Journal of Network and Systems Management (2025) 33:75
https://doi.org/10.1007/s10922-025-09940-1

A Dynamic Network Intrusion Detection Model for Infrastructure as Code Deployed Environments

Adilson G. Filho[1] · Eduardo K. Viegas[1] · Altair O. Santin[1] · Jhonatan Geremias[1]

Received: 13 January 2025 / Revised: 13 January 2025 / Accepted: 3 June 2025

Abstract
The dynamic nature of Infrastructure as Code (IaC) provisioned infrastructures presents substantial challenges for traditional Machine Learning (ML) Network Intrusion Detection Systems (NIDSs). In such settings, continuously evolving configurations lead to difficulties in maintaining detection accuracy, as ML models struggle to adapt to rapidly changing network behaviors and new attack patterns. This paper introduces a novel ML-based NIDS framework tailored to address the non-stationary behavior of IaC-provisioned infrastructures. The framework integrates two key components: multi-objective feature selection and dynamic classification. The multi-objective feature selection enhances the model's generalization capabilities during training, enabling it to better handle the evolving behaviors characteristic of IaC environments. The dynamic classification component complements this by actively selecting the most appropriate subset of classifiers at the inference phase, ensuring adaptability to the current infrastructure state. By incorporating these components, the proposed scheme achieves real-time adaptability to the dynamic nature of IaC-provisioned infrastructures, providing reliable intrusion detection. Experimental evaluations conducted on a realistic IaC-generated testbed with over 19 configurations demonstrate significant improvements in detection performance. Specifically, the proposed model can increase the F1-Score by up to 0.31 when compared to traditional approaches on newly provisioned IaC infrastructures.

Keywords Infrastructure as Code · Network Intrusion Detection · Machine Learning · Feature Selection

Extended author information available on the last page of the article

Published online: 25 June 2025

1 Introduction

Over the past few years, the provisioning of Information Technology (IT) infrastructures was predominantly performed manually, a costly and time-consuming task that often required continuous supervision by operators [1]. As the complexity of the IT infrastructures expanded, the demand for automated resource provisioning techniques became increasingly evident. In this context, Infrastructure as Code (IaC) paved the way to automate the infrastructure provision task in which the software will be deployed [2]. In practice, IaC enables the automatic configuration of system dependencies and the provisioning of local and remote instances to facilitate continuous deployment in accordance with service requirements [3]. It accomplishes this by utilizing provisioning scripts (*code*) that act as configuration templates for service provisioning tasks, enabling IT organizations to reduce their deployment time significantly. Due to the advantages brought by automating infrastructure provisioning, several IaC technologies have emerged in recent years, including Chef [4], Puppet [5], and Terraform [6] for private infrastructures, and AWS CloudFormation [7], and Azure Resource Manager [8] for public clouds.

Securing IaC-provisioned infrastructures presents a significant challenge due to the dynamic nature of infrastructure configurations [9]. These configurations change based on the provided IaC *script*, demanding that security solutions can effectively adapt to these variations [2]. Conversely, despite the automation of infrastructure provisioning brought by IaC, the configuration of security solutions still often requires manual intervention. This includes configuring firewall rules, access control policies, Virtual Local Area Network (VLAN), and Network Intrusion Detection System (NIDSs), among other measures. As a result, there is a gap between the automation of infrastructure provisioning tasks enabled by IaC techniques and the automation of security solution configurations required to secure the newly provisioned infrastructure [10]. This gap hinders the ability to address security concerns in rapidly changing IaC-provisioned environments effectively.

Unfortunately, automating security configurations based on IaC code presents significant challenges, especially for NIDS solutions [11]. Most of the literature focuses on designing *behavior-based* intrusion detection schemes by typically relying on constructing a behavioral Machine Learning (ML) model [12]. These models assume a static baseline environment configuration, making the adaptation to the ever-changing nature of IaC-provisioned infrastructures difficult [13]. In practice, the infrastructure configuration must be known beforehand to properly train and evaluate the ML scheme for reliable use. As a consequence, when the infrastructure configuration changes due to a new IaC *script* for instance, the previously designed ML-based NIDS becomes unreliable for production deployment. Typically, this issue can only be addressed through model updates, a process that often requires several days or even weeks to complete. This delay poses a considerable challenge to maintaining effective intrusion detection in highly dynamic environments [14].

Developing ML-based NIDSs for dynamic environments is particularly challenging, as ML models are typically optimized to maximize detection

accuracy within a static, predefined training environment [15]. This inherent design assumption limits their ability to adapt to the continuous changes characteristic of dynamic IaC-deployed infrastructures [12]. This limitation often results in intrusion detection schemes that inadequately generalize from the training dataset, leading to performance issues in real-world settings. Consequently, such schemes struggle to detect new and evolving behaviors effectively, a situation usually observed in IaC-provisioned infrastructures. In practice, intrusion detection models tend to achieve high detection accuracies in environments that closely resemble their training datasets. However, their performance often degrades significantly when exposed to behavioral changes or unseen variations in the operational environment. Conversely, developing a reliable ML-based NIDS for IaC-provisioned infrastructures requires that the designed scheme effectively accounts for behaviors that are not observed during the training phase [16].

The development of a generalizable ML-based NIDS has been the focus of numerous studies in recent years, yet its application within IaC environments remains largely overlooked. In such environments, building a training dataset that accurately reflects the behavior of the to-be-deployed infrastructure is not easily feasible due to the dynamic nature of IT configurations, which changes based on the IaC *script* and result in corresponding shifts in environment behavior [17]. Deploying a new service configuration alters network traffic behavior, impacting normal operations and potential attacker activities. In traditional settings, changes in network traffic behavior lead to unreliable ML-based NIDSs, necessitating model updates for effective remediation [18]. However, in IaC-deployed infrastructures, the environment's behavior is only revealed after the provisioning task is completed. Therefore, for ML-based NIDS to remain reliable in such dynamic settings, proposed approaches must adapt to new environment behaviors in real-time without relying on frequent model updates. This is particularly important given the challenges and delays associated with constructing updated training datasets for constantly evolving IaC environments.

Traditional ML-based NIDSs assume that the behavior of the deployed environment is stationary and does not change over time. In practice, the literature assumes that the network traffic behavior is known entirely during training and remains unchanged once the system is deployed in production. Otherwise, the system's error rates will increase compared to those observed during the testing phase, leading to an unreliable system [19]. Unfortunately, developing a ML-based NIDS that can adapt to potentially unseen network traffic behavior is not readily achievable, particularly in IaC environments. In contrast, researchers often overlook the generalization capabilities of their designed schemes in favor of achieving higher detection accuracies during testing [20]. This approach frequently results in overfitting, where models are tailored to a single environment's behavior and perform poorly when exposed to environmental changes. Consequently, despite the growing adoption of IaC-deployed infrastructures in recent years, a research gap persists in the development of ML-based NIDSs capable of addressing the shifts in environment behavior caused by the IaC configuration *scripts*.

Contribution. In light of this, this paper proposes a novel ML-based NIDS tailored for IaC-deployed infrastructures. The proposed model is implemented twofold,

simultaneously leveraging a multi-objective feature selection and dynamic classifier selection. First, the multi-objective feature selection identifies a subset of features that can simultaneously enhance the system's accuracy and generalization capabilities. Our key insight is to improve generalization during the model-building phase, addressing potential behavior changes introduced by the IaC configuration *scripts*. Second, the dynamic selection of classifiers actively chooses the most appropriate subset of classifiers based on the behavior of the currently deployed IaC environment. This approach ensures that infrastructure behavior changes caused by the IaC *script*, which could otherwise degrade accuracy, are mitigated by dynamically adjusting the classifiers used for the classification at the inference phase. As a result, our proposed model effectively tackles the challenges of generalization and novelty detection posed by the non-stationary behavior introduced by IaC *scripts*.

In summary, the main contributions of our work are:

- A new ML-based NIDS for IaC-deployed infrastructures. The proposed scheme addresses model generalization at the training phase and actively selects the classifiers during the inference phase for novelty detection. Our proposal improves the F1-Score by up to 0.31;
- A new publicly available IaC-provisioned intrusion dataset with 19 different configurations, generated with 100 normal clients for 5 services, as well as 14 different attacker behaviors;

Roadmap. The remainder of this paper is organized as follows. Section 2 discusses the fundamentals of ML-based NIDS and IaC. Section 3 overviews the current literature on ML-based NIDS for IaC. Section 4 introduces our proposed scheme, Sect. 5 describes its implementation, and 6 evaluates its performance. Finally, we conclude our work on Sect. 7.

