A Hierarchical Attack Detection Solution for 6G MEC-based IoT Networks

Luiz Henrique B. A. da Silva*, José R. de Souza Silva*, Caio B. B. de Souza*†, and Andson M. Balieiro*
*Centro de Informática (CIn), Universidade Federal de Pernambuco (UFPE), Recife, Brazil.
† Universidade de Pernambuco (UPE), Brazil.
{lhbas, jrss, cbbs, amb4}@cin.ufpe.br, caiobruno.souza@upe.br

Abstract*—The growth of Internet of Things (IoT) devices and the emergence of sixth-generation (6G) networks introduce new cybersecurity challenges in Multi-access Edge Computing (MEC) environments. This work proposes and implements a two-phase hierarchical Intrusion Detection System (IDS) for IoT/MEC environments: Phase 1 runs a binary LightGBM classifier on the edge node, optimized by the F_2-score, and Phase 2 classifies the attack type with a multiclass LightGBM, both fed by a pipeline that converts PCAP captures from two CIC datasets into a 190 GB SQLite database with *netflower*, a flow extractor developed for this project. To keep detection from stalling packet capture under flooding, inference is decoupled from capture: a separate worker consumes flows from an unbounded queue and runs both phases in batches. In a fully scripted testbed validation on a VIM 4 edge node covering all eight attack categories, detection is scored *per flow*, labelling each flow by the attack window in which it started, which makes the metric independent of the flow-extractor emission delay (median 181–428 s). Over a calibrated ($+2$ s) window the binary IDS reaches a precision of 97% at a recall of 70% (F1 81%), strongest on volumetric floods (spoofing 95%, DoS 90%) and weaker on DDoS (56%) and low-volume attacks, whose single-packet flows fall below the 0.9 threshold. Phase 2 separates reconnaissance (F1 75%) but merges the remaining categories onto DoS (macro-F1 19%). Most consequentially, the measurements expose a throughput ceiling: at $\sim$1,050 flows/s the VIM 4 cannot process traffic in real time and buffers it, with a mean end-to-end latency of 116 s for the binary pipeline and 343 s for the hierarchical one, adding Phase 2 roughly tripling this latency while raising average power from 3.4 to 5.1 W.

***Index Terms*—6G, MEC, IoT, intrusion detection, LightGBM, edge computing.**

I. Introduction

The convergence of sixth-generation (6G) mobile networks, the Internet of Things (IoT) and Multi-access Edge Computing (MEC) promises to transform the digital communication infrastructure, with estimated densities of 10^6 IoT devices per km² and latencies below 1 ms, where MEC acts as an essential technology to meet these requirements [1].

The massive proliferation of IoT devices expands the cyberattack surface and introduces severe security challenges, as these endpoints with limited resources cannot support traditional, computationally demanding security solutions. Consequently, Intrusion Detection Systems (IDSs) relying on rules and signatures have been shifted to edge nodes to bring threat detection closer to the network periphery, ensuring high detection accuracy for known threats [2]. Nevertheless, novel attacks and zero day variants can easily evade signature detection mechanisms, requiring frequent manual updates and human intervention, which makes this paradigm fundamentally incompatible with the 6G scale.

Machine learning (ML)-based IDSs deployed at the network edge have emerged as a promising alternative, capable of classifying network flows in real time and generalizing to unseen attack patterns with low inference overhead [3]. For instance, classifiers trained on Canadian Institute for Cybersecurity (CIC) IoT datasets have achieved high detection accuracy in [4], [5]. Furthermore, recent works propose collaborative edge-cloud architectures [6] and two-stage frameworks that filter traffic using a lightweight classifier before activating more computationally intensive models [7]. Despite these advances, critical challenges remain. Most existing studies rely on datasets collected in controlled environments, risking severe distribution shifts when deployed under real network conditions. Many solutions are evaluated in cloud environments, thereby overlooking the strict energy and computational constraints of actual embedded devices. Additionally, the attack heterogeneity, coupled with conflicting optimization objectives, strongly motivates the adoption of hierarchical approaches.

