An International Journal of Optimization and Control: Theories & Applications
ISSN: 2146-0957 eISSN: 2146-5703
Vol.X, No.X, pp.X-X (2026)
https://doi.org/10.36922/IJOCTA026400309

RESEARCH ARTICLE

RC-PORT-IDS: Resource-constrained two-stage model-portfolio planning with observable-state recourse for edge intrusion detection

Ameen Shaheen[1]*, Ala'a Al-Shaikh[2], Wael Alzyadat[1], and Aysh Alhroob[1]

[1]Department of Software Engineering, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan

[2]Department of Cyber Security, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan

a.shaheen@zuj.edu.jo, a.shaikh@zuj.edu.jo, wael.alzyadat@zuj.edu.jo, aysh@zuj.edu.jo

ARTICLE INFO

Article history:
Received: October 2, 2026
Revised: October 31, 2026
Accepted: November 2, 2026
Published Online: November 17, 2026

Keywords:
Intrusion detection system
Internet of Things
Cybersecurity
Model portfolio
Resource-constrained optimization

AMS Classification 2010:
90C29; 90C90

ABSTRACT

Fixed single-model deployment can be restrictive for edge intrusion detection when traffic conditions and available computational resources change after deployment. This study proposes RC-PORT-IDS, a two-stage model-portfolio planning framework with observable-state recourse. The first stage selects a constrained portfolio of heterogeneous intrusion-detection models before the operating state is observed. The second stage selects a feasible deployed model using observable traffic and resource states only. Seven tabular classifiers were evaluated. Canadian Institute for Cybersecurity Internet of Things 2023 dataset was used for development analysis, including duplicate-conflict control, observable-regime construction, resource benchmarking, and internal recourse-gain analysis. Canadian Institute for Cybersecurity Internet of Medical Things 2024 dataset was used for independent external replication with retraining, three-fold out-of-fold regime-complete calibration, policy freezing, and final test evaluation. Internal development produced positive recourse gain in 13 of 15 deployment cases, with a mean of 0.001512. In the external replication, the calibration-selected constrained static policy obtained a mean balanced accuracy of 0.697275, a constrained test-oracle static policy obtained 0.840868, and full traffic-plus-resource recourse obtained 0.888692. Full recourse therefore retained a mean advantage of 0.047825 over the test-oracle static policy. Traffic-only recourse reached 0.837476, while resource-only recourse remained at 0.697275. Sensitivity analyses over regime count, regime seed, and ± 5% latency-threshold perturbations preserved positive recourse gain in 12 of 15 cases. The results show that a resource-constrained pre-deployed portfolio can support state-dependent model selection under controlled storage and latency constraints without using ground-truth attack-family information at runtime.

CC BY

1. Introduction

The continued growth of the Internet of Things (IoT) and edge-connected systems has increased the need for intrusion detection systems (IDS) that can operate under changing traffic conditions and limited computational resources. Machine learning-based IDS are widely used because they can learn traffic patterns directly from data and can support binary or multi-class attack detection without depending only on predefined signatures. Reviews of network and IoT intrusion detection have reported the extensive use of supervised learning, ensemble methods, and deep learning for

*Corresponding Author

this purpose.[1,2] An optimized supervised-learning IDS has also been evaluated specifically for IoT traffic, showing that classifier selection remains an important part of the detection pipeline.[3]

The availability of recent IoT benchmarks has made it possible to examine IDS behavior over a wider range of attacks and device conditions. Canadian Institute for Cybersecurity Internet of Things 2023 (CICIoT2023) contains traffic generated from a large IoT topology and includes multiple attack categories, which makes it suitable for evaluating heterogeneous candidate models under the same feature space.[4] However, high classification performance alone does not resolve the deployment problem. IoT and edge environments may impose restrictions on memory, storage, inference time, processor availability, and batch processing. Lightweight machine learning has therefore been examined as a way to reduce the computing burden of IoT security models.[5] Similar work on edge-based industrial IoT (IIoT) intrusion detection has treated detection quality and computational payload as joint deployment concerns rather than independent reporting measures.[6]

Runtime conditions can also change after a model has been deployed. Traffic composition, load, and available processing capacity may vary across operating periods, and a model that is suitable under one state may not remain the preferred model under another state. Adaptive IDS work has addressed changing network behavior through online or adaptive learning while retaining deployment on constrained edge devices.[7] Feature selection and supervised learning have also been combined to reduce learning time and detection cost in IoT IDS pipelines.[8] These approaches support the need for deployment-aware IDS design, but they mainly focus on constructing or adapting a detector rather than deciding which set of heterogeneous detectors should be installed before deployment and which installed detector should be used after the operating state becomes observable.

A related issue appears in model selection. Conventional selection usually identifies one model from a candidate set and then keeps that model fixed during operation. This can be restrictive when candidate models have different latency, storage, and predictive characteristics. Earlier work on Learned Acquisition and Reconstruction Optimization-IDS showed that the preferred IoT IDS model can change when predictive performance, cross-family behavior, and deployment cost are considered together instead of using only the best validation score.[9] The remaining question is different from single-model selection. Before deployment, storage capacity determines which models can be installed on the edge system. After deployment, runtime selection operates within that pre-deployed portfolio using the observed traffic and resource state. The first decision is therefore made before the operating state is known, while the second decision is made after that state is observed.

This decision structure is consistent with two-stage stochastic programming. In a two-stage problem, first-stage decisions are fixed before the uncertain state is observed, while second-stage recourse decisions are allowed after the state becomes available.[10] Classical stochastic-programming value measures distinguish decisions made before uncertainty from recourse and perfect-information benchmarks.[11] In the IDS setting considered here, the first stage determines the deployed model portfolio under a storage budget, and the second stage selects a feasible deployed model for the observed traffic and resource state. This formulation does not require the runtime system to know the ground-truth attack family. The recourse decision uses only observable state information and the measured resource feasibility of the deployed models.

Optimization has been used in other parts of IDS design, including feature selection, model tuning, and data generation. Recent work on CICIoT2023, for example, combined conditional generative modeling with an adaptive whale optimization procedure to support IDS dataset synthesis.[12] In the present study, optimization is applied to a different decision level: pre-deployment portfolio planning and portfolio-constrained runtime recourse. The objective is to determine which models should be available before deployment and how the deployed portfolio should be used after the operating state is observed.

To address this problem, this study proposes RC-PORT-IDS, a resource-constrained two-stage model-portfolio planning framework with observable-state recourse for edge intrusion detection. The framework considers a heterogeneous pool of seven classifiers. The first-stage decision selects a portfolio subject to storage and portfolio-size constraints. The second-stage decision selects a model from the deployed portfolio subject to the current latency feasibility condition. Observable traffic regimes and controlled resource states are used to define the recourse states. Balanced error is used in

the final external policy objective so that false positives and false negatives are weighted equally under the highly imbalanced external test distribution. The ground-truth attack family is not used by the runtime policy.

The experimental design separates development evidence from independent external replication. CICIoT2023 is used for the internal development analysis. Canadian Institute for Cybersecurity Internet of Medical Things 2024 (CICIoMT2024) is then used as an independent external dataset; its Wi-Fi and Message Queuing Telemetry Transport (MQTT) data provide compatible flow features together with separate training, profiling, and test partitions.[13] For the external replication, the same model families and fixed hyperparameter settings are trained on the external training split. Out-of-fold predictions from the training data are combined with the profiling split to complete the observable-regime calibration before the policy is frozen. The external test split is used only after the portfolio and recourse policy have been fixed. Duplicate overlap between training, profiling, and test data is removed before the final evaluation. The main contributions of this study are as follows:

- Pre-deployment IDS planning is formulated as a two-stage stochastic model-portfolio problem in which a constrained set of heterogeneous models is selected before the operating state is observed, and a feasible deployed model is selected afterward through recourse.
- Observable-state recourse is defined using traffic and resource states only. The runtime decision operates over the previously deployed portfolio and does not use the ground-truth attack family as an input to model selection.
- A development and external-replication protocol is introduced that includes duplicate-conflict control, resource re-benchmarking, balanced-error optimization, and out-of-fold regime-complete calibration before the external test policy is frozen.
- The two-stage decision is evaluated across storage budgets, latency-service-level agreement (SLA) levels, and portfolio sizes using recourse gain, a constrained test-oracle diagnostic, and policy sensitivity to observable-regime construction and latency thresholds under the same deployment constraints.
- The two-stage decision procedure is independently replicated on CICIoMT2024 after retraining the same model families and re-benchmarking their resource behavior, providing a dataset-level test of the decision structure rather than direct transfer of fitted classifiers.

The remainder of this study is organized as follows. Section 2 reviews related work on IoT intrusion detection, adaptive and multi-model detection, resource-constrained deployment, and stochastic decision models. Section 3 presents the problem formulation and the RC-PORT-IDS framework. Section 4 describes the datasets, preprocessing, candidate models, observable-state construction, resource benchmarking, and evaluation protocol. Section 5 reports and discusses the internal and external results, including the recourse ablation and sensitivity analysis. Section 6 concludes the study and identifies directions for further evaluation.

2. Literature review

Research related to RC-PORT-IDS can be grouped into four areas: IoT intrusion detection, adaptive and multi-model detection, resource-constrained IDS deployment, and stochastic decision models with recourse. The first three areas address detection quality, adaptation, and deployment cost from different directions. The fourth area provides the decision structure used when some choices must be fixed before an uncertain operating state is observed and other choices can be made afterward. This section reviews these areas and identifies the position of the present study.

2.1. Edge and Internet of Things intrusion detection

Internet of Things intrusion detection has increasingly used flow-level datasets that contain heterogeneous devices, protocols, and attack categories. Edge-IIoTset was generated from an IoT and IIoT testbed and includes traffic, system, and application information for centralized and federated intrusion detection experiments.[14] Its design reflects the use of machine learning in environments where sensing, communication, and computing are distributed across edge-connected devices.

Telemetry and operating network_IoT extended this direction by combining network, operating-system, and IoT telemetry data within one dataset family. The dataset was designed for data-driven IDS evaluation across IoT and IIoT sources and includes normal and malicious observations from different telemetry layers.[15]

This type of data supports model comparison beyond a single traffic source, but the evaluated detector is still normally selected and assessed as a fixed model after training.

Earlier device-focused work also showed that IoT attacks can be detected from behavioral network features. N-BaIoT used deep autoencoders to model traffic from commercial IoT devices infected with Mirai and BASHLITE and evaluated anomaly detection separately for the monitored devices.[16] These studies established the usefulness of heterogeneous machine learning models and realistic IoT traffic for IDS evaluation. They do not, however, define the deployment decision as a choice of which subset of candidate models should be installed before runtime state information becomes available.

2.2. Adaptive and multi-model intrusion detection

Adaptive IDS research addresses the fact that network traffic may change after a detector has been trained. Shao *et al.* used adaptive online learning with online ensemble learning for real-time IoT botnet detection and explicitly considered concept drift in streaming traffic.[17] The model is updated during operation so that the detector can respond to changes in the observed data stream.

Multi-model approaches provide another form of adaptation. Alalhareth and Hong[18] proposed a performance-driven meta-learning IDS for Internet of Medical Things in which classifier voting weights are adjusted using model accuracy, loss, and confidence information. The approach combines several classifiers within one ensemble and changes their contribution to the final decision rather than retaining one fixed classifier for all inputs.

Dynamic model selection has also been considered together with resource allocation. Auto-intrusion detection as a service uses a Deep Q-network to select IDS models under changing traffic conditions and uses twin delayed deep deterministic policy gradient for capacity allocation in an intrusion-detection-as-a-service setting.[19] This work is closely related to RC-PORT-IDS because it treats model selection and computational capacity as coupled operational decisions. Its decision process is reinforcement-learning based and focuses on runtime configuration, whereas RC-PORT-IDS separates a pre-deployment portfolio decision from the recourse decision made after state observation over the models installed by the first stage.

A recent dynamic-selection framework, Framework for Ensemble Defense with Dynamic Model Selection, combines heterogeneous deep learning, conventional machine learning, and statistical anomaly-detection models for adversarially resilient intrusion detection. The framework uses confidence information to select models dynamically and reports both detection and processing-latency measurements.[20] This confirms that model diversity can be useful at runtime, but it does not make the installed model set a first-stage optimization variable under a deployment-storage budget. Resource-aware model selection has also been studied in edge-cloud anomaly detection, where reinforcement learning was used to adapt model choice under computational and latency constraints.[21] A multi-model fallback architecture for edge devices similarly used a deployed model pool with deterministic runtime switching while balancing prediction quality, latency, and model size.[22]

2.3. Resource-constrained intrusion detection system deployment

Resource-constrained IDS studies usually reduce the cost of one detector or compare deployment locations. Resource-constrained IoT IDS work has also combined detection performance with explicit computational indicators in small computing environments.[23] Alwaisi *et al.*[24] examined machine learning for constrained IoT systems across device, edge, and cloud settings and compared models using resource measures that included energy and memory use. Their results show that predictive performance alone is not sufficient for selecting a deployment configuration on small devices.

Feature reduction is another common method for decreasing IDS cost. Ensemble Feature Selection for Lightweight Intrusion Detection System combines several filter-based feature-selection methods and evaluates the resulting classifiers using training time, testing time, CPU use, memory use, and cross-domain testing.[25] The objective is to construct a lighter classifier by reducing the input representation before deployment.

Almalawi[26] used recursive clustering for data condensation together with Monte Carlo Cross-Entropy feature selection to reduce the amount of training data and the number of retained features. The reported evaluation considered

N-BaIoT, Edge-IIoTset, and CICIoT2023 and measured reductions in training time, testing time, and memory use. This is another model-construction approach in which resource use is reduced before the final detector is selected.

