Meta-IDS: A Multi-stage Deep Intrusion Detection System with Optimal CPU Usage

Nadia Niknami, Vahid Mahzoon, and Jie Wu
Center for Networked Computing, Temple University, USA
Emails: {nadia.niknami, vahid.mahzoon, jiewu}@temple.edu

***Abstract*—The exponential growth in network traffic coupled with the intricacies of neural network methodologies has presented formidable challenges for conventional single-machine architecture Network Intrusion Detection Systems (NIDS). In this paper, we propose a novel approach to address these challenges drawing inspiration from meta-computing principles. Meta-computing, characterized by dynamic resource allocation, offers a promising solution for optimizing NIDS performance in the face of escalating network traffic. We introduce a hierarchical NIDS that employs varying levels of model complexity to efficiently process different scales of network traffic. In designing the proposed lightweight IDS, our efforts focus on selecting the optimal set of features for each level. By integrating dynamic resource allocation techniques, our Meta-IDS is able to adapt proactively to fluctuations in network activity, thus mitigating the risk of resource depletion while preserving its efficacy in detecting intrusions. Through experiments conducted on a benchmark dataset, we demonstrate the effectiveness of our proposed Meta-IDS in enhancing network intrusion detection in high-performance environments. Our approach not only ensures efficient resource management but also enhances the accuracy and timeliness of intrusion detection, thereby providing a robust solution for modern network security challenges.**

***Index Terms*—Accuracy, CPU time, Network Intrusion Detection System(NIDS), Lightweight IDS, Meta-computing.**

I. Introduction

The crucial function of Network Intrusion Detection Systems (NIDS) [1] in safeguarding network perimeters by diligently monitoring and swiftly identifying security breaches remains of essential importance. NIDS employs two primary detection methodologies: signature-based and anomaly-based [2] [3]. Signature-based NIDS establishes a knowledge base through state modeling or string matching beforehand, detecting aberrant behavior by comparing data flow against existing signatures. While signature-based NIDS demonstrates robust performance against known attacks, it may falter in addressing attacks absent from the knowledge base. Conversely, anomaly-based NIDS possesses the capability to discern unknown attacks by evaluating deviations between detected activity and normal patterns, a domain currently undergoing vigorous development. Detecting malicious network traffic is a critical element of network intrusion detection, with the goal of the timely and precise identification of hidden malicious attacks within network traffic. Consequently, accurately identifying malicious traffic stands as a pivotal area of research focus. With the advent of Deep Learning (DL), a multitude of DL-based intrusion detection models have emerged, significantly enhancing the accuracy and resilience of intrusion detection. However, despite the commendable accuracy attained by these models, researchers have struggled to implement them on resource-constrained devices due to their high computational overhead and large model size. The dynamic nature of large-scale networks introduces unique challenges for effective network intrusion detection. Moreover, the traditional single-machine architecture of NIDS has trouble with the continual growth in traffic volume. Despite considerable efforts from researchers, NIDS continues to encounter hurdles in deployment on resource-constrained devices. Various approaches have been suggested to alleviate the model size burden for NIDS. However, reducing network complexity often entails disregarding channel correlations, resulting in diminished detection accuracy.

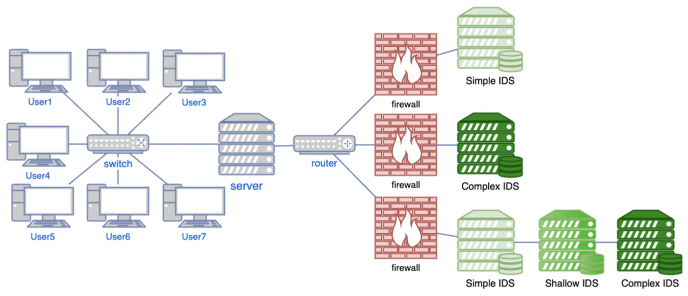
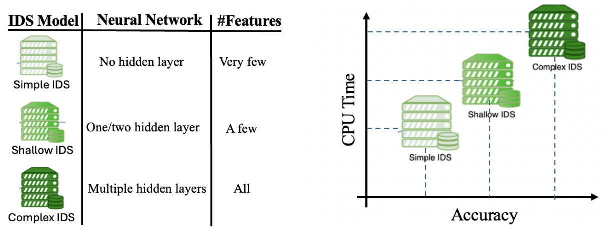


Fig. 1: Different structures for IDS in a network.

Furthermore, the dynamics of large-scale networks introduce additional complexities. The varied traffic patterns observed in such environments, characterized by diverse application mixes and temporal fluctuations, significantly influence NIDS performance . Notably, the depth of analysis required for different application protocols varies, necessitating adaptive strategies. Moreover, the prevalence of heavy-tailed data transfers leads to sudden peaks in traffic volume, presenting a considerable challenge for NIDS operations. In high-performance environments, the efficacy of a NIDS depends not only on its capability to handle average traffic loads but also on its resilience in the face of frequent traffic bursts. Therefore, robust mechanisms must be in place to ensure the NIDS's effectiveness under dynamic and demanding conditions.

The time it takes to detect an intrusion in cybersecurity, referred to as the "Time To Detect" (TTD), is a crucial metric for organizations to gauge their ability to swiftly identify and respond to security incidents. Several factors influence TTD, including the complexity of the security infrastructure, the effectiveness of security monitoring systems, and the level of



IDS Model	Neural Network	#Features
Simple IDS	No hidden layer	Very few
Shallow IDS	One/two hidden layer	A few
Complex IDS	Multiple hidden layers	All

(a) Different IDS models (b) Accuracy/CPU time trade-off.

Fig. 2: Different IDS models and their efficiency.

training and expertise of the security team.

For some datasets, a simpler model might suffice and even surpass a more complex model because of its superior generalization capability. However, for large and complex datasets, the simplicity of the model may prove inadequate in achieving desired outcomes. In such cases, the utilization of complex models becomes necessary to ensure acceptable efficiency. Complex models can achieve higher accuracy but they demand greater computational resources for both training and inference. This encompasses memory, computing power, and time. Utilizing a complex model may not be viable in situations where resources are constrained. To tackle these challenges, we advocate for implementing a multi-level approach to NIDS instances. This approach inherently offers scalability, allowing for a linear expansion of hardware resources to accommodate increasing system loads while maintaining consistent service levels. This incremental scalability proves particularly advantageous in large-scale environments, where capacity planning is complicated by numerous unknown variables.

Meta-computing arises as an enticing methodology for acquiring resources to efficiently handle extensive computational tasks. Expanding on this notion, we introduce a hierarchical architecture for NIDS deployment. This architecture dynamically allocates analysis tasks, reducing communication and processing overheads while enhancing the system's flexibility, scalability, and fault tolerance. Through the adoption of this innovative approach, our goal is to augment the effectiveness of network intrusion detection in high-performance environments, thereby paving the way for more robust and adaptable security solutions.