To address these challenges, this work proposes a two-phase hierarchical intrusion detection system (IDS) for IoT/MEC environments. Phase 1 utilizes a lightweight binary classifier to isolate malicious traffic, which then triggers a Phase 2 multiclass classifier to categorize the specific attack type. The primary contributions of this paper include the design of this hierarchical, ML-based architecture, alongside a reproducible, SQLite-based data pipeline optimized for large-scale balanced sampling. Furthermore, this work introduces *NetFlower*, a parallel tool for converting PCAP traces into network flows, which has been made publicly available on PyPI. The final core contribution consists of an experimental validation on a Khadas VIM4 embedded platform under real online traffic, comprehensively evaluating CPU, RAM, energy, and throughput, which characterizes the throughput ceiling of the hierarchical pipeline on an ARM edge node under volumetric attack and empirically identifies the point where the 55 adopted flow features lose their discriminatory effectiveness.

The remainder of the paper covers related work (Section II), the methodology (Section III), the results (Section IV) and the conclusion (Section V).

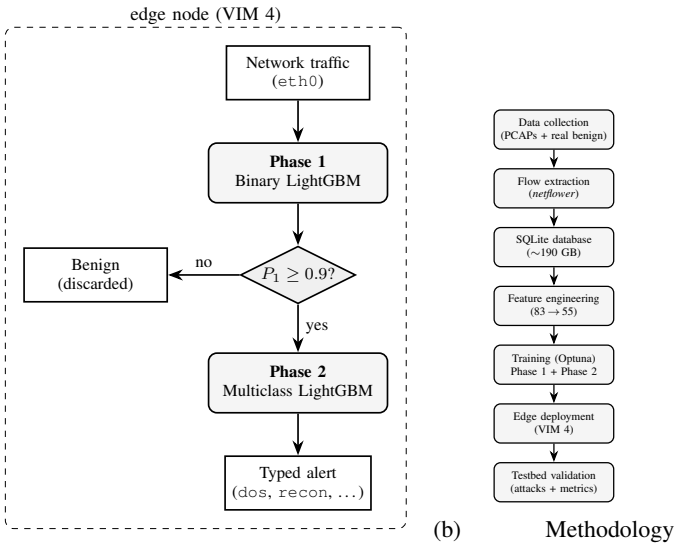
TABLE I: Positioning of this work with respect to the literature.

Work	Real edge deployment	Online evaluation	On-device cost	Multi-stage	6G focus
Houichi et al. [4]	No	No	No	No	No
Chen et al. [5]	No	No	No	No	No
Gyamfi and Jurcut [3]	*literature review*				
Yang et al. [6]	Partial	No	No	Yes	No
Aldaej et al. [7]	Yes	No	No	Yes	No
Chinnasamy et al. [10]	No	No	No	No	Yes
This work	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

II. Related Work

IDS can be signature-based, which identify known patterns with high precision but cannot detect new threats, or anomaly-based, which learn a profile of normal behavior and flag deviations [2]. Combining both approaches in hierarchical architectures is a interesting alternative for resource-constrained environments such as IoT devices and MEC nodes [8]. Regarding anomaly-based solutions, CIC datasets have been widely used to train and evaluate IDS in IoT networks. For instance, by adpting the CIC IoT Dataset 2023 [9], which reproduces a topology of 105 devices with 33 attacks in seven categories, Houichi et al. [4] compare classical classifiers for smart-city security, and Chen et al. [5] combine LightGBM with feature selection by an improved salp swarm algorithm. These works, however, report only offline metrics on the dataset's own test split, without evaluating the model deployed on a real edge node or the shift between dataset and production traffic. Gyamfi and Jurcut [3] review IDS design combining MEC, machine learning and datasets, and Yang et al. [6] propose detection through cloud–edge collaboration. In such architectures, however, the heaviest stage is typically moved to the cloud, and the cost of keeping detection on the edge node itself, in CPU, RAM and energy, is rarely measured. Aldaej et al. [7] propose a two-stage IDS for IoT-edge platforms. Our proposal shares this philosophy, but both phases run on the same edge node and the marginal cost of the second phase is isolated and measured directly on the device. On the 6G horizon, unprecedented device density and sub-millisecond latency targets [1] shrink the window to detect and react, making inspection on the MEC node itself a design requirement. Moreover, Chinnasamy et al. [10] reinforce the need for detectors that combine accuracy and low computational cost at the edge.