Latency has also been identified as an underreported IDS deployment measure. Dash and Acken[27] reviewed IDS latency reporting and showed that the term is used for different quantities, including inference, communication, and processing time, which makes operational comparisons difficult when the measured component is not stated. For a resource-constrained portfolio problem, the latency quantity used as a feasibility constraint therefore needs to be measured under a fixed and reproducible benchmarking procedure.

These studies address deployment cost by simplifying features, reducing data, selecting lighter models, or measuring resource use. RC-PORT-IDS uses a different decision unit. The candidate models are retained as heterogeneous alternatives, and deployment cost determines which subset can be stored and which deployed models remain feasible under the observed runtime resource state.

2.4. Two-stage stochastic programming and recourse

Stochastic programming is used when an optimization decision must be made without complete knowledge of future conditions. A central modeling issue is the information structure: decisions made before uncertainty is observed must be separated from decisions that can respond after additional information becomes available.[28] In a two-stage formulation, the first-stage variables represent the decisions that are fixed in advance, while the second-stage variables represent recourse actions associated with realized scenarios.

The standard stochastic-programming formulation represents uncertain quantities by scenarios or probability distributions and evaluates the first-stage decision together with its scenario-dependent recourse cost. This structure allows feasibility and expected performance to depend on what can be changed after the uncertain state is known.[29] It is therefore suitable when deployment and operation occur at different decision times.

This separation has been used in IoT edge-computing problems. Cao *et al.*[30] formulated task offloading with uncertain connection time using stochastic programming and used posterior recourse to compensate for differences between the predicted and realized connection state. The first decision concerns task assignment and server selection, while recourse actions respond to the realized operating condition.

A related IoT-eHealth study formulated offloading, resource allocation, and migration as a multi-stage stochastic program under uncertain device-to-fog connectivity. The model used sample average approximation to obtain a tractable policy while retaining the distinction between decisions made before and after uncertainty was realized.[31] These applications show that recourse is useful when edge resources and connectivity vary after an initial planning decision.

The same information structure can be applied to IDS deployment. Model installation is a planning decision because storage and deployment choices are fixed before the current operating state is known. Model use is an operational decision because the available traffic and resource state can be observed at runtime. The reviewed stochastic-programming studies establish this separation in edge-resource problems, but the reviewed IDS studies do not formulate heterogeneous model availability itself as the first-stage decision.

2.5. Positioning of the present study

The reviewed literature leaves a specific decision problem between static model selection and unrestricted runtime adaptation. IoT IDS studies provide heterogeneous candidate models and realistic traffic datasets. Adaptive and ensemble methods allow detector behavior to change during operation. Lightweight IDS studies reduce model or feature cost, while stochastic programming separates planning decisions from recourse decisions under uncertainty. These components are related, but they are normally treated as separate design problems.

RC-PORT-IDS combines these components at the deployment-decision level. The first stage selects a resource-constrained portfolio of heterogeneous IDS models before the operating state is observed. The second stage selects from that deployed portfolio after the observable traffic state and resource state are available. Storage constraints define first-stage portfolio feasibility, while latency constraints define second-stage model feasibility. The runtime policy uses observable traffic and resource states without ground-truth attack-family information.

Earlier Learned Acquisition and Reconstruction Optimization-IDS work addressed a different deployment decision: it selected one IDS model by combining predictive performance, family-aware behavior, and deployment cost.[9] RC-PORT-IDS changes the decision unit from one model to a deployable portfolio and introduces a state-dependent recourse mapping over observable traffic and resource states. Storage constrains the installed subset before deployment, while measured latency constrains the model that can be activated after the traffic regime and resource state are observed. A model that was not deployed cannot be selected at runtime.

The experimental design also separates development evidence from independent external replication. The internal analysis is used to determine whether model complementarity and resource constraints create value for a portfolio decision. The external replication then rebuilds the candidate models on a separate dataset, calibrates observable regimes without using the external test labels for policy selection, and evaluates the frozen recourse policy using balanced error. This design is used to test whether the decision structure, rather than a particular fitted classifier, is retained on a second IoT security dataset.

3. Problem formulation and RC-PORT-IDS framework

RC-PORT-IDS formulates edge intrusion-detection deployment as a two-stage model-portfolio decision problem. The first stage selects a constrained subset of candidate IDS models before the runtime operating state is known. After deployment, the second stage observes the traffic regime and the current resource state and selects one model from the deployed portfolio. The runtime rule does not use the ground-truth attack family. The formulation below defines the candidate set, observable state, first-stage portfolio variables, second-stage recourse variables, resource constraints, calibrated state-loss contributions, and the frozen policy used during evaluation.

3.1. Problem definition

Let the candidate IDS model set be defined as **Equation 1**:

$$\mathcal{M}=\{m_1, m_2, \ldots, m_K\} \tag{1}$$

where $\mathcal{M}$ is the finite set of K trained candidate models, and m_K denotes the $k-th$ candidate. The observable runtime state is formed from a traffic regime and a resource state. The joint state space is shown in **Equation 2**:

$$\boldsymbol{\Omega} = \boldsymbol{\mathcal{R}} \times \boldsymbol{\mathcal{S}},\ \boldsymbol{\omega} = (\mathbf{r}, \mathbf{s}) \in \boldsymbol{\Omega} \tag{2}$$

where $\boldsymbol{\mathcal{R}}$ is the set of observable traffic regimes, $\boldsymbol{\mathcal{S}}$ is the set of controlled resource states, and ω denotes one joint operating state. The deployment decision is made before ω is observed. The recourse decision is made after r and s are available.

Table 1 shows the notation used in the RC-PORT-IDS formulation. The notation separates the pre-deployment portfolio variables from the runtime recourse variables and distinguishes storage feasibility from latency feasibility.

The complete RC-PORT-IDS workflow is summarized in **Figure 1**. **Figure 1** shows the separation between model and state calibration, first-stage portfolio planning, and runtime recourse. Calibrated state-loss contributions and resource measurements are used to select a fixed deployed portfolio. At runtime, the observable traffic regime and resource state determine which feasible model is activated from that portfolio.

3.2. Candidate model pool

Each candidate model is represented by a learning algorithm and a fixed hyperparameter configuration (**Equation 3**):

$$\mathbf{m_k} = \mathbf{A_k}(\boldsymbol{\theta}_\mathbf{k}), \mathbf{k} = 1, \ldots, \mathbf{K} \tag{3}$$

where $\mathbf{A_k}$ is the learning algorithm used by candidate $\mathbf{m_k}$, and $\boldsymbol{\theta}_\mathbf{k}$ is its frozen configuration. Model training is completed before portfolio optimization. The candidate pool is therefore treated as fixed during the first-stage decision. Section 4 reports the seven model families and configurations used in the experiments.

The candidate models can differ in predictive behavior, storage cost, and latency under different resource states. RC-PORT-IDS therefore treats model suitability as state-dependent rather than assuming one candidate is preferred under all operating conditions. The optimization problem determines which admissible deployed subset minimizes state-dependent loss under the defined deployment constraints.

3.3. Observable traffic and resource states

The traffic state is obtained from observable flow features. Let $h(\cdot)$ denote the frozen regime-assignment function. For a d-dimensional feature

vector x, the observable traffic regime is shown in **Equation 4**:

Table 1. Notation used in the RC-PORT-IDS formulation

Symbol	Definition
$\mathcal{M}$	Set of candidate IDS models.
$\mathcal{R}$	Set of observable traffic regimes.
$\mathcal{S}$	Set of controlled resource states.
Ω	Joint state space, $\Omega = \mathcal{R} \times \mathcal{S}$.
m, r, s	Indices for model, traffic regime, and resource state.
x_m	Binary first-stage deployment variable.
y_{mrs}	Binary second-stage model-selection variable.
u_s	Binary infeasibility indicator for resource state s.
c_m	Serialized storage cost of model m.
B	Available storage budget.
K_{max}	Maximum deployed portfolio size.
ℓ_{ms}	Measured inference latency of model m in resource state s.
τ_s	Latency limit for resource state s.
a_{ms}	Binary latency-feasibility indicator.
FP_{mr}, FN_{mr}	Calibration false-positive and false-negative counts for model m in traffic regime r.
N_0, N_1	Total benign and attack observations in the calibration data.
q_{mr}	Calibrated loss contribution of model m in traffic regime r.
π_s	Weight assigned to resource state s, with $\Sigma_s \pi_s = 1$.
Z^{RP}	Objective value of the two-stage recourse problem.
Z^{static}	Objective value of the best feasible static single-model policy.
$G_{re}c$	Recourse gain relative to the constrained static policy, $G_{re}c = Z^{static} - Z^{RP}$.
ρ_r	Calibration probability of observable traffic regime r in the internal development analysis.

$$\mathbf{r} = \mathbf{h}(\mathbf{x}),\ \mathbf{h} : \mathbb{R}^d \to \mathcal{R} \tag{4}$$

where r is produced without using the attack label or attack-family identity. The resource state s represents the measured execution condition used by the latency benchmark. In the experiments, s is defined from the CPU-core setting and inference batch size. For a fixed latency-SLA setting, model feasibility in resource state s is shown in **Equation 5**:

$$\mathbf{a}_{ms} = \mathbb{1}\left[\ell_{ms} \le \boldsymbol{\tau}_s\right] \tag{5}$$

where ℓ_{ms} is the measured inference latency of model m under state s, τ_s is the corresponding latency limit, and $\mathbb{1}[\cdot]$ is the indicator function. A model is runtime-feasible only when $a_{ms} = 1$.

3.4. First-stage portfolio decision

The first stage determines which models are installed before the runtime state is observed. The binary deployment variable is shown in **Equation 6**:

$$\mathbf{x}_m \in \{0,1\},\ \forall \mathbf{m} \in \mathcal{M} \tag{6}$$

where $x_m = 1$ indicates that model m is included in the deployed portfolio. The deployed portfolio is shown in **Equation 7**:

$$\mathcal{P}(\mathbf{x}) = \{\mathbf{m} \in \mathcal{M} : \mathbf{x}_m = 1\} \tag{7}$$

Deployment feasibility is defined by the available storage budget (**Equation 8**):

$$\boldsymbol{\Sigma}_m \in \mathcal{M} \mathbf{c}_m \mathbf{x}_m \le \mathbf{B} \tag{8}$$

where c_m is the serialized storage cost of model m, and B is the storage budget. The portfolio-size constraint is shown in **Equation 9**:

$$\boldsymbol{\Sigma}_m \in \mathcal{M} \mathbf{x}_m \le \mathbf{K}_{max} \tag{9}$$

where K_{max} is the maximum portfolio size. **Equations 8** and **9** define the first-stage feasible portfolio. This portfolio remains fixed when r and s become available.

3.5. Second-stage recourse decision

After the joint state (r,s) is observed, the second stage selects one model from the deployed and latency-feasible set. The binary recourse variable is shown in **Equation 10**:

$$\mathbf{y}_{mrs} \in \{0,1\},\ \forall \mathbf{m} \in \mathcal{M}, \mathbf{r} \in \mathcal{R}, \mathbf{s} \in \mathcal{S} \tag{10}$$

where $y_{mrs} = 1$ indicates that model m is used for traffic regime r under resource state s. Recourse model availability is determined by the first-stage deployment decision (**Equation 11**):

$$\mathbf{y}_{mrs} \le \mathbf{x}_m,\ \forall \mathbf{m}, \mathbf{r}, \mathbf{s} \tag{11}$$

The selected model must also satisfy the

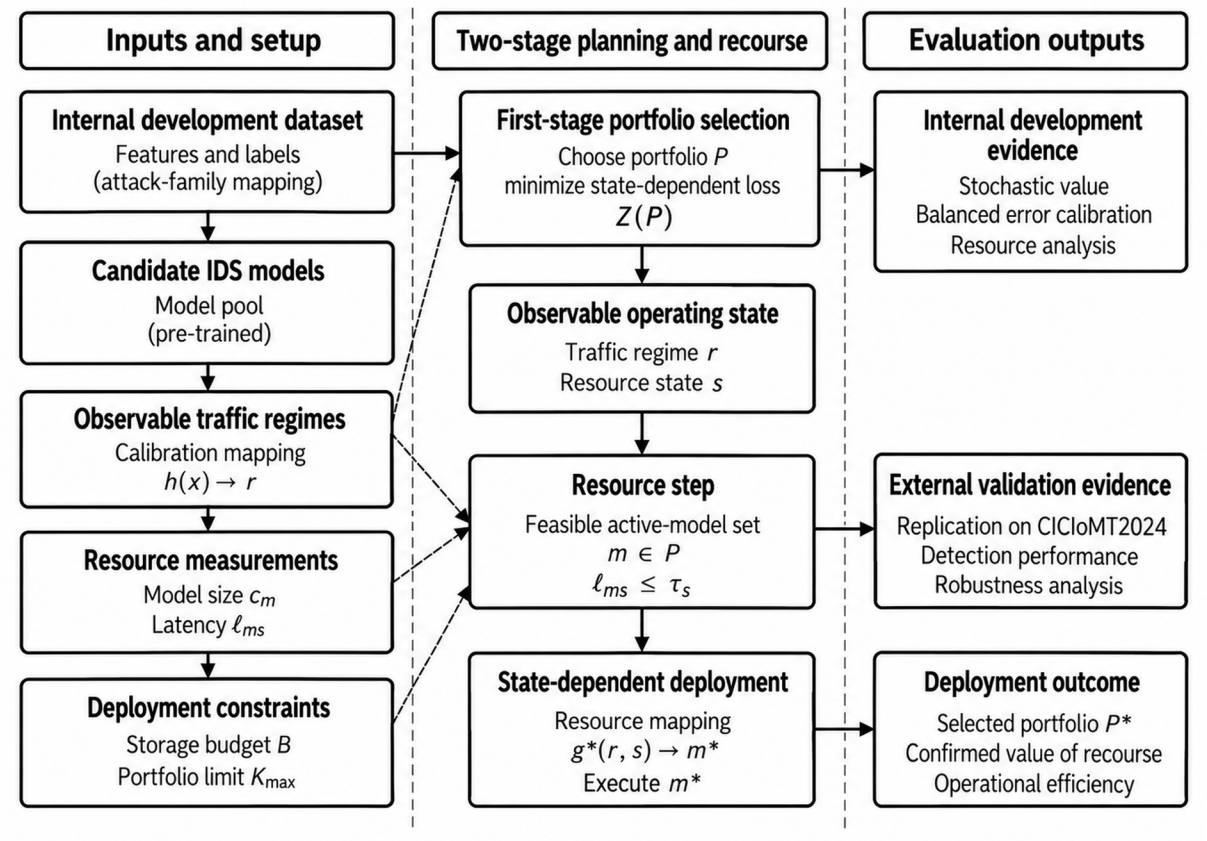


Figure 1. Overall architecture of the proposed RC-PORT-IDS framework

latency condition (**Equation 12**):

$$\mathbf{y}_{mrs} \leq \mathbf{a}_{ms}, \ \forall \mathbf{m}, \mathbf{r}, \mathbf{s} \tag{12}$$