2 Preliminaries

The utilization of IaC-deployed infrastructures has consistently increased over the past few years. This section further describes the fundamentals of IaC-based provisioning techniques, followed by a discussion on the development of ML-based NIDSs for IaC-deployed infrastructures.

2.1 Infrastructure as Code (IaC)

IaC is an approach to manage and provision IT infrastructures through machine-readable definition files. In practice, rather than relying on physical hardware configurations or manual configuration tools, this approach allows developers and operations teams to automate the deployment and management of infrastructure resources through code [2]. This automation ensures consistency, scalability, and rapid iteration by treating infrastructure similarly to software development, as IaC enables version control, testing, and team collaboration. These capabilities reduce

the likelihood of configuration drift and promote more efficient resource utilization. Additionally, IaC enables organizations to implement Continuous Integration and Continuous Deployment CI/CD practices, enhancing agility and responsiveness to evolving business needs [2].

The advantages introduced by IaC have led numerous companies to develop new solutions for automating infrastructure deployment. These include private platforms such as AWS CloudFormation [7] and Azure Resource Manager [8], alongside publicly available alternatives like Chef [4], Puppet [5], and Terraform [6]. The latter is an open-source tool that enables users to define and provision infrastructure resources using a high-level configuration language called Hashicorp Configuration Language (HCL). Terraform integrates the management of both cloud services and on-premises resources through declarative configuration files that specify the desired infrastructure state [21]. These capabilities extend to automating the deployment of resources such as virtual machines, networks, and security groups. During the deployment phase, Terraform supports integration with various public and private cloud platforms via a provider-based architecture, enabling the seamless provisioning of diverse infrastructure components. It utilizes a resource graph that visualizes resource dependencies, ensuring parallel execution during provisioning. Its state management feature also tracks the current infrastructure state, facilitating automated updates and modifications while maintaining consistency and preventing configuration drift.

Deploying an IT infrastructure through IaC facilitates the seamless modification of configurations, allowing for straightforward adjustments that can significantly alter the behavior of services. This flexibility enables teams to quickly implement changes in response to evolving requirements or performance metrics, streamlining the process of adapting infrastructure to meet specific needs [22]. However, while this ease of configuration change enhances agility, it also necessitates careful consideration of potential impacts, as even minor adjustments can lead to major changes in service behavior.

2.2 Machine Learning for Network Intrusion Detection

The utilization of ML techniques for NIDS has steadily increased over the past few years due to the capability of ML to detect new attack behaviors [12]. To achieve this goal, proposed schemes are typically implemented through a four-phase process. First, the *Data Acquisition* module continuously captures network packets from a monitored Network Interface Card (NIC). The collected packets serve as input for the *Feature Extraction* module, which aims to extract behavioral features representing the network traffic behavior between hosts and their services. In general, network traffic behavior is represented through flow-based features, which include metrics such as the number of exchanged network packets and bytes between a specific service over a defined time frame (e.g., over a 60 seconds interval). Table 1 shows the set of network-level features used in our work. In total, we consider 31 features that summarize the communication between client and server in a window interval of up to 60 seconds. The resulting feature vector is then classified by a *Classification*

Table 1 Feature set extracted at the network level in a time window interval of 60s for every client and server communication

#	Feature	Description
1	Total Fwd Pkts	Total number of packets sent from client to server
2	Total Fwd Vol	Total volume of data (in bytes) sent from client to server
3	Total Bwd Pkts	Total number of packets sent from server to client
4	Total Bwd Vol	Total volume of data (in bytes) sent from server to client
5	Fwd Pkt Len Std	Standard deviation of the length of packets sent from client to server
6	Bwd Pkt Len Max	Maximum length of packets sent from server to client
7	Bwd Pkt Len Std	Standard deviation of the length of packets sent from server to client
8	Fwd IAT Mean	Mean inter-arrival time of packets sent from client to server (in milliseconds)
9	Fwd IAT Max	Maximum inter-arrival time of packets sent from client to server (in milliseconds)
10	Fwd IAT Std	Standard deviation of inter-arrival time of packets sent from client to server (in milliseconds)
11	Bwd IAT Max	Maximum inter-arrival time of packets sent from server to client (in milliseconds)
12	Bwd IAT Std	Standard deviation of inter-arrival time of packets sent from server to client (in milliseconds)
13	Duration	Total duration of the network flow (in milliseconds)
14	Active Min	Minimum time the flow was active (in milliseconds)
15	Active Mean	Mean time the flow was active (in milliseconds)
16	Active Max	Maximum time the flow was active (in milliseconds)
17	Active Std	Standard deviation of the time the flow was active (in milliseconds)
18	Idle Min	Minimum time the flow was idle (in milliseconds)
19	Idle Mean	Mean time the flow was idle (in milliseconds)
20	Idle Max	Maximum time the flow was idle (in milliseconds)
21	Idle Std	Standard deviation of the time the flow was idle (in milliseconds)
22	SFlow Fwd Pkts	Sampled number of packets sent from client to server
23	SFlow Fwd Bytes	Sampled number of bytes sent from client to server
24	SFlow Bwd Pkts	Sampled number of packets sent from server to client
25	SFlow Bwd Bytes	Sampled number of bytes sent from server to client
26	FPSH Count	Number of PUSH flags in packets sent from client to server
27	BPSH Count	Number of PUSH flags in packets sent from server to client
28	FURG Count	Number of URGENT flags in packets sent from client to server
29	BURG Count	Number of URGENT flags in packets sent from server to client
30	Total FHLen	Total header length of packets sent from client to server
31	Total BHLen	Total header length of packets sent from server to client

module, which performs this task using a previously trained ML model. The *Alert* module then signals events classified as intrusion to a network operator for decision-making.

The reliability of a ML-based NIDS depends on an adequately trained ML model [14]. To achieve this, researchers usually employ a three-phase process,

namely *training*, *validation*, and *testing*. The *training* phase involves extracting a behavioral ML model from a training dataset. Consequently, the dataset must reliably represent the behavior of the environment in which the system will be deployed. The extracted ML model is then assessed using a *validation* dataset, which facilitates feature selection and model fine-tuning. Finally, the accuracy of the optimized ML model is evaluated on a *testing* dataset, with the expectation that this accuracy will be reflected when the system is deployed in a production environment.

As a result, changes in environment behavior necessitate re-executing the entire model training process [23]. This process involves not only retraining the model but also regenerating an updated training dataset that accurately captures the new environment behavior. Such dataset generation is a time-consuming task, often requiring several days or even weeks to gather, preprocess, and label sufficient data to reflect the updated conditions accurately. Additionally, the continuous need for retraining introduces operational delays and increases computational costs, making it impractical for environments with frequent configuration changes [24]. This highlights the significant challenges associated with maintaining reliable performance in dynamic IaC-deployed infrastructures.

2.3 When ML-based NIDS meets IaC

Applying ML-based NIDS in IaC environments presents several challenges, primarily stemming from the dynamic nature of service configurations. As infrastructure configuration is frequently modified through automated deployment scripts, any changes in service configuration can significantly impact network behavior, leading to alterations in provided services and shifts in the threat landscape [2]. These modifications necessitate continuous model retraining to ensure that the ML-based NIDS can accurately detect anomalies and threats based on the current operational environment. However, retraining models in a timely and efficient manner can be resource-intensive and complex, requiring access to updated training data and adequate validation processes [25]. Additionally, integrating these retrained models into the IaC pipeline must be meticulously managed to prevent disruptions in service availability and to maintain the effectiveness of security measures amid ongoing configuration changes.

In this context, there remains a significant gap in developing new ML-based NIDSs that can effectively navigate the dynamic nature of IaC environments. While researchers often assume that challenges arising from frequent configuration changes can be adequately addressed through periodic model retraining, this approach is frequently not feasible. This is due to the resource-intensive nature of retraining processes and the necessity for extensive, up-to-date training data. Consequently, this reliance on retraining overlooks the challenges associated with real-time operational adjustments and the rapid evolution of network behaviors in IaC, making traditional approaches inapplicable due to the time frame and effort required to build an updated ML model.