Although these studies provide valuable results and insights, none presents a hierarchical IDS validated end-to-end, from model training to deployment and operation on a real edge device. This work addresses that gap on a resource-constrained ARM-based platform, complementing conventional offline evaluation with experiments under real attack traffic, measuring the incremental cost of the second detection phase and characterizing operational phenomena that purely offline evaluations cannot expose, such as distribution shift and flow-extractor saturation under flooding attacks. Table I summarizes this positioning.



(a) Two-phase detection architecture. (b) Methodology pipeline.

Fig. 1: Overview of the proposed system: (a) detection architecture and (b) end-to-end methodology, both on the edge node (VIM 4).

III. Methodology

The methodology covers the full system cycle: traffic is collected in Packet Capture (PCAP) files, converted into bidirectional flows by the *netflower* tool, consolidated into an SQLite database, submitted to feature engineering and used to train the two classifiers, which are then deployed on the edge node for testbed validation.

A. Two-phase hierarchical architecture

Since most traffic in any network is benign, applying a heavy classifier to every flow would unnecessarily increase energy consumption and overload edge-node resources. Thus, we adopted the hierarchical two-stage architecture of Fig. 1a, in which each stage focuses on a distinct objective: binary detection favors recall, so that attacks do not slip through, while type classification favors per-class precision. Phase 1 employs a binary LightGBM classifier to distinguish between *benign* and *malicious* traffic at minimal cost while processing all flows captured at the network interface. Phase 2 is invoked only when the attack probability estimated by Phase 1 exceeds a predefined decision threshold, identifying the corresponding attack category. Otherwise the flow is discarded without additional processing, preventing the large volume of legitimate traffic from reaching the second classifier. The computational overhead of Phase 2 is therefore incurred only for suspicious flows. To keep inference from stalling packet capture under flooding, the capture thread only enqueues completed flows, while a separate worker drains the queue and runs Phase 1 on each batch and Phase 2 on the subset it flags. The queue is unbounded, so no captured flow is dropped, which turns overload into latency and memory rather than packet loss, as quantified in Section IV.

B. Data pipeline

The database was built from datasets whose PCAPs are separated by category, with labels derived from folder names, which avoids schema inconsistencies across sources [11]. Two CIC datasets were employed: the CIC IoT Dataset 2023 [9], focused on multi-attack traffic on IoT devices, and the CIC IIoT Dataset 2025[1], which comprises industrial IoT intrusions. The benign class was complemented with real traffic captured by the authors over dozens of hours of everyday personal computer use (browsing, streaming, gaming, SSH sessions and system updates), converted into flows and labeled `benign`. The benign traffic of the CIC datasets is scripted, with regular temporal patterns, whereas real interactive traffic exhibits SSH bursts, HTTP keepalives and short flows over a fast local network. Mixing real traffic into training reduces the distribution shift between training and deployment.

Given the data scale (raw captures on the order of terabytes), each dataset is consolidated into a single SQLite table, which is lightweight, portable and capable of performing aggregations and statistical queries directly on disk. The construction process is automated and reproducible: all PCAP files within each attack-category directory are converted into network flows by *netflower* with eight parallel processes, and the resulting per-class CSV files are bulk-loaded into the database and indexed by the label column. By aggregating multiple packets into a single flow, a reduction in the storage footprint by approximately 5× is achieved: the resulting database occupies $\sim$190 GB, compared with $\sim$1 TB for the original PCAP files.

The *netflower* is a flow-extraction tool developed in this work, published on the Python Package Index (PyPI)[2] under the MIT license [12]. From PCAPs or a live network interface, it reconstructs bidirectional IPv4 flows (TCP/UDP) and produces 82 CICFlowMeter-compatible features (flow identity, duration and rate statistics, packet-size and inter-arrival-time distributions, TCP-flag counts, active/idle periods, bulk, subflow and initial-window metrics). Designed for edge devices, it uses the *dpkt* library (10–25$\times$ faster than Scapy on ARM), accumulates statistics with Welford's online algorithm [13] in $O(1)$ memory per flow, accesses libpcap via `ctypes`, and emits each flow only upon completion, by connection termination (TCP FIN/RST), inactivity timeout (`idle_timeout`, default 30 s) or maximum duration (`flow_timeout`, default 120 s). These two timeouts govern the detection latency in live capture.