In the network depicted in Fig. 1, various users are connected to a server, which forwards their traffic to a firewall and NIDS for monitoring purposes. These IDSs analyze the passing traffic and generate alert messages if anomalies are detected. This figure illustrates three potential strategies for IDS deployment: 1) *Simple Model*, 2) *Complex Model*, and 3) *Mixture Model*. Each of these IDS frameworks varies in structure and complexity, resulting in divergent requirements for accuracy and CPU time. Depending on factors such as the scale of traffic and the sensitivity of attack detection, organizations can select the most suitable framework from these options to effectively monitor and protect their network infrastructure. Fig. 2(a) delineates the distinct configurations for *Simple Model*, *Shallow Model*, and *Complex Model*, with complexity determined by neural network architecture and feature count. Fig. 2(b) illustrates the trade-off between accuracy and processing time across different IDS models.

Most of the models that have been proposed so far suffer from various issues, including large model parameters, high complexity, and lengthy computing times. Additionally, these models require continuous training after deployment. The primary goal of malicious traffic detection and identification is to detect the intrusion of malicious traffic, which plays a key role in ensuring security. Deep learning has shown great success in this field. However, due to resource limitations such as computational weaknesses and low-edge network node storage capacity, deploying and applying high-complexity deep learning models becomes challenging. In this paper, we delve into the complexities of network intrusion detection in high-performance environments, exploring innovative approaches and strategies to overcome inherent challenges and enhance the efficacy of NIDS deployments. Through empirical analysis and practical insights, we aim to contribute to the advancement of NIDS technology, enabling more robust and resilient security solutions for modern network infrastructures. This paper makes the key contributions to address the existing issues above as follows:

- We proposed a lightweight intrusion detection system to reduce model complexity while maintaining acceptable attack detection accuracy.
- We introduce a hierarchical intrusion detection system with multiple stages, each tailored to balance CPU time and accuracy. This framework incorporates the concept of meta-computing, adjusting parameters and features to mitigate performance declines in lightweight IDS implementations.
- We use different feature selection methods to select the most informative features from CICFlowMeter features to design a lightweight intrusion detection system.
- Extensive experiments demonstrate that our Meta-IDS achieves significant advantages in accuracy and efficiency on a challenging dataset in different scenarios with varying sensitivities and complexities.

II. Background and Related Works

A. *Network Intrusion Detection*

A NIDS detects potential security breaches and alerts are communicated through various means such as textual alerts, log files, or a graphical user interface. Alerts are then either analyzed by human analysts or automatic postprocessing facilities. Correctly identified intrusions are true positives, while false alarms are false positives. Failure to recognize an ongoing intrusion results in a false negative, while correctly remaining inactive when no breach occurs is a true negative. DL approaches have shown promise in enhancing the performance of IDS, particularly in terms of accuracy and load balancing [2], [4], [5]. Complex DL models provide high detection accuracy for IDS but bring challenges to deployment with resource-constrained devices.

Yazdinejadna *et al.* [6] introduced a zone-based architecture for intrusion detection to enhance scalability and anomaly detection in NIDS. Niknami *et al.* [7] proposed an approach deploying a chain of IDSs in the data plane interconnected with switches for efficient data flow grouping and therefore balancing the load on the controller. Zhao *et al.* [8] proposed a light IDS model, albeit with reduced detection rates. Wang *et al.* [9] proposed a knowledge distillation model to reduce model complexity, although designing appropriate teacher and student models remains challenging. Verkerken *et al.* [10] introduced a multistage and scalable IDS capable of detecting unknown and zero-day attacks.

Yang et al. [11] proposed a lightweight intrusion detection method achieving a balance between accuracy and efficiency through self-knowledge distillation. Hocine *et al.* [12] presented a collaborative NIDS based on a multi-agent framework, utilizing dynamic load balancing of traffic analysis to enhance intrusion detection, particularly against DDoS attacks, while minimizing excessive communication. Ge *et al.* [13] introduced MetaCluster, a versatile classification framework for cybersecurity. MetaCluster filters and combines classification semantics through feature prototypes and dynamic graph learning layers, offering a comprehensive solution for interpretable classification tasks.

To address the challenge of balancing detection accuracy with model complexity, which often leads to increased CPU consumption, this paper introduces a lightweight multistage NIDS. Leveraging meta-computing, we propose dynamic resource allocation for each level of the NIDS. Additionally, we present a novel method for feature selection aimed at reducing model complexity.

B. Meta-computing

Meta-computing involves managing and orchestrating computing resources in a dynamic and adaptive manner to optimize performance, efficiency, and resource utilization [14]. Dynamically allocating resources based on workload demands and system conditions is a key aspect of meta-computing because it allows for the efficient utilization of available resources and adaptation to changing computational requirements. Meta-computing involves the utilization of computing resources in a dynamic, adaptive, and efficient manner to solve computational problems effectively. When integrated with few-shot learning, meta-computing techniques can help optimize resource allocation, enhance parallel processing capabilities, and improve overall computational efficiency.

C. Feature Selection

Feature selection plays a pivotal role in machine learning and data analysis, aiming to identify a subset of relevant features from the original. This process not only enhances model performance but also helps mitigate overfitting while improving interpretability. By prioritizing informative features, model simplification, reduced computational complexity, and potential accuracy enhancement can be achieved. The advantages of feature selection are as follows:

- Improved Model Performance: By eliminating irrelevant or redundant features, feature selection can enhance the performance of machine learning models, leading to better predictive accuracy and generalization on unseen data.
- Reduced Overfitting: Including only the most relevant features helps in reducing overfitting, where the model learns noise present in the data rather than capturing the underlying patterns. This leads to better generalization on new data.
- Enhanced Interpretability: Simplifying the model by selecting only the most important features makes it easier to interpret and understand the relationships between input variables and the target variable.
- Faster Training and Inference: By reducing the dimensionality of the feature space, feature selection can significantly decrease the computational cost associated with training and evaluating machine learning models.

