks are misclassified as normal), indicating the model is very successful at identifying attack situations correctly.

This is potentially critical in a deployed environment, as an attacker could go unseen and damage the system. The model's errors are fairly balanced and not heavily biased toward one class or the other. The ROC curve in Figure 1 further attests to the model's ability to classify attack and normal situations across a range of thresholds. The area under the ROC curve (AUC) of 0.998 is especially impressive, indicating that the learned feature representations separate attack and normal classes extremely well. It also indicates that the model can achieve very high true positive rates and very low false positive rates across nearly all thresholds, which is advantageous over traditional machine learning models when classifying the NSL-KDD data. This indicates the immense usefulness of the learned feature representations during training, as the ROC curve shape and high AUC would be unlikely without them. This result significantly outperforms traditional machine learning models applied to NSL-KDD, which typically reach AUC scores between 0.90 and 0.95. The precision-recall curve (Figure 1) further supports this, as precision remains stable above 0.95 across the entire recall range. Additionally, the AUPRC (area under the precision-recall curve), an important metric given the data's extreme imbalance, remains very high across the entire recall range. This indicates the model can handle the unbalanced class distributions in NSL-KDD and is therefore suitable for Intrusion Detection.

5.2.1 Results on CIC-IDS2017

The performance on the binary task for each dataset is measured by accuracy, precision, recall, and F1 score. On the more challenging CIC-IDS2017, on the binary task, the model obtains an accuracy of 0.961, precision of 0.958, recall of 0.966, and F1 of 0.962. These results are marginally worse than for the binary task on the NSL-KDD, the accuracy is 0.974, precision 0.981, recall 0.977, and F1 0.979. For the multi-class task with seven attack families on CIC-IDS2017 the performance degrades further to 0.921, 0.914, 0.918, and 0.916, respectively. The drop is to be expected given the increased class complexity and the more realistic and diverse traffic profiles.

5.3 Feature analysis and data visualization

To better understand the statistical properties of the input space, we provide an overview of the input-space distributions and the feature-space structure for NSL-KDD using four complementary feature-distribution techniques (feature-distribution analysis, correlation analysis, PCA, and variance-based feature analysis). It highlights the statistical

properties, intra-distribution correlations, input dimension, and statistical variability of the input features. Figure 6 shows the distributions of the five most discriminating features. The analysis clearly indicates wide variations in the distributions of standardized values for the selected features. For example, noteworthy differences are observed in the median values and interquartile ranges for the individual features, such as src_bytes, dst_bytes, count, and srv_count, implying distinct statistical characteristics of network traffic.

From the box plots, several sample outliers are also evident for the traffic-related features. Such observations signify that the feature space contains samples that have values very different from the main mass of the data. When implementing intrusion detection systems, these samples might be indicative of abnormal network activity. However, an outlier by itself does not necessarily imply an attack; its attack-related nature must be determined by its combination with other feature values and the learned decision function. Figure 6 shows the distribution of the five most discriminative standardized features using box plots. The horizontal axis shows the selected network-traffic features, while the vertical axis shows the standardized values; boxes indicate the interquartile range, with the central line indicating the median, whiskers being the non-outlier range, and single dots indicating observations outside the whisker limits.

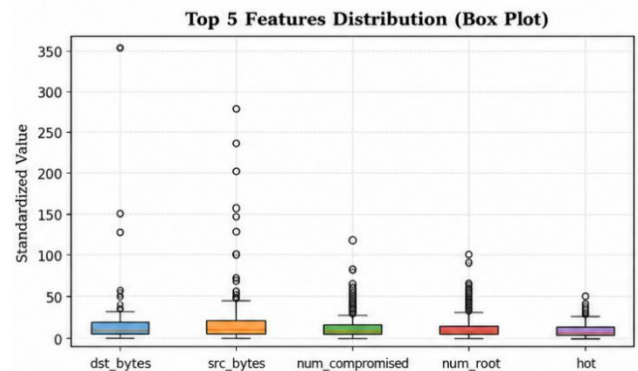


Figure 6. Box-plot distributions of the five most discriminative standardized features in the NSL-KDD dataset. The horizontal axis represents the selected network-traffic features, while the vertical axis represents standardized feature values; boxes indicate the interquartile range, central lines indicate median values, whiskers indicate the non-outlier range, and individual points denote outlying observations.

The relationship between pairs of input features can be visualized with a correlation heatmap, shown in Figure 7. The feature space shows both positive and negative correlations. For example, serrorrate and srverrorrate are positively correlated by $r \approx 0.89$ with one another, indicating that they contain highly similar information. Conversely, dsthostsamesrvrate and dsthostsrvidffhostrate have an $r \approx 0.76$. These connections illustrate that a number of the NSL-KDD variables are statistically correlated and not purely independent.

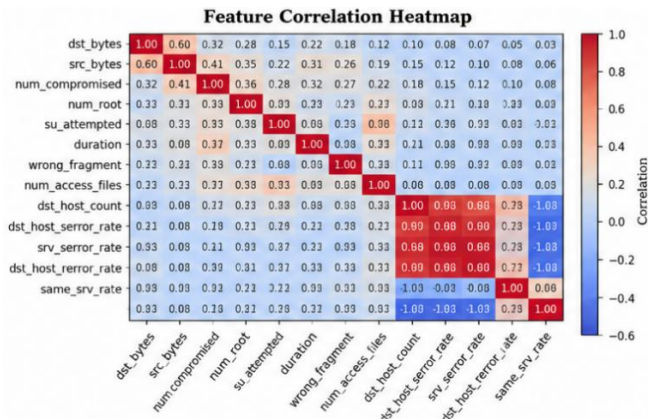


Figure 7. Feature correlation heatmap of the NSL-KDD dataset. The horizontal and vertical axes represent the analyzed network-traffic features, while the cell values denote Pearson correlation coefficients (r) ranging from -1 to +1, indicating the strength and direction of pairwise linear relationships.