C. Label mapping

Since DDoS and DoS attacks aim to exhaust resources and their network flows (high packet rate, short duration, flag anomalies) are similar, maintaining two separate classes may introduce noise without providing a significant discriminative gain for the classifier. Furthermore, from the perspective of the target node, the distinction between DoS and DDoS is not observable at the individual flow level. The received flows are statistically similar, while the main discriminative characteristic (source-IP diversity) requires aggregation across flows, which is not captured by the 55 per-flow features considered in this work. Therefore, we combined `ddos` and `dos` attacks into a single class before training, reducing the space to 8 classes: `benign`, `dos` (resource exhaustion), `malware` (including Command and Control communication), `bruteforce` (SSH, HTTP, Telnet), `mitm` (ARP poisoning), `web` (application scanning and fuzzing), `spoofing` (source forgery) and `recon` (network scanning).

D. Feature engineering

Starting from the 82 raw features generated by *NetFlower*, a preprocessing stage removed the attribute groups that could cause the model to learn dataset-specific artifacts rather than intrinsic traffic characteristics: the features identifying specific hosts and services (e.g., IP addresses and port numbers), which lead the model to memorize endpoints observed during training instead of generalizable traffic patterns; the initial TCP window features carrying the sentinel value -1, which does not occur in real network traffic and could become a source of false positives; and the bulk-rate features, which *NetFlower* always outputs as zero during live operation whereas the training datasets contain non-zero values, a discrepancy that would not exist in production. Two statistical filters completed this stage, discarding eleven highly correlated features ($|\rho| > 0.95$, Pearson) and three features with near-zero variance ($\sigma^2 < 10^{-4}$), which resulted in a final set of 55 numerical features. According to the LightGBM feature importance based on gain, the two most informative features were the flow byte rate per second (`flow_byts_s`) and the minimum packet inter-arrival time (`flow_iat_min`).

E. Training the two phases

The models comprising the two phases were trained independently using the SQLite database as the data source. The hyperparameters were optimized with Optuna, while class-balanced samples were extracted directly from the disk, avoiding the need to load the entire dataset into memory.

1) Phase 1 - Binary classifier

The binary classifier was trained using 615,317 benign flows and all 554,795 attack flows. After duplicate removal, the resulting dataset comprised 703,558 unique instances, which were partitioned into training and test sets using a stratified 80/20 split. Hyperparameters and the decision threshold were jointly optimized with Optuna by maximizing the F_2-score, which assigns four times greater importance to recall than to precision, reflecting the significantly higher security cost of false negatives compared with false positives. Evaluation on the test set (140,712 flows) using the optimal decision threshold of 0.157 yielded an accuracy of 80%, an attack recall of 96%, a precision of 68%, and an F_2 score of 88.6%. Although this threshold provides interesting accuracy, recall, and F_2 score, the resulting false-positive ratio, approximately one false alarm for every three alerts, is impractical for deployment in production environments. Therefore, the decision threshold was increased to 0.9 for operational use, sacrificing a small

[1] https://www.unb.ca/cic/datasets/iiot-dataset-2025.html

[2] https://pypi.org/project/netflower/

TABLE II: Phase 2 offline evaluation per class — test set.

Class	Precision	Recall	F1	Support
benign	83%	69%	76%	83,313
bruteforce	20%	77%	31%	2,191
dos	**99%**	**96%**	**98%**	64,582
malware	90%	78%	84%	81,338
mitm	42%	72%	53%	13,823
recon	83%	68%	75%	47,915
spoofing	28%	73%	41%	12,289
web	77%	84%	80%	11,930
Accuracy			**78%**	317,381
Macro F1			**67%**	

amount of recall to substantially reduce false positives (see Section IV).

2) Phase 2 - Multiclass classifier

The multiclass classifier was trained using independently sampled data, with the number of flows per attack category capped at 615,317. We applied `class_weight=balanced` to compensate for the scarcity of the `bruteforce` attack ($\sim$16k samples), while Optuna targeted the macro F_1 metric to weigh all classes equally. Table II summarizes the classifier performance considering the test set (317,381 flows), illustrating an offline baseline pattern that later intensifies on the physical testbed. Specifically, `dos` traffic achieves near perfect classification ($F_1 = 98\%$), whereas `bruteforce`, `spoofing`, and `mitm` display low precision (20%, 28%, and 42%) paired with high recall (72–77%) driven by class balancing. A macro F_1 score of 67% confirms that features extracted from an individual flow isolate attacks with distinct structural signatures effectively but perform poorly against volumetric floods.

