Fault-Resilient Adaptive Offloading with Priority Buffers for Early-Exit Deep Neural Networks

Ricardo Soares da Silva*, Pedro Cintra Silveira‡, Roberto G. Pacheco*,
Daniel Sadoc Menasché‡, Miguel Elias M. Campista‡
*Universidade Federal Fluminense
{ri_soares; robertopacheco}@id.uff.br
‡Universidade Federal do Rio de Janeiro
{pedrocintra, miguel}@gta.ufrj.br, sadoc@dcc.ufrj.br

***Abstract*—Early-exit Deep Neural Networks (EE-DNNs) embed intermediate classifiers along the model architecture, enabling local inference when the confidence of an early exit surpasses a threshold. While this design reduces dependence on cloud processing, fixed thresholds are unable to adapt to performance degradation caused by transient *software faults* or *hardware–software interaction faults*. Such faults may manifest as latency spikes, processing slowdowns, or increased drop rates, especially in edge–cloud collaborative inference scenarios. This work proposes a fault-resilient adaptive offloading framework that combines multi-armed bandit (MAB) threshold selection with a priority-based input buffer. The adaptive controller responds to varying conditions, including simulated network degradation, cloud-side slowdowns, and burst overloads, by jointly optimizing the accuracy–latency trade-off and ensuring timely processing of high-priority inputs. We evaluate three buffer policies—FIFO, Strict Priority, and a Hybrid Priority-Admission FIFO—in a simulation-based environment using MobileNetV2 with a single early exit. Results show that the hybrid policy consistently achieves higher goodput and lower drop rates under fault-injection scenarios, demonstrating the benefits of combining adaptive thresholding with priority-aware buffering for robust EE-DNN operation in fault-prone edge–cloud systems.**

***Index Terms*—Early-Exit Deep Neural Networks, multi-armed bandits, adaptive thresholding, priority-aware buffering, fault resilience, edge–cloud computing**

I. Introduction

Deep Neural Networks (DNNs) have achieved state-of-the-art performance in a wide range of tasks [1], [2], but their computational demands and inference latency make deployment challenging in *fault-prone, latency-sensitive* environments. In edge–cloud collaborative inference, parts of the model are executed locally on the edge device while the remainder is processed in the cloud [3], [4]. This architecture can be disrupted by *software faults* (e.g., misconfigurations, resource leaks) and *hardware–software interaction faults* (e.g., network interface degradation, memory bottlenecks), which increase inference delays, reduce throughput, and cause input drops.

One promising approach for mitigating these effects is the *Early-Exit DNN (EE-DNN)* architecture [5], [6], which inserts intermediate classifiers (side branches) into the main network. When intermediate classification confidence exceeds a threshold, inference can terminate locally, avoiding further computation and transmission. This flexibility can reduce the impact of fault-induced slowdowns in remote processing. However, most EE-DNN systems use *fixed thresholds*, which cannot adapt to dynamic fault conditions. A static policy may either overload the cloud when thresholds are too high or unnecessarily sacrifice accuracy when thresholds are too low.

In this work, we enhance EE-DNN fault resilience through:

1) **Adaptive Threshold Selection via Multi-Armed Bandits (MABs):** A UCB-based algorithm dynamically selects the early-exit confidence threshold in response to changing system conditions.
2) **Priority-Aware Buffer Management:** A Hybrid Priority-Admission FIFO policy that preserves high-priority inputs under overload, ensuring critical data is processed even when faults reduce system capacity.

Beyond the inherent computational constraints of edge devices, real deployments often face unpredictable runtime conditions. For instance, in mobile and IoT-based vision systems, wireless link instability may cause large fluctuations in transmission delays, while edge servers co-located with other workloads can suffer from contention in CPU, GPU, or memory resources. These factors create a dynamic operating context in which inference latency, drop rates, and accuracy may vary abruptly over short timescales. Static design choices, such as fixed early-exit thresholds, are ill-suited to such conditions and may lead to either excessive offloading—overburdening the cloud and increasing latency—or overly aggressive local inference that sacrifices accuracy.

While recent studies have proposed methods for early-exit threshold selection [7], [8], most have been evaluated under relatively stable conditions, with little attention paid to the interaction between threshold adaptation and buffer management under faults. Likewise, research on priority-aware buffering in edge–cloud AI pipelines [9] has not considered the feedback loop between queue state and inference policy selection. Our work addresses these gaps by jointly considering threshold selection and buffer admission policies in a framework that explicitly models transient faults and performance degradation, thus offering a more resilient design for fault-prone edge–cloud deployments.

Our simulation results with MobileNetV2 show that the proposed hybrid policy maintains higher throughput and lower drop rates compared to FIFO and Strict Priority baselines, particularly under severe fault conditions. These findings demonstrate that integrating adaptive thresholding with priority-aware buffering can significantly improve EE-DNN reliability in fault-prone distributed AI deployments.

The remainder of this paper is organized as follows. Section II reviews the state of the art in Early-Exit DNNs, priority-aware buffering strategies, and techniques for handling software and hardware–software interaction faults. Section III presents the proposed adaptive offloading framework, detailing the EE-DNN model, the Multi-Armed Bandit formulation for threshold selection, and the overhead modeling approach. Section IV describes the input buffer management policies evaluated in this work. Section V reports the simulation-based experimental results, including performance under fault-injection scenarios. Finally, Section VI summarizes our main findings and outlines directions for future research.

