Received 19 February 2025, accepted 16 March 2025, date of publication 26 March 2025, date of current version 8 April 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3555201

RESEARCH ARTICLE

Protocol-Agnostic and Packet-Based Intrusion Detection Using a Multi-Layer Deep-Learning Architecture at the Network Edge

RODOLPHE PICOT[1], **FELIPE GOHRING DE MAGALHÃES**[1], **AHMAD SHAHNEJAT BUSHEHRI**[1], **MAROUA BEN ATTI**[2], **GABRIELA NICOLESCU**[1], **AND ALEJANDRO QUINTERO**[1], **(Senior Member, IEEE)**
[1]Department of Computer Engineering, École Polytechnique de Montréal, Montreal, QC H3T 0A3, Canada
[2]Humanitas Solutions, Montreal, QC H2W 1V1, Canada

Corresponding author: Rodolphe Picot (rodolphe-laurent-louis.picot@polymtl.ca)

ABSTRACT Intrusion Detection (ID) faces multiple challenges, including the diversity of intrusion types and the risk of false positives and negatives. In an edge computing context, resource constraints further complicate the process, particularly during the training phase, which is computationally intensive. This paper presents a novel approach to ID in network traffic within edge computing environments using a Neural Network (NN) model. The proposed model is designed to align with the layered structure of network packets and has been trained and evaluated on the widely used CIC-IDS2017 cybersecurity dataset. Its protocol-agnostic design and customized preprocessing method enable it to efficiently detect network attacks across multiple protocols while preserving the original packet structure. Unlike existing approaches that transform packets into alternative representations such as images or NLP-based techniques, which introduce additional overhead, our method processes packets directly, eliminating the need for complex components like Recurrent Neural Networks (RNNs) or convolutional layers. Our model is optimized for edge computing by employing a centralized training approach that minimizes resource consumption while allowing flexible deployment on edge devices. Experimental results demonstrate that our approach outperforms existing methods in terms of accuracy, F1-score, recall, and precision when evaluated on a real-world dataset. This work highlights the potential of deep learning in enhancing network security while respecting edge computing constraints.

INDEX TERMS Intrusion detection, edge computing, neural network, preprocessing, deep-learning

I. INTRODUCTION

The rapid expansion of the Internet has transformed global connectivity. It has enabled unprecedented communication between devices and driven innovation in various domains, such as commerce, healthcare, and entertainment. This rapid evolution, however, comes with specific challenges, particularly in the domain of cybersecurity. Malicious actors increasingly exploit vulnerabilities in network infrastructures, organizing sophisticated attacks that pose serious threats to private companies, public institutions, and other organizations. These attacks range from data breaches and system outages to physical infrastructure damage, with consequences from financial losses including reputational harm and operational disruptions [1].

In response to these threats, organizations must deploy robust mechanisms to analyze and identify malicious network traffic effectively. The domain dedicated to this critical task is known as ID [2]. The automation of this process is achieved through Intrusion Detection Systems (IDS), which aim to detect and mitigate threats as fast as possible, in real-time if possible. The ability of IDS to perform timely and accurate detections is vital to preserving the security and integrity of networks. However, this task is complex, as IDS must

The associate editor coordinating the review of this manuscript and approving it for publication was Ayaz Ahmad.

process and analyze vast amounts of network data, often in heterogeneous and dynamic environments. Traditional rule-based approaches, while effective in specific contexts, struggle with scalability and adaptability, leading to high rates of false positives and missed detections. To address these limitations, researchers have made benchmark datasets such as KDD-99 [3], NSL-KDD [4], and CIC-IDS2017 [5], which provide labeled samples of normal and malicious network traffic. These datasets often include traffic represented in formats such as raw data, CSV files, etc., serving as essential tools for evaluating and comparing different detection methods. The application of Machine Learning (ML) and Deep Learning (DL) techniques, including Support Vector Machines (SVM), k-Nearest Neighbors (KNN), Decision Trees (DT), Artificial Neural Networks (ANN), and Recurrent Neural Networks (RNN), demonstrated significant potential in improving the efficiency of IDS [6], [7], [8], [9].

Beyond computational efficiency, Adversarial Machine Learning (AML) introduces additional challenges for intrusion detection in edge computing. Attackers can manipulate network traffic to mislead models. To mitigate this risk, adversarial training techniques have been explored to enhance resilience [10], [11]. Additionally, unsupervised learning methods have been investigated to detect unknown threats without labeled data [12], [13]. These techniques leverage structural and temporal patterns to improve ID in resource-constrained environments.

Beyond detection, preprocessing plays an essential role in the development of IDS by transforming raw network traffic into structured inputs suitable for analysis. Despite its importance, many preprocessing methods fail to adhere to the hierarchical organization of network packets defined by the Open Systems Interconnection (OSI) model. The OSI model is a conceptual framework that divides network communication into seven distinct layers, each with specific functions and associated protocols. By aligning to this hierarchical structure in IDS design, it becomes possible to preserve the semantic and contextual relationships between layers, enabling more precise ID and improved interpretability, as demonstrated by our approach aligned with this architecture in processing network packets. Researchers often resort to simplifying transformations such as converting packets into images, aggregating data into flows [14], selecting a minimal set of features that are common across packet types [15], etc. These methods, while practical, often overlook the semantic and structural dependencies between protocol layers, potentially discarding critical contextual information. A preprocessing strategy that respects the OSI model's layered architecture can enhance both the interpretability and effectiveness of IDS, providing a more nuanced understanding of network behavior.

Overall, the paper presents the following key contributions:

- We introduce a novel deep learning-based approach for detecting network intrusions
- We design a model to reflect the layered structure of network traffic, ensuring compatibility with diverse protocols and enabling protocol-agnostic detection
- We evaluate the model's performance across multiple protocols, including File Transfer Protocol (FTP) and Secure Shell (SSH), using the CIC-IDS2017 dataset. It provides comprehensive raw traffic data, offering a more realistic representation of network activity that makes it an ideal choice for our study
- We propose a flexible deployment strategy that balances the need for computational efficiency with robust detection capabilities

The rest of the paper is organized as follows. Section II reviews the state-of-the-art in network intrusion detection, with a focus on preprocessing techniques (Section II-A) and detection methods suited to edge computing environments (Section II-B). Section III presents our proposed model, while Section IV provides a detailed account of our data and preprocessing methodology. In Section V, we discuss the experimental results. Finally, Section VI concludes the paper and outlines directions for future research.

II. STATE-OF-THE-ART

This section introduces the state-of-the-art in packet preprocessing and ID in edge computing. It presents the advantages and limitations of various existing methods, serving as a base for decisions and discussion throughout the paper.

A. PREPROCESSING

The first step in setting up a deep learning model is defining the data processing required to prepare it as input for the model. In the context of network ID at the protocol level, we need to implement processing capable of handling the fields of various protocols represented in network traffic data.