3 Related Works

The development of new ML-based techniques for NIDS has been a widely explored topic in the literature over the past few years [12]. In general, proposed schemes often pursue higher accuracies, albeit the tradeoffs on model generalization, a characteristic that challenges the application on IaC-provisioned architectures. As an example, S. Tariq et al. [26] proposed an intrusion detection scheme implemented through a long Short-Term Memory (LSTM) model. The proposed approach improves accuracy by training their scheme using a transfer learning implementation. Unfortunately, the authors assume that model updates can be conducted as required and overlook model generalization challenges. Similarly, K. Wolsing et al. [27] relies on an ensemble of classifiers to improve detection accuracy. Their scheme builds an ensemble through a time-aware and transfer learning framework to reduce false positives. The impact of model generalization and changes in environmental behavior is overlooked. B. Mbarek et al. [28] proposes replicating intrusion detection models to detect network attacks. Their scheme improves accuracy but also leaves model generalization unaddressed. R. Lazzarini et al. [29] proposes using an ensemble stacking approach through deep learning classifiers to reduce false-positive rates. Their scheme significantly increases accuracy on a single dataset while overlooking the generalization challenges. R. Zhao et al. [30] proposes a dynamic autoencoder model to address the non-stationary behavior of intrusions. Their approach improves detection accuracy but assumes that model updates can be conducted as required.

Feature selection is a widely used approach in intrusion detection for accuracy improvements. M. Rashid et al. [31] relies on a tree-based classifier built using feature selection to improve intrusion detection accuracy. Their approach reduces false positives but overlooks the generalization impact on the resulting model. Z. Ye et al. [32] proposes an evolutionary-based feature selection approach for combining multiple classifiers. Their approach improves accuracy on widely used datasets; however, the authors assume a static environment behavior during the evaluation. Similarly, S. Das et al. [33] evaluates feature selection impact on multiple intrusion datasets while overlooking the generalization capabilities of the resulting model. Z. Halim et al. [34] utilizes a genetic search algorithm for feature selection aiming to reduce the model's error rate. Their approach improves detection accuracy in several intrusion datasets but overlooks the impact on model generalization. Z. Chkirbene et al. [35] relies on a dynamic intrusion detection scheme coped with a feature selection approach. The proposal reduces the error rate but overlooks model generalization and assumes a stationary environment behavior. Y. Zhou et al. [36] proposes a feature selection approach for an ensemble of classifiers to reduce intrusion detection error rates. The authors evaluate the efficacy of their scheme on several datasets but neglect the impact on model generalization. C. Khammassi et al. [37] uses a multi-objective feature selection scheme aiming for higher accuracy and fewer features. Their model improves intrusion detection with less computational costs, unfortunately, model generalization is neglected.

In general, to address the non-stationary behavior of intrusion detection environments, such as those originating from IaC-deployed infrastructures, researchers rely on dataset generation approaches. M. Laundauer et al. [38] aims at simulating multiple normal user behavior to address dataset generation challenges. Their approach could be applied to model updates in dynamic environments. However, the application in such a context is not evaluated. V. Kumar et al. [39] utilizes a generative adversarial network for the generation of synthetic attack samples. Their approach improves accuracy in a single dataset but fails at evaluating their scheme under non-stationary behavior. L. Syne et al. [40] address model updates in intrusion detection through a federated learning scheme. Their model enables model updates to be conducted by multiple parties but neglects the dataset generation challenges associated with model updates. Similarly, S. S. Woo et al. [41] proposes a reinforcement learning strategy to update intrusion detection models over time. Their approach enables model updates to be conducted more easily but overlooks the generalization of the resulting intrusion detection scheme.

As a consequence, there is still a substantial research gap in the development of ML-based NIDSs that can function effectively in IaC-deployed environments. This gap exists primarily because current schemes struggle to handle the non-stationary behavior of such dynamic settings. As infrastructure configurations change rapidly and frequently in IaC, the underlying network environment becomes unpredictable, causing traditional NIDS models to fail in adapting to these shifts. To address this issue, proposed schemes must generalize the network behavior to adapt to evolving conditions and detect and respond to unseen or novel network traffic patterns without solely depending on frequent retraining.

3.1 Discussion

Table 2 overviews the current literature concerning the implementation of ML-based NIDSs. The existing literature on ML-based NIDSs demonstrates limited attention to several characteristics essential for securing IaC-provisioned infrastructures. Feature selection, while explored in several works, often focuses solely on improving detection accuracy without considering generalization to unseen network behaviors. Similarly, addressing new network behaviors remains underexplored, with most approaches relying on static training datasets that fail to capture the dynamic nature of real-world environments. Moreover, the application of these schemes to IaC environments is rarely discussed, leaving a significant gap in adapting to the infrastructure's variability driven by configuration scripts. Few studies employ realistic datasets that reflect practical deployment scenarios, further limiting the applicability of these solutions in production environments. Finally, dynamic inference, crucial for adapting to runtime changes in network behavior, is largely absent from existing approaches. In contrast, our proposed framework addresses these shortcomings by incorporating multi-objective feature selection to enhance generalization, applying dynamic classification to adapt to new network behaviors, leveraging realistic

Table 2 A summary of related work and the characteristics of their ML-based NIDS implementation

Work	Feature Selection	New Net. Behavior	IaC Environment	Realistic Dataset	Dynamic Inference
S. Tariq et al. [26]	×	✓	×	✓	×
K. Wolsing et al. [27]	×	×	×	✓	✓
B. Mbarek et al. [28]	×	✓	×	×	×
R. Lazzarini et al. [29]	×	✓	×	×	×
R. Zhao et al. [30]	×	✓	×	✓	×
M. Rashid et al. [31]	✓	×	×	×	×
Z. Ye et al. [32]	✓	✓	×	✓	×
S. Das et al. [33]	✓	✓	×	×	×
Z. Halim et al. [34]	✓	×	×	✓	×
Z. Chkirbene et al. [35]	✓	×	×	×	✓
Y. Zhou et al. [36]	✓	×	×	✓	×
C. Khammassi et al. [37]	✓	✓	×	×	×
M. Laundauer et al. [38]	×	×	×	✓	×
V. Kumar et al. [39]	×	✓	×	✓	×
L. Syne et al. [40]	×	✓	×	✓	×
S. S. Woo et al. [41]	×	✓	×	✓	×
Ours	✓	✓	✓	✓	✓

datasets that simulate IaC environments, and enabling dynamic inference to ensure reliable operation in non-stationary settings.

4 A Dynamic Classification Model for Network Intrusion Detection in IaC-deployed Infrastructures

Algorithm 1 Dynamic Classifier Selection - Inference Phase

Require:
Feature vector x ▷ Event to be classified
Region of competence $\mathcal{D}$ ▷ Region of competence of previously correctly classified events
Ensemble $\{C_1, C_2, \ldots, C_N\}$ ▷ Ensemble of classifiers
Number of classifiers to select k

procedure DynamicClassifier(x, $\mathcal{D}$, $\{C_1, C_2, \ldots, C_N\}$, k)
  Initialize an empty list $\mathcal{S}$ to store distances for each classifier
  for each event x_i in $\mathcal{D}$ **do** ▷ Compute event distance to region of competence
    Compute distance d from x_i to x
    Add d to $\mathcal{S}$
  end for
  Rank classifiers $\{C_1, \ldots, C_N\}$ based on their minimum distances in $\mathcal{S}$
  Return top-k classifiers $\{C_1, C_2, \ldots, C_k\}$ with smallest distances ▷ Select closest classifiers to event
end procedure

To address the generalization challenges caused by the dynamic behavior of IaC-deployed infrastructures in ML-based NIDS, we propose a dynamic classification model implemented through a feature selection scheme. The operation of our scheme is illustrated in Fig. 1, and is implemented in two phases, namely *Multi-objective Feature Selection*, and *Dynamic Classification*.

The *Multi-objective Feature Selection* aims at finding a feature space that optimizes classifier accuracy and generalization capabilities. To achieve this, we frame feature selection as a multi-objective optimization task, where the classifier is expected to improve accuracy on the anticipated IaC environment behavior while also enhancing generalization capabilities on unseen network traffic behavior. In practice, we measure generalization by evaluating the detection accuracy of unseen network events during the training phase, which is a common challenge in IaC-deployed infrastructures. Our main insight is to improve both classification accuracy and generalization during the training phase as a multi-objective feature selection task. This leads to a classifier capable of addressing the non-stationary behavior inherent to IaC-deployed infrastructures.