For each resource state, the policy either selects exactly one feasible model for every traffic regime or assigns the predefined infeasible-state loss. This condition is written as **Equation 13**:

$$\mathbf{\Sigma}_{m} \in \mathcal{M} \mathbf{y}_{mrs} = 1 - \mathbf{u}_{s}, \ \forall \mathbf{r}, \mathbf{s}; \ \mathbf{u}_{s} \in \{0,1\} \tag{13}$$

where $u_s = 0$ requires one model to be selected for every r under resource state *s*. For a deployed portfolio P, ordinary second-stage selection is feasible in resource state *s* when at least one deployed model satisfies the latency limit for every required traffic regime. If this condition fails, $u_s = 1$ and the resource state receives the predefined worst policy loss.

Figure 2 shows the two-stage decision timing. **Figure 2** separates the pre-deployment variables x_m from the state-dependent variables y_{mrs}. The state pair (r,s) is observed only after the deployed portfolio has been fixed.

3.6. Resource feasibility constraints

RC-PORT-IDS separates deployment feasibility from runtime feasibility. Storage is evaluated before deployment through **Equation 8**, while latency is evaluated after the resource state is observed through **Equation 12**. This structure enforces runtime selection among models that are both deployed and latency-feasible in the observed resource state.

For each controlled resource state *s*, the latency limit is obtained from the measured latency distribution across the candidate pool. For a selected latency-SLA quantile α, the limit is shown in **Equation 14**:

$$\boldsymbol{\tau}_{s}(\boldsymbol{\alpha}) = \mathbf{Q}_\boldsymbol{\alpha}\left(\{\ell_{ms} : \mathbf{m} \in \mathcal{M}\}\right), \ 0 < \boldsymbol{\alpha} < 1 \tag{14}$$

where $Q_\alpha(\cdot)$ is the empirical α-quantile. The experiments evaluate fixed α values and solve the deployment problem separately for each setting. The resource-state weights π_s are defined independently of test outcomes and satisfy $\Sigma_s \pi_s = 1$.

3.7. Calibrated state-loss objective

RC-PORT-IDS uses a calibrated loss contribution q_{mr} for model m in observable traffic regime *r*. The definition of q_{mr} is obtained from calibration

evidence and depends on the evaluation stage. In the CICIoT2023 development analysis, the traffic-regime probability and regime-level F1 are combined as in **Equation 15**:

$$\mathbf{q}_{\mathrm{mr}}^{\mathrm{dev}} = \rho_{\mathrm{r}}\left(1-\mathbf{F1}_{\mathrm{mr}}\right) \tag{15}$$

where ρ_r is the calibration probability of traffic regime r, and $F1_{mr}$ is the regime-level binary F1 of model m. In the independent external replication, the confirmatory objective uses balanced-error contributions so that benign and attack errors are weighted equally under the strongly imbalanced test distribution (**Equation 16**):

$$\mathbf{q}_{\mathrm{mr}}^{\mathrm{ext}} = 1/2\left(\mathbf{FP}_{\mathrm{mr}}/\mathbf{N}_0 + \mathbf{FN}_{\mathrm{mr}}/\mathbf{N}_1\right) \tag{16}$$

where FP_{mr} and FN_{mr} are the calibration false-positive and false-negative counts for model m in regime r, and N_0 and N_1 are the total benign and attack observations in the external calibration data. For either definition of q_{mr}, the loss of the recourse policy in resource state s is shown in **Equation 17**:

$$\mathbf{L}_{\mathrm{s}} = \boldsymbol{\Sigma}_{\mathrm{r}} \in \boldsymbol{\mathcal{R}} \boldsymbol{\Sigma}_{\mathrm{m}} \in \boldsymbol{\mathcal{M}} \mathbf{q}_{\mathrm{mr}} \mathbf{y}_{\mathrm{mrs}} + \mathbf{u}_{\mathrm{s}} \tag{17}$$

where the first term accumulates the calibrated loss contributions of the selected models. When $u_s = 1$, the selection variables are zero under **Equation 13**, and L_s is set to 1. The resource-weighted policy loss is shown in **Equation 18**:

$$\mathbf{Z} = \boldsymbol{\Sigma}_{\mathrm{s}} \in \boldsymbol{\mathcal{S}} \boldsymbol{\pi}_{\mathrm{s}} \mathbf{L}_{\mathrm{s}}, \boldsymbol{\pi}_{\mathrm{s}} \geq 0, \boldsymbol{\Sigma}_{\mathrm{s}} \in \boldsymbol{\mathcal{S}} \boldsymbol{\pi}_{\mathrm{s}} = 1 \tag{18}$$

where π_s is the predefined weight of resource state s. The experiments assign equal weight to the 12 controlled resource states, so $\pi_s = 1/12$ for every $s \in \mathcal{S}$. These weights define a controlled design distribution and are not estimates of production-state frequencies. For the external balanced-error formulation, the corresponding policy balanced accuracy is shown in **Equation 19**:

$$\mathbf{BA} = 1 - \mathbf{Z} \tag{19}$$

where BA is the main external policy metric.

3.8. Two-stage stochastic formulation

The calibrated policy loss serves portfolio and recourse selection, while the individual candidate classifiers retain their original training objectives. Combining the deployment, recourse, and resource constraints gives the finite-state two-stage optimization problem. The first-stage vector x is shared across all states, while y and u are state-dependent recourse variables (**Equation 20**):

$$\begin{aligned}
&\min_{\mathrm{X},\gamma,\mathrm{u}} \boldsymbol{\Sigma}_{\mathrm{s}} \in \boldsymbol{\mathcal{S}} \boldsymbol{\pi}_{\mathrm{s}} \left[\boldsymbol{\Sigma}_{\mathrm{r}} \in \boldsymbol{\mathcal{R}} \boldsymbol{\Sigma}_{\mathrm{m}} \in \boldsymbol{\mathcal{M}} \mathbf{q}_{\mathrm{mr}} \mathbf{y}_{\mathrm{mrs}} + \mathbf{u}_{\mathrm{s}}\right] \\
&\mathbf{s.t.} \boldsymbol{\Sigma}_{\mathrm{m}} \in \boldsymbol{\mathcal{M}} \mathbf{c}_{\mathrm{m}} \mathbf{x}_{\mathrm{m}} \leq \mathbf{B},\ \boldsymbol{\Sigma}_{\mathrm{m}} \in \boldsymbol{\mathcal{M}} \mathbf{x}_{\mathrm{m}} \leq \mathbf{K}_{\max}, \\
&\mathbf{y}_{\mathrm{mrs}} \leq \mathbf{x}_{\mathrm{m}},\ \mathbf{y}_{\mathrm{mrs}} \leq \mathbf{a}_{\mathrm{ms}},\ \forall \mathbf{m}, \mathbf{r}, \mathbf{s}, \\
&\boldsymbol{\Sigma}_{\mathrm{m}} \in \boldsymbol{\mathcal{M}} \mathbf{y}_{\mathrm{mrs}} = 1 - \mathbf{u}_{s},\ \forall \mathbf{r}, \mathbf{s}, \\
&\mathbf{x}_{\mathrm{m}}, \mathbf{y}_{\mathrm{mrs}}, \mathbf{u}_{\mathrm{s}} \in \{0,1\}
\end{aligned} \tag{20}$$

where the objective minimizes the resource-weighted calibrated policy loss, the first two constraints define the admissible deployed portfolio, the next two constraints enforce runtime selection within the deployed and latency-feasible model set, and the final selection constraint assigns the infeasible-state penalty when required. With seven candidate models and $K_{max} = 3$, the admissible portfolio space is finite and can be evaluated exactly, so the implementation returns the globally optimal portfolio for each defined storage-budget and latency-SLA condition.

3.9. Recourse gain relative to static deployment

To quantify the benefit of runtime recourse, the optimized portfolio is compared with the best feasible static single-model deployment selected from the same calibration evidence under the same storage budget and latency-SLA setting. Let Z^{static} denote the calibrated loss of the constrained static policy, and Z^{RP} denote the calibrated loss of the optimized recourse policy. The recourse gain is shown in **Equation 21**:

$$\mathbf{G}_{\mathrm{re}}\mathbf{c} = \mathbf{Z}^{\mathrm{static}} - \mathbf{Z}^{\mathrm{RP}} \tag{21}$$

where $G_{re}c > 0$ indicates lower policy loss for the recourse solution than for the constrained static comparator. In the external balanced-error

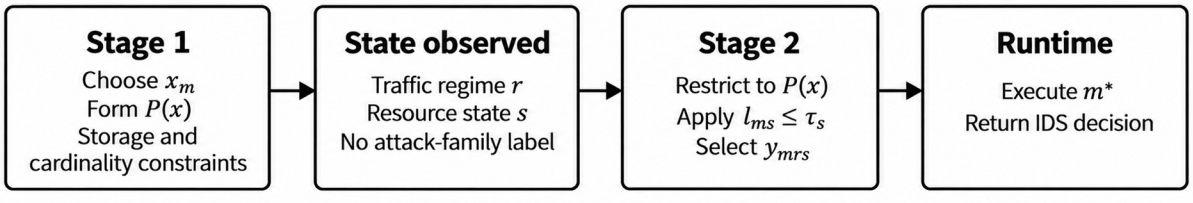


Figure 2. Two-stage decision structure of RC-PORT-IDS
Abbreviation: IDS: Intrusion detection system.

evaluation, this quantity is numerically equal to $BA^{RP} - BA^{static}$. Traffic-only and resource-only policies use the same calibration inputs and deployment constraints but remove one recourse component. These variants are used as ablations in Section 5.

3.10. RC-PORT-IDS solution procedure

Because the candidate pool is finite and the portfolio-size constraint is fixed at $K_{max} = 3$, RC-PORT-IDS solves the first-stage portfolio problem by exact enumeration of all admissible portfolios. For each candidate portfolio and resource state, the latency-feasible model set is identified from the measured latency lookup. For each observable traffic regime, the feasible model with the minimum calibrated loss contribution q_{mr} is selected, with latency used only as a tie-breaker. The resulting state losses are aggregated using π_s, and the portfolio with the minimum weighted loss is retained. This exact search returns the globally optimal solution for the defined finite portfolio problem and deployment constraints. Algorithm 1 summarizes the portfolio search and construction of the frozen recourse mapping used in the experiments.

The output of **Algorithm 1** is the deployed portfolio P* and the deterministic mapping g*(r,s). Both are frozen after calibration. During test evaluation, the method assigns the observed traffic to *r*, reads the current resource state *s*, and executes g*(r,s). The deployed portfolio and recourse mapping remain fixed throughout test evaluation and are not updated from test labels.

Algorithm 1. RC-PORT-IDS portfolio selection and frozen recourse construction.

Input: $\mathcal{M}$, $\mathcal{R}$, $\mathcal{S}$, $\{q_{mr}\}$, $\{c_m\}$, B, K_{max}, $\{\ell_{ms}\}$, $\{\tau_s\}$, $\{\pi_s\}$
Output: P* and frozen mapping g*(r,s)

$\mathcal{C} \leftarrow \emptyset$
for each $P \subseteq \mathcal{M}$ such that $|P| \le K_{max}$ and $\Sigma_{m \in P} c_m \le B$ do
  $Z(P) \leftarrow 0$
  for each $s \in \mathcal{S}$ do
    $F_s(P) \leftarrow \{m \in P : \ell_{ms} \le \tau_s\}$
    if $F_s(P) = \emptyset$ then
      $L_s(P) \leftarrow 1$
    else
      $L_s(P) \leftarrow 0$
      for each $r \in \mathcal{R}$ do
        $m^* \leftarrow \arg\min_{m \in F_s(P)} (q_{mr}, \ell_{ms})$
        $g_P(r,s) \leftarrow m^*$
        $L_s(P) \leftarrow L_s(P) + q_{m^*r}$
      end for
    end if
    $Z(P) \leftarrow Z(P) + \pi_s L_s(P)$
  end for
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{(P,Z(P))\}$
end for
$P^* \leftarrow \arg\min_(P,Z(P)) \in \mathcal{C}\ Z(P)$
$g^*(r,s) \leftarrow g_P^*(r,s)$
return P*, g*

4. Datasets and experimental protocol

This section describes the datasets, preprocessing steps, candidate model pool, observable-state construction, resource benchmarking, internal development protocol, and independent external replication used to evaluate RC-PORT-IDS. The protocol was designed to separate development evidence from the final external replication. CICIoT2023[4] was used for model-pool development, complementarity analysis, and internal recourse-gain analysis. CICIoMT2024[13] was used for an independent replication in which the policy was calibrated from external training and profiling data before the external test split was evaluated.