TABLE I
TABLE OF NOTATION

Symbol	Description
$\boldsymbol{x}$	Input sample to the EE-DNN model.
$\boldsymbol{z}_I$	Intermediate feature vector at the side branch.
$\boldsymbol{p}_I(\boldsymbol{x})$	Probability vector from the side branch classifier.
$C_I(\boldsymbol{x})$	Intermediate classification confidence.
$C_L(\boldsymbol{x})$	Final classification confidence (cloud output).
α	Early-exit confidence threshold.
α^*	Optimal early-exit confidence threshold.
$\mathcal{A}$	Set of possible confidence threshold actions.
o	Processing overhead for offloading to the cloud.
$o(q)$	Overhead as a function of buffer occupancy q.
$\Delta C(\boldsymbol{x})$	Gain in confidence from processing to the final layer.
$r(\alpha_t)$	Instantaneous reward for action α_t.
$\mathbb{E}[r(\alpha)]$	Expected reward for threshold α.
$N_t(\alpha)$	Number of times threshold α is selected up to round t.
$Q_t(\alpha)$	Average reward for threshold α at round t.
λ	Input arrival rate (Poisson process).
B	Buffer capacity (number of samples).

II. Related Work

This section reviews prior work relevant to our contributions, organized into three areas: Early-Exit Deep Neural Networks, priority-based buffering in inference systems, and the impact of software and hardware–software interaction faults on system performance.

A. Early-Exit Deep Neural Networks

The use of Early-Exit Deep Neural Networks (EE-DNNs) has been extensively explored to reduce inference latency and computational cost. Foundational architectures such as BranchyNet [5] introduced intermediate exit points, with early exit decisions driven by entropy-based confidence. Subsequent works, including SPINN [6] and SEE [10], refined this approach by leveraging classification confidence to determine early termination points.

Other research has adapted early-exit mechanisms for different objectives, such as selecting optimal model depth in FlexDNN, implementing latency-aware partitioning strategies like Edgent [4], and optimizing for energy efficiency via FPGA-based designs, as seen in DynExit [11]. These methods demonstrate that intermediate classifiers can yield substantial efficiency gains, but most rely on static thresholds, limiting adaptability under changing runtime conditions.

B. Priority-Aware Buffering in Inference Systems

The use of input buffers to enhance EE-DNN performance has emerged more recently. Pacheco *et al.* [12] proposed a calibration-aided partitioning method where buffer backlog influences early-exit decisions. Priority policies are long-established in queueing theory [13], yet systematic evaluations of different priority schemes in EE-DNN buffers remain limited [7], [8].

Our work builds on these ideas by comparing FIFO, Strict Priority, and a hybrid Priority-Admission FIFO in a unified framework. We also integrate buffer state into the decision-making loop of a Multi-Armed Bandit (MAB) controller, extending adaptive threshold selection methods such as AdaEE [14] and UCBEE [15].

C. Software Faults and Hardware–Software Interaction Faults

While early-exit and buffering strategies focus on performance optimization, relatively little work addresses *fault resilience* in EE-DNN systems. In distributed AI pipelines, *software faults*—such as scheduler misconfigurations, resource leaks, or driver bugs—and *hardware–software interaction faults*—such as network interface degradation or memory contention—can degrade performance or cause dropped inputs.

Fault injection and simulation are common in software reliability research [3], but have not been widely applied to EE-DNN performance evaluation. Our work closes this gap by explicitly modeling such faults in simulation, including network degradation, cloud-side processing slowdowns, and burst arrivals. By integrating MAB-based adaptation with priority-aware buffering, we provide a fault-resilient inference framework envisioning high goodput and service quality under degraded operating conditions.

III. Early-Exit DNN and MAB Framework

A. Early-Exit DNN Model

Figure 1 illustrates an EE-DNN model with a single side branch employed in a collaborative inference scenario between the cloud and the edge. In this scenario, after receiving an input $\boldsymbol{x}$, the EE-DNN model processes it layer-by-layer until reaching the side branch (intermediary classifier), where the side branch computes the intermediate feature vector $\boldsymbol{z}_I$. Then, a softmax function is applied to obtain an intermediate probability vector given by $\boldsymbol{p}_I(\boldsymbol{x}) = \text{softmax}(\boldsymbol{z}_I(\boldsymbol{x}))$. Each component of $\boldsymbol{p}_I(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ belongs to a specific class. From this vector, the intermediate classification confidence for the input $\boldsymbol{x}$ is calculated as $C_I(\boldsymbol{x}) = \max(\boldsymbol{p}_I(\boldsymbol{x}))$.

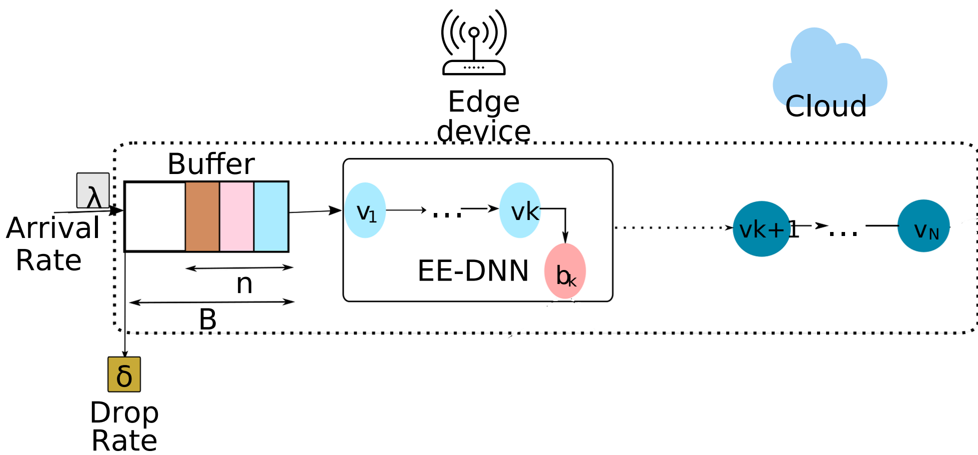


Fig. 1. EE-DNN with a single side branch.