The goal of *Dynamic Classification* is to handle the classification of new environment behaviors during the inference phase in IaC-deployed infrastructures. To achieve this, we propose a dynamic classifier selection approach, where the subset of classifiers used for inference is actively chosen based on the current environment behavior. This enables the model to detect new behaviors generated by the dynamic nature of IaC *scripts* when deployed in production. As a result, our approach not only enhances generalization capabilities, driven by the feature selection process but also improves the detection of new network traffic behaviors. This paves the way for the implementation of ML-based NIDSs in IaC-deployed infrastructures in production environments.

The following subsections further describe the implementation of our proposal, including the modules that implement it.

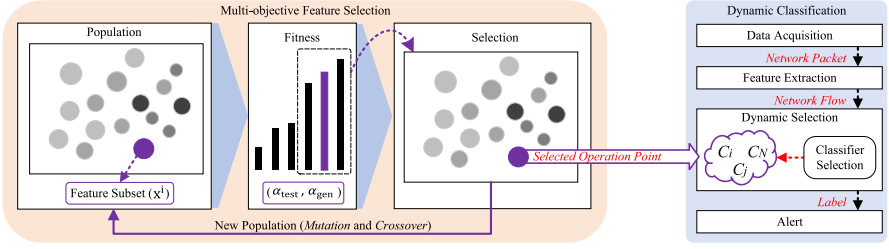

Fig. 1 Overview of our proposed dynamic classifier selection scheme for IaC-deployed infrastructures. The multi-objective feature selection aims to increase the resulting classifiers' generalization capabilities. The dynamic classification actively selects the subset of classifiers to address novelty detection at the IaC-deployment phase

4.1 Dynamic Classification

Implementing a reliable ML-based NIDS in IaC-deployed infrastructures is challenging due to the dynamic nature of the resulting environment. This presents difficulties for traditional ML-based NIDSs, which are not well-suited to handle the non-stationary behavior of network traffic (see Section 2.3). To address this, our proposed scheme frames the inference task as a dynamic classification task. The aim is to actively select the subset of classifiers that are most suited to classify the current environment behavior.

The classification overview is illustrated in Figure 1, and follows a traditional ML-based NIDS operation. It starts with the collection of network packets by the *Data Acquisition* module. The behavior of the collected network packets are then extracted by a *Feature Extraction* module, generating a network flow vector. The network flow is classified by a *Dynamic Selection* module, and signaled accordingly by the *Alert* module.

To conduct the classification, we frame it as a dynamic classifier selection task. Let $x \in \mathbb{R}^D$ denote a D-dimensional feature vector input and $y \in \{n, a\}$, where n denotes a *normal* event, and a denotes *attack* labeled events. Our model aims to learn a dynamic classifier ensemble, each of which models the probabilistic predictive distribution $p(y|x)$ over ground truth labels. To achieve such a goal, we build a N-sized classifier ensemble (Fig. 1, C_i). The dynamic classifier then finds a subset of classifiers from the ensemble that best fits the classification task based on the provided network event x.

Algorithm 2 Genetic Algorithm for solving Eq. 4.

```
Initialize population P with random solutions
Evaluate fitness of P based on Eq. 2 and Eq. 3                ▷ Initialize first generation
while not stopping condition met do
    Select parents from P based on fitness (Eq. 2 and Eq. 3)  ▷ Filter best individuals
    child ← Crossover(parents)
    Mutate(child, M_rate)
    Append child to offspring
    Evaluate fitness of offspring (Eq. 2 and Eq. 3)
    Replace less fit solutions in P with offspring            ▷ Create new generation
    Update best_solution if better found
end while
```

This is achieved by building a region of competence $\mathcal{D}$ from a given labeled test set. The region of competence comprises the set of events β from the test set that were correctly classified by each single classifier from the ensemble. In practice, $\mathcal{D}$ denotes correctly classified events from each classifier. At the inference phase, the dynamic classifier goal is to find a subset of classifiers K, such that the events from the region of competence $\mathcal{D}$ are similar to the to-be-classified event x. Therefore, we find the subset of classifiers based on the following equation:

$$\min_{\beta \in \mathcal{D}} \mathrm{k} \sqrt{\sum_{i=1}^{n} (\beta_i - x_i)^2} \tag{1}$$

where k denotes the number of classifiers to be selected, β the event from the region of competence $\mathcal{D}$, x the current environment event, and n the number of features. As a result, k classifiers are found based on the minimum Euclidean distance from the current evaluated event. Recalling that the region of competence $\mathcal{D}$ contains correctly classified events. Thus, we actively select classifiers based on a similarity measure with these past correct classifications.

Algorithm 1 overviews the implementation of our proposed dynamic classifier selection at the inference phase. The dynamic classifier selection algorithm operates by identifying a subset of classifiers from an ensemble that are most suitable for classifying a given event, represented as a feature vector x. This process begins by computing the distance between x and each event in the region of competence $\mathcal{D}$, which comprises previously correctly classified events. The distances are stored in a list $\mathcal{S}$. Next, the classifiers in the ensemble $\{C_1, C_2, \ldots, C_N\}$ are ranked based on the minimum distances computed from their respective regions of competence. Finally, the algorithm selects the top-k classifiers with the smallest distances to x, returning them as the most relevant classifiers for the task. This dynamic selection mechanism ensures that the classification task is performed using classifiers that have demonstrated competence in similar scenarios.

4.2 Multi-objective Feature Selection

ML-based NIDSs designed for IaC-provisioned infrastructures must generalize the behavior of the training environment. This requirement stems from the variability in IaC-deployed infrastructures, which change according to the provided configuration *script*. This variability presents a major challenge for effectively deploying such schemes in these dynamic settings. To tackle this challenge, we approach the *Dynamic Classification* training phase (sec. 4.1) as a multi-objective feature selection task, as illustrated in Figure 1. In practice, our goal is to optimize classification accuracy in the training environment and also account for a subset of attack and normal activities not encountered during the training phase.

To achieve such a goal, our multi-objective feature selection task aims at finding a feature space such that it minimizes the error rate in the training environment through the following equation:

$$\alpha_{test}(h, x_i, \mathcal{D}_{test}) = error(h(x_i, \mathcal{D}_{test})) \tag{2}$$

where *error* denotes a function that measures the error rate of the classification system h, using a feature space x_i, on a given test set $\mathcal{D}_{test}$. Here, the test set $\mathcal{D}_{test}$ holds a set of samples with a behavior similar to the initially expected from the to-be-deployed IaC infrastructure. Therefore, it contains the set of normal and attack samples expected to be generated from the initial IaC configuration *script*.

To measure the generalization capabilities of the designed model, we make use of the following equation:

$$\alpha_{gen}(h, x_i, \mathcal{D}_{gen}) = error(h(x_i, \mathcal{D}_{gen})) \tag{3}$$

where *error* also denotes a function that measures the error rate of the classification system h, using a feature space x_i, on a given test set $\mathcal{D}_{gen}$. Conversely, $\mathcal{D}_{gen}$ holds a set of normal and attack samples generated using new services and attack variants. As a result, it contains samples that are not expected to be generated from the current IaC *script* version. As a result, our multi-objective feature selection approach aims to solve the following equation:

$$\begin{aligned} &\arg\min_{x_0,\ldots,x_n} \ \alpha_{test}(h, x_i, \mathcal{D}_{test}) \\ \text{and} \\ &\arg\min_{x_0,\ldots,x_n} \ \alpha_{gen}(h, x_i, \mathcal{D}_{gen}) \end{aligned} \tag{4}$$

where $\{x_0, \ldots, x_n\}$ denotes all feature spaces variations, h the dynamic classification scheme (see Sec. 4.1), α_{test} the testing error objective (Eq. 2), and α_{gen} the generalization error objective (Eq. 3). As a result, we aim to find a feature space that simultaneously reduces the resulting dynamic classifier error rate on previously seen events (α_{test}) and new events (α_{gen}).

Given that searching over the complete feature space is not feasible, we solve Eq. 4 through a genetic search implementation. Algorithm 2 overviews the genetic search implementation on our scheme. It starts by initiating a population set P with random feature spaces, and computing their fitness using Eqs. 2 and 3. It then proceeds to select the most fitted individuals based on their associated fitness to crossover and mutate them to generate a new population. The less fitted individuals are replaced with the new offspring, and the procedure is repeated for N generations. Finally, the resulting set of individuals can be selected by the network operator based on the desired error rate trade-offs. For instance, in scenarios where the IaC configuration script remains relatively stable over time, the operator may prioritize a lower error rate on the test set while tolerating higher error rates for new events. Conversely, in environments where the IaC configuration changes frequently, the operator might favor a lower error rate on the generalization set to better handle the dynamic nature of the infrastructure.