There are different methods of feature selection and extraction [15]–[17]. We list out the famous ones as follows:

- Mutual Information: Mutual information [18] measures the amount of information obtained about one variable through the other variable revealing a non-linear relationship. In feature selection, it quantifies the dependency between the feature and the target variable, and features with high mutual information are considered more informative.
- Sparse Sensor Placement Optimization for Reconstruction (SSPOR): SSPOR [19] involves selecting a subset of features from a larger feature set in such a way that the selected features provide the most informative representation of the data while minimizing redundancy and computational costs. Similar to its application in sensor placement, SSPOR for feature selection aims to achieve sparsity by identifying a minimal set of features that are most relevant for accurately describing the underlying patterns in the data. This optimizer arranges features in descending order of significance using the QR with a column pivoting algorithm.
- Lasso (Least Absolute Shrinkage and Selection Operator): Lasso [20] is a regularization technique that penalizes the absolute value of the regression coefficients. It tends to shrink regression coefficients of less informative features towards zero, effectively performing feature selection.
- Random Feature Selection: Random feature selection involves randomly selecting a subset of features from the original feature set. This method is often used as a baseline for comparing the performance of other feature selection algorithms.
- Minimum Redundancy Maximum Relevance (MRMR): MRMR [21] aims to select features that are highly relevant to the target variable while being minimally redundant with each other. It evaluates both the relevance and redundancy of features and selects the subset that maximizes the relevance and minimizes redundancy.
- Gini Importance: Gini importance measures the importance of a feature by calculating the total decrease in

node impurity (e.g., Gini impurity) caused by splitting the data based on that feature in a decision tree and random forest. Features with higher Gini importance are considered more important for classification tasks.

- Principal Component Analysis (PCA): PCA [22] is a dimensionality reduction and feature extraction technique that transforms the original features into a new set of orthogonal features called principal components. These components are ordered by the amount of variance they explain in the data. By selecting a subset of principal components that capture most of the variance, PCA effectively performs feature selection while preserving the most important information in the data.

III. Methodology

In this section, we present a detailed discussion of the proposed lightweight Meta-IDS. If an IDS fails to process the incoming packet stream at full wire speed, it risks encountering buffer overflows or packet loss, potentially leading to system crashes. Given the detrimental impact of packet loss on attack detection, it is imperative to prevent such occurrences. From another angle, for malicious network traffic detection and identification it is very important to design a model with fewer parameters, low complexity, and high performance. We propose a deep learning model, called the lightweight intrusion detection approach. In our hierarchical approach, we dynamically distribute analysis tasks across the multi-level NIDS, thereby reducing communication and processing overheads. This dynamic allocation enhances the flexibility and scalability of the system while bolstering its fault tolerance capabilities.

Fig. 3 presents the hierarchical architecture of our intrusion detection framework, strategically designed to overcome server capacity limitations and bolster detection efficiency by harnessing powerful CPU systems. Inspired by meta-computing principles, our framework divides the intrusion detection process into three stages, each equipped with unique capabilities and resource allocations. In the initial stage, a simple neural network (without hidden layers) is utilized to identify straightforward attacks based on predefined thresholds and probabilities. If the model confidently predicts a traffic flow as benign, it requires no further processing. If a sample's probability of belonging to a class falls below a threshold τ_1 or the model predicts the sample as an attack, it advances to the next stage as suspicious traffic. The next stage involves a simple multi-class neural network with more features.

In the event that the classification probability falls below τ_1, the sample is forwarded to the second stage, where a classifier determines whether it corresponds to any known attack class (ATK_i) or the benign class. This stage employs distinct thresholds and an advanced anomaly detector capable of identifying more intricate attacks across various types. Should the attack classifier fail to categorize the sample into a known attack class with a level of certainty exceeding threshold τ_2, the sample proceeds to the third and final stage. Here, traffic with probabilities falling below τ_2 in the second stage undergoes further analysis. Equipped with substantial allocations of storage and CPU resources, this stage plays a pivotal role in detecting unseen or complex attacks. By adopting this hierarchical approach, our framework ensures comprehensive intrusion detection while optimizing resource utilization for efficient and effective threat mitigation.

Algorithm 1 Meta-IDS Algorithm

```
 1: Input $(X, Y) = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, $\mathcal{F}_1$, $\mathcal{F}_2$, $\tau_1$, $\tau_2$
 2: Output $\hat{Y} = \{\hat{y}_1, \ldots, \hat{y}_N\}$: Predicted Labels
 3: for each data point $x_i$ in $X$ do
 4:     $p_1, c_1 \leftarrow$ BinaryModel($x_i, \mathcal{F}_1$)
 5:     if $p_1 < \tau_1$ or $c_1 ==$ malicious then
 6:         $p_2, c_2 \leftarrow$ SimpleMultiClassModel($x_i, \mathcal{F}_2$)
 7:         if $p_2 < \tau_2$ then
 8:             $\hat{y}_i \leftarrow$ ComplexMultiClassModel($x_i$)
 9:         else
10:             $\hat{y}_i \leftarrow c_2$
11:         end if
12:     else
13:         $\hat{y}_i \leftarrow c_1$
14:     end if
15: end for
```

The Meta-IDS algorithm, outlined in Algorithm 1, is designed to predict labels for incoming data points based on a cascading decision-making process. Given a dataset (X, Y) consisting of input features X and corresponding labels Y, along with two sets of feature representations $\mathcal{F}_1$ for the number of features in *Stage 1* and $\mathcal{F}_2$ for the number of features in *Stage 2*, and two threshold values τ_1 and τ_2 for *Stage 1* and *Stage 2* respectively. In *Stage 1*, for each data point x_i in X, it first employs a binary classification model (`BinaryModel`) using $\mathcal{F}_1$ to determine if the data point is potentially malicious. If the probability of being malicious or benign is below the threshold τ_1 or the predicted class is explicitly marked as malicious, the algorithm proceeds to *Stage 2* which is a simple multi-class classification model (`SimpleMultiClassModel`).

In *Stage 2* with $\mathcal{F}_2$ features, if the probability (p_2) of belonging to any class is below the threshold τ_2, the algorithm predicts the class using a complex multi-class classification model (`ComplexMultiClassModel`) in *Stage 3*; otherwise, it assigns the class predicted by the simple multi-class model. If the initial binary classification indicates non-malicious behavior with high confidence ($p_1 \geq \tau_1$), the algorithm directly assigns the class predicted by this model without further evaluation. Through this sequential decision-making process, the Meta-IDS algorithm aims to provide robust and accurate predictions for network intrusion detection.