F. Testbed validation: environment and protocol

The experimental validation was conducted using two machines connected to the same Gigabit Ethernet local network, shared with other hosts: a personal computer - PC (AMD Ryzen 5 7600, 32 GB DDR5 RAM, and an NVIDIA RTX 5070 GPU) and a Khadas VIM 4 board. The former was responsible for training the machine learning models (LightGBM on CPU), orchestrating the experiments via SSH, and generating attack traffic targeting the Khadas VIM 4. The latter served as the MEC node, executing the IDS on its `eth0` network interface. The Khadas VIM 4 is a low-cost edge computing platform suitable for IoT/MEC deployments, equipped with an Amlogic A311D2 system-on-chip (SoC) featuring eight heterogeneous ARM cores (4× Cortex-A73 at 2.2 GHz and 4× Cortex-A53 at 1.8 GHz), 8 GB of LPDDR4x RAM, and a 2.5–9 W power envelope.

On the PC, the orchestrator (`run_experiment.sh`) deploys the scripts to the VIM 4 via SSH, coordinates the evaluation sessions and triggers the computation of the performance metrics. The attack generator (`attack_generator.py`) sequentially executes the eight attack categories and produces a JSON report containing the time intervals associated with each attack, which serves as the ground-truth reference. The metrics script then correlates this report with the IDS logs collected from the VIM 4 to compute the detection metrics. On the VIM 4, the IDS (`network_ids.py`) executes Phase 1 and, when enabled, Phase 2, logging per-flow detection events together with periodic snapshots of CPU utilization, memory usage, processor temperature, and operating frequency, from which the computational cost metrics are derived. The attacks use real tools, including nmap and masscan (SYN, UDP and aggressive scans), hping3 (SYN/UDP/ICMP floods, DDoS with random sources and spoofing with a forged source), medusa (SSH/HTTP/Telnet brute force), nikto and gobuster (web scanning and enumeration), curl (HTTP flood), arpspoof (bidirectional ARP poisoning for MITM), and netcat (simulated C2 beacon). Brute force runs medusa over several usernames and a generated password list, and the MITM step drives TCP and UDP traffic through the poisoned route so that *netflower*, which parses only TCP and UDP, records it as flows.

The proposed approach is evaluated through two sequential sessions executed under the same script: Session A enables only the binary IDS (Phase 1), whereas Session B activates both classifiers (Phases 1 and 2), which isolates the incremental resource cost of the second phase. Each session begins and ends with a 60 s benign baseline period and includes a 120 s idle interval between consecutive attacks, which keeps the attack windows apart (labelling by flow start does not depend on this gap). The binary classifier adopts a decision threshold of $P_1 \geq 0.9$. Detection is scored *per flow*: each classified flow is labelled attack or benign according to the ground-truth window in which it *started* (`flow_ts`, the capture time of its first packet), never by when the alert was emitted. This is deliberate: under flooding the ARM node saturates and emits flows with a median delay of 181–428 s (Section IV-C), so any emission-anchored metric contaminates neighbouring seconds and windows. Two window definitions are reported: *strict*, using the orchestrator's recorded start/end times, and *calibrated* (+2 s), which widens each window by 2 s per side to correct a systematic lag. The orchestrator records `start_time` after the attack tool is already sending, so the first attack flows begin up to $\sim$2 s before the logged instant, and 88% of the strict false positives are exactly such real attack flows that the +2 s slack reabsorbs (the correction saturates at 2 s). Labelling remains by `flow_ts`, without reintroducing emission time. For the multiclass classifier, the predicted category is the Phase 2 label when Phase 1 alerts, and benign otherwise. In addition to detection metrics, CPU utilization, RAM usage, alert throughput, and estimated energy consumption are reported for all experiments.

IV. Results

This section presents the experimental evaluation of the proposed hierarchical IDS deployed on the Khadas VIM4 platform under online attack traffic. The binary and multiclass configurations are systematically analyzed in terms of detection performance and computational, memory, and energy costs, guiding their deploying on resource-constrained edge nodes.