4.3 Discussion

Addressing the dynamic behavior of IaC-deployed environments is a challenging task for current ML-based NIDSs. In light of this, our proposed model frames intrusion detection as a dynamic classifier selection (Section 4.1) built through a feature selection operation (Section 4.2). The first aims at actively selecting the subset of classifiers that should be used for the classification task. The goal is to adjust the behavior of the classification module at inference phase based on the current IaC-deployed environment behavior. The latter aims to select the subset of features that simultaneously minimize the dynamic classifier error rate on the test set and the generalization set (Eq. 4). Hence, it improves our scheme generalization capabilities caused by variations on the used IaC *script*. As a result, our proposed model paves the way for reliable ML-based NIDS implementation on IaC infrastructures.

5 Prototype and Testbed

We implemented a proposal prototype on a IaC-provisioned infrastructure executed in a private environment. Fig. 2 overviews the implementation of our proposed scheme. We considered a IaC environment implemented through Terraform [6] that deploys generated *scripts* through a Kubernetes [42] cluster *v*.1.31. The Kubernetes deploy the associated IaC scripts as Pods making use of Docker [43] *v*.24.0. We considered 5 different normal services, as follows:

- **DNS**. A Domain Name Service (DNS) server implemented through BIND *v*.9.11. The normal network traffic is generated through name resolution queries executed towards the server;
- **HTTP**. A web server implemented through Apache Tomcat *v*.11.0 hosting the top 500 websites from Alexa listing. The normal network traffic is generated through workload that randomly queries hosted websites;
- **SMTP**. A mail server implemented through Postfix *v*3.9. The normal network traffic is generated by sending random-sized e-mails from 100 to 1, 000 bytes;
- **SNMP**. A Simple Network Management Protocol (SNMP) server hosted on Ubuntu. The normal network traffic is generated by randomly querying the MIB tree;
- **SSH**. A Secure Shell (SSH) server implemented through openssh-server *v*.9.3 on an Ubuntu 22.04 container. The normal network traffic is generated by randomly executing a command from a 100-sized list in random intervals;

To generate realistic normal network traffic, we deploy 100 containers executing workload according to the deployed service in random intervals from 0 to 4 second periodicity. We also generate 14 categories of attack behaviors through Kali OS. For the attack generation, we vary the attack frequency and throughput.

We implemented our proposed ML-based NIDS (see Fig. 1) as an isolated service that continuously assesses the generated network traffic in our testbed. In this case, the generated network traffic is continuously acquired by a *Data Acquisition* module implemented using Scapy API *v*.2.6. The behavior of the collected network

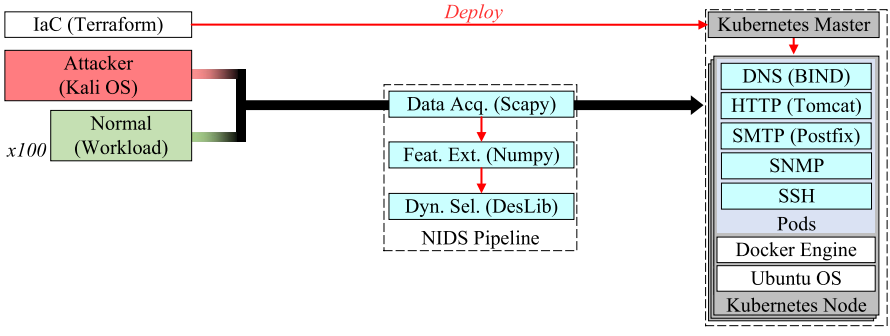

Fig. 2 Overview of our proposed prototype implementation on a private infrastructure

packets is extracted using numpy API *v*.1.26. Table 1 lists our prototype's extracted network flow features. We extract a total of 31 features considering the IP and services grouping in a 60 second interval. The extracted network flow features are input for our dynamic classifier selection scheme (see Section 4.1) implemented using DesLib API *v*.0.4. The used parameters and the implementation of our proposed feature selection scheme (see Alg. 2) are later discussed in Section 6.1.

Table 3 presents the statistics of our generated testbed. We executed 10 different service configurations, each for 30 minutes, to simulate a realistic IaC configuration scenario. In practice, the behavior of the environment, considering both attack and normal variants, varies with each deployment, leading to a realistic representation of IaC-deployed environment dynamics. As a result, our generated testbed creates a realistic IaC situation that can vary the environment behavior based on the utilized *script* configuration.

6 Evaluation

Our conducted experiments aim at answering the following Research Question (RQs):

- ***RQ1.*** *What is the accuracy impact of IaC behavior changes on traditional ML-based NIDS?*

Table 3 Statistics of our generated testbed concerning the network flows and packets

Env.	Behavior	Net. Flows	Net. Packets
$\mathcal{D}_{test}$	ACK Scan	13.2M	26.4M
	DDoS	734.8k	66.7M
	ICMP Echo Discover	13.2M	26.4M
	UDP Scan	51.8k	98.1k
	HTTP (*Normal*)	516.5k	5760
	SNMP (*Normal*)	2.7k	5760
$\mathcal{D}_{gen}$	MySQL Brute Force	70.2k	39.5M
	Nikto	1.6M	111.3M
	Scan Vuln	13.7M	53.5M
	SSH Brute Force	12.2k	472.5k
	Stealth Scan	13.2M	26.4M
	SYN Scan	13.1M	26.4M
	Brute Force DIRB	481.5k	32.9M
	Wapiti	1.4k	83.9k
	CMS Scan	1.1M	48.8M
	Full Connect Scan	13.2M	26.4M
	DNS (*Normal*)	49.1k	5760
	SMTP (*Normal*)	286.8k	5760
	SSH (*Normal*)	18.2k	5760

- ***RQ2.*** *Does our proposed feature selection technique improve the generalization capabilities of our scheme?*
- ***RQ3.*** *What is the accuracy performance of our model under IaC-deployed environments?*

The following subsections further describe the performance of our scheme including the model-building aspects and its performance on our new dataset.

6.1 Model Building

We evaluate our proposed *Dynamic Classifier Selection* (Alg. 1) implemented with a dynamic selection of classifiers, namely k-Nearest Oracle-Eliminate (KNORA-E). We use 5 neighbors to estimate the competence region and k-Nearest Neighbor (kNN) for distance computation (Eq. 1). The parameters were empirically set, and no significant influence on the results was observed when varying them. The classifiers were implemented on DESLib API *v*.0.4.

We also compare the performance of our proposed model *vs*. widely used traditional ensemble-based ML classifiers. The ensemble classifiers, namely Random Forest (RF), and Bagging (Bag), were implemented with 100 decision trees as their base learners, where each one of them also uses *gini* as the node split quality metric. The Decision Tree (DT) classifier relies on a gini node split quality metric without a maximum depth of the tree. Finally, the Multilayer Perceptron (MLP) classifier was implemented with 256 hidden neurons, with a Relu activation function and adam optimizer. The traditional ensemble classifiers were implemented through *scikit-learn* API *v*0.24.

We split the original dataset (Table 3) into *training*, *testing*, and *validation* datasets, each composed by 40%, 30%, and 30% respectively of each behavior. In addition, we generated two distinct datasets (see Table 3) based on these splits as follows:

- $\mathcal{D}_{test}$. Dataset used for training purposes. It contains DNS, HTTP, and SMTP normal network traffic, and ACKScan, Bruteforce DIRB, CMSScan, DDoS, and FullConnectScan attack network traffic;
- $\mathcal{D}_{gen}$. Dataset used for evaluating the generalization capabilities (see Section 4.2). It contains SNMP and SSH normal network traffic and the remaining attack network traffic. These services and attacks are not used for training;

Therefore, all selected classifiers are trained using the $\mathcal{D}_{test}$ *training* split dataset, evaluated using the *testing* split, whereas evaluated also using the $\mathcal{D}_{gen}$ *testing* split. The goal is to enable the generalization evaluation of our scheme to generate a similar behavior that would be evidenced in IaC-deployed infrastructures. In this case, the ML-based NIDS is subject to a specific training environment and should be able to generalize the behavior without retraining, as caused by variations on the used IaC *script*.