If this confidence meets or exceeds a given threshold α, such that $C_I(\boldsymbol{x}) \geq \alpha$, the side branch classifies the input as $\hat{y} = \arg\max(\boldsymbol{p}_I(\boldsymbol{x}))$, and the inference process terminates. Otherwise, if $C_I(\boldsymbol{x}) < \alpha$, the input is processed by the subsequent layers until the final layer at the cloud server, which generates a final confidence score $C_L(\boldsymbol{x})$. This additional processing introduces an overhead, denoted by o, which may become significant depending on the system conditions.

In fully mobile scenarios, where the entire model is executed on the mobile device, the overhead is primarily associated with the execution time of the subsequent layers and potentially with energy consumption. In our edge–cloud collaborative inference scenario, the overhead includes both the communication time required to transfer data from the edge device to the cloud and the processing time of the remaining layers in the cloud.

It is important to note that in real-world deployments, the inference process can also be affected by *software faults* and *hardware–software interaction faults* [16], [17]. Examples include scheduler misconfigurations, resource leaks, and network stack bugs, as well as faults arising from **adverse environmental conditions or hardware malfunctions**. Such faults can introduce additional delays, packet loss, or even inference failures, effectively altering the perceived overhead o.

While prior EE-DNN studies often assume fault-free execution, our work is motivated by these effects. We model them in simulation by exposing the system to a stream of low-quality inputs (**High Ambiguity scenario**) to test its resilience, in contrast to its performance under nominal conditions (**Low Ambiguity scenario**), aligning with established practices in fault injection [17], [18].

Although EE-DNNs can feature multiple side branches, this work considers only a single lateral branch to simplify the analysis. Specifically, we adopt a MobileNetV2 [19] architecture with a single side branch as the EE-DNN model for our study.

B. MAB Formulation for Threshold Selection

Traditionally, EE-DNN models employ a fixed confidence threshold α. Nevertheless, they cannot adapt the context changing in real-world applications as explained previously. To this end, these applications require context-adaptive thresholds. To dynamically select the optimal confidence threshold α^* according to the context, we model the problem using a Multi-Armed Bandit (MAB) framework early-exit decisions.

Model: after the edge device receives a given input image $\boldsymbol{x}$, the algorithm chooses an action $\alpha \in \mathcal{A}$, with each action of the bandit corresponding to a discrete confidence threshold value, where $\mathcal{A}$ is the set of possible actions. At this stage, it is important to emphasize that $C_I(\boldsymbol{x})$ and $C_L(\boldsymbol{x})$ are unknown until the input is processed by the DNN model. Then, we compare the intermediary-layer confidence $C_I(\boldsymbol{x})$ to the chosen threshold α. If $C_I(\boldsymbol{x}) \geq \alpha$, the inference is concluded and the input $\boldsymbol{x}$ is classified at the side branch. Otherwise, $C_I(\boldsymbol{x}) < \alpha$, the input is offloaded to the cloud. Thus, the pair $C_I(\boldsymbol{x})$ and α decide if the early exit occurs.

Upon selecting an action α, an instantaneous reward $r(\alpha)$ is associated with the threshold. The instantaneous reward function, $r(\alpha_t)$, is defined as follows:

$$r(\alpha_t) = \begin{cases} 0, & \text{if } C_I(\boldsymbol{x}) \geq \alpha \\ \Delta C - o, & \text{otherwise,} \end{cases} \tag{1}$$

where $\Delta C(\boldsymbol{x}) = \max(C_L(\boldsymbol{x}) - C_I(\boldsymbol{x}), 0)$ represents the potential gain in confidence from processing the sample to the final layer. The overhead o represents the cost associated with this additional processing and associated delays. In this work, the overhead o is normalized to be commensurate with ΔC.

Based on the instantaneous $r(\alpha_t)$, we can compute the expected reward $\mathbb{E}[r(\alpha)] = \mathbb{E}[\Delta C(\boldsymbol{x}) - o | C_I < \alpha] \cdot P[C_I(\boldsymbol{x}) < \alpha]$. Our goal is to find an optimal confidence threshold α^* that achieves the optimal expected reward given by $\alpha^* = \arg\max_{\alpha \in \mathcal{A}} \mathbb{E}[r(\alpha)]$. We define the performance of selecting a policy Π_{MAB} over $n \in \mathbb{N}$ in terms of expected regret given