4.1. CICIoT2023 development dataset

The primary development experiments used CICIoT2023, which provides flow-level features extracted from IoT traffic containing benign activity and 33 attack labels. The official dataset includes comma-separated value (CSV) feature files derived from captured traffic and was introduced as a large-scale benchmark for IoT attack detection.[4]

The local CICIoT2023 archive contained 309 CSV files with 46,776,697 rows. All files used the same 39-column numeric feature schema. The class identity was obtained from the source directory structure because the downloaded CSV files did not contain a separate label column. The 34 fine labels were retained for stratification and diagnostic analysis. They were also mapped to eight broader traffic families: Benign, BruteForce,

distributed denial of service (DDoS), denial of service (DoS), Mirai, Recon, Spoofing, and Web. The family mapping was not used as a runtime input to the RC-PORT-IDS recourse rule.

A maximum of 50,000 rows was sampled from each fine label using seed 2,026, while all available rows were retained for labels below the cap. This produced an initial working set of 1,292,371 rows. **Table 2** shows the roles and retained sample counts for the two datasets used in the study. The table separates the CICIoT2023 development subsets from the CICIoMT2024 external-replication subsets so that the evaluation role of each split is explicit.

4.2. Data preprocessing and duplicate-conflict control

All 39 CICIoT2023 feature columns were treated as numeric model inputs. Quality control identified 104 NaN values and 60 infinite values in the capped working set. Because the invalid entries overlapped at the row level, 60 rows were removed, leaving 1,292,311 valid rows before duplicate handling.

Feature-vector duplication was audited before the development split. Records were treated as identical when all 39 numeric feature values matched exactly after invalid-value handling; a conflicting group contained the same feature vector under more than one fine label. All 107,193 rows belonging to conflicting fine-label groups were removed rather than assigning one label to the shared vector. After conflict removal, 64,987 additional same-label duplicate rows were removed so that the retained data contained no repeated feature vectors. The resulting development dataset contained 1,120,131 rows, 34 fine labels, and eight traffic families. This step was applied before the split to reduce direct duplicate leakage between development subsets.

The final development data were stratified by fine label using seed 2,026 into 70% training, 15% calibration, and 15% development-test subsets. The resulting counts were 784,091, 168,020, and 168,020 rows, respectively. The development-test subset was used for internal method development and evaluation, while CICIoMT2024 provided independent external replication with separate training, profiling, and final test partitions.

Figure 3 shows the complete experimental protocol used in the study. The left branch summarizes the CICIoT2023 development path from the raw CSV archive to the final stratified development subsets. The right branch shows the independent CICIoMT2024 replication, including the external training set, three-fold out-of-fold predictions, the profiling split, policy freezing, and the final test evaluation. The resource-state grid is shared by the portfolio-selection steps.

4.3. Candidate detection models

The candidate pool contained seven supervised binary classifiers: Logistic regression, Random forest, ExtraTrees, HistGradientBoosting, extreme gradient boosting (XGBoost), Light gradient boosting machine (LightGBM), and CatBoost. The binary target was defined as benign versus attack. These model families were retained because they represent linear, bagged-tree, randomized-tree, histogram-based boosting, and gradient-boosting alternatives with different prediction-time and storage characteristics. No attack-family label was supplied to any classifier or to the runtime recourse rule.

The same seven model families were used throughout the internal development stage and the external replication. No hyperparameter search space was explored: each model family used the single configuration reported in **Table 3**, so the candidates did not receive unequal tuning budgets. The classifiers consumed the same 39-column numeric feature frame without model-specific feature preprocessing. No Platt, isotonic, or other probability-calibration step was applied; probability-based diagnostics used each fitted estimator's native positive-class probability. Internal fitted models were frozen after training and were used without retraining in the internal portfolio analyses. For the independent external replication, the same model specifications were retrained on the CICIoMT2024 external training split. The CICIoMT2024 test split was not used for model fitting, model-family selection, threshold tuning, clustering, or policy calibration. **Table 3** reports the hyperparameters set explicitly in the implementation; remaining estimator options used the defaults of the installed library versions. No post-training decision-threshold tuning was applied, and class labels were obtained with each fitted estimator's native predict() rule.

4.4. Observable-state construction

Runtime traffic states were constructed from the 39 observable feature values only. The 39-variable vector comprised Header_Length, protocol type, Time_To_Live, Rate; fin, syn, rst, psh, ack, ece, and cwr flag-number variables; ack, syn, fin, and rst counts; HTTP, HTTPS, DNS, Telnet, SMTP, SSH, IRC, TCP, UDP, DHCP, ARP, ICMP,

Table 2. Dataset roles and retained samples used in RC-PORT-IDS

Dataset/subset	Role	Rows	Features	Use in the protocol
CICIoT2023 final set	Development	1,120,131	39	Internal development evidence
Training	Development	784,091	39	Model fitting and regime construction
Calibration	Development	168,020	39	Policy calibration/model comparison
Development test	Development	168,020	39	Internal development evaluation
CICIoMT2024 train	External replication	674,977	39 shared	External model fitting
CICIoMT2024 profiling	External replication	361,081	39 shared	Full-train calibration predictions
CICIoMT2024 test	External replication	1,611,992	39 shared	Final external policy evaluation

Abbreviations: CICIoMT2024: Canadian Institute for Cybersecurity Internet of Medical Things 2024; CICIoT2023: Canadian Institute for Cybersecurity Internet of Things 2023.

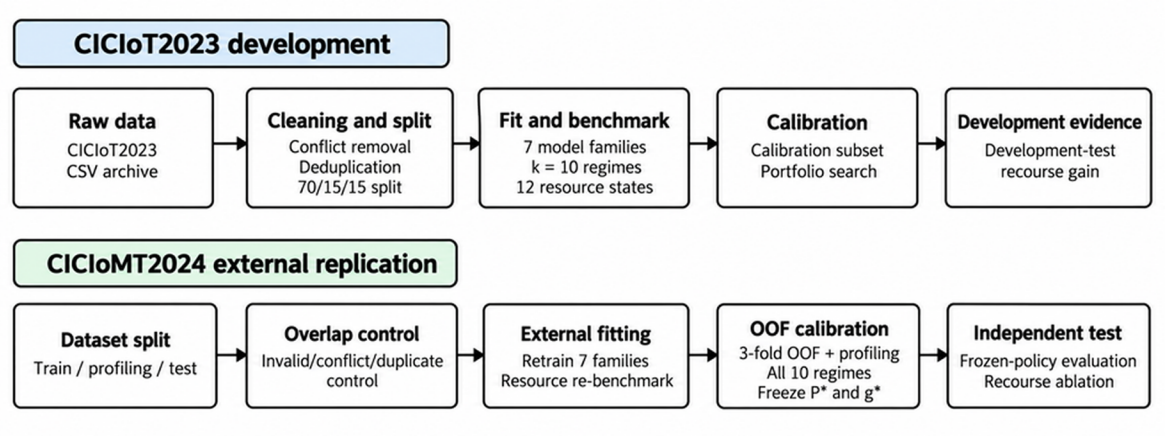


Figure 3. Experimental protocol for internal development and independent external replication
Abbreviations: CICIoMT2024: Canadian Institute for Cybersecurity Internet of Medical Things 2024; CICIoT2023: Canadian Institute for Cybersecurity Internet of Things 2023; CSVs: Comma-separated values; OOF: Out-of-fold.

IGMP, IPv, and LLC indicators; and Tot sum, Min, Max, AVG, Std, Tot size, IAT, Number, and Variance. The training features were standardized using StandardScaler and then partitioned using MiniBatchKMeans. Mini-batch k-means was used because it supports clustering on large tabular datasets while updating the cluster centers from subsets of observations.[32]

The initial state-construction analysis evaluated $k \in \{4, 6, 8, 10\}$. The highest silhouette score among these evaluated settings was obtained at $k = 10$, which was fixed for the development pipeline. The fitted scaler and cluster model were then used without label input to assign each calibration and development-test observation to one of the ten observable regimes. The runtime state variable was the observable cluster index; source-directory labels, the fine attack label, the binary target, and attack-family identity were excluded from regime construction. Protocol and flow variables may correlate with particular attacks, but none is derived from a ground-truth class label.

4.5. Resource-state benchmarking

Resource state was represented by the number of

Table 3. Candidate detection model pool

Model	Model family	Frozen configuration
Logistic regression	Linear classifier	max_iter=400; solver=liblinear; random_state=2026
Random forest	Bagged tree ensemble	n_estimators=160; max_depth=None; min_samples_leaf=2; class_weight=balanced_subsample; n_jobs=-1; random_state=2026
ExtraTrees	Randomized tree ensemble	n_estimators=160; max_depth=None; min_samples_leaf=2; class_weight=balanced; n_jobs=-1; random_state=2026
HistGradientBoosting	Histogram boosting	max_iter=180; learning_rate=0.08; max_leaf_nodes=31; random_state=2026
XGBoost	Gradient-boosted trees	n_estimators=220; max_depth=6; learning_rate=0.08; subsample=0.9; colsample_bytree=0.9; objective=binary:logistic; eval_metric=logloss; tree_method=hist; n_jobs=-1; random_state=2026
Light gradient boosting machine	Gradient-boosted trees	n_estimators=220; learning_rate=0.08; num_leaves=31; subsample=0.9; colsample_bytree=0.9; n_jobs=-1; random_state=2026; verbosity=-1
CatBoost	Gradient-boosted trees	iterations=220; depth=8; learning_rate=0.08; loss_function=Logloss; random_seed=2026; thread_count=-1; verbose=False

available CPU cores and the inference batch size. Four CPU settings, {1, 2, 4, 16}, were crossed with three batch sizes, {1, 64, 1024}, producing 12 controlled resource states. The internal benchmark used a deterministic 20,000-row sample from the CICIoT2023 development-test feature matrix. Process affinity was restricted to the requested core count. For batch sizes 1, 64, and 1,024, the timed evaluations processed 1,000, 10,000, and 20,000 rows, respectively. Each state used one warm-up prediction followed by three timed passes. The median elapsed time was divided by the number of processed rows to obtain latency in microseconds per row.

The external fitted models were re-benchmarked on a deterministic 20,000-row sample from the profiling split. When supported by the estimator, n_jobs or thread_count was set to the requested core count, and library thread pools were limited to the same count. Each resource state again used a warm-up prediction and three timed passes, and the median per-row latency was retained. All candidate-model inference used the CPU; the RTX 4050 was not used in the latency measurements. Serialized model size was measured from the fitted model artifact used in the corresponding stage. Because deployment uses a portfolio rather than an isolated classifier, the same host was also used to measure complete-portfolio storage, process resident-set size (RSS) and peak-RSS increments, portfolio loading time, regime-assignment time, selector-dispatch time, end-to-end latency, and throughput for representative 1-core/batch-1 and 4-core/batch-64 states. All portfolio models were loaded before runtime selection, so switching changed the dispatch target without reloading a model artifact. Device energy was not measured because the host did not provide a calibrated CPU or system power meter.

The resource measurements were used as feasibility inputs rather than as labels for the intrusion-detection task. For a given resource state, latency-SLA thresholds were defined from the 0.25, 0.50, and 0.75 quantiles of the seven measured model latencies in that state. A model was feasible only when its measured latency did not exceed the corresponding threshold. The complete 7×12 latency lookup was used directly by the portfolio and recourse procedures. Resource states were assigned equal weight, $\pi_s = 1/12$.

4.6. Internal development protocol

The internal development protocol was conducted in stages. First, the seven candidate models were

trained on the CICIoT2023 training subset and evaluated on the calibration and development-test subsets. Overall binary performance and regime-specific performance were recorded. Second, model complementarity was examined across traffic regimes and resource states. Third, the pre-deployment portfolio decision was evaluated under storage and latency constraints with a maximum portfolio size of three models.

The internal recourse-gain analysis compared the recourse problem with the best feasible static singleton under the same storage budget and latency-SLA setting. The development loss used q_{mr}^{dev} from **Equation 15**, which combines the calibration probability of regime r with $1 - F1_{mr}$. The development-test subset served as the internal evaluation set for model complementarity and recourse analysis, while the independent CICIoMT2024 replication was reserved for the confirmatory policy comparison using the balanced-error loss in **Equation 16** and balanced accuracy as the reported policy metric.

4.7. Portfolio, storage-budget, and latency-SLA settings

The final portfolio search used $K_{max} = 3$, selected from the development frontier. The main marginal gain occurred when portfolio capacity increased from one to two models, while the third model provided an average additional gain of 0.000017. This configuration retains a third deployment option when storage and latency feasibility make it useful and defines a finite decision space that is solved exactly.

Table 4 shows the fixed settings used for the portfolio and resource analysis. The storage budgets and SLA quantiles were applied as controlled deployment conditions rather than as learned parameters. The same resource-state grid was retained in the external replication, while the external models were re-benchmarked after retraining so that latency and storage feasibility were based on the external fitted artifacts rather than on the CICIoT2023 model files.