A. Stage 1: First Level IDS

The first stage of our methodology involves assessing the potential maliciousness of each data point within the dataset. In this initial stage, denoted as *Stage 1*, we employ a binary classification model referred to as `BinaryModel`. This

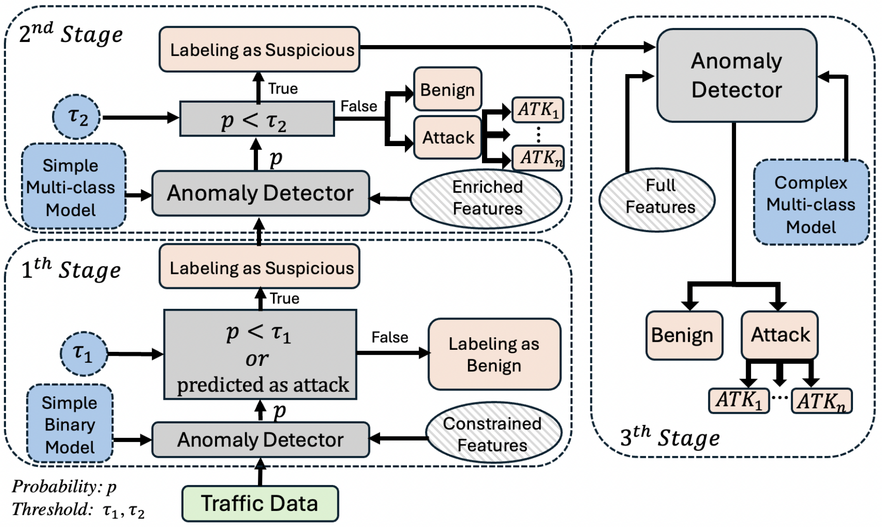


Fig. 3: The proposed architecture of the multi-stage hierarchical intrusion detection system.

model is specifically trained on a very small portion of the dataset, which comprises two distinct classes: benign and an attack label that encompasses all types of attacks. This training aims to discern and understand the normal behavior patterns within the monitored network. By analyzing the behavior of each data point using some features, $\mathcal{F}_1$, the binary model determines whether it aligns with the learned normal and malicious patterns. If the probability of a data point being malicious or benign falls below a predefined threshold τ_1, or if the predicted class is explicitly flagged as malicious, the algorithm progresses to *Stage 2*. For this level, we use a simple logistic regression model.

B. Stage 2: Middel-level IDS

In the second stage, data samples identified as potentially malicious during the anomaly detection phase undergo multi-class classification. This classifier is exclusively trained on data comprising a benign label and multiple attack types. It demonstrates proficiency in accurately categorizing samples into the known attack classes present within the training dataset, as well as identifying benign cases. During this stage, a straightforward multi-class classification model, denoted as `SimpleMultiClassModel`, is utilized for further evaluation. This sequential decision-making process is designed to efficiently detect potential malicious data points while optimizing computational resources. The classifier generates probabilities for each known attack class as well as the benign class, and the class with the highest probability is assigned as the predicted class. Samples with probabilities falling below a predefined threshold τ_2 are classified as unknown and forwarded to the subsequent stage. This threshold is carefully determined to achieve the highest level of accuracy in classification. At this level, a feedforward neural network with no hidden layers has been applied. This model takes input, applies a linear transformation (through weights and biases), and then uses a softmax function to output probabilities for each of the classes.

C. Stage 3: Full-detection

The last stage involves the utilization of a complex neural network, encompassing all available features, to conduct comprehensive detection on data samples that were not satisfactorily classified during *Stage 2*. The main aim of this stage is to address false positives, particularly instances where benign network activity is incorrectly identified as malicious. False positives represent a significant challenge for anomaly-based IDS. Furthermore, since only a subset of data is forwarded to *Stage 3*, the computational workload is significantly reduced compared to analyzing the entire dataset. This approach helps optimize CPU time consumption while maintaining robust detection capabilities. For the complex model, a feedforward neural network with three hidden layers and a ReLU activation function is utilized.

IV. Evaluation

In this section, we begin by assessing various feature selection techniques across different numbers of features within our dataset. This evaluation aims to determine the correlation between the quantity of features and the accuracy of models used in the initial two stages of our methodology, and to find the best feature selection method for k number of features. Then, we proceed to examining a simpler application of our methodology, where we combine a simple and a complex binary model (comprising only two stages) to demonstrate the efficiency of our approach in the binary classification problem. Following this, we delve into multi-class classification. We explore three distinct scenarios, each characterized by a different number of features in the first two stages. These variations result in differing levels of model accuracy and inference time in the initial stages, closely mimicking real-world conditions. In each case, we determine the optimal thresholds using a proposed utility function, and subsequently illustrate how our methodology not only reduces processing time but also maintains accuracy within acceptable limits when compared to a complex model. The outcomes of these three scenarios are then analyzed and compared to underscore the effectiveness of our approach.

A. Experimental Settings

We conducted our experiments on a system equipped with an Intel(R) Xeon(R) W-2225 CPU @ 4.10GHz, featuring an $x86\_64$ architecture with 8 CPUs, each with 2 threads per core and 4 cores per socket, reaching a maximum frequency of 4.6 GHz. We conduct each experiment multiple times and then calculate the average to present the results.

B. Dataset and evaluation metrics

In order to evaluate our method, experiments were carried out on CICIDS2017. This dataset was generated from real network recordings. It contains benign samples along with the samples of most up-to-date common attacks. The generation of the dataset spanned a period of 5 days using 14 machines. The benign traffic is simulated from the benign behavior of a group of 25 humans using statistical techniques and machine learning. On the other hand, the malicious traffic is generated by executing existing attack tools at specific time windows. In order to preprocess the data for dataset CICIDS2017, several preprocessing steps were undertaken. Firstly, data samples containing NaN values were eliminated. Secondly, data samples featuring negative values for attributes that necessitate non-negative values were discarded. Next, one-hot encoding was applied to the Protocol column, resulting in three new features. Subsequently, columns representing Source IP, Source Port, Destination IP, and Destination Port were removed, leaving us with 79 features and one column designated for labeling. Finally, the features were standardized, a process involving centering them around zero by subtracting the mean and scaling them to unit variance. Additionally, we excluded the dataset pertaining to infiltration attacks from our analysis due to the exceedingly small number of samples available for this type of attack.

After the preprocessing stage, our dataset was categorized into six attack types along with benign instances, as detailed in Table I. For our binary model, we trained on 10% of this data, while the multi-class models were trained on just 2%. Utilizing such a limited dataset significantly reduced our model training times and CPU usage. For testing purposes, we selected 1 million records, not included in the training set, comprising $500,000$ benign instances and $500,000$ attacks. In our experiments, to simulate larger traffic volumes, we repeatedly utilized this 1 million test dataset multiple times to process a total of 8 million samples.

In order to evaluate the proposed method, we considered measurements of *Accuracy* and *Inference time*. Inference time in deep learning refers to the time it takes for a trained model to make predictions on new, unseen data. During inference, the model processes input data and produces an output, such as a classification or a regression prediction. Inference time is an important consideration for deploying deep learning models in real-world applications, as it impacts the speed and responsiveness of the system.

C. Feature Selection Results

We implemented our feature selection techniques on a small subset of the data for different numbers of features

TABLE I: Categorizing labels in datasets after pre-processing.