Algorithm 1: UCB-based algorithm for selecting the confidence threshold

Input: Policy (a selection policy object), K (number of thresholds), o (overhead), $\tilde{c} > 0$
Initialization: Call Policy.Initialize();
For each arm $k = 1, \dots, K$, set $Q_k \leftarrow 0$ and $N_k \leftarrow 0$
1 **for** $t = 1, 2, \dots, T$ **do**
2   **if** *sample* $\boldsymbol{x}_{new}$ *arrives* **then**
3   Policy.OnSampleArrival($\boldsymbol{x}_{new}$)
4   **end**
5   **if** *device is free* **then**
6     $\boldsymbol{x} \leftarrow$ Policy.SelectSampleForProcessing()
7     $q \leftarrow$ Policy.GetQueueState()
8     $\alpha_t \leftarrow \arg\max_k (Q_k + \tilde{c} \cdot \sqrt{2 \ln t / N_k})$
9   **if** $C_I(\boldsymbol{x}) \geq \alpha_t$ **then**
10   Process locally
11       $r_t \leftarrow 0$
12   **else**
13   Offload to Cloud
14       $r_t \leftarrow \max(C_L(\boldsymbol{x}) - C_I(\boldsymbol{x}), 0) - o(q)$
15   **end**
16     $N_t(\alpha_t) = N_{t-1}(\alpha_t) + 1$
17     $Q_t(\alpha_t) = \sum_{u=1}^{t} r_u(\alpha_u) \mathbb{1}\{\alpha_u = \alpha_t\} / N_t(\alpha_t)$
18   **end**
19 **end**

by $R(\Pi_{\text{MAB}}, n) = n \cdot \mathbb{E}[r(\alpha^*)] - \sum_{t=1}^{n} \mathbb{E}[r(\alpha_t) | \Pi_{\text{MAB}}]$, where α_t denotes the confidence threshold chosen by the algorithm at the round t.

Algorithm: Among the many algorithms available for the MAB framework [20], this work employs an algorithm based on Upper Confidence Bound (UCB) [21], which is introduced in [14], [15]. Algorithm 1 presents the pseudocode of the employed algorithm (see also Figure 2).

The algorithm takes as input the trade-off parameter between exploration and exploitation, $\tilde{c}$, and the number of thresholds, K. Initially, each threshold is applied once to the first K inputs, generating one input per arm. In the subsequent rounds, whenever a new sample $\boldsymbol{x}_{\texttt{new}}$ arrives, it is stored in a buffer on the edge device (lines 2 – 4). Next, when the edge device is available, an input $\boldsymbol{x}$ is drawn from the buffer according to the priority policy described in Section IV (lines 5 – 7). At each round, the algorithm selects the threshold associated with the highest UCB index (line 8). If the intermediate-layer confidence $C_I(\boldsymbol{x})$ exceeds the selected threshold α_t, an early exit is performed (lines 9 – 11). Otherwise, the input is processed through all subsequent layers at the cloud, and the prediction from the layer with the highest confidence score is considered the final output (lines 12 – 15). The statistics $N_t(\alpha_t)$ and $Q_t(\alpha_t)$ are then updated (lines 16 – 17), where $N_t(\alpha_t)$ represents the number of times action α_t has been selected up to round t, and $Q_t(\alpha_t)$ denotes the average reward associated with action α_t at round t.

Overhead Function: we employ the concept of side information, which refers to observable variables that capture the real-time state of the system or environment, making our algorithm a contextual bandit [22].

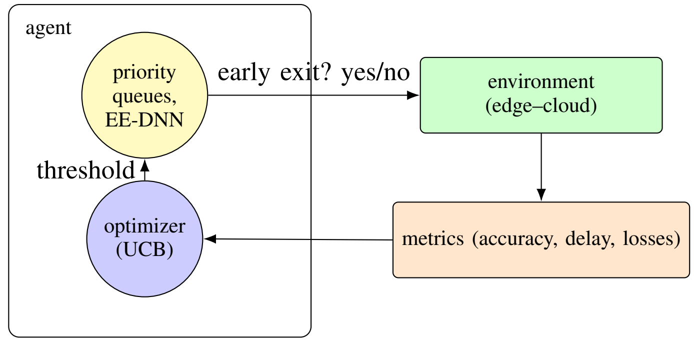


Fig. 2. Agent–environment loop for adaptive early exit in edge–cloud inference.

We will focus on the impact of overhead on buffer performance. According to Little's law [23], which states that the average queue length is proportional to the arrival rate and the average waiting time, we can express our overhead $o(q)$ as a function of buffer occupancy. Based on this, we define a quadratic overhead function $o(q)$, which captures the following components: (i) delay-related penalty that grows quadratically with queue accumulation, (ii) a per-sample cost incurred when offloading data to the cloud, and (iii) fixed infrastructure and operational costs, modeled as a constant term. The resulting overhead function is given by $o(q) = o_1 q^2 + o_2 q + o_3$, where o_1, o_2, and o_3 reflect the relative weight of each overhead component. This overhead model will serve as the basis for the adaptive offloading strategy evaluated in the experiments section.

IV. Input Buffer

Given the adaptive offloading scenario, depicted in Figure 1, each time an input arrives at an edge device, it is placed into a finite input buffer with capacity B. Moreover, we assume the input arrivals are modeled as a Poisson process with rate λ. Next, each stored sample is subsequently retrieved from the buffer and processed by the EE-DNN model. In this work, we examine the following buffer management policies (see Figure 3):

- **FIFO:** This policy employs a single buffer with a strict First-In, First-Out (FIFO) rule. Both admission and removal are sequential, making the system unaware of sample priority.
- **Strict Priority (SP):** This policy maintains a separate FIFO queue for each priority class. Each of these queues individually has the same capacity as the single buffer in the baseline FIFO system. At each service opportunity, the scheduler selects the first sample from the highest-priority non-empty queue.
- **Priority-Admission FIFO (Hybrid):** A single shared queue is used with FIFO removal order. However, admission is priority-aware: if the buffer is full, an arriving

sample can replace an existing one with strictly lower priority.