Different methods are available for preprocessing. First, manual preprocessing represents the most straightforward approach, involving the application of specific treatments to each field in each protocol. If the data is categorical, we apply one-hot encoding, whereas standardization or normalization may be used for continuous data [16]. This method offers the advantage of interpretability, but it lacks scalability since each field requires manual treatment.

Another method is image transformation, which consists of converting the binary representation of a packet into an image [17], [18], [19]. This approach enables the application of image processing techniques, such as Convolutional Neural Networks (CNNs). However, it loses the layered structure inherent to packet data. In some cases, transformation into streams is used, where methods work not on individual packets but rather on groups of packets organized into streams [20], [21], [22], [23]. These streams represent device communications over time. Tools like CICFlowMeter [24] can convert a Packet Capture (PCAP) file into a set of flows, providing contextual data for each packet. Unfortunately, much data is lost, as streams are often summaries of multiple packets. While information about the number of packets in each flow or the flow's duration is preserved, individual packet fields are not.

Finally, Natural Language Processing (NLP) techniques could be applied by modifying word2vec [25] for network packets [26], [27], [28]. Similar to image transformation, NLP methods tend to lose the structural information of the packet. Given these limitations, we propose a customized automatic method that retains the advantages of manual preprocessing while reducing the time and effort required to set it up. Once data preprocessing is defined, the next challenge in ID for edge computing is to design learning models capable of efficiently handling the preprocessed data within the resource constraints of edge devices. This transition from data preparation to model training is crucial, as effective preprocessing enhances the performance of machine learning models in limited-resource environments.

B. TRAINING CHALLENGES IN EDGE COMPUTING

ID in edge computing environments presents more challenges than in conventional systems, particularly due to the limited processing power of edge devices [29]. However, different approaches are available. For example, [30] uses Principal Component Analysis (PCA) to reduce data dimensionality and resource usage during the training step. Other works avoid deep learning and use classical machine-learning techniques for ID. For instance, [31] employs Random Forests, while [32] incorporates biologically inspired algorithms.

A growing trend to address the issue of limited processing power is the use of federated learning [33]. This approach distributes the training phase across devices, allowing each device to contribute with minimal computational power. However, federated learning introduces new challenges, such as issues with training distribution, device disconnection, and synchronization during the training phase [34].

To go further, there are two main configurations in machine and deep learning models for edge computing. In a centralized configuration, data is collected from devices and sent to a central server, where the model is trained and then distributed back to the devices for the inference phase. The training phase, which is resource-intensive in terms of CPU, GPU, memory, and battery, takes place on the server [35]. This configuration simplifies data management and the training process due to the central server [36], [37], but it has drawbacks, such as limited scalability, a single point of failure, and privacy concerns [38], [39].

In a federated configuration [40], more powerful devices can handle the training phase, and methods have been developed to distribute the training workload. This setup offers improved scalability and privacy but requires additional communication overhead, introduces system heterogeneity, and increases the complexity of model management [37], [41]. Additionally, federated learning requires more computational resources for training on edge devices [36], [42].

In this work, we deliberately chose not to integrate these complex techniques into our model. Instead, we aimed to simplify the problem and propose a model that remains effective while ensuring easy deployment in edge computing environments. Additionally, we focus on centralized environments to develop a model that can be seamlessly deployed across different edge computing scenarios. This configuration also reduces power consumption on edge devices. In many cases, edge devices return to a base for battery recharging, providing an ideal opportunity to update them with a pre-trained model.

Table 1 provides a summary of key techniques and their associated challenges discussed in this section. It offers a comprehensive overview of the State-of-the-art section, with a particular focus on edge computing environments, though the insights are also applicable to broader contexts.

III. MODEL

This section introduces a new model for detecting attacks on edge computing devices. This model is based on ID and functions as a binary classifier, capable of identifying malicious packets in network traffic. We have developed a supervised model that uses packet labels during the training phase. Furthermore, to ensure that our model remains protocol-agnostic, we implement a custom packet preprocessing method, as discussed in Section II-A. We propose this new preprocessing method in Section IV to address the limitations of existing methods.

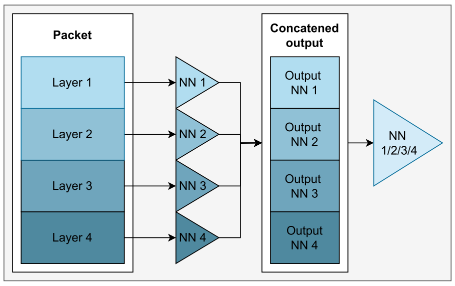


FIGURE 1. Example of a packet passing through the model, ensuring each protocol/layer maintains the same order in the internal representation when concatenating protocol NNs' outputs.

Our model is built on a new deep-learning architecture specifically designed to process network packets and consists of multiple NNs. Each network protocol (e.g., *ETH* or *IP*) and each protocol combination (e.g., *ETH*, *IP*, *TCP*, *HTTP*) is assigned its own NN. An illustration of this structure is provided in Fig. 1. Depending on the input packets, the model's architecture is dynamic. For each packet received, the model selects the NNs associated with the packet's protocols and the NN corresponding to the specific protocol combination. To train the model, as shown in Algorithm 1, we divide the packets based on protocol combinations and feed each protocol-specific NN with its respective data. For each protocol combination, we concatenate the

TABLE 1. Overview of the literature.

Category	Techniques	Limitations/Challenges	References
Preprocessing	Manual Preprocessing	Lacks scalability, labor-intensive	[16]
	Image Transformation	Loses structural information of packets	[17], [18], [19]
	Stream-based Transformation	Discards detailed packet-level data	[20], [21], [22], [23]
	NLP-Based Encoding	Overlooks structural dependencies	[25], [26], [27], [28]
Learning Models	Classical ML	Limited scalability for large datasets	[30], [31], [32]
	Deep Learning	High resource consumption	[42], [43]
	Federated Learning	Sync issues, privacy concerns	[33], [34], [37]
	Centralized Learning	Single point of failure, privacy risks	[35], [36], [37]

outputs of the protocol NNs and use these as input for the combination-specific NN. This structure is designed to carry out collaborative training across protocol combinations, create a specialized NN for each individual protocol, and develop a specialized NN for each protocol combination.

This architecture enables flexible and protocol-agnostic training while mitigating the curse of dimensionality. Using a single NN for all protocol fields would significantly increase input size, as each protocol has distinct fields, often resulting in sparse data. Sparse data, where not all fields are populated for each packet, presents challenges for NNs to handle effectively [43].

With this setup, it is clear that for having an protocol-agnostic model, the main challenge lies in the preprocessing. We discuss this preprocessing method further in Section IV.