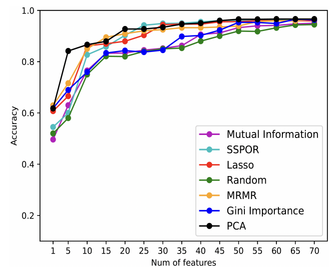
Label	Category in CICIDS2017	Number of Samples
Benign	Benign	1982759
DoS	DoS Hulk	230123
	DoS Slowloris	5796
	DoS Slowhttp	5499
	DoS GoldenEye	10293
	Heartbleed	11
Web Attack	Brute Force - XSS	2159
	Brute Force - Web	1507
	SQL Injection	21
DDoS	DDoS	128025
Brute-Force	FTP-Patator	7935
	SSH-Patator	5897
Bot	Bot	1956
PortScan	PortScan	158804

$\mathcal{F} = \{1, 5, 10, \ldots, 70\}$. For the supervised feature selection methods, we categorized the labels into two groups: "Benign" and "Attack", to form a binary classification framework. Subsequently, we trained our binary and simple multi-class models using the selected features on corresponding training data. These models were then tested on our designated testing set. According to the results depicted in Fig. 4, it is evident that PCA as a feature extraction algorithm exhibits the highest accuracy compared to different feature selection methods, particularly for both binary and multi-class classification tasks. Fig. 4 not only showcases the superior accuracy of PCA but also underscores the importance of feature selection on detection accuracy. Increasing the number of features generally enhances detection accuracy. However, despite its superior performance, PCA was not considered for our experiment due to its significant time overhead, as illustrated in Fig. 5. Unlike other methods where the chosen features can be readily applied through straightforward slicing operations, PCA necessitates transforming the original data into a new coordinate system via matrix multiplication. This process, inherently more computationally demanding, leads to increased processing time when applied to new data, as shown in Fig. 5 for extracting 10 features. Consequently, while PCA can effectively reduce complexity and retain important information, its slower computational speed compared to feature selection methods led us to exclude it from our experiment.

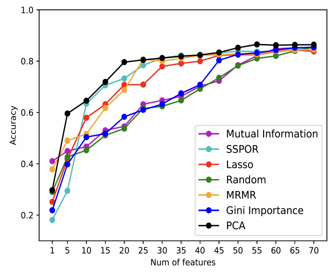
In addition, the quality of features is equally crucial in influencing detection accuracy. For instance, approximately 35 features are adequate to achieving satisfactory accuracy in binary classification, while a minimum of 50 features is necessary for acceptable accuracy in multi-class classification. This discrepancy highlights the greater complexity and the challenge associated with multi-class classification compared to binary classification. In the subsequent sections of the paper, we adopt the feature selection method that yields the highest accuracy for a given $\mathcal{F} = k$.

D. Binary Classification

Our approach is focused on multi-class classification, but we also aimed to assess Meta-IDS in a simpler case which is a binary classification with only two stages. For this purpose, we employed a logistic regression model with 10 features as a



(a) Binary Classification



(b) Multi-Class Classification

Fig. 4: Comparing methods of feature selection for Binary and Multi-class Classification.

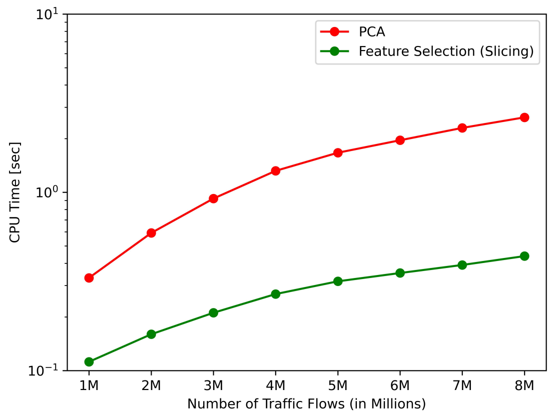


Fig. 5: Calculating CPU time for PCA.

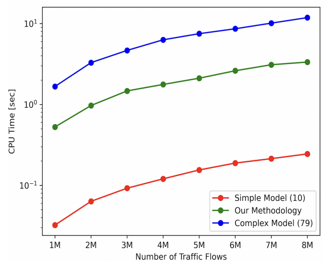
basic binary classifier, and a neural network with three hidden layers served as the more complex binary model. In this setup, if the probability predicted by the simpler model is below 0.8, those samples are then forwarded to the next stage for further evaluation by the complex binary model utilizing the full set of features. We define three models for comparison of the performance in binary classification as follows:

- *Simple Model*: A simple binary classification model with 10 features
- *Complex Model*: A complex binary classification with 79 features
- *Meta-IDS*: It starts with a binary classification with 10 features. Then suspicious traffic is forwarded to a complex binary classification with 79 features

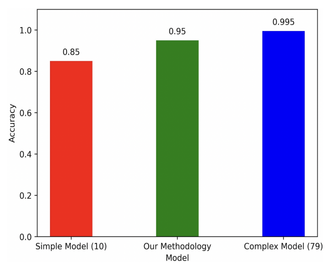
Fig. 6 illustrates the comparison of three different models on different scales of traffic, ranging from 1 million to 8 million instances. It shows the performance of models in the case of CPU time and accuracy in the context of binary classification. It is obvious that the proposed approach outperforms the *Simple Model* in terms of accuracy and the *Complex Model* in CPU time. This establishes an optimal balance, offering a desirable trade-off between accuracy and computational resource utilization.

E. Multi-class Classification

To evaluate our model, we consider three scenarios with different numbers of features in the first stage and second stage of our proposed method. These scenarios were selected



(a) CPU Time



(b) Accuracy

Fig. 6: Comparing three models for binary classification.

based on the varying levels of accuracy depicted in Fig. 4(b). Recognizing that simple models can exhibit varying degrees of accuracy in real-world applications, our approach adapts accordingly, selecting different thresholds to suit each model's accuracy level. The scenarios were:

- *Weak-Feature Scenario*: the number of features in *Stage 1* is $\mathcal{F}_1 = 5$ and the number of features in *Stage 2* is $\mathcal{F}_2 = 10$.
- *Moderate-Feature Scenario*: the number of features in *Stage 1* is $\mathcal{F}_1 = 10$ and the number of features in *Stage 2* is $\mathcal{F}_2 = 30$.
- *Heavy-Feature Scenario*: the number of features in *Stage 1* is $\mathcal{F}_1 = 40$ and the number of features in *Stage 2* is $\mathcal{F}_2 = 50$.