4.8. CICIoMT2024 independent external replication

The independent external replication used CICIoMT2024, a multi-protocol Internet of Medical Things security dataset collected from a testbed containing real and simulated medical-IoT devices and including Wi-Fi, MQTT, and Bluetooth traffic. The dataset provides predefined training, profiling, and test data and includes DDoS, DoS, Recon, MQTT, and spoofing attack categories.[13]

The WiFi/MQTT processed parquet used in this study was obtained from a cleaned CICIoMT2024 derivative archived on Zenodo. Before any external model fitting, its schema was audited against the CICIoT2023 feature schema. All 39 CICIoT2023 features were present by name after canonical space/underscore normalization. The processed file contained 9,128,910 rows and 50 columns, including label, split, and source metadata.[33]

The external training data were capped at a maximum of 50,000 rows per fine label using seed 2,026 and were subjected to the same invalid-value, feature-conflict, and duplicate controls used in the development pipeline. The retained external training set contained 674,977 rows. The profiling split retained 361,081 rows after overlap control. The final external test set contained 1,611,992 rows after removing 125 feature vectors that overlapped with the external training or profiling data. The test set contained 37,607 benign rows and 1,574,385 attack rows.

The seven model families were retrained on the external training subset. The observable-state pipeline was fitted on external training features with $k = 10$, using the same 39-feature representation. Resource latency and model size were then re-measured for the external fitted models using the same 12 controlled resource states. No external-test label or model performance was used to select the portfolio, the recourse rule, the storage budget, or the SLA setting.

4.9. Out-of-fold regime-complete calibration

The final external policy was calibrated using three-fold out-of-fold predictions on the external training subset together with predictions from the independently retained profiling split. The training folds were stratified by fine label. For each model family, two folds were used for fitting, and the remaining fold was predicted; the cycle continued until every external training observation had a prediction from a classifier that had not been fitted on that observation. The classifiers used the common numeric feature frame without fold-specific model preprocessing. The separate label-free regime pipeline, consisting of StandardScaler and MiniBatchKMeans with $k = 10$, was fitted once on the full external training

Table 4. Fixed experimental settings for RC-PORT-IDS

Setting	Value	Purpose
Random seed	2026	Primary deterministic sampling and split seed
Observable regimes	$k = 10$	Frozen traffic-state representation
CPU-core states	{1, 2, 4, 16}	Controlled resource availability
Batch-size states	{1, 64, 1024}	Controlled inference workload
Resource states	12	CPU-core × batch-size combinations
Portfolio-size limit	$K \leq 3$	Pre-deployment model subset limit
Storage budgets (MB)	{0.35, 1, 128, 200, 320}	Deployment feasibility conditions
Latency-SLA quantiles	{0.25, 0.50, 0.75}	State-specific latency limits
External OOF folds	3	Calibration without using test outcomes
Final policy metric	Balanced error / balanced accuracy	Confirmatory external comparison
Resource-state weights	$\pi_s = 1/12$	Uniform controlled-design weighting across 12 states
Static comparator	Best calibration-selected feasible singleton	Uses the same storage and latency rules as the recourse problem

Abbreviations: OOF: Out-of-fold; SLA: Service-level agreement.

feature set and was not refitted inside the classifier folds; profiling and test features were excluded from fitting this regime pipeline.

The out-of-fold training predictions and the profiling predictions were then combined at the observable-regime level. This supplied calibration evidence for all ten regimes; regimes 0, 4, and 6 were contributed by the out-of-fold training predictions, and no regime present in the external test set was absent from calibration. For each regime and model, false positives and false negatives were accumulated and converted to q_{mr}^{ext} using **Equation 16**. The portfolio and its state-dependent recourse mapping were constructed from these calibration quantities and the external resource measurements. External test labels were not used in policy construction.

For each storage budget B and SLA quantile α, the static comparator was selected from singleton portfolios using the same calibration loss, resource-state weights, storage constraint, and latency feasibility rules used by the recourse problem. If the fixed model violated the latency limit in a resource state, that state received the predefined infeasible-state loss. The static model was frozen before test evaluation. Model-level test results were therefore diagnostic outputs and did not participate in selecting either the static comparator or the recourse portfolio.

4.10. Constrained test-oracle diagnostic and policy sensitivity

The external evaluation included a constrained test-oracle static diagnostic in addition to the frozen calibration-selected static comparator. For each storage budget and SLA quantile, the test-oracle policy was defined as the feasible singleton with the highest test balanced accuracy under the same storage constraint, latency-feasibility lookup, resource-state weights, and infeasible-state rule. It therefore has privileged access to test outcomes for singleton selection but does not use per-observation ground-truth labels as a runtime input. The diagnostic does not fit a model, choose the recourse portfolio, or modify the recourse mapping; it provides a reference for the best feasible static singleton under each deployment condition.

Policy sensitivity was evaluated around the two design components that determine state

assignment and runtime feasibility. Observable-regime sensitivity used k ∈ {8, 10, 12} with seed 2,026, together with seed values 2,026, 2,027, and 2,028 at k = 10. Latency sensitivity multiplied every state-specific SLA threshold by 0.95, 1.00, or 1.05 while retaining the corresponding measured latency matrix. For each setting, the same 15 storage-budget/SLA cases were re-evaluated, and the mean full-policy balanced accuracy, recourse gain relative to the frozen static policy, number of positive-gain cases, and portfolio agreement with the baseline configuration were recorded. These analyses retained the fitted external model families and the external test split and changed only the stated regime-construction or latency-feasibility setting.

4.11. Evaluation metrics

Binary classifier performance was reported using precision, recall, specificity, F1-score, balanced accuracy, Matthews correlation coefficient (MCC), and area under the receiver-operating-characteristic curve (AUC).[34] Balanced accuracy was used as the main policy metric in the external replication because the external test split was strongly attack-dominated. This metric gives equal weight to the attack and benign classes rather than allowing the majority class to dominate the overall score. Policy-level diagnostics additionally included macro-F1, attack and benign recall, false-positive rate, MCC, precision-recall-AUC, 15-bin expected calibration error, Brier score, attack-family and fine-label recall, and observable-regime tail measures. Probability-based diagnostics used the native positive-class probability from the model selected by the frozen mapping; attack identities were used only for offline diagnostic reporting (**Equations 22**–**25**).

$$\mathbf{Precision} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FP}} \tag{22}$$

where TP denotes true-positive attack predictions, and FP denotes benign observations predicted as attacks.

$$\mathbf{Recall} = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} \tag{23}$$

where FN denotes attacks predicted as benign, and Recall is the attack-class true-positive rate.

$$\mathbf{Specificity} = \frac{\mathbf{TN}}{\mathbf{TN} + \mathbf{FP}} \tag{24}$$

where TN denotes true-negative benign predictions and Specificity is the benign-class true-negative rate.

$$\boldsymbol{F1} = \frac{2 \cdot \boldsymbol{Precision} \cdot \boldsymbol{Recall}}{\boldsymbol{Precision} + \boldsymbol{Recall}} \tag{25}$$

where F1 is the harmonic mean of precision and recall for the attack class.

Balanced accuracy was computed as the mean of Recall and Specificity, which is appropriate when the two classes have substantially different frequencies (**Equations 26 and 27**).[35]

$$\boldsymbol{BA} = \frac{1}{2}\left(\boldsymbol{Recall} + \boldsymbol{Specificity}\right) \tag{26}$$

where BA denotes balanced accuracy. The corresponding balanced error used by the final external policy is BE = 1 − BA.

$$\boldsymbol{MCC} = \frac{\boldsymbol{TP} \cdot \boldsymbol{TN} - \boldsymbol{FP} \cdot \boldsymbol{FN}}{\sqrt{\left(\boldsymbol{TP} + \boldsymbol{FP}\right)\left(\boldsymbol{TP} + \boldsymbol{FN}\right)\left(\boldsymbol{TN} + \boldsymbol{FP}\right)\left(\boldsymbol{TN} + \boldsymbol{FN}\right)}} \tag{27}$$

where MCC denotes the Matthews correlation coefficient.[36] MCC was reported as a supplementary model-level measure because it uses all four entries of the binary confusion matrix.

4.12. Computational environment and reproducibility

All experiments were executed on a Windows 10 laptop with 32 GB RAM and an NVIDIA GeForce RTX 4050 laptop GPU with 6 GB memory. The project used Python (3.11.9, Python Software Foundation, Netherlands). The final environment included NumPy 2.4.6, pandas 3.0.6, scikit-learn 1.9.1, SciPy 1.17.1, XGBoost 3.2.0, LightGBM 4.7.0, CatBoost 1.2.10, and psutil 7.2.2.

All fitted artifacts, intermediate results, scripts, and freeze records were retained in a versioned project archive. The final scientific freeze recorded SHA-256 hashes for the retained scripts, result files, model artifacts, and regime pipelines so that the evaluated experimental state could be traced after analysis. The public reproducibility package described in the Availability of Data section is intended to contain the complete deployment grid, resource measurements, frozen configurations, and scripts used to reproduce the reported analyses.

The final experimental protocol therefore separates exploratory development from confirmatory external replication. CICIoT2023 supports the internal model-pool and recourse-gain development analysis, while CICIoMT2024 provides an independent train-calibrate-test replication in which policy construction excludes the final test labels. The outputs produced by this protocol are used in Section 5 for the internal

recourse-gain results, external replication results, and recourse ablation.

5. Results and discussion

This section reports the internal development results and the independent CICIoMT2024 replication. The internal results are used to show model complementarity, resource-dependent model selection, and recourse gain relative to constrained static deployment. The external results evaluate the same decision structure after the candidate model families are retrained and the policy is calibrated without using the external test labels. Balanced accuracy is used as the main external policy measure. The external analysis also reports a constrained test-oracle static diagnostic and sensitivity of the final policy to observable-regime construction and latency-threshold perturbations.

5.1. Baseline model performance and complementarity

The seven candidate models produced similar overall binary F1 values on the CICIoT2023 development-test subset, but their latency and storage requirements differed. **Table 5** shows the development-test F1, recall, AUC, measured inference latency, stored model size, and Pareto status for the seven fitted models.

LightGBM produced the highest overall F1 of 0.9827. CatBoost, XGBoost, and HistGradientBoosting were within 0.0004 of this value. Logistic regression had a lower F1 of 0.9775 but had the smallest latency and storage footprint in the initial benchmark.

The overall ranking did not remain fixed across the offline attack-family diagnostics. ExtraTrees produced the highest family-level F1 for BruteForce, DDoS, DoS, Mirai, and Web traffic. LightGBM was highest for Recon, and random forest was highest for Spoofing. The winning margins ranged from 0.0014 for Recon to 0.0400 for BruteForce. These family-level results were used only as an offline complementarity diagnostic. Attack-family identity was not supplied to the runtime recourse rule.

5.2. Resource-constrained model-selection behavior

The resource benchmark changed the feasible and preferred models across CPU-core and batch-size settings. The initial family-by-resource analysis produced 84 family-resource combinations and nine distinct Pareto sets. A broader development-only constraint sweep then crossed 10 observable regimes, 12 resource states, seven storage budgets {0.25, 0.30, 0.35, 1, 16, 128, 200} MB, and five state-specific latency limits. The latency limits were 1.05 times the minimum measured latency, the 0.25, 0.50, and 0.75 latency quantiles, and 1.05 times the maximum measured latency. This produced $10 \times 12 \times 7 \times 5 = 4{,}200$ development cases, of which 4,160 were feasible. Five models were selected as the constrained winner across these cases: CatBoost, HistGradientBoosting,

Table 5. CICIoT2023 development-test performance and initial deployment measurements

Model	F1	Recall	AUC	Latency (µs/row)	Size (MB)	Pareto
LightGBM	0.9827	0.9892	0.9751	6.743	0.32	Yes
CatBoost	0.9824	0.9896	0.9742	0.868	0.33	Yes
XGBoost	0.9823	0.9886	0.9747	1.916	0.23	Yes
HistGradientBoosting	0.9823	0.9886	0.9745	5.570	0.29	No
Random forest	0.9818	0.9809	0.9742	10.384	120.59	No
Logistic regression	0.9775	0.9981	0.9015	0.276	<0.01	Yes
ExtraTrees	0.9714	0.9546	0.9682	11.839	188.62	No

Abbreviations: AUC: Area under the receiver-operating-characteristic curve; CICIoT2023: Canadian Institute for Cybersecurity Internet of Things 2023; LightGBM: Light gradient boosting machine; XGBoost: Extreme gradient boosting.

LightGBM, logistic regression, and XGBoost. The main analysis used the five-budget and three-SLA grid reported in **Table 4**.

The resource constraints changed the unconstrained winner in 2,051 of the 4,160 feasible cases, corresponding to 49.30%. In 423 cases, the change was associated with an F1 difference of at least 0.001. All ten observable traffic regimes had more than one winner across the evaluated budget conditions. This result established that model availability and latency feasibility were not constant across the controlled deployment conditions.

5.3. Internal two-stage recourse gain

The internal two-stage analysis evaluated 15 combinations of storage budget and latency-SLA quantile. **Table 6** summarizes the recourse gain relative to the constrained static single-model policy. The gain was positive in 13 of 15 cases and was at least 0.001 in five cases. The mean recourse gain was 0.001512, the median was 0.000158, and the maximum was 0.004597. Across the 15 paired cases, the mean gain had a t-based 95% confidence interval of [0.000348, 0.002676] and a bootstrap 95% interval of [0.000596, 0.002568]. The paired t-test gave $p = 0.0145$, the two-sided Wilcoxon signed-rank test gave $p = 0.00133$, and Cohen's d_z was 0.72. The absolute mean gain corresponds to 0.151 percentage points and is therefore interpreted as a consistent but modest development-stage effect.

Figure 4 shows the mean, median, and maximum internal recourse gain. Recourse gain varied across deployment conditions, with the larger gains concentrated under the tighter latency-SLA setting. The maximum gain, 0.004597, occurred at the 0.25 latency-SLA quantile with storage budgets of 1, 128, 200, and 320 MB. Under the same SLA quantile, the 0.35 MB budget produced a gain of 0.003392.