Algorithm 1 provides details of the training process. Backpropagation occurs only when packet data has passed through all levels (i.e., protocol-specific NNs and combination-specific NNs). To use a combination NN, we concatenate the outputs from each protocol-specific NN. This design eliminates the need for complex NNs, as our model considers the structure of the data. By training all combinations sharing one or more protocols concurrently, we ensure that the NNs for protocols with less frequent occurrences are trained as effectively as those for more common protocols. In addition, we aim to keep the NNs simple to limit resource consumption on edge devices. Concerning the algorithm, several key components commonly used in DL are employed:

- Batch Normalization (BatchNorm1D [44]):

$$x_i^{norm} = \frac{x_i - \mu_B}{\sqrt{(\sigma_B)^2 + \epsilon}} \tag{1}$$

This process involves normalizing each input feature, with x_i^{norm} representing the normalized value for x_i, the i-th value of the batch, μ_B and σ_B are the batch mean and standard deviation, respectively. An arbitrarily small constant ϵ is added to ensure numerical stability. After this step, we apply a transformation step to obtain the layer output:

$$\hat{x}^i = \lambda \cdot x_i^{norm} + \beta \tag{2}$$

where λ and β are the hyperparameters of the BatchNorm1D layer. In this way, we standardize the inputs to a layer for each mini-batch, stabilizing the learning process and reducing the number of epochs required for training.

Algorithm 1 Training Algorithm

```
Require: Number of $epochs$, list of $combinations$, $dataset$,
    $model$
 1: for $epoch$ in $range(epochs)$ do
 2:     for $combination$ in $combinations$ do
 3:         for $X_{batch}, Y_{batch}$ in $dataset[combination]$ do
 4:             Adam.zero_grad()
 5:             $\hat{X} = \text{Array}()$
 6:
 7:             # prot NN part
 8:             for $X_{prot}$ in $X_{batch}$ do
 9:                 $X_{prot}^{norm} = \text{BatchNorm1D}(X_{prot})$
10:                 $NP = model.number\_layers\_prot$
11:                 for $i$ in $range(NP)$ do
12:                     $X_{prot}^{norm} = \text{Linear}(X_{prot}^{norm})$
13:                     $X_{prot}^{norm} = \text{ReLU}(X_{prot}^{norm})$
14:                 end for
15:                 $\hat{X}.\text{append}(X_{prot}^{norm})$
16:             end for
17:
18:             # combination NN part
19:             $\hat{X}_{concat} = \text{Concat}(\hat{X})$
20:             $NC = model.number\_layers\_combination$
21:             for $i$ in $range(NC)$ do
22:                 $\hat{X}_{concat} = \text{Linear}(\hat{X}_{concat})$
23:                 $\hat{X}_{concat} = \text{ReLU}(\hat{X}_{concat})$
24:             end for
25:             $\hat{X}_{concat} = \text{Linear}(\hat{X}_{concat})$
26:             $\hat{Y} = \text{Sigmoid}(\hat{X}_{concat})$
27:             $loss = \text{BCELoss}(\hat{Y}, Y_{batch})$
28:             backward($loss$)
29:             Adam.step()
30:         end for
31:     end for
32: end for
```

- Linear layer: This is the primary layer type used in NNs [45]:

$$Z = W^T \cdot X + B \tag{3}$$

where Z represents the output matrix, W is the layer weight matrix, X is the input matrix, and B is the bias vector.

- ReLU Activation: ReLU is applied as the activation function for all layers except the final layer in our model:

$$f(x) = \max(0, x) \tag{4}$$

  This activation function is simple, so it does not require much computation time.
- Sigmoid Activation: For the final layer, we use the sigmoid activation function, which is well-suited for binary classification tasks:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

We use the Adam optimization algorithm [46] as the optimizer due to its efficient, adaptive learning rate, which enhances convergence speed. With Adam, at each step t, we have:

$$\theta_t = \theta_{t-1} - \frac{\alpha * \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{6}$$

where

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{7}$$

In Equation 6, α is the learning rate. In Equation 7, β_1 and β_2 are the decay rates, while the terms m_t and v_t are the first and second-moment estimates and defined as:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \tag{8}$$

$$v_t = \beta 2 * v_{t-1} + (1 - \beta_2) * g_t^2 \tag{9}$$

and

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{10}$$

where $f(\theta)$ is the stochastic objective function with parameter θ representing the weight vector.

Binary Cross-Entropy is used as the loss function [47]. Its formula, presented in Equation 11, calculates the loss based on the binary indicator y and the predicted probability p for each observation, making it suitable for our binary classification model.

$$\mathbf{Loss} = -(y \log(p) + (1 - y)\log(1 - p)) \tag{11}$$

The protocol-specific NNs are designed to match the size required for their use as inputs to the combination NN. We consider the following hyper-parameters:

- Categorical Heuristic: defines the threshold at which a field can be considered categorical, as explained further in Section IV.
- Minimum Feature Percentage: Determines whether the presence of a feature is sufficient to retain it, otherwise it is removed from the dataset.
- Bridge Layer's Width: Specifies a fixed width for the protocol NN outputs, facilitating the calculation of the combination NN input size.
- Adam Optimizer Parameters: Includes the learning rate α, β_1 and β_2 for optimization adjustments.
- Epochs: Number of training iterations.

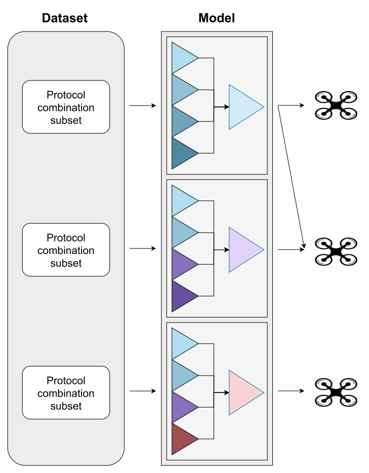

Dataset
Model
Protocol combination subset
Protocol combination subset
Protocol combination subset

FIGURE 2. The architecture of the proposed model.

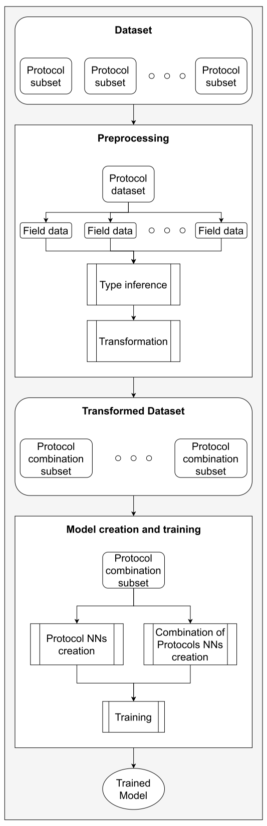


FIGURE 3. The general workflow of the proposed approach including the preprocessing step.

- Batch Size: Number of samples processed in each batch.

Fig. 2 illustrates the possible combinations of NNs that may be selected during the training or inference stages depending on the protocol combination in the dataset.