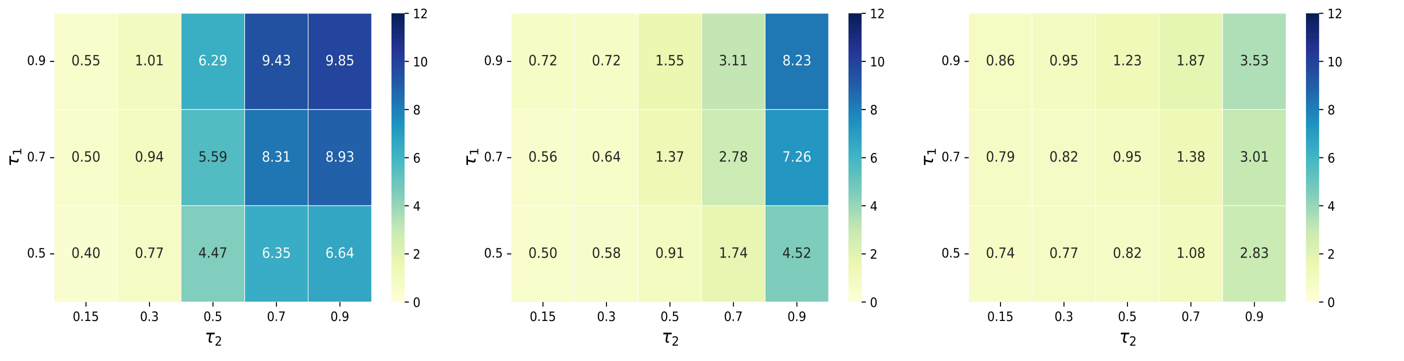
Given these scenarios, we select different values of τ_1 and τ_2 and obtain accuracy and CPU time on 8 million of testing data points for each case. With these results, we employ an optimization technique to determine the optimal values for thresholds τ_1 and τ_2, considering both CPU time and accuracy. We provide a formulation in Equation 1 to find the best τ_1 and τ_2 based on the expectation for CPU time and accuracy.

$$Utility(\tau_1, \tau_2) = \begin{cases} Score(\tau_1, \tau_2), & \text{if } Accuracy(\tau_1, \tau_2) > 0.8 \\ 0 & \text{otherwise} \end{cases}$$

where

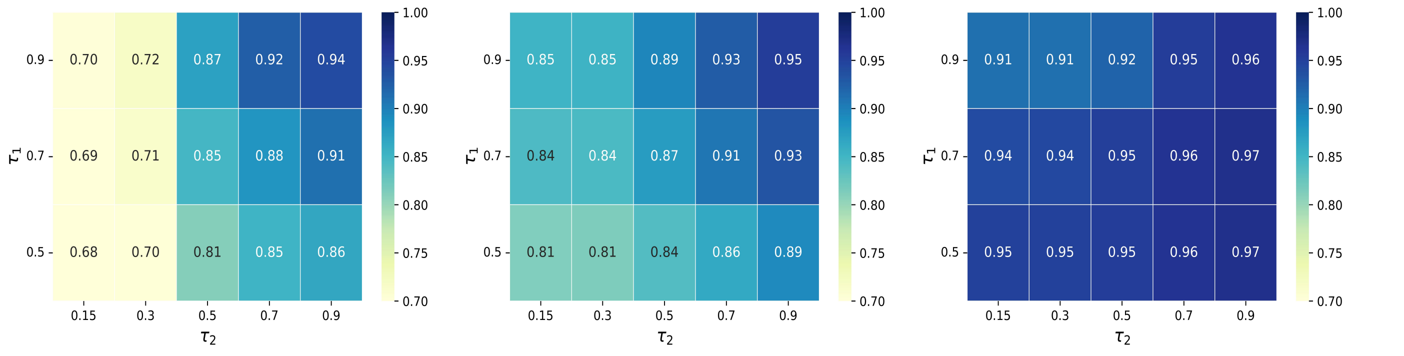
$$\begin{aligned} Score(\tau_1, \tau_2) =& (W_{ACC} \times Accuracy(\tau_1, \tau_2)) \\ & - (W_T \times Time(\tau_1, \tau_2)). \end{aligned} \tag{1}$$

Here, W_{ACC} represents the weight of accuracy, and W_T denotes the weight of CPU time. These two parameters assign importance to the accuracy and CPU time in the utility calculation, respectively. The $Utility$ function essentially measures the effectiveness of the threshold values in terms of achieving high accuracy and low computational costs. A higher utility score indicates a more desirable balance between accuracy and CPU time, guiding the selection of τ_1 and τ_2. Let us consider a scenario where an admin proposes that if we can enhance accuracy by 0.01, they are prepared to tolerate an additional 0.3 seconds of CPU time. Under such circumstances, we may set weights as follows: $W_{ACC} = 30$ and $W_T = 1$. After obtaining optimal thresholds for each scenario, we conduct a thorough comparison of the performance among three distinct methods the Simple multi-class model with $\mathcal{F}_2$ features, the proposed model (Meta-IDS), and the Complex model with 79 features.



(a) *Weak-Feature Scenario* (b) *Moderate-Feature Scenario* (c) *Heavy-Feature Scenario*

Fig. 7: CPU time (sec) for different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) on varying thresholds τ_1 and τ_2.



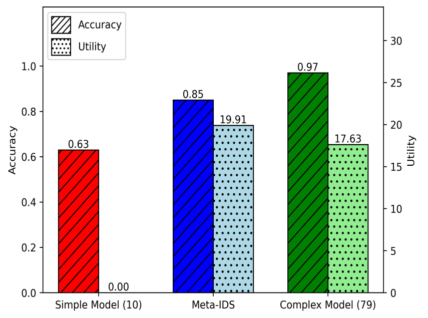
(a) *Weak-Feature Scenario* (b) *Moderate-Feature Scenario* (c) *Heavy-Feature Scenario*

Fig. 8: Accuracy for different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) on varying thresholds τ_1 and τ_2.

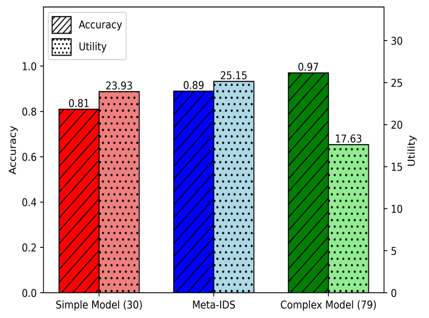
Impact of Different Values for τ_1 and τ_2 on Accuracy and CPU Time: As detailed in Section III, samples remaining after the first stage, with a classification probability lower than the threshold τ_1, are forwarded to the next stage. In *Stage 2*, the classification probability is compared with τ_2. Therefore, carefully selecting these thresholds is crucial, as highlighted in Figs. 7 and 8, to ensure accurate results. If the threshold τ_1 is set too high, numerous samples are forwarded to the next stage, resulting in increased computational resource consumption. Conversely, setting τ_1 too low may lead the first stage to classify actual intrusions as benign, if the binary model is not very accurate. The threshold τ_2 determines the confidence level of predictions by the attack classifier in the second stage. If set to a small value, only a few samples are forwarded to the third stage leading to low accuracy, if the model in *Stage 2* is not accurate enough. On the contrary, a higher τ_2 permits more samples to have further analysis in *Stage 3*, enhancing accuracy but at the cost of increased CPU time. Thus, the thresholds τ_1 and τ_2 balance computational efficiency and final classification performance. Setting them too high will lead to high CPU consumption while setting them too low may have unacceptable accuracy.