TABLE III: Per-flow binary IDS metrics on the testbed (threshold 0.9), under strict and calibrated (+2 s) ground-truth windows. Precision is the reliable specificity measure. The FPR denominator is tiny (few hundred benign flows in a flood-saturated capture).

Metric	Strict	Calibrated (+2 s)
Accuracy	67.9%	**69.4%**
Precision	94.5%	**96.8%**
Recall / TPR	69.6%	**70.1%**
F1-score	80.2%	**81.3%**
FPR	55.3%	42.3%
FNR	36.0%	35.7%

TABLE IV: Per-flow recall per attack type, binary IDS, calibrated (+2 s) window (Section IV-A).

Attack	Flows	Detected	Recall
spoofing	148,763	141,664	**95.2%**
dos	166,428	149,028	89.6%
ddos	456,514	256,523	56.2%
recon	3,526	2,094	59.4%
bruteforce	233	39	16.7%
mitm	193	15	7.8%
malware	103	3	2.9%
web	8,168	51	0.6%

A. Binary IDS (Phase 1)

Table III summarizes the binary-IDS results under the per-flow metric, over both strict and calibrated (+2 s) ground-truth windows. Under the calibrated window the detector reaches a precision of 96.8%, a recall of 70.1% and an F1 of 81.3%. Under the strict window, these become 94.5%, 69.6% and 80.2%. Precision is the reliable specificity measure in this testbed: as the capture is dominated by flooding (∼99% of flows are attack), genuinely benign traffic amounts to only a few hundred flows, so the FPR (42.3% calibrated, 55.3% strict) inherits a tiny, noisy denominator and is not comparable to an FPR measured on a clean benign baseline. The recall of 70% reflects the operating threshold rather than a weak model: the single-packet UDP flows produced by the UDP floods score around 0.807, just below the 0.9 threshold, and count as misses.

Table IV shows that recall is unevenly distributed across attack types. Volumetric floods carrying a mix of TCP and UDP are detected well (spoofing 95.2% and DoS 89.6% of flows), but DDoS drops to 56.2% because its random-source flood is dominated by single-packet UDP flows that score just below the threshold, and reconnaissance reaches 59.4% because nmap's UDP scan emits too few packets per port to accumulate features before the flow expires (30 s `idle_timeout`). Low-volume attacks are largely missed at this threshold: brute force 16.7%, MITM 7.8%, web 0.6% and malware 2.9%. These categories generate few flows by nature (brute force 233, MITM 193, malware 103), so their rates rest on small samples rather than on a generator artifact.

B. Multiclass IDS (Phases 1+2)

All eight categories now generate flows (Table V). Phase 2 classifies `recon` well (F1 = 74.6%), whereas the other attacks collapse onto `dos`, which absorbs most attack flows (precision 74.3%, recall 51.2%, F1 60.6%). The 142,195 spoofing flows are labelled `dos` almost in full, and web, brute force, MITM and malware are essentially never assigned their own class. Type accuracy is 41.7% (40.6% strict) and macro-F1 over the attack classes is 19.4%. Restricted to the flows Phase 1 detected, type accuracy rises to 64.3%.

TABLE V: Multiclass IDS in the testbed (calibrated +2 s window; dos includes ddos).

Class	Sup.	Prec.	Rec.	F1
recon	1,638	94.9%	61.4%	**74.6%**
dos	629,291	74.3%	51.2%	60.6%
spoofing	142,195	6.9%	0.0%	0.0%
web	8,131	100.0%	0.1%	0.2%
bruteforce	264	0.2%	0.4%	0.3%
mitm	116	0.0%	0.9%	0.0%
malware	103	0.0%	1.0%	0.0%
Type accuracy		**41.7%**	(40.6% strict)	
Macro F1 (attacks)				19.4%

The type label thus identifies reconnaissance scans but does not separate the other attacks from generic flooding. Within the 55 per-flow attributes, short, high-rate flows are statistically indistinguishable from a flood. For `spoofing` the `dos` label is in fact appropriate, since the attack is a forged-source flood. Separating the rest would require state aggregated across flows, such as authentication-failure counts or source-IP diversity, which per-flow features lack by construction.