Protocol NNs are reused across different combinations whenever a protocol is part of a combination. It is indicated by the matching colors in the figure. Note that, protocol NNs follow the sequence determined by the OSI model [48], which reflects the hierarchical order of protocols within a packet. Combination NNs, on the other hand, are unique to each protocol set. For example, the NN for *ETH* is applied across the entire dataset, as all combinations start with this protocol. Conversely, the NN for *HTTP* is used only in cases where *HTTP* is present in the combination. This architecture promotes collaborative learning and thus enables simple NNs to adapt effectively to heterogeneous data, such as network packets.

Before processing packets in our model, we need to preprocess them as outlined in Fig. 3. First, we divide the dataset by protocol combinations to establish the associated NNs. This preliminary subsetting groups data fields from each protocol, as shown in the general workflow allowing us to preprocess each protocol's fields individually. Then, we group them according to protocol combinations. Finally, we can proceed with the training.

Our model can be deployed in part or as a whole, depending on the requirements of the target device. For example, Fig. 2, shows how each device, such as a drone, can incorporate only the model components needed to analyze its specific traffic patterns.

IV. DATA AND PREPROCESSING

In this subsection, we explain the rationale behind our choice of dataset. Due to a lack of datasets specifically designed for edge computing environments [31], we decided to use the CIC-IDS2017 dataset [5].

A. DATASET PRESENTATION

This dataset consists of raw network packets, making it ideal for analyzing communication patterns in edge networks. The CIC-IDS2017 dataset was created through simulations involving various attack scenarios such as $SSH - Patator$, and $FTP - Patator$. Figure 5 illustrates the architecture of the network used to generate the dataset, which comprises a sub-network of target machines (victims), a sub-network of attackers, and a connecting firewall. While this dataset is not explicitly tailored to edge computing, its structure and data characteristics align closely with real-world edge communications. Moreover, several types of attacks, such as *SSH*-based attacks, are highly relevant to edge computing scenarios.

The dataset spans five days of network traffic, each day dedicated to a specific attack type, except for Monday, which consists solely of normal traffic. Table 2 shows the distribution of attacks across the days. Since the dataset contains a significantly higher proportion of normal packets compared to malicious ones, we extracted an equal number of each type to create a balanced dataset for our experiments. To manage the large volume of data, we utilized a quarter of this balanced subset in our experiments.

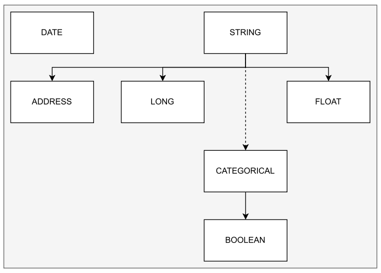


FIGURE 4. The architecture of the final data types.

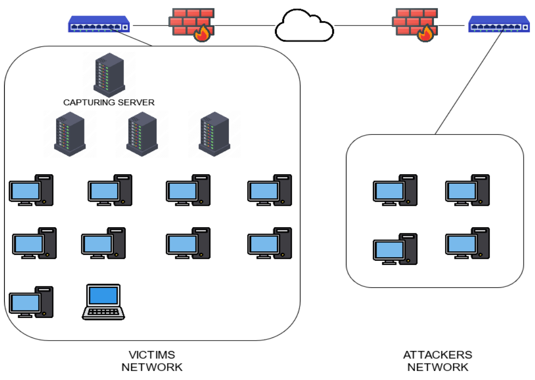


FIGURE 5. Network architecture used to generate the CIC-IDS2017 dataset.

TABLE 2. Description of the dataset traffic.

Day	Slot	Attack
Monday	Morning	Without Attack
	Afternoon	Without Attack
Tuesday	Morning	FTP-Patator (9:20am-10:20am)
	Afternoon	SSH-Patator (2:00pm-3:00pm)
Wednesday	Morning	DoS Slowloris (9:47am-10:10am)
		DoS Slowhttptest (10:14am-10:35am)
		DoS Hulk (10:43am-11:00am)
		DoS GoldenEye (11:10am-11:23am)
	Afternoon	Heartbleed Port 444 (3:12pm-3:32pm)
Thursday	Morning	Web Attack - Brute Force (9:20am-10:00am)
		Web Attack - XSS (10:15am-10:35am)
		Web Attack - SQL Injection (10:40am-10:42am)
	Afternoon	Dropbox infiltration (2:19pm, 2:20pm-2:21pm, 2:33pm-2:35pm, 3:04pm-3:45pm)
		Cool Disk - Infiltration (2:53pm-3:00pm)
Friday	Morning	Botnet ARES (10:02am-11:02am)
	Afternoon	DDoS LOIT (3:56pm-4:16pm)

B. PREPROCESSING METHOD FOR NETWORK PACKET DATA

To perform data preparation, we developed an automated method to preprocess each field of each protocol. We relied on Wireshark [49], to analyze network packets, as it is the most widely used, comprehensive, and easiest network analysis tool to integrate with Python, the language we used. Wireshark provides detailed metadata, including field names, descriptions, types, and the specific protocol version in which a field appears. Table 3 summarizes these metadata. For each field, we extracted values across all combinations of training subsets. Using these values and predefined heuristics, we determined how to preprocess fields into types such as categorical or continuous variables. We employed a hierarchical type determination system, illustrated in Fig. 4.

The fields defaulted to a string type if Wireshark provided insufficient information. Step-by-step discriminative rules then refined the type, except for the transition to the categorical type, which relied on a heuristic. Especially, if the number of unique elements in a field exceeded a certain threshold (a hyperparameter of our model), it was defined as noncategorical. The type assignment process begins by consulting the Wireshark documentation for each field, which provides detailed information such as field names, descriptions, and types. Based on this metadata, an initial type is assigned. For example, fields described as 'Unsigned integer, 8 bytes' in the documentation are initially classified as integers. Once this preliminary type is determined, refinement rules are applied to further specify the field's role in the data. For example, if only five unique values are observed in the dataset for a numerical field, it would be reclassified as a categorical variable. Finally, Wireshark's original types are mapped to the custom type hierarchy of the model, ensuring consistency and compatibility with the processing framework, as illustrated in Table 4. For example, if a field is described in the Wireshark documentation as "Unsigned integer, 2 bytes", it matches the "signed" pattern according to our rules. This field is then cast to the "Long" type in our system before being passed, along with its data, into the hierarchical processing method.

TABLE 3. Example of field information provided by Wireshark.