Table 6. Internal recourse-gain summary

Measure	Value	Rate
Deployment cases	15	–
Positive recourse-gain cases	13/15	86.7%
Recourse gain ≥ 0.001	5/15	33.3%

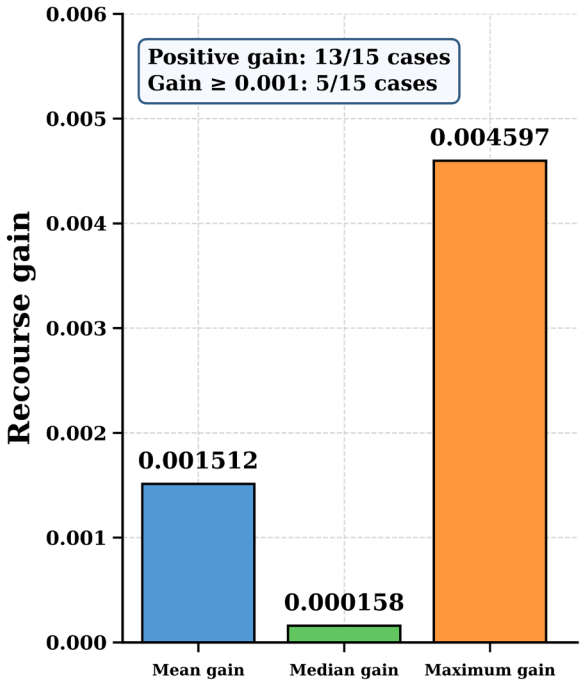


Figure 4. Internal recourse gain relative to constrained static deployment across the evaluated development cases

5.4. Effect of portfolio size

The development frontier showed that the main marginal gain occurred when the portfolio-size limit increased from one model to two models. **Table 7** shows the mean development gain associated with each increment. Increasing the limit from one model to two produced a mean gain of 0.001499, while increasing the limit from two models to three produced an additional mean gain of 0.000017. The resulting frontier shows diminishing marginal gain after two-model capacity while retaining the third model as an additional feasible deployment option.

Table 7. Marginal effect of the portfolio-size limit

Portfolio-size change	Mean development gain	Observed effect
$1 \rightarrow 2$	0.001499	Main marginal gain
$2 \rightarrow 3$	0.000017	Diminishing marginal gain

Figure 5 shows the same marginal comparison. The development frontier exhibits a clear diminishing-return pattern after the

second model. The final search therefore retains $K_{max} = 3$: two models capture the main average recourse gain, while the third model preserves an additional feasible deployment option when the storage and latency conditions favor a different portfolio composition.

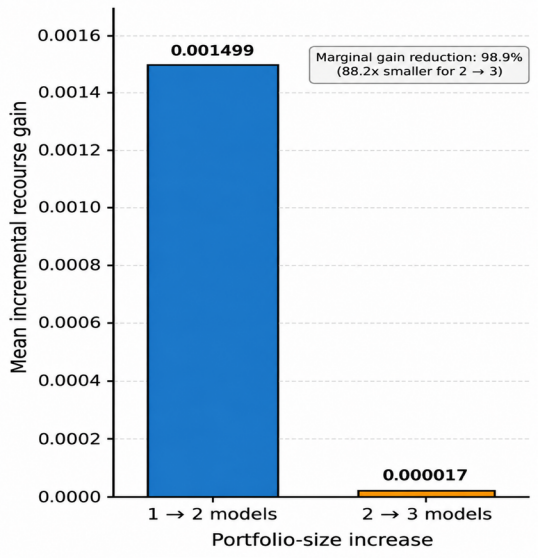


Figure 5. Mean marginal development gain when the portfolio-size limit is increased

5.5. Independent external model replication

The seven model families were retrained on the CICIoMT2024 external training subset before the external policy was calibrated. The final test subset contained 37,607 benign rows and 1,574,385 attack rows after overlap control. An all-attack classifier would therefore obtain an attack-class F1 of 0.988198 while its balanced accuracy would remain 0.500000. Balanced accuracy was therefore used alongside attack-class F1 to represent performance under this class distribution.

Table 8 shows the external test results for the seven retrained model families. ExtraTrees produced the highest balanced accuracy, 0.9824, with specificity of 0.9658 and AUC of 0.9999; random forest followed with balanced accuracy 0.9725. The five boosting or tree-ensemble models other than logistic regression produced balanced accuracy between 0.9647 and 0.9683. Logistic regression retained a high attack-class F1 of 0.9894 but had balanced accuracy of 0.6973 and specificity of 0.4016. These model-level test scores were recorded after policy freezing and were not used to select the static comparator or the recourse portfolio.

5.6. Out-of-fold complete external portfolio results

The final external policy used three-fold out-of-fold predictions from the external training subset together with the retained profiling split. This calibration covered all ten observable regimes before the test policy was evaluated. For each deployment condition, the static comparator was the calibration-selected feasible singleton defined in Section 4.9. Applying the same storage, latency-feasibility, resource-weight, and calibrated-loss rules selected logistic regression as the frozen static model in all 15 conditions. Full recourse produced a positive test gain in 12 of the 15 deployment cases. Its mean balanced accuracy was 0.888692, compared with 0.697275 for the frozen static policy, giving a mean recourse gain of 0.191417. The constrained test-oracle static policy defined in Section 4.10 obtained a mean balanced accuracy of 0.840868. The mean frozen-static regret to this oracle was 0.143593, while full recourse remained 0.047825 above the oracle mean. Across the same 15 frozen policy cases, mean macro-F1 was 0.908128, attack recall 0.997756, benign recall 0.779629, false-positive rate 0.220371, MCC 0.818851, precision-recall-AUC 0.993185, 15-bin expected calibration error 0.020419, and Brier score 0.007812. Weighted attack-family recall was lowest for Recon (mean 0.876647; range 0.603068-0.988897), followed by Spoofing (0.911047), while MQTT, DoS, and DDoS were at least 0.996733 on average. Recon VulScan was the lowest-recall fine attack label in every deployment case, ranging from 0.549951 to 0.830861. Among multi-model cases, the worst mixed-class regime balanced accuracy ranged from 0.623475 to 0.717630, and the 10th-percentile regime balanced accuracy ranged from 0.624338 to 0.742835.

Table 9 shows selected deployment cases from the frozen external policy. At the 0.75 SLA quantile and a 1 MB storage budget, the selected portfolio contained logistic regression and HistGradientBoosting. Its test balanced accuracy was 0.961941 compared with 0.697275 for the frozen static logistic regression policy, giving a recourse gain of 0.264666. With storage budgets from 128 to 320 MB at the same SLA quantile, the selected portfolio contained logistic regression, random forest, and LightGBM and produced a balanced accuracy of 0.961077. The complete 15-case deployment grid is retained as a machine-readable result and is included in the

Table 8. CICIoMT2024 independent external model replication

Model	F1	Balanced accuracy	Specificity	MCC	AUC	Size (MB)
ExtraTrees	0.9991	0.9824	0.9658	0.9597	0.9999	111.2298
Random forest	0.9991	0.9725	0.9453	0.9624	0.9999	61.7423
HistGradientBoosting	0.9990	0.9683	0.9371	0.9574	0.9999	0.6530
LightGBM	0.9990	0.9671	0.9346	0.9561	0.9999	0.7348
XGBoost	0.9990	0.9655	0.9313	0.9558	0.9999	0.6314
CatBoost	0.9990	0.9647	0.9296	0.9583	1.0000	0.8935
Logistic regression	0.9894	0.6973	0.4016	0.4710	0.5556	0.0017

Abbreviations: AUC: Area under the receiver-operating-characteristic curve; CICIoMT2024: Canadian Institute for Cybersecurity Internet of Medical Things 2024; LightGBM: Light gradient boosting machine; MCC: Matthews correlation coefficient; XGBoost: Extreme gradient boosting.

reproducibility package described on Zenodo at https://doi.org/10.5281/zenodo.23047315.

Figure 6 compares the three external decision references across the same 15 deployment cases. The frozen calibration-selected static policy averaged at 0.697275, the constrained test-oracle static policy averaged at 0.840868, and full traffic-plus-resource recourse averaged at 0.888692. Full recourse exceeded the oracle in four cases, matched it in three cases, and was lower in eight cases, while retaining the highest mean balanced accuracy across the evaluated grid.

5.7. Recourse ablation

The final ablation separated traffic-state recourse from resource-state recourse. **Table 10** shows the mean, median, minimum, and maximum test balanced accuracy for the four variants. Full traffic-plus-resource recourse produced the highest mean balanced accuracy, 0.888692, and improved over static deployment in 12 of 15 cases. Traffic-only recourse produced a lower mean of 0.837476 and improved over static deployment in eight cases. Resource-only recourse produced the same mean balanced accuracy as static deployment.

Figure 7 shows the mean balanced accuracy of the four ablation variants. Full recourse exceeded resource-only recourse in 12 of 15 cases and exceeded traffic-only recourse in seven cases. Traffic-only recourse reached a slightly higher maximum balanced accuracy in one deployment region, but it did not retain the same mean performance across the full grid. The resource-only variant did not change the selected outcome relative to the static baseline.

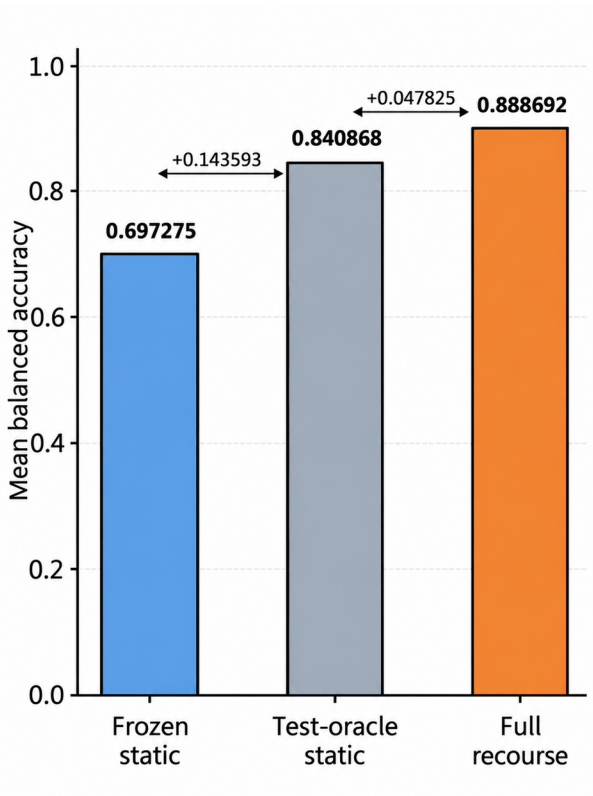


Figure 6. Mean external balanced accuracy of the frozen calibration-selected static policy, the constrained test-oracle static policy, and full traffic-plus-resource recourse

Table 9. Selected frozen external portfolio cases

SLA q	Storage (MB)	Frozen portfolio	Static model	Full BA	Static BA	Recourse gain
0.75	1.00	Logistic regression + HistGradientBoosting	Logistic regression	0.961941	0.697275	0.264666
0.75	128–320	Logistic regression + Random forest + LightGBM	Logistic regression	0.961077	0.697275	0.263802
0.50	1.00	Logistic regression + CatBoost	Logistic regression	0.958144	0.697275	0.260869
0.50	128–320	Logistic regression + XGBoost + LightGBM	Logistic regression	0.960358	0.697275	0.263082
0.25	128–200	Logistic regression + LightGBM + CatBoost	Logistic regression	0.915717	0.697275	0.218442

Abbreviations: BA: Balanced accuracy; LightGBM: Light gradient boosting machine; XGBoost: Extreme gradient boosting.

Table 10. External recourse ablation across 15 deployment cases

Mode	Mean BA	Median BA	Min BA	Max BA	Mean Δ vs. static	Positive cases
Full	0.888692	0.958144	0.697275	0.961941	0.191417	12/15
Traffic only	0.837476	0.958144	0.697275	0.962233	0.140201	8/15
Resource only	0.697275	0.697275	0.697275	0.697275	0.000000	0/15
Static	0.697275	0.697275	0.697275	0.697275	0.000000	0/15

Abbreviation: BA: Balanced accuracy.

5.8. Static diagnostic and policy sensitivity

The three external reference levels characterize different parts of the deployment decision. The frozen static policy represents the calibration-selected singleton available before test outcomes are known. The constrained test-oracle static policy represents the best feasible singleton for each deployment condition after the test outcomes are available. Full recourse retains a pre-deployed portfolio and selects among its feasible models after the observable state is known. Their mean balanced accuracies were 0.697275, 0.840868, and 0.888692, respectively. Across the 15 deployment cases, full recourse was above, equal to, or below the oracle static policy in 4, 3, and 8 cases, respectively.

Table 11 shows policy sensitivity to the observable-regime configuration and to ± 5% perturbations of the state-specific latency thresholds. The baseline configuration is k = 10, seed 2,026, and a threshold factor of 1.00.