We evaluate the selected classifiers using the following classification performance metrics:

- *True Positive (TP)*: number of attack samples correctly classified as an attack.
- *True Negative (TN)*: number of normal samples correctly classified as normal.
- *False Positive (FP)*: number of normal samples incorrectly classified as an attack.
- *False Negative (FN)*: number of attack samples incorrectly classified as normal.

The F-Measure was computed according to the harmonic mean of precision and recall values while considering attack samples as positive and normal samples as negative, as shown in Eq. 7.

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

$$F\text{-}Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{7}$$

6.2 ML-Based NIDS in IaC-deployed Infrastructures

Our first experiment aims at answering *RQ1* and investigates the accuracy performance of traditional ML-based NIDSs on IaC-deployed infrastructures. To achieve such a goal, we train the traditional ML classifiers using the $\mathcal{D}_{test}$ subset, and evaluate their performance on the entire dataset.

Table 4 shows the classification performance of the selected classifiers when subjected to variations in IaC environments. It is evident that while all selected schemes maintain significantly high detection accuracies when evaluated on environments similar to those used during training ($\mathcal{D}_{test}$), their performance drops markedly when assessed on unseen environments with new services and attack patterns ($\mathcal{D}_{gen}$). This demonstrates the limited ability of traditional ML-based NIDSs to generalize to dynamic, real-world settings. For instance, the RF classifier achieved perfect performance on the $\mathcal{D}_{test}$ environment, with average TN and TP rates of 1.0 and 1.0, respectively. However, under the $\mathcal{D}_{gen}$ environment–representing new conditions generated by variations in IaC *scripts*–the same classifier experienced a notable decline in performance. Specifically, it achieved average TN and TP rates of only 0.77 and 0.56, corresponding to degradations of 0.23 and 0.44, respectively. This significant difference highlights the inability of the classifier to adapt to the evolving network behavior introduced by the dynamic nature of IaC deployments. As a consequence, it is possible to note the critical limitations of traditional ML-based NIDSs in dynamic and non-stationary environments, where IaC *scripts* introduce

Table 4 Classification accuracy of selected classifiers on our IaC dataset. Detection accuracy is reported in the TP rate for attack events and TN rates for normal events

Env.	Behavior	Detection Accuracy				
		Random Forest	Decision Tree	Bagging	Multilayer Perceptron	***Ours***
$\mathcal{D}_{test}$	ACK Scan	1.000	1.000	1.000	0.325	1.000
	DDoS	1.000	1.000	1.000	0.415	1.000
	ICMP Echo Discover	1.000	1.000	1.000	0.325	1.000
	UDP Scan	1.000	1.000	0.999	0.999	1.000
	HTTP (*Normal*)	1.000	1.000	0.999	1.000	1.000
	SNMP (*Normal*)	1.000	0.996	0.996	0.995	0.999
$\mathcal{D}_{gen}$	MySQL Brute Force	0.493	0.495	0.491	0.466	0.944
	Nikto	0.004	0.000	0.006	0.103	0.066
	Scan Vuln	0.962	0.962	0.962	0.332	0.964
	SSH Brute Force	0.999	0.999	0.999	0.984	0.997
	Stealth Scan	1.000	1.000	1.000	0.407	1.000
	SYN Scan	1.000	1.000	1.000	0.311	1.000
	Brute Force DIRB	0.000	0.000	0.006	0.113	0.067
	Wapiti	0.192	0.290	1.000	0.290	0.033
	CMS Scan	0.000	0.000	1.000	0.042	0.012
	Full Connect Scan	1.000	1.000	0.032	0.140	1.000
	DNS (*Normal*)	0.364	0.000	0.000	0.969	1.000
	SMTP (*Normal*)	1.000	1.000	0.998	0.998	1.000
	SSH (*Normal*)	0.964	0.990	0.990	0.995	1.000
Average		0.736	0.723	0.725	0.524	0.794

continuous variations to infrastructure behavior. These findings emphasize the importance of developing new approaches to address these challenges and ensure consistent performance even under dynamic environmental conditions.

Our second experiment aims at answering *RQ2*, and investigates how our proposed feature selection scheme can be used to improve the system generalization capabilities. To achieve such a goal, we implement our scheme as a wrapper-based feature selection using the *Non-dominated Sorting Genetic Algorithm* (NSGA-II) [44] on top of *pymoo* API (see Alg. 2). In such a case, the NSGA-II uses a 100 population size, 100 generations, a crossover of 0.3, and a mutation probability of 0.1. The multi-objective feature selection aims at decreasing the α_{test} (Eq. 2) and α_{gen} (Eq. 3), by implementing Alg. 2. Using our dynamic classification scheme, we measure the resulting objectives for every evaluated individual. In this case, we evaluate our proposed *Dynamic Selection* implemented with a dynamic selection of classifiers (KNORA-E) using a *bagging* pool of 100 estimators with replacement event selection and 5 neighbors (Alg. 1).

Figure 3 presents the Pareto curve of our wrapper-based feature selection model, highlighting the tradeoff between testing accuracy ($\mathcal{D}test$) and generalization

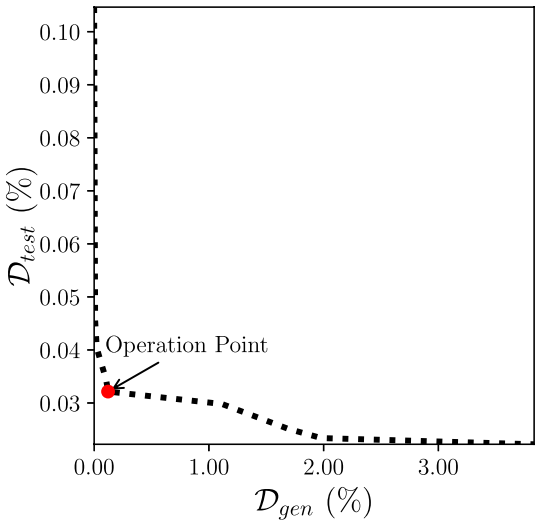

Fig. 3 Pareto curve of our proposed multi-objective feature selection mechanism

capabilities ($\mathcal{D}gen$). The curve consists of optimal solutions from the final generation of the optimization process. The results demonstrate that improving generalization to handle IaC-deployed infrastructure variations often requires tolerating slightly higher error rates in the training environment. Despite this tradeoff, our approach significantly enhances generalization with minimal impact on training accuracy. For instance, the model achieves a generalization error rate as low as 0.01%, while maintaining a training error rate of just 0.1%. This balance ensures adequate performance in dynamic environments caused by frequent changes in IaC *scripts*. In addition, the flexibility of the Pareto curve allows network operators to choose models that align with their specific needs, prioritizing either testing accuracy or generalization capabilities. This adaptability makes the proposed model a reliable solution for deploying ML-based NIDSs in real-world IaC infrastructures, reducing retraining requirements and improving operational efficiency.

Finally, to answer *RQ3* we investigate how the feature-selected dynamic classification can improve the resulting model's accuracy. To achieve such a goal, we select an average operation point that provides the best average values concerning both proposal's objectives (Fig. 3, *Operation Point*). It is important to note that the operation point used should be defined at the operator's discretion. As an example, one can favor higher generalization capabilities to address a more dynamic IaC-deployed infrastructures albeit the impact on the resulting model's accuracy. Therefore, we select the average operation point to provide a comparison baseline for our model efficacy.

Table 4 presents the detection accuracy achieved by our scheme at the selected operational point (*Ours*). The results demonstrate that our proposed model provides substantial improvements in detection accuracy across both the training and generalization environments. Specifically, the model achieves an average detection accuracy of 0.794, which surpasses the performance of traditional methods. Compared to random forest, decision tree, naive Bayes, and multilayer perceptron, our

approach achieves improvements of 0.058, 0.071, 0.056, and 0.270, respectively. This improvement highlights the effectiveness of our model in addressing the challenges posed by the dynamic nature of IaC-provisioned infrastructures. By leveraging the multi-objective feature selection and dynamic classification approach, the model enhances its ability to adapt to varying network behaviors while maintaining strong detection capabilities. These results show the reliability improvement of our proposed scheme for deployment in real-world IaC-deployed infrastructures.