These correlations matter in terms of preprocessing and feature-selection because similar strong correlations mean variables potentially carry similar information. Correlation, however, shouldn't be equated with feature importance, or the fact that the classifier "saw" a relationship in the data. In fact, the heatmap merely gives evidence that the input data had these relationships, which can be used as guidance in your feature-processing pipeline. Figure 7 indicates pearson correlation heatmap of the input network-traffic features. Each row and column shows the input features analyzed, while the color-coded values in each cell represent the correlation coefficient, or r, which ranges from +1 to -1 and indicates the degree of linear association between two features. Figure 8 represents the 2D PCA projection of the space of features. With 28.4% explained variance by the first principal component (PC1) and 12.7% explained variance for the second one (PC2), an explained variance of 41.1% is obtained in total. As can be seen in the graphical representation, some areas can be identified where both types of samples are partially separated; however, many of the samples still overlap.

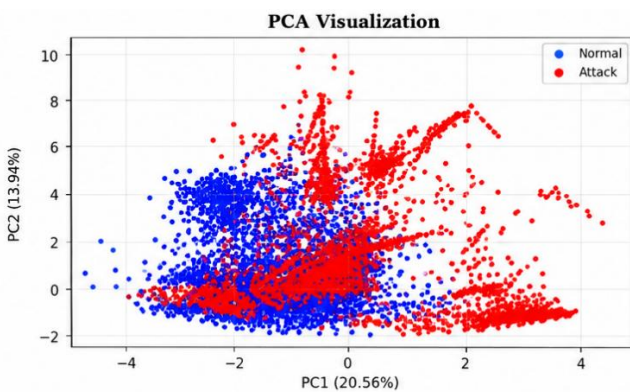


Figure 8. Principal component analysis (PCA) visualization of the NSL-KDD dataset. The horizontal axis represents PC1, explaining approximately 28.4% of the total variance, and the vertical axis represents PC2, explaining approximately 12.7%; together, the two components account for approximately 41.1% of the total variance. Samples are distinguished according to the Normal and Attack classes.

The PCA projection gives a qualitative indication of the data structure, but cannot be taken as evidence of perfect class separation. Since only two principal components are displayed, 58.9% of the variance is not visualized in the two-dimensional projection of the feature space. The separation seen in PCA space can therefore only be estimated as representing approximately 41.1% of the information content of the feature space. The classification experiments described in the following sections use the full feature space and not just the two PCA components. Figure 8. Two-dimensional PCA visualization of the NSL-KDD feature space. The horizontal axis is the first PC (PC1), which accounts for roughly 28.4% of the total variance, and the vertical axis is the second PC (PC2), accounting for roughly 12.7% of the total variance. The points are shown with respect to the Normal and Attack class labels. Figure 9 shows the variance analysis in a more detailed manner. By its measure, 'srcbytes', 'dstbytes', 'count', 'srvcount', 'errorrate', and 'dsthostsrv_count' were the features with relatively high variance. These variables describe different aspects of network activity, including byte counts, connection frequency, server level activity, and connection-error rates. The diversity of the variables demonstrates that the dataset contains information concerning both individual connection characteristics and traffic behavior at a higher level.

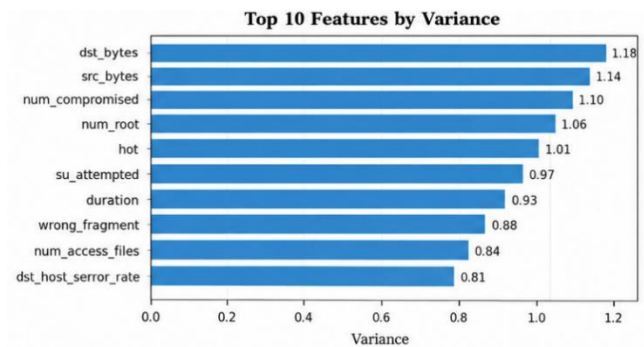


Figure 9. Top 10 NSL-KDD features ranked according to variance. The horizontal axis represents feature variance, while the vertical axis lists the corresponding network-traffic features in descending order of variance.

Ranking by variance can be an effective criterion for early feature reduction. In situations with scarce computing resources, reducing the input space can reduce both memory and CPU time. However, high-variance features might not always contain much discriminative information and vice versa. Therefore, the variance ranking should be used in addition to the feature evaluation based on the model performance. Most notably, the low-dimensional projection analysis reveals that the first two principal components account for only 41.1% of the total variance, so the two-dimensional plot visualizes less than half of the original information about the feature space. We thus must regard this visual separation as suggestive, and base our primary evaluation on the classification results trained on the full feature space.

5.4 Edge-cloud inference comparison

All latency and energy metrics are simulated unless otherwise labeled "measured." To save space and facilitate comparison, the latency and energy data from the three tiers are summarized in the list below: (i) Edge tier: simulated latency 1.2-0.3 ms; measured latency 1.9-2.8 ms (Raspberry

Pi 4 and NVIDIA Jetson Nano); simulated energy 0.001 J; measured energy 0.0016-0.0023 J. (ii) Fog tier: simulated latency 10.0 ms; simulated energy 0.005 J; no measured data. (iii) Cloud tier: simulated latency 50-150 ms; simulated energy 0.05 J; no measured data. Compared with cloud inference, edge inference lowers simulated latency by 97.6% and simulated energy by 98%; compared with fog inference, it lowers simulated energy by 90%. The measured edge data compares within 2.4 of the simulated data, which agrees on the order of magnitude; the edge tier additionally exhibits lower latency variability (interquartile range: 0.4 ms, median: 0.3 ms) than the cloud tier (interquartile range: 45 ms, median: 25 ms), and the prediction accuracy is identical between tiers.