FIELD NAME	DESCRIPTION	TYPE	VERSIONS
eth.addr	Address	Ethernet or other MAC address	1.0 to 4.0
eth.addr.oui	Address OUI	Unsigned integer (3 bytes)	3.2 to 4.0
eth.addr.oui_resolved	Address OUI (resolved)	Character string	3.2 to 4.0
eth.addr_resolved	Address (resolved)	Character string	1.12 to 4.0
eth.dst	Destination	Ethernet or other MAC address	1.0 to 4.0
eth.dst.ig	IG bit	Boolean	3.2 to 4.0
eth.dst.lg	LG bit	Boolean	3.2 to 4.0
eth.dst.oui	Destination OUI	Unsigned integer (3 bytes)	3.2 to 4.0
eth.dst.oui_resolved	Destination OUI (resolved)	Character string	3.2 to 4.0
eth.dst_resolved	Destination (resolved)	Character string	1.12 to 4.0
eth.fcs	Frame-check sequence	Unsigned integer (4 bytes)	1.8 to 4.0
eth.fcs.status	FCS Status	Unsigned integer (1 byte)	2.2 to 4.0
eth.fcs_bad	Bad checksum	Label	1.8 to 4.0
eth.fcs_bad.expert	Expert Info	Label	1.12 to 2.0.1
eth.fcs_good	FCS Good	Boolean	1.8 to 2.0.16
eth.ig	IG bit	Boolean	1.0 to 4.0
eth.invalid_lentype	Invalid length/type	Unsigned integer (2 bytes)	1.8 to 4.0
eth.invalid_lentype.expert	Invalid length/type	Label	1.12.3 to 4.0
eth.len	Length	Unsigned integer (2 bytes)	1.0 to 4.0

As part of the preprocessing phase, two critical objectives were addressed: data filtering and memory optimization. Data filtering aimed to eliminate fields that were irrelevant for training purposes. This included fields that were overly correlated with labels, such as packet source fields in the CIC-IDS2017 dataset, where attacks were consistently associated with specific sources. Memory optimization, on the other hand, involved handling outlier fields (See Fig. 4) through deterministic methods combined with targeted exception-processing strategies to improve storage efficiency.

An additional challenge arose from the observation that not all protocol fields are consistently present in corresponding packets. Therefore, we had to choose between two solutions to address this: either set a default value for absent fields or introduce a new dimension to indicate field present. Assigning a default value was considered impractical, as arbitrary

TABLE 4. Mapping between Wireshark data types and custom internal types using pattern matching.

Type	Pattern	Wireshark Type
Long	Signed	Signed integer, 1 byte
		Signed integer, 2 bytes
		Signed integer, 3 bytes
		Signed integer, 4 bytes
		Signed integer, 8 bytes
		Unsigned integer, 1 byte
		Unsigned integer, 2 bytes
		Unsigned integer, 3 bytes
		Unsigned integer, 4 bytes
		Unsigned integer, 5 bytes
		Unsigned integer, 6 bytes
		Unsigned integer, 7 bytes
		Unsigned integer, 8 bytes
	Frame number	Frame number
	Time offset	Time offset
Address	Address	Ethernet or other MAC address
		IPv4 address
		IPv6 address
		EUI64 address
		AX.25 address
		VINES address
Float	Floating	Floating point (single-precision)
		Floating point (double-precision)
		IEEE-11073 Floating point (16-bit)
		IEEE-11073 Floating point (32-bit)
Label	Identifier	ASN.1 object identifier
		Globally Unique Identifier
		ASN.1 relative object identifier
	Label	Label
	Protocol	Protocol
	Character	Character, 1 byte
	OSI System-ID	OSI System-ID
	IPX network number	IPX network number
String	Fibre Channel WWN	Fibre Channel WWN
		EBCDIC character string
	Character string	Character string
	Sequence of bytes	Sequence of bytes
Date	Date	Date and time
Boolean	Boolean	Boolean

assignments (e.g., 0 or 1) might hold specific meanings for certain fields, leading to potential inaccuracies. Instead, a "shadow feature" dimension was introduced. Additionally, a new dimension was added for each field to prevent errors during inference. For example, if a variable was categorized as categorical during preprocessing, the observed values were recorded at that stage. However, due to data distribution differences between the training and test sets, an unknown value could appear in the test set, potentially causing an error. If the new dimension is set to 1, it indicates an issue with transforming this field for the given packet. Otherwise, it remains 0.

Our preprocessing approach offers significant advantages over existing methods. It is highly interpretable and preserves the hierarchical structure inherent to network packets, making it well-suited to the layered architecture of our model. Additionally, it limits the size of preprocessed data, particularly for categorical variables. For example, if a field is interpreted as categorical but contains thousands of unique values, its size could become unmanageably large and sparsely populated with many zeros. In such cases, our method allows the field to be reinterpreted as numerical, balancing interpretability with computational efficiency.

V. RESULTS & DISCUSSION

For our experiments, we extracted data from the CIC-IDS2017 dataset, obtaining 62029 packets. Table 5 details the breakdown by protocol combination and the corresponding attacks. These protocol combinations were selected based on their ability to balance normal and malicious packets while maintaining a sufficient number for model training. For example, the $ETH/IP/TCP$ protocol combination is the most common in our dataset, with 19,512 normal packets and 19,475 attack packets, including those associated with $FTP-Patator$, and $SSH-Patator$. Table 6 lists the features present in each protocol, while Table 7 presents the number of shadow features per protocol. The percentage of shadow features varies depending on the protocol. For instance, the IP protocol has 100% shadow fields, meaning that each field is missing in at least one instance within the IP layer of a packet. Not all features are present for every protocol due to our hyperparameter selection. If a feature lacks sufficient data for training, we exclude it to prevent excessive sparsity in both training and testing datasets.

TABLE 5. Characteristics of packets extracted from the CIC-IDS2017 dataset.

Protocols combination	# of Normal Packets	# of Attack Packets	Attack(s)
ETH/IP/TCP	19512	19475	Web attacks, FTP-Patator, SSH-Patator
ETH/IP/TCP/FTP	575	542	FTP-Patator
ETH/IP/TCP/HTTP	1827	1770	Web attacks
ETH/IP/TCP/SSH	9164	9194	SSH-Patator

To ensure the stability of our model, we conducted multiple training runs and fine-tuned the hyperparameters to achieve optimal performance. To do this, we used Optuna [50], a Python library for searching for hyperparameter optimization. Optuna allows the scores of all attempts to be stored in a database (in our case, a PostgreSQL database) and supports defining multiple objectives. We defined four objectives for optimization: Average Accuracy, F1-Score, Precision, and Recall. These metrics were used to evaluate the performance of our model during the hyperparameter tuning process.

We relied on resources provided by Compute Canada [51] to perform the optimization. Using Optuna, we systematically explored the hyperparameter space, running each trial on a $P100$ GPU and recording the results in a PostgreSQL database. This approach enabled us to achieve 99.99% efficiency across all objectives, as illustrated in Fig. 6a (accuracy), Fig. 6b (F1-score), Fig. 6c (precision), and Fig. 6d (recall). Our results show an improvement over previous works, as summarized in Table 8.