We further analyze how our proposed model enhances classification accuracy for each of the evaluated attacks in our testbed. Figure 4 presents the Cumulative Distribution Function (CDF) of the selected classifiers across all 14 attacks generated in the IaC environment. The results clearly indicate that our model achieves significant improvements in detection accuracy for all evaluated attacks. For instance, our approach attains a TP rate of 90% for 11 out of the 14 attacks, outperforming traditional methods. By comparison, random forest, decision tree, naive Bayes, and multilayer perceptron achieve a TP rate of 90% for only 9, 9, 9, and 5 attacks, respectively. This improvement can be attributed to the synergy between our proposed multi-objective feature selection process (Alg.2) and the dynamic classifier selection mechanism (Alg.1) employed during the inference phase. The enhanced accuracy across a broader range of attack scenarios demonstrates the robustness and adaptability of our model to the dynamic nature of IaC-provisioned infrastructures. These results underline the importance of tailored feature selection and adaptive classification techniques in achieving reliable intrusion detection in such complex and variable environments.

Figure 5 presents a comparison of the average F1-Scores across the evaluated schemes. The results show that our proposed model substantially enhances generalization capabilities compared to traditional approaches (Fig.5b), while also achieving slight improvements in training environment accuracies (Fig.5a). Notably, our model achieves an increase in the resulting F1-Score by up to 0.31 on the generalization

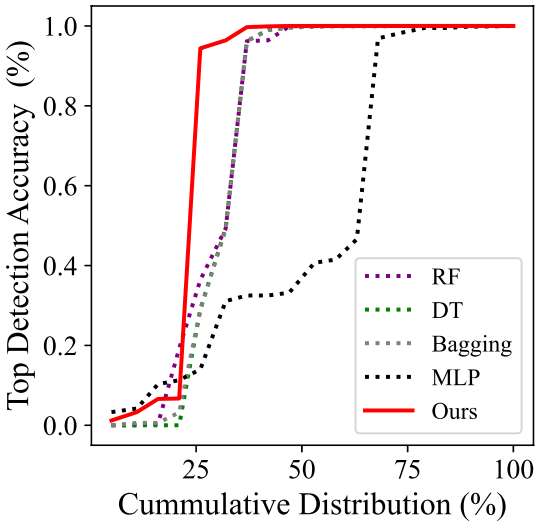

Fig. 4 Cumulative Distribution Function (CDF) of the true-positive accuracies of the selected classifiers on our built IaC dataset

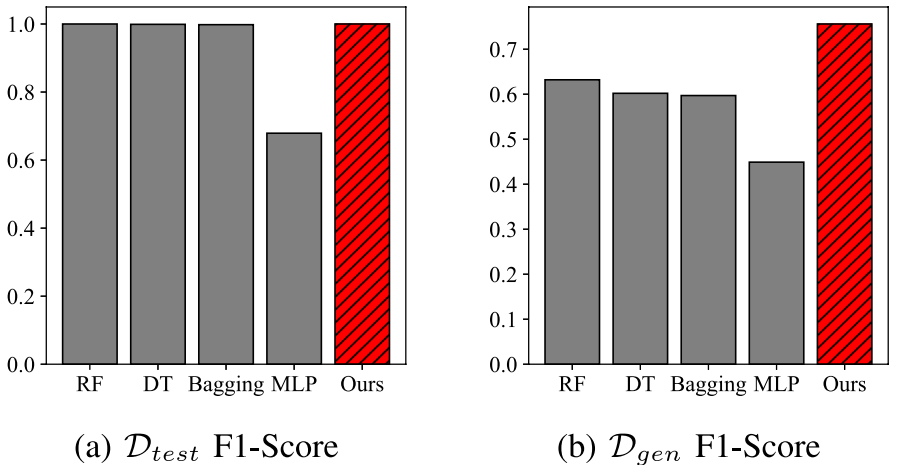

Fig. 5 F1-Score of the selected classifiers on the training and generalization datasets

set, highlighting its ability to effectively detect both known and novel behaviors in dynamic IaC-provisioned environments. This improvement stems from the synergy between our dynamic classifier selection mechanism and the multi-objective feature selection strategy, which collectively optimize the model's adaptability to diverse infrastructure scenarios. These findings emphasize the potential of our approach to enable reliable deployment of ML-based NIDS in IaC-provisioned infrastructures, addressing the challenges posed by their dynamic and non-stationary nature.

7 Conclusion

Securing IaC-provisioned infrastructures presents significant challenges due to the dynamic nature of these environments. The infrastructure undergoes continuous changes driven by the provided configuration scripts, leading to constantly shifting environment behaviors. This inherent dynamism makes maintaining consistent security difficult and renders traditional ML-based NIDSs unreliable in such settings. To address these challenges, this paper proposes a novel ML-based NIDS framework specifically designed for IaC-deployed infrastructures. The approach incorporates two key components: multi-objective feature selection and dynamic classifier selection. The multi-objective feature selection focuses on identifying a subset of features that enhance the model's generalization capabilities, while the dynamic classifier selection proactively chooses the subset of classifiers best suited to adapt to the current environment behavior. Experiments conducted on a newly developed IaC intrusion testbed demonstrate the feasibility and effectiveness of the proposed approach. The results show significant improvements in detection accuracy over traditional methods, paving the way for more reliable ML-based NIDSs in dynamic and non-stationary IaC environments.

As future work, we aim to extend the model to include unsupervised model updates after the IaC provisioning phase.

Acknowledgements This work was partially sponsored by the Brazilian National Council for Scientific and Technological Development (CNPq), grants no 304990/2021-3, 407879/2023-4, 302937/2023-4, and 442262/2024-8.

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection, and analysis were performed by Adilson G. Filho. Paper writing and reviewing was performed by Eduardo K. Viegas. Paper reviewing and supervision was performed by Altair O. Santin. Experiments reviewing and paper reviewing was performed by Jhonatan Geremias.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no Conflict of interest.

References

1. Rahman, A., Mahdavi-Hezaveh, R., Williams, L.: A systematic mapping study of infrastructure as code research. Inf. Softw. Technol. **108**, 65–77 (2019). https://doi.org/10.1016/j.infsof.2018.12.004
2. Opdebeeck, R., Zerouali, A., De Roover, C.: "Control and data flow in security smell detection for infrastructure as code: Is it worth the effort?" In: 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR). IEEE, (May 2023). https://doi.org/10.1109/MSR59073.2023.00079
3. Konjaang, J.K., Xu, L.: Meta-heuristic approaches for effective scheduling in infrastructure as a service cloud: A systematic review. J. Netw. Syst. Manag (2021). https://doi.org/10.1007/s10922-020-09577-2
4. ProgressChef, Chef - Extend DevOps Value, October 2024, https://www.chef.io/
5. Perforce, Puppet Infrastructure and IT Automation at Scale, October 2024, https://www.puppet.com/
6. Hashicorp, Terraform Deliver Infrastrcture as Code, October 2024, https://www.terraform.io/
7. Services, A.W.: AWS CloudFormation Speed up Cloud Provisioning with Infrastructure as Code, October 2024, https://aws.amazon.com/cloudformation/
8. Microsoft, Azure Resouce Manager, October 2024, https://azure.microsoft.com/get-started/azure-portal/resource-manager
9. Thakkar, P., Patel, A.S., Shukla, G., Kherani, A.A., Lall, B.: Dynamic microservice provisioning in 5g networks using edge-cloud continuum. J. Netw. Syst. Manag. (2024). https://doi.org/10.1007/s10922-024-09859-z
10. Saavedra, N., Ferreira, J.F.: Glitch: Automated polyglot security smell detection in infrastructure as code. Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ser. ASE '22. ACM, Oct. 2022. https://doi.org/10.1145/3551349.3556945
11. Cherfi, S., Lemouari, A., Boulaiche, A.: Mlp-based intrusion detection for securing iot networks. J. Netw. Syst. Manag. (2024). https://doi.org/10.1007/s10922-024-09889-7
12. Molina-Coronado, B., Mori, U., Mendiburu, A., Miguel-Alonso, J.: Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process. IEEE Trans. on Netw. Service Manag. **17**(4), 2451–2479 (2020). https://doi.org/10.1109/TNSM.2020.3016246
13. Rahman, A., Parnin, C.: Detecting and characterizing propagation of security weaknesses in puppet-based infrastructure management. IEEE Trans. Softw. Eng. p. 1–18 (2023). https://doi.org/10.1109/TSE.2023.3265962
14. Catillo, M., Pecchia, A., Villano, U.: Machine learning on public intrusion datasets: Academic hype or concrete advances in nids? In: 2023 53rd Annual IEEE/IFIP International Conference on