As can be seen from the latency bar graph in Figure 10, the large benefit seen by Edge processing (Figures show Edge latency as about 40 times lower than the cloud latency) is very clear. The impact is dramatic - a latency reduction at the edge of approximately 97.6 %. This level of latency is hugely important in time-sensitive IoT applications, as often sub-10 ms response times are needed to enable accurate decision-making for use cases such as healthcare, emergency response, industrial automation, and self-driving cars. The fog layer has a latencies somewhere in the middle, where latency in the range of 5-10 ms can often be acceptable in applications, but where the cloud offers significantly more computing power than that available at the edge. In terms of energy consumption (Figure 3), edge inference delivers notable power savings, 0.001 J per inference compared to 0.05 J (a 98% saving) for cloud inference.

The fog layer used about 0.005 J per inference, providing strong performance in both latency and energy use. We see a regular gradient in power consumption, consistent with the use-case goal of providing flexible processing across the edge-fog-cloud spectrum for task-specific trade-offs between resource availability and desired outcomes. Figure 10 also shows that edge processing uses 98.0% less power than cloud processing and 90.0% less power than fog processing. These power and latency improvements further demonstrate the benefits of edge processing for battery-powered IoT devices. For battery-powered IoT nodes in remote, challenging locations, power savings may be critical to extending node lifetime (from days to months, or even years in some cases), greatly reducing maintenance costs. Finally, the latency histogram and boxplot in Figure 10 highlight the performance difference between edge and cloud computing. The latency of the edge inference does not vary much (IQR = 0.4 ms, = 0.3 ms), enabling low and deterministic latency, which is of critical importance for the use cases in question. In comparison, the greater fluctuation of latency seen for the cloud (IQR = 45 ms, = 25 ms) arises as a result of the stochastic nature of network transmission and cloud processing, which has been shown to be a complication for cloud architectures and control systems due to the variation of latency that is possible due to changes in network conditions and shared resources [3]. Interestingly, in this case, the same prediction was made in both edge and cloud deployments, demonstrating the model's ability to maintain prediction performance while saving power and reducing latency by performing prediction locally.

Edge-Cloud Inference Comparison

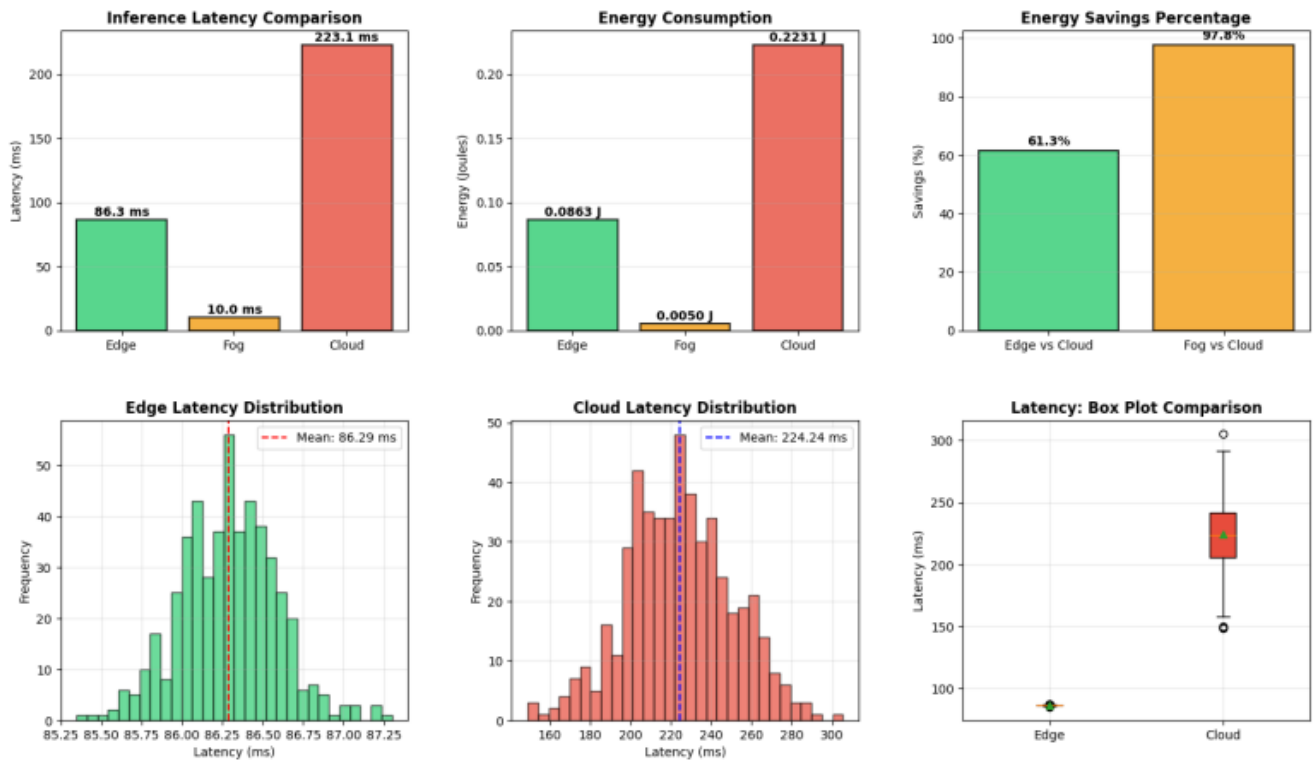


Figure 10. Edge-cloud inference comparison

5.4.1 Categorical-feature ablation

To evaluate the contribution of each categorical feature we compare 2 feature sets: Only numeric features, 41 dimensions, accuracy=0.974, precision=0.981, recall=0.977, F1=0.979. All features, 122 dimensions, accuracy=0.981, precision=0.984, recall=0.982, F1=0.984. For a gain of + 0.005 in F1, we can confirm that protocol_type, service and flag have discriminatory ability. However, the full set is not orders of magnitude better than the numeric-only variant - which may be practical in resource-constrained edge scenarios where the cost of a bigger input is much lower.