C. Distribution shift and deployment artifacts

Three deployment phenomena, absent from offline evaluation, help explain the results. The most consequential is a throughput ceiling. At about 1,050 flows/s the ARM node cannot process traffic in real time: the inference queue grows to a peak of 433,469 flows for the binary pipeline and 491,880 for the hierarchical one, RAM rises to 2.4–2.6 GB, and the mean end-to-end latency from capture to classification is 116 s and 343 s respectively, with maxima of 182 s and 550 s. Because the queue is unbounded and fully drained at shutdown, no captured flow is dropped, so the overload surfaces as latency and memory rather than packet loss. Scoring by flow start (`flow_ts`) makes the detection metrics immune to this displacement, which changes *when* an alert appears, not *which* flow it is. The second phenomenon is a distribution shift in benign traffic: `flow_iat_min` lies in the millisecond range in training but drops to 1–10 µs in real TCP bursts, which argues for capturing benign traffic in the deployment environment itself. The third is a *netflower* artifact: each TCP teardown produces a single-packet FIN/ACK micro-flow scored at ∼84% attack probability, mitigated by a minimum-packet guard before inference.

D. Binary IDS vs. multiclass IDS

Table VI compares both configurations. The two columns are the two independent testbed sessions, which share the Phase 1 model but generate their own attack traffic. Each session drains its queue completely before exiting, so the flow totals (827,458 vs 809,448) carry no dropped residual; the ∼2% gap is almost all benign background, as the attack traffic is nearly identical (783,928 vs 781,738, 0.3%). Adding Phase 2 is far from free on this node. Average CPU rises from 12.3% to 41.6% (peaks 43% to 73%), average power during attacks

TABLE VI: Pipeline comparison on the VIM 4 platform. Columns are the two independent testbed sessions (they share the Phase 1 model but generate their own attack traffic, so Phase 1 metrics differ). †The FPR denominator is tiny (few hundred benign flows in a flood-saturated capture); precision is the reliable specificity measure.

Metric	Binary (Phase 1)	Multiclass (Phases 1+2)
Binary detection (Phase 1), per flow, +2 s window		
Precision	**96.8%**	97.4%
Recall / TPR	**70.1%**	64.3%
F1-score	**81.3%**	77.5%
Accuracy	**69.4%**	63.9%
FPR†	42.3%	47.8%
Type classification (Phase 2)		
Accuracy (type)	—	**41.7%**
Macro F1 (attacks)	—	19.4%
Resources (VIM 4, during session)		
CPU avg / max	12.3% / 43.0%	**41.6% / 72.8%**
RAM avg	1,866 MB	2,041 MB
RAM max	2,449 MB	2,606 MB (**+157 MB**)
Throughput and latency		
Flows classified	827,458	809,448
Alerts emitted	567,836	516,182
Attack traffic	∼985 flows/s	∼1,050 flows/s
Queue peak	433,469	491,880
Mean e2e latency	116 s	**343 s**
Estimated energy		
Avg. power (attack)	3.49 W	**5.20 W**
Avg. power (idle)	2.77 W	4.81 W
Total energy (band)	4,375–6,672 J	6,478–10,043 J

from 3.49 to 5.20 W, and the mean end-to-end latency from 116 to 343 s, while peak RAM grows by about 157 MB to 2.61 GB. Batched inference itself is cheap (0.14 ms per flow in Phase 1 and 1.22 ms in Phase 2), so the cost comes from Phase 2 running on the traffic Phase 1 flags, not from the model.

The energy consumption in Table VI is a calibrated estimate, not a direct measurement, since the VIM 4 exposes no energy telemetry interfaces (RAPL or an INA226 sensor). Power is modeled linearly on CPU utilization,

$$P(t) = P_{idle} + (P_{max} - P_{idle}) \times \text{CPU}\%(t), \qquad (1)$$

with $P_{idle} = 2.5$ W and $P_{max} = 9.0$ W anchored on the VIM 4 power envelope, calibrated at rest and under full load, and integrated over the periodic samples (∼3 s), separating attack from idle by the ground-truth windows. The resulting sensitivity band is 4,375 to 6,672 J for the binary pipeline and 6,478 to 10,043 J for the multiclass one. What matters is the relative comparison: the hierarchical pipeline raises average power during attacks from 3.49 to 5.20 W, a direct consequence of Phase 2's CPU load under flooding rather than a memory-only overhead.