The Decision Maker (DM) must resolve the trade-off between early exits (possibly lower accuracy, lower delay) and full processing (possibly higher accuracy, higher delay), a decision made critical by the varying importance of each sample. *The objective is to maximize throughput, which prioritizes the correct classification of high-value observations.*

This creates a challenge: a full buffer risks dropping high-priority inputs. Therefore, optimizing throughput requires a policy that balances early-exit usage against the value of each specific input.

V. Experimental Results

This section evaluates the performance of our adaptive offloading system under different buffering policies, as explained in Section IV. In these experiments, we assume the inputs arrive according to Poisson with rate λ and we vary $\lambda \in [0.5, 3.0]$. The simulation environment is driven by synthetic data samples designed to emulate the output of an early-exit DNN model with a single side branch. Specifically, we employ a MobileNetV2 with a single side branch. For the experimental evaluation, we conduct 30 simulation runs per scenario, each spanning 5000 turns. The buffer capacity is fixed at $B = 7$ items for all buffer policies. This buffer capacity is a calibrated trade-off. A smaller capacity (e.g., $B < 4$) would be insufficient to absorb traffic bursts, resulting in high data loss, while a larger one (e.g., $B > 12$) would permit excessive queuing delay that would distort the learning agent's cost-sensitive reward function.

Performance metrics are weighted to reflect the intrinsic value of each service class, with inputs categorized by priority based on their certainty and importance. The chosen weights (1-Low, 2-Medium, 3-High) are critical for a fair evaluation across scenarios with shifting class distributions, as they prevent the final metric from being skewed by the sheer volume of low-priority traffic. This linear scheme quantifies the relative importance, where the value of a high-priority input is equivalent to the combined value of a low- and a medium-priority input.

The offloading cost is modeled as follows: a variable network delay and a queuing-sensitive cost function. The network delay for each transmission includes a base latency of 30 ms and a random jitter uniformly distributed in the range of ± 10 ms to simulate channel volatility. Separately, the abstract offloading cost is defined by three components: a queue accumulation penalty of $o_1 = 0.009$ which scales with buffer occupancy; a per-item cost of $o_2 = 0.02$; and a constant baseline term of $o_3 = 0.01$. These parameter values were calibrated to be commensurate with the average confidence gain of the network, $\Delta C = 0.15$, based on an independent simulation of 1000 turns with parameter calibration through regression-based curve approximation.

To assess system performance under varying conditions, we define two distinct scenarios for the generation of confidence scores. In all scenarios, scores are sampled from Normal distributions ($\mathcal{N}(\mu, \sigma)$) and subsequently clipped to the range $[0.1, 0.99]$. The scenarios are as follows:

- **Scenario 1: Low Ambiguity.** This scenario simulates the system's nominal operating conditions, where the model is well-adapted to the input data stream. The intermediate-layer confidence $C_I(\boldsymbol{x})$ is drawn from $\mathcal{N}(0.85, 0.15)$, while the final-layer confidence $C_L(\boldsymbol{x})$ is drawn from $\mathcal{N}(0.90, 0.15)$. The high mean confidence ensures a steady stream of high-quality, high-priority samples, allowing for the evaluation of the system's throughput and efficiency.
- **Scenario 2: High Ambiguity.** This scenario stress-tests the system's robustness against a sudden context shift or the influence of external failures, which results in a mass of low-quality input samples. The intermediate-layer confidence $C_I(\boldsymbol{x})$ is drawn from $\mathcal{N}(0.50, 0.15)$ and the final-layer confidence $C_L(\boldsymbol{x})$ is drawn from $\mathcal{N}(0.60, 0.15)$. The low mean confidence simulates conditions where the model's reliability is compromised, flooding the system with low-priority, high-uncertainty data. This critically tests the policy's ability to remain stable and effectively prioritize the rare high-value samples that may still occur.

System performance is quantified using the following metrics, analyzed separately for each priority level. Here, *screening* refers to an initial confidence-based assessment of an input's classification result, performed before deciding whether to process it locally or offload it to the cloud.

- **High priority:** screening score ≥ 0.8
- **Medium priority:** $0.8 >$ screening score ≥ 0.6
- **Low priority:** screening score < 0.6

The following metrics are evaluated per priority level:

- **Cloud Offloading:** proportion of samples sent to the cloud for final processing.
- **Average Latency:** mean end-to-end inference time per sample.
- **Drop Rate:** fraction of samples discarded due to buffer overflow or policy preemption.
- **Throughput:** rate of classified observations per unit time.

A. Fault Resilience

To illustrate the system's adaptive capabilities, we evaluate the response of the policy computed by Algorithm 1 against cloud failure events. Impact of the agent's offloading decision is represented by the blue Edge Usage and yellow Cloud Usage lines in Figure 4.

The analysis in this section focuses on the Hybrid policy. Since the UCB algorithm's offloading decisions are guided exclusively by queue occupancy rather than the priority of individual items, any of the buffer policies could demonstrate this adaptive behavior. The Hybrid policy was selected as a representative case due to its promising performance in our experiments.