We achieved these results using the following hyperparameters:

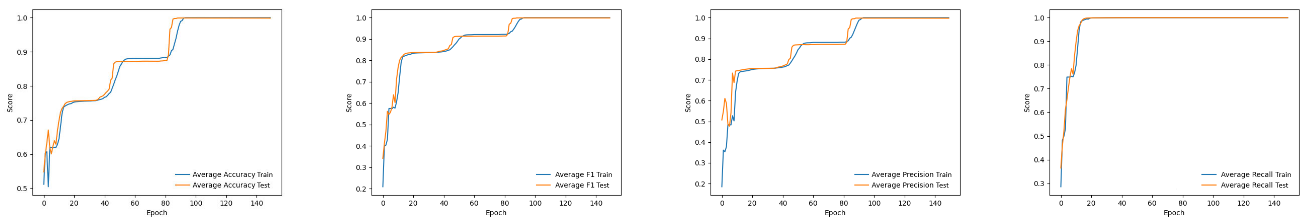
- Categorical Heuristic: 0.05
- Minimum Feature Percentage: 0.1
- Bridge Layer's Width: 90
- Adam Optimizer Parameters: Learning rate of 7.494×10^{-6}, β_1 of 0.6, and β_2 of 0.95
- Epochs: 150
- Batch Size: 50

Despite these achievements, there are variations in step-like patterns in Accuracy (Fig. 6a), F1-score (Fig. 6b), Precision (Fig. 6c) and Recall (Fig. 6d). These variations originate from the architecture of our model, which integrates multiple NNs. Each NN independently seeks to minimize its loss function, leading to local optimizations that contribute to the overall global minimum of the model. For example,

TABLE 6. Protocols' features details.

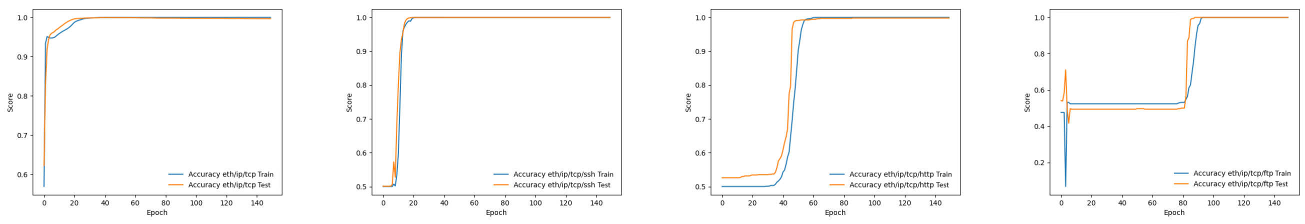
Protocol	Feature
ETH	dst.ig
	dst.lg
	ig
	lg
	padding
	src.ig
	src.lg
	type
FTP	request
	request.arg
	request.command
	response
	response.arg
	response.code
HTTP	accept
	accept_language
	bad_header_name
	connection
	prev_request_in
	referer
	request
	request.line
	request.version
	request_number
	user_agent
IP	checksum.status
	dsfield
	dsfield.dscp
	flags.df
	flags.mf
	flags.rb
	frag_offset
	hdr_len
	len
	proto
	ttl
	version
SSH	direction
	message_code
	packet_length
	packet_length_encrypted
	padding_length
	padding_string
TCP	ack_raw
	analysis
	analysis.ack_rtt
	analysis.bytes_in_flight
	analysis.initial_rtt
	checksum.status
	completeness
	connection.rst
	flags
	flags.ack
	flags.cwr
	flags.fin
	flags.push
	flags.res
	flags.reset
	flags.syn
	flags.urg
	hdr_len
	len
	options.mss
	options.mss_val
	options.timestamp.tsecr
	options.timestamp.tsval
	options.wscale.multiplier
	options.wscale.shift
	option_len
	port
	seq_raw
	time_delta
	time_relative
	urgent_pointer
	window_size
	window_size_scalefactor
	window_size_value

this behavior is reflected in Figs. 7a-7d for the accuracy metric across various protocol combinations. Each step-like pattern indicates a sub-model (e.g., an NN for protocol combinations or individual protocols) reaching its global maximum, which corresponds to a local maximum within



(a) Accuracy (b) F1-score (c) Precision (d) Recall

FIGURE 6. Performance metrics for the proposed model.



(a) ETH/IP/TCP (b) ETH/IP/TCP/SSH (c) ETH/IP/TCP/HTTP (d) ETH/IP/TCP/FTP

FIGURE 7. Accuracy results for various protocol configurations.

TABLE 7. Distribution of shadow features across protocols in the extracted data.

Protocol	# of fields	# of shadows fields	Ratio
ETH	8	7	87%
IP	12	12	100%
TCP	34	21	61.7%
FTP	6	2	33.34%
HTTP	11	0	0%
SSH	6	1	16.67%

TABLE 8. Performance comparison of the proposed model with other approaches.

Source	Accuracy (%)	F1 Score (%)	Precision (%)	Recall (%)
[54]	99.34	99.38	99.07	99.69
[55]	99.60	99.62	99.60	99.54
[56]	99.99	-	-	-
[57]	98.61	98.09	97.05	96.95
[58]	99.99	99.90	100	99.80
Our Model	**99.99**	**99.99**	**99.99**	**99.99**

the overall model. Minor (local) instabilities, such as those in Fig. 6d, are inherent to NN training. It can result from short-term fluctuations during the learning process.

VI. CONCLUSION

In this paper, we introduced a novel preprocessing method and a new deep-learning structure. The proposed model enhances the management of diverse protocols within heterogeneous networks. This approach leverages the inherent nature of packets to achieve superior performance, including 99.99% accuracy on the CIC-IDS2017 dataset, over-performing previous works. Furthermore, our approach demonstrates the potential for reducing on-board energy consumption while enabling a maximum number of devices to use only portions of the trained model, which can be hosted on a more powerful server.

Regarding scalability, our model maintains a modular architecture, enabling seamless adaptation as network complexity increases. Instead of requiring larger NNs, additional protocol-specific NNs can be integrated to accommodate new protocols and protocol combinations. This flexibility ensures that the model adapts efficiently to evolving network configurations without introducing unnecessary complexity. While training time may increase as the data volume expands, this is a standard tradeoff for most models. During inference, devices that do not rely on the newly added components will not experience additional resource consumption, as they will process only the parts relevant to their specific traffic patterns.

Future optimizations could involve model compression techniques such as pruning or quantization to further improve efficiency on resource-constrained edge devices. Although our model has demonstrated strong performance in ID, further research is needed to assess its resilience against adversarial attacks, such as evasion tactics or manipulated network packets. Addressing these challenges will be essential for ensuring robust security in real-world deployments. Future work could explore adversarial training, intrusion explanation methods, and unsupervised learning to enhance the model's resistance to sophisticated cyber threats and its adaptability to unlabeled data. Additionally, investigating training time, inference time, and memory usage, particularly for resource-constrained edge computing devices, could provide valuable insights into the practical feasibility of deploying our approach in real-world IoT environments.