Comparison of Thresholds' Impact on Different Scenarios: In the previous section, we delved into the general impact of various thresholds on our methodology across all scenarios. In this section, our attention shifts to a detailed analysis of how these thresholds distinctly influence each individual scenario. It is noteworthy that we have designed the number of features in the first two stages so that the accuracy of the simple binary and multi-class models will be considered low for the *Weak-Feature Scenario*, moderate for the *Moderate-Feature Scenario*, and high for the *Heavy-Feature Scenario*. This design has significant implications, particularly evident when $\tau_2 > 0.5$. For $\tau_2 > 0.5$, the CPU time for *Heavy-Feature Scenario* is notably lower compared to the other scenarios. This efficiency stems from the increased confidence of the models in the first two stages (owing to their higher accuracy), resulting in fewer traffic flows progressing to *Stage 3*.

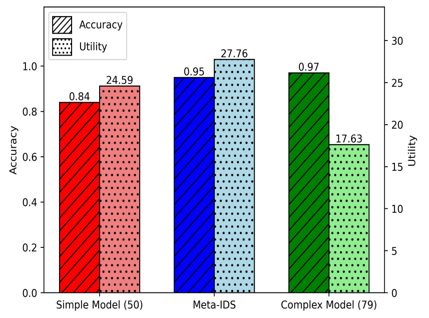
An intriguing observation in the *Heavy-Feature Scenario* is that a low threshold already yields high accuracy, unlike other scenarios. This outcome is attributed to the presence of more accurate models in the first two stages, which significantly enhances the overall performance. In the *Heavy-Feature Scenario*, an intriguing pattern emerges with small values of τ_2. For $\tau_2 < 0.7$, when τ_1 increases, there is a noticeable decrease in accuracy. Initially, this might seem counterintuitive. However, the underlying reason is that certain samples, while correctly predicted by the binary model, have a confidence



(a) *Weak-Feature Scenario*

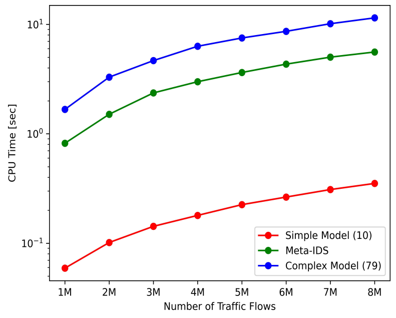


(b) *Moderate-Feature Scenario*

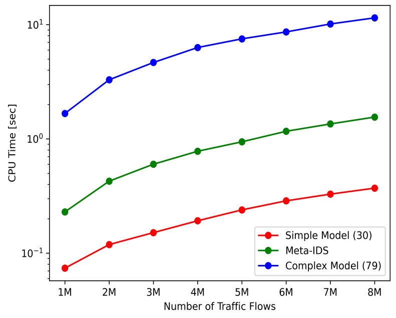


(c) *Heavy-Feature Scenario*

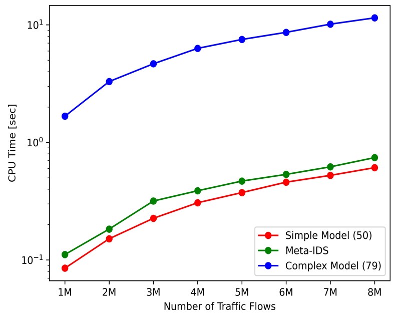
Fig. 9: Accuracy and Utility for the best thresholds in different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) in Figs. 7 and 8. Part (a) shows accuracy for *Weak-Feature Scenario* when $\tau_1 = 0.7$ and $\tau_2 = 0.5$. Part (b) shows accuracy for *Moderate-Feature Scenario* when $\tau_1 = 0.9$ and $\tau_2 = 0.5$. Part (c) shows accuracy for *Heavy-Feature Scenario* when $\tau_1 = 0.5$ and $\tau_2 = 0.15$. We considered $W_{ACC} = 30$.



(a) *Weak-Feature Scenario*



(b) *Moderate-Feature Scenario*



(c) *Heavy-Feature Scenario*

Fig. 10: CPU time for the best thresholds in different scenarios *Weak-Feature Scenario*:($\mathcal{F}_1 = 5$, $\mathcal{F}_2 = 10$), *Moderate-Feature Scenario*:($\mathcal{F}_1 = 10$, $\mathcal{F}_2 = 30$), and *Heavy-Feature Scenario*:($\mathcal{F}_1 = 40$, $\mathcal{F}_2 = 50$) in Figs. 7 and 8. Part (a) shows CPU time for *Weak-Feature Scenario* when $\tau_1 = 0.7$ and $\tau_2 = 0.5$. Part (b) shows CPU time for *Moderate-Feature Scenario* when $\tau_1 = 0.9$ and $\tau_2 = 0.5$. Part (c) shows CPU time for *Weak-Feature Scenario* when $\tau_1 = 0.5$ and $\tau_2 = 0.15$. We considered $W_{ACC} = 30$.

level lower than τ_1. Consequently, these samples are forwarded to the simple multi-class model, which misclassifies them. This misclassification can be attributed to the inherently more challenging nature of multi-class classification compared to binary classification as shown in Fig. 4. Nonetheless, as τ_2 increases, these same samples are routed to the complex model which compensates for this accuracy shortfall. This observation underscores the significance of each component in our three-stage methodology, particularly highlighting the crucial role played by the binary stage in enhancing the overall effectiveness of the process in this scenario.

Optimal Threshold for Each Scenario: For each scenario, the optimal threshold values were identified using the proposed utility function, as detailed in Table II for different values of W_{ACC}. In the *Heavy-Feature Scenario*, the notably high accuracy of the models in the first two stages contributed to achieving a superior utility value for each case. Additionally, with threshold values such as $(\tau_1 = 0.5, \tau_2 = 0.15)$ in this scenario with $W_{ACC} = 30$, all samples identified as benign in the first stage were not passed to the second stage, and no sample was analyzed with the *Stage 3* which is highly efficient. It is evident that as we increase the value of W_{ACC}, indicating a greater emphasis on accuracy, our optimizer consistently selects higher (or equal) thresholds in each scenario.