The results demonstrate the Upper Confidence Bound (UCB) algorithm's capability to promptly respond to dynamic

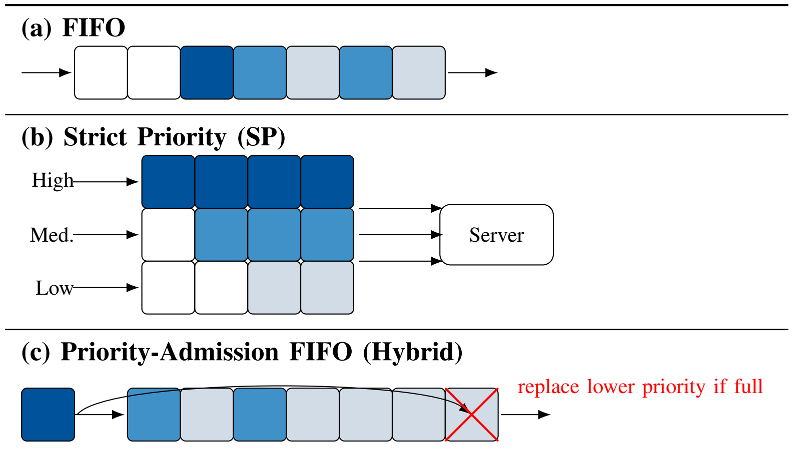


Fig. 3. Queue management policies: (a) FIFO; (b) Strict Priority (SP); (c) Priority-Admission FIFO (Hybrid).

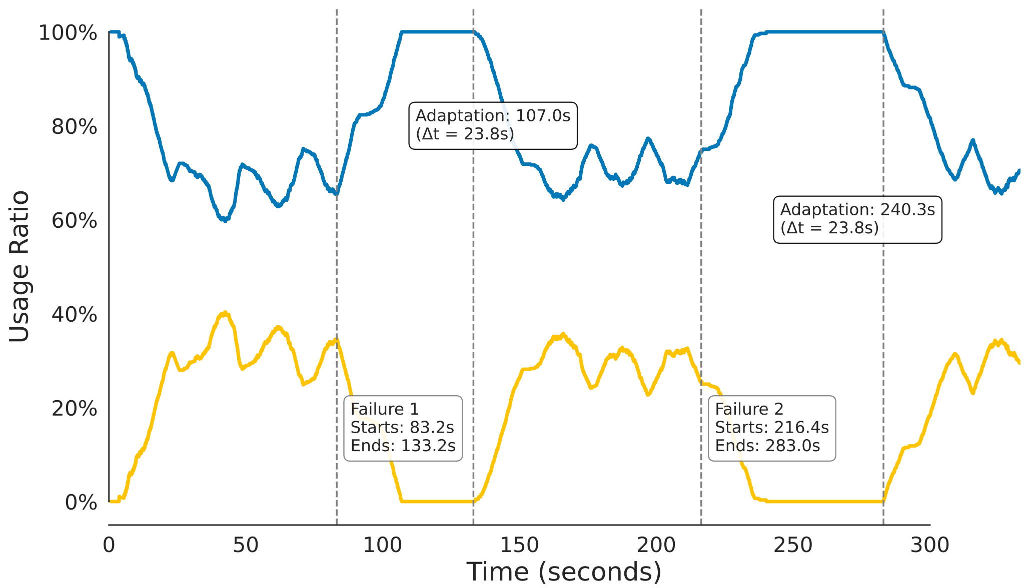


Fig. 4. Response of the Hybrid Buffer policy to cloud failures, where usage data is smoothed by a moving average. The system's immediate shift to edge-only processing is visualized as a gradual transition (Δt) whose duration is determined by the smoothing window's size.

changes in the cloud connection state. As depicted in Figure 4, the hybrid buffer reacts to each failure event by shifting its offloading policy entirely to the edge, effectively abandoning the cloud. This adaptation occurs along a measurable delay (Δt), representing the time required for the agent to accumulate sufficient evidence against the underperforming cloud option. This reaction time highlights a fundamental trade-off, which can be tuned via internal constants: a slower, smoother transition can be configured for systems sensitive to abrupt policy changes, whereas a more granular and rapid response can be prioritized for systems where immediate adaptation is critical.

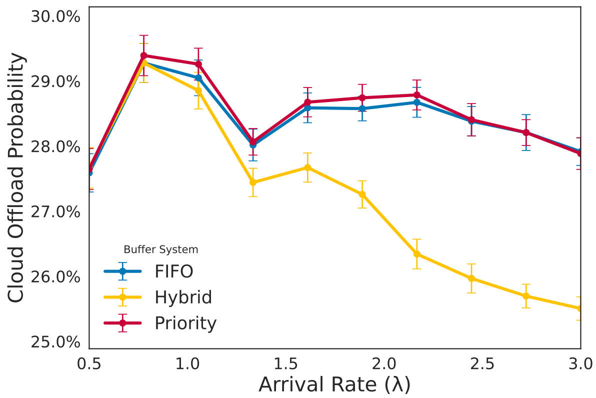
B. Cloud Offloading

The cloud offloading probability, defined as the fraction of samples fully processed in the cloud, reveals distinct behaviors across scenarios. In the Low Ambiguity scenario, all buffering strategies demonstrate a clear trend of reduced cloud utilization under increasing arrival rates (λ). The hybrid buffer, however, consistently achieves the lowest offloading probability, indicating a strong synergy between its preemptive admission policy and the Multi-Armed Bandit (MAB) learning agent.