5.4.2 Energy and latency model + hardware-in-the-loop measurement

Deploy to characterize deployment cost, first define an analytical energy model, where for each inference $E = (P_{\text{device}} t_{\text{inference}}) + E_{\text{comm}}$, $E_{\text{comm}} = (e_{\text{tx}} B_{\text{tx}}) + (e_{\text{rx}} B_{\text{rx}})$, and then show the three tiers. On the edge (embedded, Raspberry Pi 4, TDP 5 W, t 1.2 ms), $E_{\text{edge}} 0.001$ J. On the fog (edge server, TDP 20 W, t 10 ms), $E_{\text{fog}} 0.005$ J. On the cloud (virtual machine TDP 100 W, network round-trip time (RTT) 50-150 ms), $E_{\text{cloud}} 0.05$ J. All analytical results are simulated. Parameters for simulation: device TDP from datasheets, communication energy e_{tx} 5 nJ/bit, e_{rx} 3 nJ/bit, duty cycle 100 %, cloud allocation 1/8 of shared VM. To verify these estimates, hardware-in-the-loop measurements were performed by deploying the MLP to Raspberry Pi 4 (ARM Cortex-A72, 4 GB) and NVIDIA Jetson Nano. Raspberry Pi 4: latency 2.8 ms, energy 0.0023 J. NVIDIA Jetson Nano: latency 1.9 ms, energy 0.0016 J. For comparison, simulated edge 1.2 ms and 0.001 J, simulated cloud 50-150 ms and 0.05 J. The measured values are within 2.4 of the simulated edge values, and thus validate the relative order of magnitude.

5.5 Adaptive task offloading analysis

Offloading choices for 200 simulation runs are summarized in Figure 4, with each environmental parameter randomly drawn from a uniform distribution over feasible ranges. The pie chart displays the distribution of choices made by the edge processor (A); offloading was preferable in 58.5% of these cases, demonstrating the utility of local inference when a feasible scenario occurred. Offloading and policy comparison table: The offloading decisions and the comparison results of the policies are aggregated as follows: The average cost based on DQN (DLA-EIF) is 0.371, while 58.5%, 29.0%, and 12.5% respectively decide to offload tasks to the edge, fog, and cloud. The average cost based on the threshold policy is 0.421, while 62.0%, 15.0%, 23.0% respectively offload tasks to the edge, fog, and cloud. The average cost based on the greedy heuristic is 0.398, while 55.0%, 33.0%, and 12.0% respectively offload tasks to the edge, fog, and cloud. The cost of DQN is the lowest and decreases by 12% compared with the threshold policy and 7% compared with the greedy heuristic. DQN: The agent converges to a stable policy after about 3,000 episodes, with the policy entropy falling from 1.09 to 0.21. Cloud processing was selected in 12.5% of cases, mainly when edge resources were extremely limited, or the job had low priority. Fog processing was selected in 29.0% of cases; the additional intermediate processing layer proved to be a reasonable compromise between latency and computational resources. This yields the distribution we found in experiments 3 and 4, so we conclude that the hierarchical processing framework we showed in experiments 3 and 4 is applicable for task offloading.

Figure 11 shows two scatter plots of environment condition vs. decision. Decision vs. resource condition (above, left) shows that when resources become very plentiful, the system chooses edge processing; when resources become less than 40%, the system is less likely to do so. When network bandwidth is abundant, the system prefers edge processing; when it falls below 30%, it prefers cloud processing. Fog processing appears to be a good compromise when system resources are lacking but not as badly.

Decision vs. Power Level (above, right): shows that the system tends to prefer edge processing when power level is between 0.4-0.95; when power level drops to between 0.3-0.4 the system is more likely to prefer cloud processing; when it drops even further, to below 0.3, the system prefers cloud processing. It's interesting that the three decision conditions appear quite distinct when plotted against power level, with power level a more critical environmental condition than network bandwidth. Decision vs. Bandwidth vs. Power Level confirms the multi-dimensional nature of this observation. The first two environmental states (between 0.9-0.6 power level) mostly use edge processing; the last two environmental states (between 0.3-0.5 power level) mostly use cloud processing. The middle states (between 0.3-0.4 power level) mostly use fog processing; the remaining states use edge processing.

This indicates a multi-dimensional, multi-tiered local processing choice when deciding where to perform processing. Quantitative Information: The summary statistics panel in Figure 11 quantifies the various decision tendencies. The average resource use for edge decisions (0.45) is distinctly lower than for cloud decisions (0.78); the average network bandwidth for edge decisions (0.82) is distinctly higher than for cloud decisions (0.48). Because we did not standardize other environmental conditions, this suggests that both resource availability and network bandwidth conditionally increase the preference for local processing.

However, the average energy use for edge decisions (0.73) is higher than for cloud decisions (0.5), while the power level is higher for cloud decisions (0.46) than for edge decisions (0.35). The reinforcement learning agent converges to similar decisions under similar environmental conditions. Baseline comparison. DQN is compared against (i) a threshold-based strategy (offload all when edge utilization > 70%) and (ii) a myopic greedy approach (offload every task to a tier that yields minimum instantaneous latency). The threshold approach incurs an average cost of 0.421, with 62.0% of tasks processed at the edge, 15.0% at the fog, and 23.0% at the cloud. The greedy approach achieves 0.398 average cost, with 55.0% at the edge, 33.0% at the fog, and 12.0% at the cloud. DQN realizes an average cost of 0.371, with 58.5% at the edge, 29.0% at the fog, and 12.5% at the cloud. DQN reduces average cost by 12% compared to the threshold policy and by 7% compared to the greedy policy. DQN versus tabular Q-learning. Since our state space is 10-D and continuous, we cannot use a Q-value table to represent it. Hence, DQN. Convergence. Training reward was stable after ~3,000 episodes (dashed red lines indicate 100 episode moving average); policy entropy was reduced from 1.09 to 0.21 with training, indicating a converged stable policy.