REFERENCES

[1] S. Romanosky, "Examining the costs and causes of cyber incidents," *J. Cybersecur.*, vol. 2, no. 2, pp. 121–135, Aug. 2016, Art. no. tyw001, doi: 10.1093/cybsec/tyw001.

[2] H.-J. Liao, C. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *J. Netw. Comput. Appl.*, vol. 36, no. 1, pp. 16–24, Sep. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804512001944

[3] (2007). *KDD Cup 1999 Dataset*. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[4] R. Zhao, "NSL-KDD," Tech. Rep., 2022, doi: 10.21227/8rpg-qt98.

[5] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. 4th Int. Conf. Inf. Syst. Secur. Privacy*, Jun. 2018, pp. 1–12.

[6] S. Wang, J. F. Balarezo, S. Kandeepan, A. Al-Hourani, K. G. Chavez, and B. Rubinstein, "Machine learning in network anomaly detection: A survey," *IEEE Access*, vol. 9, pp. 152379–152396, 2021.

[7] M. Macas, C. Wu, and W. Fuertes, "A survey on deep learning for cybersecurity: Progress, challenges, and opportunities," *Comput. Netw.*, vol. 212, Jul. 2022, Art. no. 109032. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622001864

[8] Y. Wu, D. Wei, and J. Feng, "Network attacks detection methods based on deep learning techniques: A survey," *Secur. Commun. Netw.*, vol. 2020, pp. 1–17, Aug. 2020.

[9] M. Pawlicki, R. Kozik, and M. Choraś, "A survey on neural networks for (cyber-) security and (cyber-) security of neural networks," *Neurocomputing*, vol. 500, pp. 1075–1087, Aug. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222007184

[10] K. He, D. D. Kim, and M. R. Asghar, "Adversarial machine learning for network intrusion detection systems: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 538–566, 1st Quart., 2023.

[11] A. Alotaibi and M. A. Rassam, "Adversarial machine learning attacks against intrusion detection systems: A survey on strategies and defense," *Future Internet*, vol. 15, no. 2, p. 62, Jan. 2023.

[12] W. Wang, S. Jian, Y. Tan, Q. Wu, and C. Huang, "Robust unsupervised network intrusion detection with self-supervised masked context reconstruction," *Comput. Secur.*, vol. 128, May 2023, Art. no. 103131. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740482300041X

[13] P. F. de Araujo-Filho, M. Naili, G. Kaddoum, E. T. Fapi, and Z. Zhu, "Unsupervised GAN-based intrusion detection system using temporal convolutional networks and self-attention," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 4, pp. 4951–4963, Apr. 2023.

[14] W. Jo, S. Kim, C. Lee, and T. Shon, "Packet preprocessing in CNN-based network intrusion detection system," *Electronics*, vol. 9, no. 7, p. 1151, Jul. 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/7/1151

[15] Y. Lyu, Y. Feng, and K. Sakurai, "A survey on feature selection techniques based on filtering methods for cyber attack detection," *Information*, vol. 14, no. 3, p. 191, Mar. 2023. [Online]. Available: https://www.mdpi.com/2078-2489/14/3/191

[16] M. K. Hooshmand and D. Hosahalli, "Network anomaly detection using deep learning techniques," *CAAI Trans. Intell. Technol.*, vol. 7, no. 2, pp. 228–243, Jun. 2022.

[17] G. Bendiab, S. Shiaeles, A. Alruban, and N. Kolokotronis, "IoT malware network traffic classification using visual representation and deep learning," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2020, pp. 444–449.

[18] T. Kim, S. C. Suh, H. Kim, J. Kim, and J. Kim, "An encoding technique for CNN-based network anomaly detection," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2018, pp. 2960–2965.

[19] S. Garg, K. Kaur, N. Kumar, G. Kaddoum, A. Y. Zomaya, and R. Ranjan, "A hybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 3, pp. 924–935, Sep. 2019.

[20] R.-H. Hwang, M.-C. Peng, C.-W. Huang, P.-C. Lin, and V.-L. Nguyen, "An unsupervised deep learning model for early network traffic anomaly detection," *IEEE Access*, vol. 8, pp. 30387–30399, 2020.

[21] N. Rust-Nguyen and M. Stamp, "Darknet traffic classification and adversarial attacks," 2022, *arXiv:2206.06371*.

[22] M. Turcaník and J. Baráth, "Intrusion detection by artificial neural networks," in *Proc. New Trends Signal Process. (NTSP)*, Oct. 2022, pp. 1–6.

[23] N. Yadav, L. Truong, E. Troja, and M. Aliasgari, "Machine learning architecture for signature-based IoT intrusion detection in smart energy grids," in *Proc. IEEE 21st Medit. Electrotechnical Conf. (MELECON)*, Jun. 2022, pp. 671–676.

[24] A. H. Lashkari, "Cicflowmeter-v4. 0 (formerly known as iscxflowmeter) is a network traffic bi-flow generator and analyser for anomaly detection," Can. Inst. Cyber Secur., Fredericton, NB, Canada, Tech. Rep., 2019.

[25] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., Red Hook, NY, USA: Curran Associates, Dec. 2013, pp. 3111–3119.

[26] E. L. Goodman, C. Zimmerman, and C. Hudson, "Packet2 Vec: Utilizing Word2 Vec for feature extraction in packet data," 2020, *arXiv:2004.14477*.

[27] M. Hassan, M. E. Haque, M. E. Tozal, V. Raghavan, and R. Agrawal, "Intrusion detection using payload embeddings," *IEEE Access*, vol. 10, pp. 4015–4030, 2022.

[28] R. Bar and C. Hajaj, "SimCSE for encrypted traffic detection and zero-day attack detection," *IEEE Access*, vol. 10, pp. 56952–56960, 2022.

[29] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Gener. Comput. Syst.*, vol. 97, pp. 219–235, Feb. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18319903

[30] 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 Things J.*, vol. 9, no. 12, pp. 9960–9972, Jun. 2022.

[31] V. U. Ihekoronye, S. O. Ajakwe, D.-S. Kim, and J. M. Lee, "Cyber edge intelligent intrusion detection framework for UAV network based on random forest algorithm," in *Proc. 13th Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2022, pp. 1242–1247.

[32] H. Bangui and B. Buhnova, "Lightweight intrusion detection for edge computing networks using deep forest and bio-inspired algorithms," *Comput. Electr. Eng.*, vol. 100, May 2022, Art. no. 107901. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790622001859

[33] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.