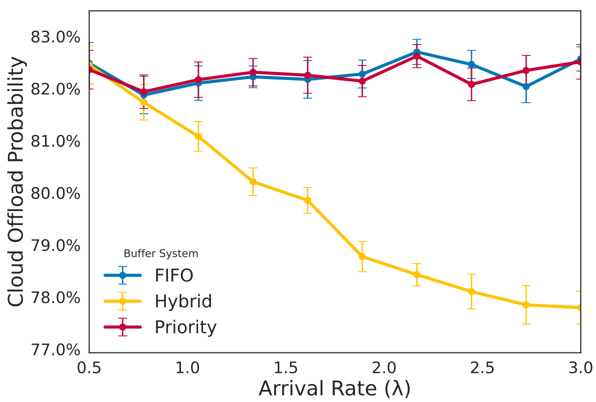
In contrast, the High Ambiguity scenario triggers a more conservative strategy, resulting in significantly higher cloud utilization across all buffer types. This behavior highlights the system's adaptive response to uncertainty; by offloading lower-confidence samples, it prioritizes the superior classification accuracy of the final-layer processor available in the cloud.

C. Inference Time

The average inference time results highlight the superior performance of the strict-priority policy. Across both sce-



(a) Low Ambiguity Scenario.



(b) High Ambiguity Scenario.

Fig. 5. Cloud Offload Probability results.

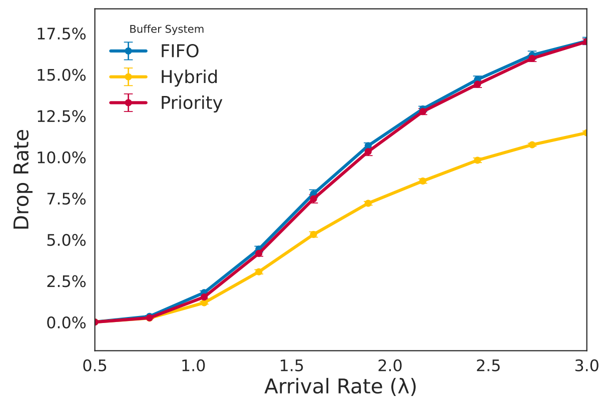
narios, this strategy consistently achieves the lowest latency, showcasing its effective response to increasing system load. This is attributed to its dedicated queue for high-priority samples, which guarantees expedited service irrespective of lower-priority traffic.

In contrast, the FIFO and hybrid buffers, both implemented on single-queue structures, exhibit tightly coupled performance characteristics. Their average latency scales predictably with the arrival rate (λ), demonstrating stable, albeit higher, service times as the system approaches saturation. This predictable increase in latency under load is a key factor that informs the MAB's offloading decisions, likely contributing to the steeper decline in cloud offloading observed for the hybrid buffer.

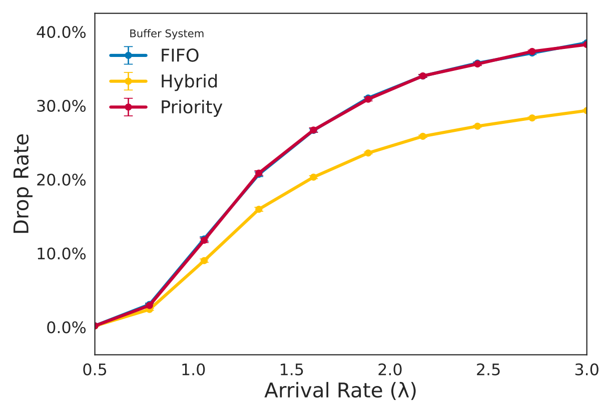
D. Drop Rate

Regarding the weighted drop rate, the hybrid buffer demonstrates markedly superior performance, consistently outperforming both the FIFO and strict-priority policies. Its effectiveness stems from a preemptive admission control mechanism: upon arrival, a high-priority sample can displace a lower-priority sample already in the queue. This policy ensures that the most critical data is preserved, resulting in the lowest overall weighted drop rate.

Interestingly, the strict-priority and FIFO buffers exhibit comparable performance under this metric. While the strict-priority architecture aggressively discards low-priority samples under load, its perfect preservation of high-priority samples is heavily rewarded by the weighted scoring system. This advantage is balanced by the indiscriminate dropping of all priority levels by the FIFO buffer, ultimately placing the two policies on par.



(a) Low Ambiguity Scenario.



(b) High Ambiguity Scenario.

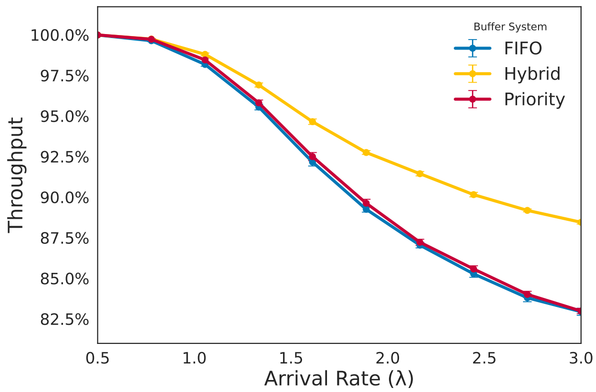
Fig. 6. Drop Rate results.