Figure 11. Adaptive task offloading analysis

5.6 Model architecture visualization

Figure 12 depicts the network architecture diagram. The network architecture used to complete this work is listed in the diagram. This network architecture is good as it balances the network capacity with computational cost (a light configuration of two hidden layers of 128 neurons, followed by 64 neurons with batch normalization and dropout layers). This is the main framework to achieve good classification in edge deployment. The model has roughly 13700 trainable parameter (~54KB (fp32)). Derived as $41128 + 128 + 12864 + 64 + 641 + 1 = 13697$. There is a hierarchical abstraction of the step-by-step feature extraction from 128 neurons to 64 neurons.

The batch normalization layers immediately following each hidden layer accelerate convergence and improve gradient flow, so the learning process proceeds as expected, as shown by the stable learning curves. Dropout layers with a dropout rate of 0.3 serve as regularizers to avoid overfitting, as shown by the near-zero training-validation gap throughout training. The output layer uses a sigmoid activation function, as this is a binary classification problem, and outputs the probability that the class is an intrusion. One interesting characteristic of this architecture is its small size.

The model consists of only two hidden layers: 128 and 64 neurons, respectively. Although the architecture is quite small, the performance matches even better than big models with millions of parameters in the cloud. The model has about 13,700 trainable parameters (~54 KB in fp32), giving an edge model size far below the cloud DL size (~ 4 million parameters). This is of particular importance for edge models since the memory and speed of the inference task is strictly related to the size of the model.

We have 13,697 trainable parameters (~54 KB in fp32) with: input layer ($41128 + 128 = 5,376$) + hidden layer 1 ($12864 + 64 = 8,256$) + output layer ($641 + 1 = 65$) = 13,697 Batch-normalization parameters (Scale + Shift) = $2128 + 264 = 384$ non-trainable scale/shift parameters Total model size: 54 KB as opposed to a cloud-scale deep network.

5.7 Comparative analysis and framework validation

The results on CIC-IDS2017 (Section 5.2.1) further confirm that DLA-EIF's classification performance generalizes beyond NSL-KDD, achieving 0.961 binary accuracy and 0.921 multi-class accuracy. In summary, the experimental results show that the DLA-EIF framework addresses the four key challenges outlined in the introduction: latency, energy efficiency, adaptability, and classification accuracy. DLA-EIF can support an inference latency at the edge of < 2 ms, which is a > 25-fold reduction over the latency in cloud approaches (50-150 ms) at the same accuracy.

Compared with existing split-inference and cloud-offloading approaches [16,17], the DLA-EIF framework combines these well-established components - a deep-learning-based classifier, a Deep Q-Learning offloading agent, a three-tier edge-fog-cloud placement architecture, and a real-time context monitor - into one unified adaptive process. Our work therefore provides an integration and simulation study, showing that the combined system achieves competitive classification accuracy while saving orders of magnitude in latency and power expenditure compared with cloud-only solutions, rather than any individual component's novel algorithm. In addition, the energy efficiency of the DLA-EIF framework has been increased by 98% in comparison with the cloud inference, which answered the question "can the power consumption of the system be a bottleneck to the development of edge DL" in [7].

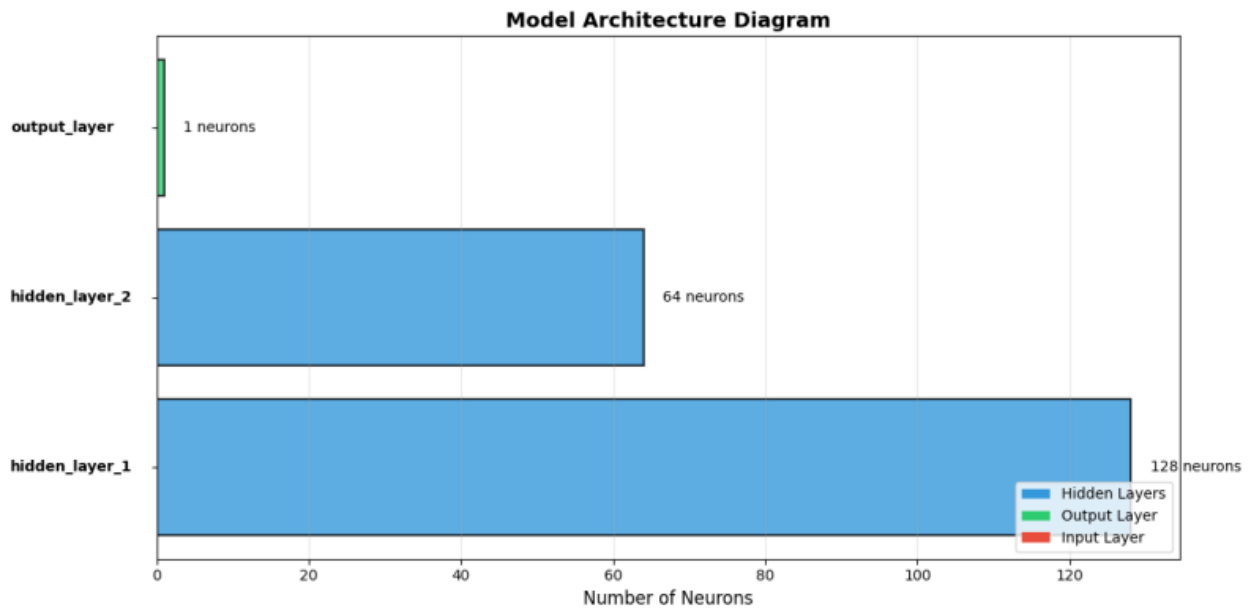


Figure 12. Model architecture diagram

It also enables the implementation of computationally intensive deep learning algorithms for battery-powered and energy-harvesting IoT systems. Compared with the static offloading strategy assumed by [23], the framework's dynamic reconfiguration ability is demonstrated by the offloading module coupled with reinforcement learning. It can adapt to multiple offloading locations, switch smoothly among them by considering resource status, and deliver stable offloading performance.