Dependable Systems and Networks - Supplemental Volume (DSN-S). IEEE (2023). https://doi.org/10.1109/DSN-S58398.2023.00038
15. Santos, K.C., Miani, R.S., de Oliveira Silva, F.: "Evaluating the impact of data preprocessing techniques on the performance of intrusion detection systems. J. Netw. Syst. Manag. (2024). https://doi.org/10.1007/s10922-024-09813-z
16. Rong, C., Geng, J., Hacker, T.J., Bryhni, H., Jaatun, M.G.: Openiac: open infrastructure as code - the network is my computer. J. Cloud Comput. (2022). https://doi.org/10.1186/s13677-022-00285-7
17. Andresini, G., Pendlebury, F., Pierazzi, F., Loglisci, C., Appice, A., Cavallaro, L.: Insomnia: Towards concept-drift robustness in network intrusion detection. In: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, ser. CCS '21. ACM, (Nov. 2021). https://doi.org/10.1145/3474369.3486864
18. Viegas, E., Santin, A.O., Abreu, V., Jr.: Machine learning intrusion detection in big data era: A multi-objective approach for longer model lifespans. IEEE Trans. Netw. Sci. Eng. **8**(1), 366–376 (2021). https://doi.org/10.1109/TNSE.2020.3038618
19. Olímpio, G., Camargos, L., Miani, R.S., Faria, E.R.: Model update for intrusion detection: Analyzing the performance of delayed labeling and active learning strategies. Comput. Secur. **134**, 103451 (2023). https://doi.org/10.1016/j.cose.2023.103451
20. Wang, Z., Liu, Y., He, D., Chan, S.: Intrusion detection methods based on integrated deep learning model. Comput. Secur. **103**, 102177 (2021). https://doi.org/10.1016/j.cose.2021.102177
21. de Carvalho, L.R., Patricia Favacho de Araujo, A.: Performance comparison of terraform and cloudify as multicloud orchestrators. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). IEEE, (May 2020), p. 380–389. https://doi.org/10.1109/CCGrid49817.2020.00-55
22. Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., Tamburri, D.A.: Devops: Introducing infrastructure-as-code. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), vol. 21. IEEE, (May 2017), p. 497–498. https://doi.org/10.1109/ICSE-C.2017.162
23. Horchulhack, P., Viegas, E.K., Santin, A.O.: Toward feasible machine learning model updates in network-based intrusion detection. Comput. Netw. **202**, 108618 (2022). https://doi.org/10.1016/j.comnet.2021.108618
24. Sarhan, M., Layeghy, S., Moustafa, N., Portmann, M.: Cyber threat intelligence sharing scheme based on federated learning for network intrusion detection. J. Netw. Syst. Manag. (2022). https://doi.org/10.1007/s10922-022-09691-3
25. Braun, T., Pekaric, I., Apruzzese, G.: Understanding the process of data labeling in cybersecurity. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, ser. SAC '24, vol. 35. ACM, Apr. 2024, p. 1596–1605. https://doi.org/10.1145/3605098.3636046
26. Tariq, S., Lee, S., Woo, S.S.: Cantransfer: transfer learning based intrusion detection on a controller area network using convolutional lstm network. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, ser. SAC '20. ACM, Mar. 2020. https://doi.org/10.1145/3341105.3373868
27. Wolsing, K., Kus, D., Wagner, E., Pennekamp, J., Wehrle, K., Henze, M.: One IDS Is Not Enough! Exploring Ensemble Learning for Industrial Intrusion Detection. Springer Nature Switzerland, p. 102–122 (2024). https://doi.org/10.1007/978-3-031-51476-0_6
28. Mbarek, B., Ge, M., Pitner, T.: Enhanced network intrusion detection system protocol for internet of things. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, ser. SAC '20, vol. 28. ACM, (2020), p. 1156–1163.https://doi.org/10.1145/3341105.3373867
29. Lazzarini, R., Tianfield, H., Charissis, V.: A stacking ensemble of deep learning models for iot intrusion detection. Knowl.-Based Syst. **279**, 110941 (2023). https://doi.org/10.1016/j.knosys.2023.110941
30. Zhao, R., Yin, J., Xue, Z., Gui, G., Adebisi, B., Ohtsuki, T., Gacanin, H., Sari, H.: An efficient intrusion detection method based on dynamic autoencoder. IEEE Wirel. Commun. Lett. **10**(8), 1707–1711 (2021). https://doi.org/10.1109/LWC.2021.3077946
31. Rashid, M., Kamruzzaman, J., Imam, T., Wibowo, S., Gordon, S.: A tree-based stacking ensemble technique with feature selection for network intrusion detection. Appl. Intell. **52**(9), 9768–9781 (2022). https://doi.org/10.1007/s10489-021-02968-1
32. Ye, Z., Luo, J., Zhou, W., Wang, M., He, Q.: An ensemble framework with improved hybrid breeding optimization-based feature selection for intrusion detection. Future Generation Comput. Syst. **151**, 124–136 (2024). https://doi.org/10.1016/j.future.2023.09.035

33. Das, S., Saha, S., Priyoti, A.T., Roy, E.K., Sheldon, F.T., Haque, A., Shiva, S.: Network intrusion detection and comparative analysis using ensemble machine learning and feature selection. IEEE Trans. Netw. Serv. Manag. **19**(4), 4821–4833 (2022). https://doi.org/10.1109/TNSM.2021.3138457
34. Halim, Z., Yousaf, M.N., Waqas, M., Sulaiman, M., Abbas, G., Hussain, M., Ahmad, I., Hanif, M.: An effective genetic algorithm-based feature selection method for intrusion detection systems. Comput. Secur. **110**, 102448 (2021). https://doi.org/10.1016/j.cose.2021.102448
35. Chkirbene, Z., Erbad, A., Hamila, R., Mohamed, A., Guizani, M., Hamdi, M.: Tidcs: A dynamic intrusion detection and classification system based feature selection. IEEE Access **8**, 95864–95877 (2020). https://doi.org/10.1109/ACCESS.2020.2994931
36. Zhou, Y., Cheng, G., Jiang, S., Dai, M.: Building an efficient intrusion detection system based on feature selection and ensemble classifier. Comput. Netw. **174**, 107247 (2020). https://doi.org/10.1016/j.comnet.2020.107247
37. Khammassi, C., Krichen, S.: A nsga2-lr wrapper approach for feature selection in network intrusion detection. Comput. Netw. **172**, 107183 (2020). https://doi.org/10.1016/j.comnet.2020.107183
38. Landauer, M., Skopik, F., Frank, M., Hotwagner, W., Wurzenberger, M., Rauber, A.: Maintainable log datasets for evaluation of intrusion detection systems. IEEE Trans. Depend. Secure Comput. **20**(4), 3466–3482 (2023). https://doi.org/10.1109/TDSC.2022.3201582
39. Kumar, V., Sinha, D.: Synthetic attack data generation model applying generative adversarial network for intrusion detection. Comput. Secur. **125**, 103054 (2023). https://doi.org/10.1016/j.cose.2022.103054
40. Syne, L., Caballero-Gil, P., Hernandez-Goya, C.: "Improving privacy in federated learning-based intrusion detection for iot networks. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, ser. SAC '24, vol. 23. ACM, Apr. 2024, p. 580–583. https://doi.org/10.1145/3605098.3636183
41. Woo, S.S., Yoon, D., Gim, Y., Park, E.: Raad: Reinforced adversarial anomaly detector. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, ser. SAC '24, vol. 393. ACM, Apr. 2024, p. 883–891. https://doi.org/10.1145/3605098.3635920
42. Kubernetes, Kubernetes Open-source, October 2024, https://kubernetes.io/
43. Docker, Docker develop faster, October 2024, https://www.docker.com/
44. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)

Authors and Affiliations

Adilson G. Filho[1] · Eduardo K. Viegas[1] · Altair O. Santin[1] · Jhonatan Geremias[1]

✉ Eduardo K. Viegas
eduardo.viegas@ppgia.pucpr.br

Adilson G. Filho
galiano@ppgia.pucpr.br

Altair O. Santin
santin@ppgia.pucpr.br

Jhonatan Geremias
jgeremias@ppgia.pucpr.br

[1] Graduate Program in Computer Science, Pontifical Catholic University of Parana (PUCPR), Curitiba, Brazil