Across the regime settings, mean full-policy balanced accuracy ranged from 0.880636 to 0.893119 and mean recourse gain ranged from 0.183361 to 0.195844, with 12 positive-gain cases in every configuration. Portfolio agreement ranged from 0.60 to 1.00, indicating that alternative regime partitions could change the deployed model set while preserving the direction of the aggregate result. Under ± 5% latency-threshold perturbations, mean full-policy balanced accuracy ranged from 0.881228 to 0.894419 and mean recourse gain ranged from

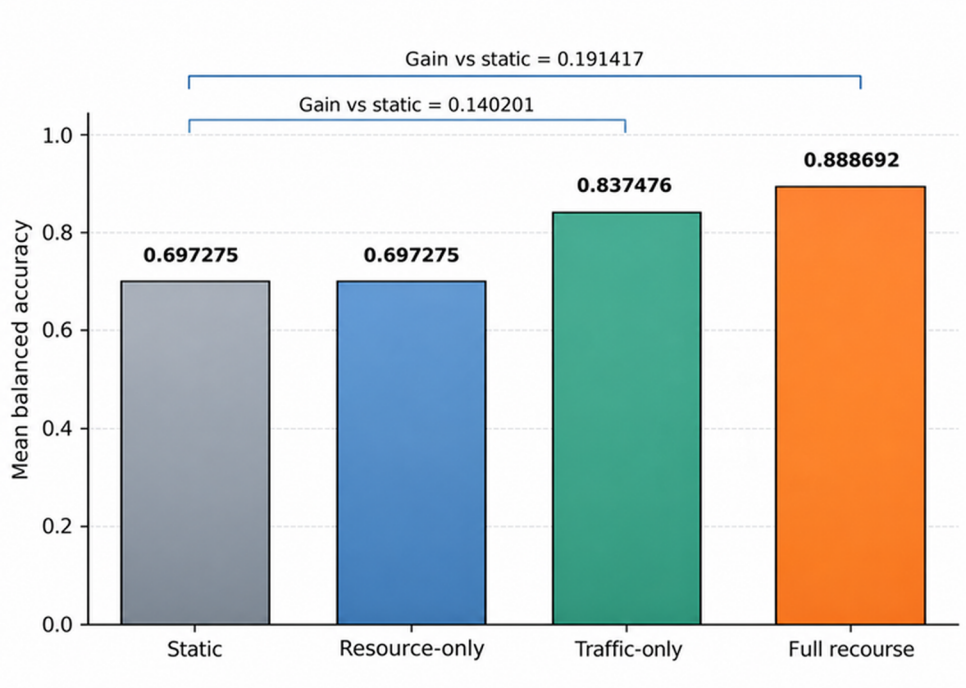


Figure 7. Mean external balanced accuracy for the static, resource-only, traffic-only, and full recourse variants

Table 11. External policy sensitivity under regime and latency perturbations

Sensitivity axis	Setting	Mean full BA	Mean gain	Positive cases	Portfolio agreement
Regime count	k = 8, seed = 2,026	0.893119	0.195844	12/15	0.800000
Regime count	k = 10, seed = 2,026	0.888692	0.191417	12/15	1.000000
Regime seed	k = 10, seed = 2,027	0.880636	0.183361	12/15	0.800000
Regime seed	k = 10, seed = 2,028	0.888160	0.190885	12/15	0.800000
Regime count	k = 12, seed = 2,026	0.889550	0.192275	12/15	0.600000
Latency threshold	0.95×	0.881228	0.183953	12/15	0.733333
Latency threshold	1.00×	0.888692	0.191417	12/15	1.000000
Latency threshold	1.05×	0.894419	0.197144	12/15	0.933333

0.183953 to 0.197144, again with 12 positive-gain cases in every setting. The policy-level outcome therefore remained stable over the examined regime and latency perturbations even when the exact selected portfolio changed in some cases.

5.9. Storage-budget and latency-service-level agreement analysis

The storage budgets controlled which model combinations could be deployed before the traffic and resource states were observed. The external

logistic regression model occupied 0.0017 MB. The compact boosting models occupied between 0.6314 and 0.8935 MB, while random forest and ExtraTrees occupied 61.7423 and 111.2298 MB, respectively. The 0.35 MB storage condition therefore admits logistic regression as the feasible singleton, defining the single-model end of the deployment grid. Larger budgets introduce additional portfolio compositions and enable state-dependent model choice.

At the 1 MB budget, compact boosting models became available. The selected portfolios then changed with the latency-SLA quantile. The 0.50 quantile selected logistic regression with CatBoost, while the 0.75 quantile selected logistic regression with HistGradientBoosting. At budgets of 128 MB and above, the policy could also retain random forest. The selected solutions at 128, 200, and 320 MB were often identical, which indicates that additional storage did not change the preferred portfolio once the useful candidate set was already feasible.

The SLA quantile also changed the recourse gain. The 0.25 quantile imposed the tightest latency condition and produced lower external balanced accuracy than the less restrictive 0.50 and 0.75 settings in the selected multi-model cases. The highest reported gain, 0.264666, occurred at the 0.75 quantile with a 1 MB storage budget. These results show that the first-stage storage decision and the second-stage latency feasibility condition interact; neither constraint can be interpreted separately from the available model set. Whole-pipeline profiling covered seven unique frozen portfolios. Serialized portfolio size ranged from 0.0017 to 62.4788 MB, resident RSS increment from 77.64 to 189.52 MB, peak RSS increment from 77.64 to 200.91 MB, and portfolio loading time from 1.32 to 1.69 s. With four cores and batch size 64, end-to-end latency ranged from 114.8 to 359.3 µs/row (2,783–8,711 rows/s), regime assignment required 21.4–54.5 µs/row, and selector dispatch required 0.148–0.412 µs/row. A 500,000-decision dispatch benchmark required 0.107-0.305 µs per decision under mappings with as many as 279,616 target changes. In a bounded single-feature sensitivity probe on 2,000 rows, standardized perturbations of 0.05 and 0.10 changed the regime assignment for 0.05% of rows and produced no model-selection changes in the tested policies; this probe characterizes local state sensitivity rather than an adversarial-robustness guarantee.

5.10. Discussion

The external replication targets transfer of the decision procedure rather than zero-shot transfer of fitted model weights. This choice is consistent with cross-dataset network IDS evidence showing that models with high within-dataset performance can change materially when applied to traffic collected from another environment.[37]

Accordingly, the independent replication retrained the same model families and re-estimated their resource behavior before evaluating the two-stage decision structure on CICIoMT2024. Recent IoT work has similarly emphasized that dataset-specific feature distributions and distributional drift can materially affect cross-dataset IDS behavior.[38]

The use of separate fitting, out-of-fold calibration, overlap control, and a final test split was also required because preprocessing leakage can inflate IDS performance when information from evaluation data is reused during model construction.[39]

The resource results are consistent with hardware-aware IDS studies that evaluate classification quality together with latency, memory, and deployment footprint rather than using predictive performance alone.[40] Vehicular IDS work has also coupled intrusion detection with energy-aware privacy preservation and edge-cloud resource selection.[41,42]

Tiny machine learning evaluations have also shown that the preferred model family can change when flash, RAM, latency, and predictive measures are considered together on constrained devices.[43]

Reviews of collaborative and edge-oriented IDS deployment identify limited processing and memory as persistent constraints when detection is moved toward IoT devices and gateways.[44]

The external results also support separating model portability from decision-procedure replication. Recent cross-dataset IoT work has argued that feature and label interoperability must be handled explicitly before results from different datasets are treated as directly comparable.[45]

Within this setting, the ablation identifies the contribution of each observable-state component. Traffic-only recourse accounts for a substantial part of the improvement, while the resource

state acts as a feasibility modifier. Full recourse increases mean balanced accuracy above traffic-only recourse because the resource state removes models that do not satisfy the current latency condition and activates an alternative feasible model when the deployed portfolio provides one. Traffic state therefore supplies the main predictive differentiation, while resource state enforces deployment-feasible adaptation.

The static diagnostic and sensitivity results further characterize the policy behavior. The constrained test-oracle static singleton improved the mean balanced accuracy relative to the frozen calibration-selected singleton, while full recourse retained the highest mean across the three reference levels. Across the tested regime counts, regime seeds, and $\pm$ 5% latency-threshold perturbations, the mean full-policy balanced accuracy remained within a narrow range, and the number of positive-gain deployment cases remained 12 of 15. Portfolio agreement varied more than aggregate performance, which indicates that multiple candidate portfolios can produce similar state-dependent outcomes under the evaluated constraints.

The internal and external analyses use different calibrated loss contributions within the same two-stage decision structure. The CICIoT2023 development analysis uses regime-probability-weighted $1 - F1$, whereas the external confirmatory analysis uses balanced-error contributions. The external formulation gives equal class weight and therefore exposes differences in benign-class behavior that are less visible in attack-class F1. Accordingly, the two analyses quantify complementary objectives and are reported on their respective evaluation scales.

5.11. Limitations and threats to validity

The resource measurements were obtained on one Windows 10 laptop and were used as controlled deployment states. The measured latency values are therefore hardware- and software-dependent. The relative feasibility relationships can change on embedded CPUs, edge accelerators, or different runtime libraries. Whole-pipeline profiling on the same host measured process RSS and peak RSS, but device energy, thermal effects, and network communication overhead were not measured. On advanced reduced instruction set computer machines-based systems, Raspberry Pi-class devices, microcontrollers, or embedded artificial intelligence accelerators, cache size, memory bandwidth, process scheduling, and library parallelism may change both absolute latency and relative model feasibility. Target deployments therefore require a new resource benchmark to populate the latency and memory inputs used by the optimizer.

The observable traffic states are operational clusters produced by MiniBatchKMeans on the 39 flow features rather than semantic attack-family or protocol labels. The sensitivity analysis across neighboring regime counts and multiple seeds showed stable aggregate policy performance despite changes in portfolio composition. Deploying to traffic distributions outside the calibrated regime structure would require re-estimating the regime mapping before policy freezing.

The external replication uses the 39 features shared by CICIoT2023 and the processed CICIoMT2024 WiFi/MQTT data with a binary Benign-versus-Attack target. The empirical decision space comprises seven heterogeneous tabular classifiers and $K_{max} = 3$, matching the exact portfolio formulation evaluated in this study. For multiclass attribution, the first-stage variables and storage/latency feasibility rules remain unchanged, while q_{mr} can be calibrated from a class-balanced or macro-averaged multiclass loss for each model and observable regime. The selected model then returns one of the benign/attack classes, which requires class-wise calibration and evaluation but not a different two-stage information structure. Additional model families or new feature schemas would broaden the deployment frontier evaluated by the optimizer.

The storage budgets, SLA quantiles, and uniform resource-state weights define a controlled deployment design rather than a fitted production-state distribution. This design isolates the interaction between portfolio availability and runtime feasibility across a reproducible grid. Deployment-specific use would instantiate the same formulation with latency and storage measurements from the target hardware and empirical operating-state frequencies.

6. Conclusion

This study formulated edge intrusion-detection deployment as a two-stage model-portfolio problem. The first-stage decision selects a constrained set of candidate models before the operating state is observed, and the second-stage decision selects a feasible deployed model after observable traffic and resource states

become available. The runtime rule does not use the ground-truth attack family. The final formulation therefore separates pre-deployment model availability from state-dependent runtime selection.

The CICIoT2023 development analysis showed a positive recourse gain in 13 of 15 deployment cases. The mean development gain was 0.001512, and the maximum was 0.004597. Increasing the portfolio limit from one model to two models produced a mean development gain of 0.001499, whereas adding a third model yielded a mean gain of 0.000017. This diminishing-return pattern supports the compact $K_{max} = 3$ portfolio design used in the final search.

The independent CICIoMT2024 replication used external retraining, resource re-benchmarking, three-fold out-of-fold regime-complete calibration, and a frozen final test policy. Full traffic-plus-resource recourse produced a mean balanced accuracy of 0.888692 compared with 0.697275 for the frozen calibration-selected static policy. A constrained test-oracle static policy obtained a mean of 0.840868, leaving full recourse with a mean advantage of 0.047825 over the best feasible static singleton identified after test outcomes were known. The ablation showed that traffic-only recourse reached 0.837476, while resource-only recourse remained at 0.697275. Sensitivity analyses over regime count, regime seed, and $\pm$ 5% latency-threshold perturbations retained positive recourse gain in 12 of 15 deployment cases across every evaluated setting.

The present evaluation covers one development dataset, one independent external replication dataset, seven tabular model families, a binary Benign-versus-Attack target, and controlled latency measurements on one hardware platform. Future work can instantiate the same decision structure on embedded edge hardware, additional feature schemas, multiclass intrusion-detection tasks, and online streams in which new observable regimes appear after deployment. The results support treating IDS deployment as a constrained portfolio decision when runtime model suitability changes with observable operating conditions.

Acknowledgments

None.

Funding

None.

Conflict of interest

The authors declare no conflicts of interest.

Author contributions

Conceptualization: Ameen Shaheen
Data curation: Ameen Shaheen, Ala'a Al-Shaikh
Formal analysis: Ameen Shaheen
Investigation: All authors
Methodology: Ameen Shaheen, Ala'a Al-Shaikh
Software: Ameen Shaheen
Validation: All authors
Writing – original draft: Ameen Shaheen
Writing – review & editing: All authors

Availability of data

The CICIoT2023 and CICIoMT2024 datasets used in this study are publicly available from their original sources. The source code, preprocessing and evaluation scripts, model configurations, resource-benchmarking procedures, optimization scripts, supplementary result tables, and computational artifacts required to reproduce the reported analyses are available in the RC-PORT-IDS reproducibility package on Zenodo at https://doi.org/10.5281/zenodo.23047315.

AI tools statement

The authors used Grammarly and ChatGPT to improve the manuscript's grammar, clarity, and readability. These tools were used for language editing only and did not generate scientific content, results, interpretations, or conclusions. All authors confirm that AI tools were used only for language editing and that all final decisions and responsibility for the manuscript content remain with the authors.