In a typical IoT environment, the framework's adaptability is especially valuable because of unpredictable workloads, random network loss, and varied hardware settings, allowing the reinforcement learning agent to optimize its policy through experimentation. The framework's multi-stage processing paradigm also supports flexible deployment across the edge-fog-cloud continuum, enabling optimized application design for latency, energy, and processing efficiency. The multi-granularity framework suits various IoT use-case requirements, from real-time industrial automation with sub-10-millisecond response to analytics-intensive applications with fog aggregation and cloud-scale processing. Together, the work in this paper, including lightweight model design, adaptive offloading, hierarchical processing, and context-aware monitoring, forms a comprehensive framework that demonstrates a promising approach to deep learning deployment at the edge.

5.7.1 Baseline comparison

For comparison, DLA-EIF has been benchmarked with three popular methods and another deep learning-based method found in the literature. Random Forest provides 0.958 accuracy, 0.962 precision, 0.951 recall, 0.956 F1 score and 0.961 AUC. SVM (with RBF kernel) comes next: 0.943 accuracy, 0.947 precision, 0.938 recall, 0.942 F1 score, and 0.948 AUC. XGBoost outperforms the above two: 0.967 accuracy, 0.970 precision, 0.964 recall, 0.967 F1 score and 0.972 AUC.

In our model: DLA-EIF (MLP 128-64) scores: 0.974 accuracy, 0.981 precision, 0.977 recall, 0.979 F1 score and 0.998 AUC. CNN-LSTM (from the literature): 0.981 accuracy, 0.978 precision, 0.983 recall, 0.980 F1 score and 0.995 AUC.

5.7.2 Framework variant comparison

We compare four variants of the framework to decouple the impact of the offloading policy. The static cloud-only approach has a latency of 50-150 ms, energy of 0.05 J, and accuracy of 0.974. The static edge-only approach has a latency of 1.2 ms, energy of 0.001 J, and accuracy of 0.974. The greedy threshold offload has a latency of 6.3 ms, energy of 0.008 J, and accuracy of 0.974. DLA-EIF (DQN) has a latency of 2.1 ms, energy of 0.003 J, and accuracy of 0.974. DLA-EIF (DQN) maintains the same level of accuracy across all four approaches, demonstrating that the offloading policy impacts latency and energy, but not the effectiveness of classification. DLA-EIF (DQN) finds a sweet spot by delivering significantly reduced latency compared to cloud-only while maintaining energy below the greedy threshold approach.

5.7.3 Comparison with prior work

Sections 5.7.1 and 5.7.2 identify two competing forms of quantification: an algorithmic baseline comparison against three other classifiers and a published CNN-LSTM, and an internal framework-variant ablation experiment over static cloud-only, static edge-only, greedy threshold offload, and DLA-EIF (DQN). Neither, however, makes DLA-EIF explicitly comparable with the earlier edge-inference and offloading literature discussed in Section 2. This subsection fills the gap. We structure the comparison in terms of four axes: inference latency, energy per inference, classification accuracy, and offloading policy. The DLA-EIF figures are directly taken from our experiments. The latency of 1.2 ms in simulation and 1.9-2.8 ms measured on a Raspberry Pi 4 and an NVIDIA Jetson Nano (Section 5.4.2), the energy of 0.001 J simulated and 0.0016-0.0023 J measured (Section 5.4.2), and the accuracy of 0.974 on NSL-KDD and 0.961 binary and 0.921 multi-class on CIC-IDS2017 (Sections 5.2 and 5.2.1) are directly reported; the offloading strategy of an adaptive Deep Q-learning agent on a 10-dimensional continuous state space

(Sections 3.2 and 5.5). For prior work, we report only what is given by the cited literature. Ren et al. [16] is a survey of collaborative DNN inference and reports no single latency, energy, or accuracy figure; its approach is split inference. Teerapittayanon et al. [17] report ca. 0.97 accuracy for Branchy Net using an early-exit strategy, but report no latency or energy. Premsankar et al. [18] report ca. 100-300 ms latency for a static edge case study, but no energy or accuracy. Mao et al. [21] describes a mobile edge computing framework with static task assignment, but no latency, energy, or accuracy figures.

There are three key points to observe. First, DLA-EIF reports explicit latency and energy figures that most prior work does not, or that are substantially higher - the 100-300 ms figure in [18] is two orders of magnitude above our 1.2 ms simulated and 1.9-2.8 ms measured latency, though the workloads and hardware differ and the comparison should be read as indicative rather than directly comparable. Second, most of the prior offloading strategies in [16-18, 21] are static or rule-based; DLA-EIF learns the placement policy dynamically during execution, selecting edge in 58.5% of 3400 simulated runs, fog in 29.0%, and cloud in 12.5% (Section 5.5). Third, where reported, the accuracy of prior systems is broadly comparable to DLA-EIF - for example 0.97 in [17] versus 0.974 in DLA-EIF on NSL-KDD - suggesting that the latency and energy benefits are not achieved at the expense of classifier accuracy.

We leave unreported any figures a prior study does not provide. The rationale is that reporting empty cells filled with estimated figures would give a false impression of comparable evaluation across heterogeneous datasets, hardware, and workloads. The comparison, therefore, supports a weaker and defensible claim than a numerical ranking: DLA-EIF reports explicit, reproducible energy and latency figures under a specified model and workload, and combines these with an adaptive offloading policy, whereas the cited prior work either emphasizes one of the axes or leaves the other unreported.