E. Throughput

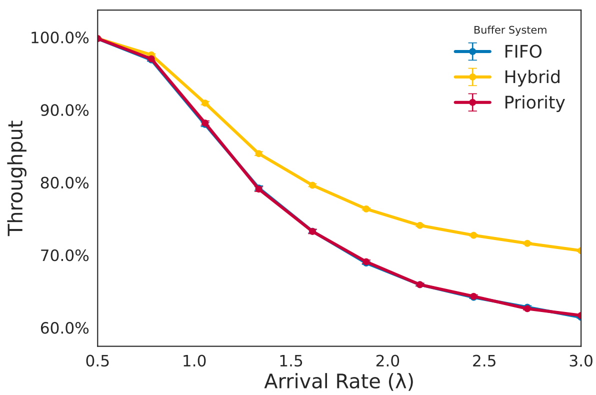
Finally, the weighted throughput results confirm the hybrid buffer's superior adaptability to increasing demand. This performance stands in contrast to the FIFO and strict-priority policies, which again exhibit tightly coupled behavior and a more pronounced degradation in throughput as the system load intensifies. The hybrid model's ability to maintain higher throughput underscores the benefits of its preemptive policy in maximizing system utility.

VI. Conclusion and Future Work

This paper investigated the impact of different priority policies on output buffers for Early-Exit Deep Neural Networks (EE-DNNs) in an edge–cloud collaborative inference setting. Our results show that a hybrid policy, combining a conventional First-In, First-Out (FIFO) buffer with preemption mechanisms, yields promising results for cloud offloading



(a) Low Ambiguity Scenario.



(b) High Ambiguity Scenario.

Fig. 7. Throughput results.

applications. Strict priority systems also prove effective in scenarios that require uninterrupted processing of high-priority samples.

A key motivation for this study stems from the observation that software faults and hardware–software interaction faults—such as scheduler misconfigurations, resource leaks, and network stack bugs—can unpredictably affect inference latency and throughput. By designing priority-aware buffering and adaptive offloading strategies, we aim to increase resilience to such performance-degrading conditions.

For future work, we plan to extend the buffer management framework to support multiple priority levels and evaluate it in other application domains, such as text classification and real-time video analytics. Additionally, we intend to investigate a broader range of Multi-Armed Bandit (MAB) algorithms, including contextual approaches like LinUCB and Thompson Sampling, with reward functions explicitly designed to account for fault-induced performance variability [22]. Finally, we envision integrating our policies into fault-injection testbeds to assess their robustness under realistic fault scenarios before deployment in production environments.

Acknowledgment

This work was funded by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, under grants 403601/2023-1, 310234/2025-5, SEI:01300.010794/2023-1 and 408255/2023-4. It was also partially sponsored by CAPES (Finance Code 001) and FAPERJ through grants CNE E-26/204.268/2024 and E-26/200.380/2023.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
[4] Y. Kang, J. Hauswald, C. Gao, A. Rovinski *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM Computer Architecture News*, vol. 45, 2017, pp. 615–629.
[5] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *IEEE Int. Conf. on Pattern Recognition (ICPR)*, 2016, pp. 2464–2469.
[6] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: synergistic progressive inference of neural networks over device and cloud," in *ACM MobiCom*, 2020, pp. 1–15.
[7] G. Casale and M. Roveri, "Scheduling inputs in early exit neural networks," *IEEE Transactions on Computers*, 2023.
[8] Y. Chen, Z. Niu, M. Roveri, and G. Casale, "Ceed: Collaborative early exit neural network inference at the edge," in *IEEE INFOCOM*, 2025, pp. 1–10.
[9] X. Wang, J. Lv, A. Slowik, B.-G. Kim, B. Parameshachari, K. Li, and G. Feng, "Augmented intelligence of things for priority-aware task offloading in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 11, no. 22, pp. 36 002–36 013, 2024.
[10] Z. Wang, W. Bao *et al.*, "SEE: Scheduling early exit for mobile dnn inference during service outage," in *ACM Conf. Modeling, Analysis and Simulation of Wireless and Mobile Sys.*, 2019, pp. 279–288.
[11] M. Wang, J. Mo, J. Lin, Z. Wang, and L. Du, "Dynexit: A dynamic early-exit strategy for deep residual networks," in *IEEE Int. Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 178–183.
[12] R. Pacheco, R. Couto, and O. Simeone, "Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks," in *IEEE Int. Conf. Communications (ICC)*, 2021, pp. 1–6.
[13] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications*. New York: Wiley-Interscience, 1976.
[14] R. G. Pacheco, M. Shifrin, R. S. Couto, D. S. Menasché, M. K. Hanawal, and M. E. M. Campista, "AdaEE: Adaptive early-exit DNN inference through multi-armed bandits," in *ICC 2023-IEEE International Conference on Communications*. IEEE, 2023, pp. 3726–3731.
[15] R. G. Pacheco, D. J. Bajpai, M. Shifrin, R. S. Couto, D. S. Menasché, M. K. Hanawal, and M. E. M. Campista, "UCBEE: A Multi Armed Bandit Approach for Early-Exit in Neural Networks," *IEEE Transactions on Network and Service Management*, 2024.
[16] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, pp. 11–33, 2004.
[17] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.
[18] J. Carreira, M. Vieira, and H. S. Madeira, "A survey of fault injection techniques for cloud computing resilience evaluation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–36, 2018.
[19] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *IEEE Conf. Comp. Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.
[20] A. Slivkins *et al.*, "Introduction to multi-armed bandits," *Foundations and Trends® in Machine Learning*, vol. 12, no. 1-2, pp. 1–286, 2019.
[21] P. Auer *et al.*, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, p. 235, 2002.
[22] A. Beygelzimer, J. Langford, L. Li, L. Reyzin, and R. Schapire, "Contextual bandit algorithms with supervised learning guarantees," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR, 2011, pp. 19–26.
[23] J. D. C. Little, "Or forum—Little's Law as viewed on its 50th Anniversary," *Operations Research*, vol. 59, no. 3, pp. 536–549, 2011.