References

1. Ahmad Z, Khan AS, Wai Shiang C, Abdullah J, Ahmad F. Network intrusion detection system: a systematic study of machine learning and deep learning approaches. *Trans Emerg Telecommun Technol.* 2021;32(1).

   https://doi.org/10.1002/ett.4150

2. Gyamfi E, Jurcut A. Intrusion detection in Internet of Things systems: a review on design approaches leveraging multi-access edge computing, machine learning, and datasets. *Sensors (Basel).* 2022;22(10):3744.

   https://doi.org/10.3390/s22103744

3. Alhomoud A. An optimized network intrusion detection system for attack detection based on supervised machine learning models in an Internet-of-Things environment. *Int J Adv Soft Comput Appl*. 2023;15(2):1-15.

   https://doi.org/10.15849/IJASCA.230720.01

4. Neto ECP, Dadkhah S, Ferreira R, Zohourian A, Lu R, Ghorbani AA. CICIoT2023: a real-time dataset and benchmark for large-scale attacks in IoT environment. *Sensors (Basel)*. 2023;23(13):5941.

   https://doi.org/10.3390/s23135941

5. Farfoura ME, Mashal I, Alkhatib A, Batyha RM, Rosiyadi D. A novel lightweight machine learning framework for IoT malware classification based on matrix block mean downsampling. *Ain Shams Eng J*. 2025;16(1):103205.

   https://doi.org/10.1016/j.asej.2024.103205

6. Tiwari RS, Lakshmi D, Das TK, Tripathy AK, Li KC. A lightweight optimized intrusion detection system using machine learning for edge-based IIoT security. *Telecommun Syst*. 2024;87(3):605-624.

   https://doi.org/10.1007/s11235-024-01200-y

7. Al Rawajbeh M, Maria Soosai AJ, Ramasamy LK, Khan F. Trustworthy adaptive AI for real-time intrusion detection in industrial IoT security. *IoT*. 2025;6(3):53.

   https://doi.org/10.3390/iot6030053

8. Shenify MA, Alghamdi AS, Alharthi AF. Hybrid supervised machine learning-based intrusion detection system of Internet of Things. *Int J Adv Soft Comput Appl*. 2024;16(2):68-84.

   https://doi.org/10.15849/IJASCA.240730.05

9. Shaheen A, Alzyadat W, Alhroob A. LARO-IDS: family-leakage-aware robust multi-objective optimization for model selection in IoT intrusion detection. *Int J Optim Control Theor Appl*. 2026;16(5):026270147.

   https://doi.org/10.36922/IJOCTA026270147

10. Birge JR, Louveaux F. *Introduction to Stochastic Programming*. 2nd ed. New York, NY: Springer; 2011.

    https://doi.org/10.1007/978-1-4614-0237-4

11. Birge JR. The value of the stochastic solution in stochastic linear programs with fixed recourse. *Math Program*. 1982;24(1):314-325.

    https://doi.org/10.1007/BF01585113

12. Hani A, El-Emam NN. Enhanced intrusion detection systems dataset synthesis using conditional generative adversarial networks with the adaptive whale optimization algorithm. *Int J Adv Soft Comput Appl*. 2026;18(1):263-285.

    https://doi.org/10.15849/ijasca.v18i1.71

13. Dadkhah S, Neto ECP, Ferreira R, Molokwu RC, Sadeghi S, Ghorbani AA. CICIoMT2024: a benchmark dataset for multi-protocol security assessment in IoMT. *Internet Things*. 2024;28:101351.

    https://doi.org/10.1016/j.iot.2024.101351

14. Ferrag MA, Friha O, Hamouda D, Maglaras L, Janicke H. Edge-IIoTset: a new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. *IEEE Access*. 2022;10:40281-40306.

    https://doi.org/10.1109/ACCESS.2022.3165809

15. Alsaedi A, Moustafa N, Tari Z, Mahmood AN, Anwar A. TON_IoT telemetry dataset: a new generation dataset of IoT and IIoT for data-driven intrusion detection systems. *IEEE Access*. 2020;8:165130-165150.

    https://doi.org/10.1109/ACCESS.2020.3022862

16. Meidan Y, Bohadana M, Mathov Y, Mirsky Y, Shabtai A, Breitenbacher D, Elovici Y. N-BaIoT: network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Comput*. 2018;17(3):12-22.

    https://doi.org/10.1109/MPRV.2018.03367731

17. Shao Z, Yuan S, Wang Y. Adaptive online learning for IoT botnet detection. *Inf Sci*. 2021;574:84-95.

    https://doi.org/10.1016/j.ins.2021.05.076

18. Alalhareth M, Hong SC. Enhancing the Internet of Medical Things (IoMT) security with meta-learning: a performance-driven approach for ensemble intrusion detection systems. *Sensors (Basel)*. 2024;24(11):3519.

    https://doi.org/10.3390/s24113519

19. Lin YD, Huang HX, Sudyana D, Lai YC. AI for AI-based intrusion detection as a service: reinforcement learning to configure models, tasks, and capacities. *J Netw Comput Appl*. 2024;229:103936.

    https://doi.org/10.1016/j.jnca.2024.103936

20. Hasan MM, Islam R, Mamun Q, Islam MZ, Gao J. Ensemble-based adversarial defense with dynamic model selection for intrusion detection systems. *Internet Things*. 2026;38:101970.

    https://doi.org/10.1016/j.iot.2026.101970

21. Forough J, Bhuyan M, Elmroth E. Reinforced model selection for resource efficient anomaly detection in edge clouds. *Future Gener Comput Syst*. 2026;176:108161.

    https://doi.org/10.1016/j.future.2025.108161

22. Kadve G, Chowdhury A, Singh VK, Pal A. Engineering a multi model fallback system for

edge devices. *Results Eng.* 2025;26:105165.

https://doi.org/10.1016/j.rineng.2025.105165

23. Chen X, Wang P, Yang Y, Liu M. Resource-constraint deep forest-based intrusion detection method in Internet of Things for consumer electronic. *IEEE Trans Consum Electron.* 2024;70(2):4976-4987.

https://doi.org/10.1109/TCE.2024.3373126

24. Alwaisi Z, Kumar T, Harjula E, Soderi S. Securing constrained IoT systems: a lightweight machine learning approach for anomaly detection and prevention. *Internet Things.* 2024;28:101398.

https://doi.org/10.1016/j.iot.2024.101398

25. Fatima M, Rehman O, Ali S, Niazi MF. ELIDS: ensemble feature selection for lightweight IDS against DDoS attacks in resource-constrained IoT environment. *Future Gener Comput Syst.* 2024;159:172-187.

https://doi.org/10.1016/j.future.2024.05.013

26. Almalawi A. A lightweight intrusion detection system for Internet of Things: clustering and Monte Carlo cross-entropy approach. *Sensors (Basel).* 2025;25(7):2235.

https://doi.org/10.3390/s25072235

27. Dash S, Acken JM. Intrusion detection latency: the neglected metric. *Cybersecurity.* 2026;9(1):144.

https://doi.org/10.1186/s42400-026-00574-7

28. King AJ, Wallace SW. *Modeling with Stochastic Programming.* 2nd ed. Cham, Switzerland: Springer Cham; 2024.

https://doi.org/10.1007/978-3-031-54550-4

29. Shapiro A, Dentcheva D, Ruszczyński A. *Lectures on Stochastic Programming: Modeling and Theory.* 2nd ed. Philadelphia, PA: SIAM; 2014.

https://doi.org/10.1137/1.9781611973433

30. Cao B, Wu T, Bai X. Stochastic programming based multi-arm bandit offloading strategy for Internet of Things. *Digit Commun Netw.* 2023;9(5):1200-1211.

https://doi.org/10.1016/j.dcan.2022.03.018

31. Zhang L, Cao B, Li Y, Peng M, Feng G. A multi-stage stochastic programming-based offloading policy for fog enabled IoT-eHealth. *IEEE J Sel Areas Commun.* 2021;39(2):411-425.

https://doi.org/10.1109/JSAC.2020.3020659

32. Sculley D. Web-scale k-means clustering. In: *Proceedings of the 19th International Conference on World Wide Web.* New York, NY: ACM; 2010:1177-1178.

https://doi.org/10.1145/1772690.1772862

33. Engel VJL, Angela D, Latif A, Kristyan SA. CICIoMT2024 cleaned dataset and EDA outputs. Zenodo. 2026.

https://doi.org/10.5281/zenodo.19452870

34. Fawcett T. An introduction to ROC analysis. *Pattern Recognit Lett.* 2006;27(8):861-874.

https://doi.org/10.1016/j.patrec.2005.10.010

35. Brodersen KH, Ong CS, Stephan KE, Buhmann JM. The balanced accuracy and its posterior distribution. In: *2010 20th International Conference on Pattern Recognition.* Piscataway, NJ: IEEE; 2010:3121-3124.

https://doi.org/10.1109/ICPR.2010.764

36. Matthews BW. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochim Biophys Acta.* 1975;405(2):442-451.

https://doi.org/10.1016/0005-2795(75)90109-9

37. Cantone M, Marrocco C, Bria A. Machine learning in network intrusion detection: a cross-dataset generalization study. *IEEE Access.* 2024;12:144489-144508.

https://doi.org/10.1109/ACCESS.2024.3472907

38. Eren KK, Küçük K, Saleh RAA, Konyar MZ, Hardy OM, Khan SA. Distributional drift in IoT intrusion detection systems: implications for cross-dataset generalisation. *Electronics (Basel).* 2026;15(11):2307.

https://doi.org/10.3390/electronics15112307

39. Bouke MA, Abdullah A. An empirical study of pattern leakage impact during data preprocessing on machine learning-based intrusion detection models reliability. *Expert Syst Appl.* 2023;230:120715.

https://doi.org/10.1016/j.eswa.2023.120715

40. Diab A, Chehade A, Ragusa E, Gastaldo P, Zunino R, Baghdadi A, Rizk M. Intrusion detection on resource-constrained IoT devices with hardware-aware ML and DL. In: *2025 IEEE International Conference on Emerging Trends in Engineering and Computing (ETECOM).* Piscataway, NJ: IEEE; 2025:1-6.

https://doi.org/10.1109/ETECOM66111.2025.11319100

41. Heidari A, Khonsari A, Rastegar SH. An energy-efficient privacy-preserving framework for intrusion detection in the Internet of Vehicles. *Comput Electr Eng.* 2026;132:111003.

https://doi.org/10.1016/j.compeleceng.2026.111003

42. Heidari A, Rastegar SH, Khonsari A. Explainable edge-cloud federated intrusion detection system for electric vehicles using GRUs, DRL selection, and integrated gradients. *J Big Data.* 2026.

https://doi.org/10.1186/s40537-026-01505-6

43. Hamdouchi A, Idri A. Evaluating the performance of TinyML singular and ensemble techniques for intrusion detection in IoT networks. *Microprocess Microsyst*. 2025;117:105172.

    https://doi.org/10.1016/j.micpro.2025.105172

44. Ieropoulos V, Anthi E, Spyridopoulos T, Burnap P, Mavromatis I, Khan A, Carnelli P. Collaborative intrusion detection in resource-constrained IoT environments: challenges, methods, and future directions: a review. *J Inf Secur Appl*. 2025;93:104127.

    https://doi.org/10.1016/j.jisa.2025.104127

45. Rodríguez M, Mehavilla Martín L, Alesanco Á, García J. A rule-based label harmonization framework for cross-dataset IoT intrusion detection. *Internet Things*. 2026;39:102030.

    https://doi.org/10.1016/j.iot.2026.102030

Ameen Shaheen *received his PhD degree in Computer Science from the University of Jordan in 2018. He is affiliated with the Department of Software Engineering, Faculty of Science and Information Technology, Al-Zaytoonah University of Jordan, Amman, Jordan. His research activities include machine learning, optimization algorithms, software engineering, artificial intelligence, and their applications in different computational and engineering problems. He has contributed to a number of research studies in these areas and has participated in collaborative research involving intelligent and data-driven systems.*
https://orcid.org/0009-0003-3120-1892

Ala'a Al-Shaikh *is an Assistant Professor in the Department of Cybersecurity at Al-Zaytoonah University of Jordan, Amman, Jordan. He holds a PhD in Computer Science from the University of Jordan. His research interests include cybersecurity, machine learning, cloud computing, data science, optimization, distributed and parallel computing, and related intelligent computing applications. He has published research in areas including intrusion detection, IoT security, cloud computing, artificial intelligence, and optimization, addressing both theoretical and applied computing problems.*
https://orcid.org/0000-0002-1888-759X

Wael Alzyadat *is an Associate Professor and Head of the Department of Software Engineering at Al-Zaytoonah University of Jordan, Amman, Jordan. His research interests include software engineering, intelligent systems, big data, streaming data, machine learning, and related data-driven applications. He has published research in several areas involving software analysis, artificial intelligence, data processing, and intelligent computing systems, and has participated in research addressing both theoretical and applied computing problems.*
https://orcid.org/0000-0002-0068-1526

Aysh Alhroob *is a Full Professor of Software Engineering and Dean of the Faculty of Science and Information Technology at Al-Zaytoonah University of Jordan, Amman, Jordan. He received his Ph.D. degree in Software Engineering from the University of Bradford, United Kingdom, in 2010. His research interests include software testing, software requirements, natural language processing, big data, data science, data analysis, computer algorithms, and artificial intelligence techniques. He has extensive academic, research, supervision, and administrative experience in software engineering and information technology.*
https://orcid.org/0000-0002-4653-7386

An International Journal of Optimization and Control: Theories & Applications (https://accscience.com/journal/ijocta)



This work is licensed under a Creative Commons Attribution 4.0 International License. The authors retain ownership of the copyright for their article, but they allow anyone to download, reuse, reprint, modify, distribute, and/or copy articles in IJOCTA, so long as the original authors and source are credited. To see the complete license contents, please visit http://creativecommons.org/licenses/by/4.0/.