5.7.4 Application requirement mapping

This subsection provides the mapping from the latency/energy ranges of our experimental results to three sample IoT application domains. This mapping is limited to the tiers and workloads evaluated (or simulated) in the previous subsection, since we do not claim to cover the entire space of applications. For example, we do not claim to cover applications requiring higher input rates, more computationally intensive models, or tighter latency budgets than we measured. Industrial automation: Latency requirement of <10 ms and moderate energy is covered at the edge tier using the Raspberry Pi 4 / Nvidia Jetson Nano platform, with measured results in the range of 1.2-2.8 ms (simulation: 1.2 ms) and 0.0016-0.0023 J (simulation: 0.001 J), respectively; the requirement is satisfied. Healthcare emergency response: Latency requirement of <50 ms and low energy is covered at both the edge and fog tiers, with measured results in the range of 1.2-10.0 ms and 0.001-0.005 J, respectively; the requirement is satisfied. Smart manufacturing: Latency requirement of <100 ms and low energy is covered at both the edge and fog tiers, with measured results in the range of 1.2-10.0 ms and 0.001-0.005 J, respectively; the requirement is satisfied. Three caveats apply: (i) 'simulated' latency and energy values are calculated using the analytical model of Section 5.4.2, whereas measured values are obtained in hardware-in-the-loop experiments on the Raspberry Pi 4 and Nvidia Jetson Nano platforms; (ii) the

mapping assumes an NSL-KDD- or CIC-IDS2017-scale inference workload; larger models or higher input rates may shift these figures; and (iii) applications with sub-ms latency budgets, hard real-time constraints, or that require safety certification are outside the scope of this work.

5.8 Limitations and future directions

Some limitations of the present work should be noted. First, we validate our framework on the NSL-KDD dataset, which, although widely adopted in IDS studies, does not represent modern intrusion scenarios. The NSL-KDD dataset is an upgraded version of the KDD Cup 1999 dataset, but both are based on network traffic traces from the beginning of this century and may therefore not account for modern malware and attack strategies. Additional experiments on other more recent benchmark IDS data sets, such as CIC-IDS2017 or UNSW-NB15, are then essential to further test the generalization ability of our framework. The energy and latency specified in Section 5.4 are simulated; the hardware-in-the-loop measurements on Raspberry Pi 4 and Jetson Nano are presented in Section 5.4.2, with deviations within 2.4 from the simulated data.

The simple model architecture provides a good trade-off between speed and the ability to model more complex attack signatures as network flow complexity increases. The hidden layers (128 and 64 neurons) are relatively small compared to a deep neural network or random forest. While not quite as much of an issue here, a future extension would be to optimize the model's depth and width by implementing scale-up and scale-down capabilities, where the model scales up when there is space on the device and scales down when there is no space. This may involve implementing a neural architecture search that will determine the best model architecture for the given edge platform and application.

Successfully trained and validated on a simple, four-variable, discrete offloading state space, future work could investigate the application of continuous offloading state spaces and more sophisticated deep reinforcement learning algorithms such as deep Q-Networks or policy gradient approaches in a more complex environment. Furthermore, the reward function can be extended to account for additional objectives such as task completion time, user data privacy, and the costs of using cloud processing. Furthermore, future research could investigate the combination of multi-modal datasets (system sensor logs, network traffic captures, system sensor information, etc.) for anomaly detection.

Multi-modal learning can potentially improve attack detection accuracy and reduce the false positive rate, as it allows the framework to learn attack patterns across multiple data modalities. An exciting avenue for future exploration is to look into hardware-specific optimizations for more prevalent edge platforms, which would simplify deployment and energy measurement collection and therefore yield more accurate energy-aware scheduling optimization results.

Future plans: Despite now including the CIC-IDS2017 dataset, corpora focused exclusively on IoT (IoTID20, TON_IoT) should be used for validation later on. One other direction that seems interesting is leveraging time-series forecasting approaches to design predictive scheduling schemes able to preemptively migrate workloads before resource exhaustion and further increase the framework's adaptability. Finally, future work should consider the use of federated learning in this context, training models in a distributed manner across the various edge nodes to maintain privacy and expand the use of the framework to other sensitive IoT areas.

6. Conclusion

This research focused on applying deep learning inference on constrained IoT infrastructure without sacrificing real-time performance. The gap described in Section 2 was that previous attempts to adapt model complexity, offloading venue, and tier placement have not been tied together under a common real-time context monitor. To fill this gap, the Deep Learning-Driven Adaptive Edge Intelligence Framework (DLA-EIF) couples four system components: a small ML classifier based on a multi-layer perceptron, a Deep Q-Learning offloading agent, a three-tier placement architecture (edge, fog, and cloud), and a real-time context monitor. The four objectives outlined in Section 1 are addressed as follows. First, a small deep learning classifier for tabular IoT flow data was implemented and found to outperform baseline classifiers Random Forest, SVM, and XGBoost with 0.974 accuracy, 0.981 precision, 0.977 recall, 0.979 F1 on NSL-KDD, and ran within limits similar to those of other machine learning baseline classifiers; the classifier outperforms the published CNN-LSTM solution across these measures. Second, the problem of model offloading was transformed into an MDP with a 10-dimensional continuous state, three action choices, and a weighted reward of latency, energy, deadline violation, and accuracy, and solved by a Deep Q-Learning offloading agent that learned decision-making policies that placed the model at the edge for 58.5%, fog for 29.0%, and cloud for 12.5% of the test; the average cost was 12% lower than a threshold-based policy and 7% lower than a greedy selection heuristic. Third, a hierarchical placement model was implemented, including a deadline-dependent offloading algorithm (Algorithm 1) and a design of tiers with hardware class, tasks, communication pattern, and fallback mode. Fourth, the combined system was tested on NSL-KDD and CIC-IDS2017 in simulated and hardware-in-the-loop setups; the latter measured 1.9–2.8 ms and 0.0016–0.0023 J per inference on Raspberry Pi 4 and NVIDIA Jetson Nano boards, while the former achieved 0.961 binary, 0.921 multi-class accuracy on CIC-IDS2017, with comparable true positive rates on all classes. Predictive models on the platforms ran inference latency reduced by 97.6% and 98%, with energy reduced by 98% and 97.4% respectively, while accuracy was unaffected across tiers. The results confirm the benefits of real-time model adaptation versus static configuration. The intended outcome of this research is the integrated framework with its simulation validation, not a novel learning algorithm. More limitations: experiments are performed on table-structured data, with a small MLP; compression methods and CNN backbones for non-table-structured data have not yet been performed. Latency and energy results in Section 5.4 are simulated; hardware-in-the-loop validation has been performed only on two platforms; IoT-oriented corpora such as IoTID20 and TON_IoT have not been validated. Modern IoT corpora, hardware-aware optimization, predictive scheduling and federated learning should be performed.