[34] J. Ding, E. Tramel, A. K. Sahu, S. Wu, S. Avestimehr, and T. Zhang, "Federated learning challenges and opportunities: An outlook," in *Proc. ICASSP - IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2022, pp. 8752–8756. [Online]. Available: https://www.amazon.science/publications/federated-learning-challenges-and-opportunities-an-outlook

[35] G. Drainakis, P. Pantazopoulos, K. V. Katsaros, V. Sourlas, A. Amditis, and D. I. Kaklamani, "From centralized to federated learning: Exploring performance and end-to-end resource consumption," *Comput. Netw.*, vol. 225, Apr. 2023, Art. no. 109657. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128623001020

[36] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[37] G. Bao and P. Guo, "Federated learning in cloud-edge collaborative architecture: Key technologies, applications and challenges," *J. Cloud Comput.*, vol. 11, no. 1, p. 94, Dec. 2022.

[38] S. Bharati, M. R. H. Mondal, P. Podder, and V. B. S. Prasath, "Federated learning: Applications, challenges and future directions," *Int. J. Hybrid Intell. Syst.*, vol. 18, nos. 1–2, pp. 19–35, May 2022, doi: 10.3233/his-220006.

[39] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: Challenges and applications," *Int. J. Mach. Learn. Cybern.*, vol. 14, no. 2, pp. 513–535, Nov. 2022.

[40] M. Abdel-Basset, N. Moustafa, and H. Hawash, "Introducing federated learning for Internet of Things (IoT)," in *Deep Learning Approaches for Security Threats in IoT Environments*, 2023, pp. 317–336.

[41] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," Tech. Rep., 2023.

[42] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, P. Zeng, M. Khan, and S. K. Das, "Edge-computing-driven Internet of Things: A survey," *ACM Comput. Surv.*, vol. 55, no. 8, pp. 1–41, Dec. 2022, doi: 10.1145/3555308.

[43] K. Poulinakis, D. Drikakis, I. W. Kokkinakis, and S. M. Spottswood, "Machine-learning methods on noisy and sparse data," *Mathematics*, vol. 11, no. 1, p. 236, Jan. 2023. [Online]. Available: https://www.mdpi.com/2227-7390/11/1/236

[44] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[45] S. Amari, "A theory of adaptive pattern classifiers," *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 3, pp. 299–307, Jun. 1967.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[47] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Ann. Data Sci.*, vol. 9, no. 2, pp. 187–212, Apr. 2022.

[48] J. D. Day and H. Zimmermann, "The OSI reference model," *Proc. IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.

[49] *Wireshark.org*. [Online]. Available: https://www.wireshark.org/

[50] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2623–2631.

[51] S. Baldwin, "Compute canada: Advancing computational research," *J. Phys., Conf. Ser.*, vol. 341, Feb. 2012, Art. no. 012001.

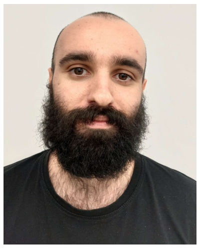
[52] A. S. Khan, Z. Ahmad, J. Abdullah, and F. Ahmad, "A spectrogram image-based network anomaly detection system using deep convolutional neural network," *IEEE Access*, vol. 9, pp. 87079–87093, 2021.

[53] A. Halbouni, T. S. Gunawan, M. H. Habaebi, M. Halbouni, M. Kartiwi, and R. Ahmad, "CNN-LSTM: Hybrid deep neural network for network intrusion detection system," *IEEE Access*, vol. 10, pp. 99837–99849, 2022.

[54] O. Faker and E. Dogdu, "Intrusion detection using big data and deep learning techniques," in *Proc. ACM Southeast Conf.*, Apr. 2019, pp. 86–93.

[55] J. Jose and D. V. Jose, "Deep learning algorithms for intrusion detection systems in Internet of Things using CIC-IDS 2017 dataset," *Int. J. Electr. Comput. Eng.*, vol. 13, no. 1, p. 1134, Feb. 2023.

[56] G. Agrafiotis, E. Makri, I. Flionis, A. Lalas, K. Votis, and D. Tzovaras, "Image-based neural network models for malware traffic classification using PCAP to picture conversion," in *Proc. 17th Int. Conf. Availability, Rel. Secur.*, Aug. 2022, pp. 1–7.



RODOLPHE PICOT received the bachelor's degree in computer sciences from Université Paris-Sud 11 (now Université Paris-Saclay), in 2016, the M.Eng. degree in cybersecurity from Sorbonne Université, in 2018, and the M.Sc.A. degree in machine learning from the École Polytechnique de Montréal, in 2023, where he is currently pursuing the Ph.D. degree with the HES Laboratory. His research interests include deep learning, cybersecurity, and unmanned autonomous vehicles.

FELIPE GOHRING DE MAGALHÃES received the double Ph.D. degree in computer engineering and computer science from École Polytechnique and PUCRS, in 2017. He is currently an Assistant Professor with École Polytechnique de Montréal. He has authored and co-authored around 40 papers and articles and two book chapters. His research interests include integrated silicon photonics, real-time systems, avionics systems, cybersecurity, implementation of multi-processor architectures, and intra-chip communication infrastructure.

AHMAD SHAHNEJAT BUSHEHRI received the Master of Science degree in telematics engineering from the Politecnico di Torino, and the Ph.D. degree in industrial engineering from Polytechnique Montréal, in 2024. He is currently an Associate Researcher with the Heterogeneous Embedded Systems Laboratory (HESL), Polytechnique Montréal. His research interests include binary instrumentation, cybersecurity, system tracing, and energy optimization in computer systems.

MAROUA BEN ATTI received the B.Eng. degree (Hons.) in telecommunications engineering from the National Institute of Applied Sciences and Technology, in 2013, and the M.Sc. degree (Hons.) in information technology engineering and the Ph.D. degree (Hons.) in computer science and engineering from the École de Technologie Supérieure (ÉTS), University of Quebec, Montreal, QC, Canada, in 2015. She is leading in research and development activities as the Research and Director of Humanitas Solutions. Her research interests include network optimization, the IoT, cloud computing, cybersecurity, machine learning, and smart city.

GABRIELA NICOLESCU is currently a Professor and the Director of the Department of Computer and Software Engineering, Polytechnique Montréal. She has published more than 240 papers in international conferences and journals. Her research interests include design methods for secure and efficient current and future architectures of embedded systems. She is very active in the field of system-level design methods, modeling, simulation, and profiling as well as security by design. She works in collaboration with several industrial partners and academics.

ALEJANDRO QUINTERO (Senior Member, IEEE) received the Engineering degree in computer engineering from the Universidad de los Andes, Colombia, in 1983, the Diploma degree in advanced studies from INPG Grenoble, in June 1989, and the Ph.D. degree in computer engineering from Université Joseph Fourier, Grenoble, France, in 1993. He is currently a Full Professor with the Department of Computer Engineering, Polytechnique Montréal, Canada. He has co-authored one book and more than 60 other technical publications, including journals and proceedings papers. His research interests include services and applications related to mobile computing, network security, network infrastructures, and next-generation mobile networks.