The CPU time, Accuracy, and Utility for the optimal case in each scenario for $W_{ACC} = 30$ is illustrated in Figs. 9 and 10. Notably, we expand our dataset by duplicating the 1 million testing data points to create datasets ranging from 2 to 8 million instances. As a result, the accuracy remains consistent across all considered scales of traffic flows. According to the results in each scenario, although the Simple model, which utilizes a minimal number of features, exhibits the lowest CPU time consumption, it performs the poorest in terms of accuracy. Moreover, the high accuracy achieved by the Complex model comes at the cost of substantial CPU time consumption. This highlights the necessity of carefully balancing model complexity and efficiency to ensure the effectiveness of intrusion detection systems. It becomes evident that our

TABLE II: Threshold and Utility Values for Different W_{ACC} Values and Scenarios when $W_T = 1$

$\mathbf{W_{ACC}}$	**Scenario**	τ_1	τ_2	**Utility**
6	*Weak-Feature*	0.5	0.5	0.39
	Moderate-Feature	0.7	0.15	4.49
	Heavy-Feature	0.5	0.15	4.98
30	*Weak-Feature*	0.7	0.5	19.85
	Moderate-Feature	0.9	0.5	25.29
	Heavy-Feature	0.5	0.15	27.88
60	*Weak-Feature*	0.9	0.9	46.67
	Moderate-Feature	0.9	0.7	52.53
	Heavy-Feature	0.5	0.7	56.63

proposed methodology adeptly navigates the intricate trade-off between CPU time and accuracy, offering a balanced approach to intrusion detection across diverse traffic scales. In this paper, we have formulated this trade-off in terms of Utility.

As illustrated in Figure 9, Meta-IDS demonstrates superiority in Utility across various scenarios. It is important to highlight that even a small improvement of 2 units in Utility holds significance. This is attributed to the specific relationship between accuracy and Utility in our context, where a 1% increase in accuracy equates to a 0.3 increment in Utility. Through this comparative analysis, we not only identify the strengths and weaknesses of each method but also highlight the efficacy of our proposed approach in optimizing resource utilization while maintaining high levels of accuracy in detecting network intrusions.

V. Conclusion

In conclusion, our hierarchical framework for intrusion detection, inspired by meta-computing principles, offers a promising approach to address the challenges in network security. Through three distinct stages equipped with unique capabilities and resource allocations, our framework effectively balances accuracy and computational efficiency. Through a series of experiments, we demonstrate the superior performance of our approach concerning CPU utilization and accuracy metrics in the binary and multi-class classification of traffic flows. Key to the success of our methodology is the careful selection of thresholds (τ_1 and τ_2) governing the transition between stages. Our analysis highlights the critical role of these thresholds in optimizing resource utilization while maintaining high levels of accuracy. Moreover, our comparative analysis across different scenarios demonstrates the versatility and effectiveness of our approach in diverse traffic environments. By leveraging a utility function to identify optimal threshold values, we achieve a desirable balance between accuracy and CPU time consumption.

Overall, our framework showcases promise in enhancing intrusion detection by efficiently utilizing computational resources while maintaining robust detection capabilities. Future research may focus on further optimizing threshold selection and exploring additional stages to enhance the framework's effectiveness in real-world network security applications.

References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[2] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, p. e4150, 2021.

[3] A. A. Ahmad, S. Boukari, A. M. Bello, and M. A. Muhammad, "A survey of intrusion detection techniques on software-defined networking," *Intl. Journal of Innovative Science and Research Technology*, 2021.

[4] A. Chen, Y. Fu, X. Zheng, and G. Lu, "An efficient network behavior anomaly detection using a hybrid dbn-lstm network," *Computers & Security*, vol. 114, p. 102600, 2022.

[5] W. Wei, H. Gu, W. Deng, Z. Xiao, and X. Ren, "Abl-tc: A lightweight design for network traffic classification empowered by deep learning," *Neurocomputing*, vol. 489, pp. 333–344, 2022.

[6] A. Yazdinejadna, R. M. Parizi, A. Dehghantanha, and M. S. Khan, "A kangaroo-based intrusion detection system on software-defined networks," *Computer Networks*, vol. 184, p. 107688, 2021.

[7] N. Niknami and J. Wu, "Enhancing load balancing by intrusion detection system chain on sdn data plane," in *Proc. of the IEEE Conf. on Communications and Network Security (CNS)*, 2022, pp. 264–272.

[8] R. Zhao, G. Gui, Z. Xue, J. Yin, T. Ohtsuki, B. Adebisi, and H. Gacanin, "A novel intrusion detection method based on lightweight neural network for internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9960–9972, 2021.

[9] Z. Wang, Z. Li, D. He, and S. Chan, "A lightweight approach for network intrusion detection in industrial cyber-physical systems based on knowledge distillation and deep metric learning," *Expert Systems with Applications*, vol. 206, p. 117671, 2022.

[10] M. Verkerken, L. D'hooge, D. Sudyana, Y.-D. Lin, T. Wauters, B. Volckaert, and F. De Turck, "A novel multi-stage approach for hierarchical intrusion detection," *IEEE Transactions on Network and Service Management*, 2023.

[11] S. Yang, X. Zheng, Z. Xu, and X. Wang, "A lightweight approach for network intrusion detection based on self-knowledge distillation," in *Proceeding of the IEEE International Conference on Communications (ICC)*, 2023, pp. 3000–3005.

[12] N. Hocine and C. Zitouni, "A multi-agent system based on dynamic load balancing for collaborative intrusion detection," in *Proceeding of the IEEE International Conference on Networking and Advanced Systems (ICNAS)*, 2023, pp. 1–6.

[13] W. Ge, Z. Cui, J. Wang, B. Tang, and X. Li, "Metacluster: a universal interpretable classification framework for cybersecurity," *IEEE Transactions on Information Forensics and Security*, pp. 1–1, 2024.

[14] X. Cheng, M. Xu, R. Pan, D. Yu, C. Wang, X. Xiao, and W. Lyu, "Meta computing," *IEEE Network*, 2023.

[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[16] Y. Saeys, I. Inza, and P. Larranaga, "A review of feature selection techniques in bioinformatics," *Bioinformatics*, vol. 23, no. 19, pp. 2507–2517, 2007.

[17] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[18] J. R. Vergara and P. A. Estévez, "A review of feature selection methods based on mutual information," *Neural Computing and Applications*, vol. 24, pp. 175–186, 2014.

[19] K. Manohar, B. W. Brunton, J. N. Kutz, and S. L. Brunton, "Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns," *IEEE Control Systems Magazine*, vol. 38, no. 3, pp. 63–86, 2018.

[20] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 58, no. 1, pp. 267–288, 1996.

[21] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[22] I. T. Jolliffe, *Principal Component Analysis for Special Types of Data*. Springer, 2002.