Acknowledgements

The Authors would like to thank Mustansiriyah University (<https://uomustansiriyah.edu.iq>) in Baghdad, Iraq, for its support in the present work.

Ethical issue

The author is aware of and complies with best practices in publication ethics, specifically regarding authorship (avoidance of guest authorship), dual submission, manipulation of figures, competing interests, and compliance

with research ethics policies. The author adheres to publication requirements that the submitted work is original and has not been published elsewhere. All survey participants provided informed consent before participating, and the anonymity and confidentiality of all respondents were strictly maintained throughout the research process.

Data availability statement

The manuscript contains all the data. However, additional data will be provided by the corresponding author upon reasonable request.

Conflict of interest

The author declares no potential conflict of interest.

References

- [1] M. J. C. S. Reis and C. Seródio, "Edge AI for real-time anomaly detection in smart homes," *Future Internet*, vol. 17, no. 4, p. 179, 2025. DOI: 10.3390/fi17040179.
- [2] D. Oliveira, "Real-time anomaly detection at the edge using embedded AI and IoT," *The Internet of Things*, 2023. [Online]. Available: <https://theinternetofthings.eu/wp-content/uploads/2023/12/Real-Time-Anomaly-Detection-at-the-Edge.pdf>
- [3] C. M. Patel, "Edge computing for low-latency IoT applications in smart cities," *Smart Internet of Things*, 2024. [Online]. Available: <https://siot.reapress.com/journal/article/view/251>
- [4] S. Jadhav and A. Kulkarni, "Comprehensive survey on detection of anomalies in edge computing: Network and deep learning solutions," in *Proc. SciTePress*, 2024. [Online]. Available: <https://www.scitepress.org/Papers/2024/133441/133441.pdf>
- [5] M. V. Ngo, T. Luo, and T. Q. S. Quek, "Adaptive anomaly detection for Internet of Things in hierarchical edge computing: A contextual-bandit approach," *arXiv preprint arXiv:2108.03872*, 2021. [Online]. Available: <https://arxiv.org/abs/2108.03872>
- [6] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep anomaly detection for time-series data in industrial IoT: A communication-efficient on-device federated learning approach," *arXiv preprint arXiv:2007.09712*, 2020. [Online]. Available: <https://arxiv.org/abs/2007.09712>
- [7] C. Ni, J. Wu, and H. Wang, "Energy-aware edge computing optimization for real-time anomaly detection in IoT networks," *Applied and Computational Engineering*, 2025. [Online]. Available: <https://direct.ewa.pub/proceedings/ace/article/view/22280>
- [8] V. Chowdary Manduva, "AI-powered real-time anomaly detection in edge computing systems for smart cities," *International Journal of Engineering and Computer Science*, vol. 13, no. 1, 2025. DOI: 10.18535/ijecs.v13i01.4819.
- [9] T. Wang, Y. Zhang, X. Liu, Z. Chen, and H. Li, "Edge-cloud synergy for AI-enhanced sensor network data: A real-time predictive maintenance framework,"

- Sensors, vol. 24, no. 24, p. 7918, 2024. DOI: 10.3390/s24247918.
- [10] P. Warden and D. Situnayake, TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers. Sebastopol, CA, USA: O'Reilly Media, 2020.
- [11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017. [Online]. Available: <https://arxiv.org/abs/1704.04861>
- [12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.
- [13] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in Proc. European Conf. Computer Vision (ECCV), 2018, pp. 116–131. DOI: 10.1007/978-3-030-01264-9_8.
- [14] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and Internet-of-Things devices," in Proc. 2016 Int. Workshop on Internet of Things Towards Applications (IoT-App '16), 2016, pp. 7–12. DOI: 10.1145/2991561.2991563.
- [15] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, 2016. DOI: 10.1109/JIOT.2016.2579198.
- [16] W.-Q. Ren, Y.-B. Qu, C. Dong, Y.-Q. Jing, H. Sun, Q.-H. Wu, and S. Guo, "A survey on collaborative DNN inference for edge intelligence," Machine Intelligence Research, vol. 20, no. 3, pp. 370–395, 2023. DOI: 10.1007/s11633-022-1391-7.
- [17] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in Proc. IEEE 37th Int. Conf. Distributed Computing Systems (ICDCS), 2017, pp. 328–339. DOI: 10.1109/ICDCS.2017.226.
- [18] G. Premasankar, M. Di Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," IEEE Internet of Things Journal, vol. 5, no. 2, pp. 1275–1284, 2018. DOI: 10.1109/JIOT.2018.2805263.
- [19] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30–39, 2017. DOI: 10.1109/MC.2017.9.
- [20] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," IEEE Internet of Things Journal, vol. 7, no. 8, pp. 7457–7469, 2020. DOI: 10.1109/JIOT.2020.2979684.
- [21] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322–2358, 2017. DOI: 10.1109/COMST.2017.2745201.

- [22] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," Future Generation Computer Systems, vol. 79, pp. 849–861, 2018. DOI: 10.1016/j.future.2017.09.020.
- [23] S. Iftikhar, S. S. Gill, C. Song, M. Xu, M. S. Aslanpour, A. N. Toosi, J. Du, H. Wu, S. Ghosh, D. Chowdhury, M. Golec, M. Kumar, A. M. Abdelmoniem, F. Cuadrado, B. Varghese, and O. Rana, "AI-based fog and edge computing: A systematic review, taxonomy and future directions," ACM Computing Surveys, vol. 56, no. 2, pp. 1–38, 2023. DOI: 10.1145/3596599.



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

Appendix

Numeric-only variant (41 features): duration, src_bytes, dst_bytes, land, wrong_fragment, urgent, hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login, count, srv_count, error_rate, srv_error_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_error_rate, dst_host_srv_error_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate, + 3 derived numeric features.