V. Conclusion

This work presented a two-phase hierarchical IDS for IoT environments in 6G networks with MEC, from data pipeline to deployment and testbed validation on the VIM 4 board. Scored per flow, the binary IDS detected volumetric floods precisely (spoofing 95%, DoS 90%) at a precision of 97%, while low-volume attacks and single-packet floods fall below the 0.9 threshold and lower the overall recall to 70%. Phase 2 separates reconnaissance but merges the other flood-like attacks into `dos`, a boundary of per-flow features rather than a pipeline fault. The main operational finding is a throughput ceiling: at ∼1,050 flows/s the VIM 4 cannot keep up and buffers traffic with a latency of minutes, and adding Phase 2 roughly triples that latency while raising CPU and power, so the hierarchical pipeline as deployed does not sustain line-rate detection under volumetric attack on this node. Future work includes bounding the queue with load shedding or sampling under overload, parallelizing inference across the node's cores, adding aggregate cross-flow features to separate flood-like attacks, evaluating against real services to reduce the training-to-production shift, and replacing the energy estimate with hardware measurements. The scripts and trained models are available in an open repository[3].

References

[1] X. You *et al.*, "Towards 6g wireless communication networks: vision, enabling technologies, and new paradigm shifts," *Science China Information Sciences*, vol. 64, no. 1, p. 110301, 2021.

[2] A. Lazarevic, L. Ertöz, V. Kumar, A. Ozgur, and J. Srivastava, "Intrusion detection: a survey," in *Managing Cyber Threats: Issues, Approaches and Challenges*, V. Kumar, J. Srivastava, and A. Lazarevic, Eds. New York: Springer, 2005, pp. 19–78.

[3] E. Gyamfi and A. Jurcut, "Intrusion detection in Internet of Things systems: a review on design approaches leveraging multi-access edge computing, machine learning, and datasets," *Sensors*, vol. 22, no. 10, p. 3744, 2022.

[4] M. Houichi, F. Jaidi, and A. Bouhoula, "Enhancing smart city security: an intrusion detection system using machine learning methods with the UNB CIC IoT 2023 dataset," *IET Smart Cities*, vol. 7, no. 1, p. e70014, 2025.

[5] W. Chen, H. Yang, L. Yin, and X. Luo, "Large-scale IoT attack detection scheme based on LightGBM and feature selection using an improved salp swarm algorithm," *Scientific Reports*, vol. 14, no. 1, p. 19165, 2024.

[6] R. Yang, H. He, Y. Xu, B. Xin, Y. Wang, Y. Qu, and W. Zhang, "Efficient intrusion detection toward IoT networks using cloud–edge collaboration," *Computer Networks*, vol. 228, p. 109724, 2023.

[7] A. Aldaej, I. Ullah, T. A. Ahanger, and M. Atiquzzaman, "Ensemble technique of intrusion detection for IoT-edge platform," *Scientific Reports*, vol. 14, no. 1, p. 11703, 2024.

[8] M. Zolanvari, M. A. Teixeira, L. Gupta, K. M. Khan, and R. Jain, "Machine learning-based network vulnerability analysis of industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6822–6834, 2019.

[9] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, "CICIoT2023: a real-time dataset and benchmark for large-scale attacks in IoT environment," *Sensors*, vol. 23, no. 13, p. 5941, 2023.

[10] P. Chinnasamy, S. Yarramsetti, R. K. Ayyasamy, E. Rajesh, V. Vijayasaro, D. Pandey, B. K. Pandey, and M. E. Lelish, "AI-driven intrusion detection and prevention systems to safeguard 6G networks from cyber threats," *Scientific Reports*, vol. 15, no. 1, p. 37901, 2025.

[11] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *International Conference on Information Systems Security and Privacy*. Funchal: SciTePress, 2018, pp. 108–116.

[12] L. H. B. A. d. Silva, "netflower: a network-flow feature extractor for edge devices," Python Package Index, 2026, disponível em: https://pypi.org/project/netflower/. Acesso em: 30 jun. 2026.

[13] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[3] https://github.com/luiz-linkezio/A-Hierarchical-Solution-based-on-MEC-and-IoT-for-Cyberattack-Detection-in-6G-Networks