Edge-Centric Security Framework for Electric Vehicle Connectivity: A Deep Learning Approach

Koustav Kumar Mondal
IDRP - IoT and Application
IIT Jodhpur
Rajasthan, India
mondal.4@iitj.ac.in

Divya Mahendia
Dept. of CSE
IIT Jodhpur
Rajasthan, India
m22cs006@iitj.ac.in

Debasis Das
Dept. of CSE
IIT Jodhpur
Rajasthan, India
debasis@iitj.ac.in

Sumit Kalra
Dept. of CSE & IDRP - IoT
IIT Jodhpur
Rajasthan, India
sumitk@iitj.ac.in

2023 28th Asia Pacific Conference on Communications (APCC) | 979-8-3503-8261-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/APCC60132.2023.10460656

***Abstract*—As connectivity in electric vehicles (EVs) expands, so does their vulnerability to cyber threats within the Internet of Vehicles (IoV). This study introduces a potent Intrusion Detection System (IDS) that leverages distributed edge computing, Convolutional Neural Networks (CNNs), and ensemble techniques to address this concern in EV systems. Performance evaluations on reputable IoV security datasets have demonstrated this IDS's efficacy, delivering detection rates and F1-scores 100%. These results affirm the system's potential to significantly enhance the cybersecurity of both internal and external vehicular networks in the context of connected EVs.**

***Index Terms*—Distributed Edge Computing, Convolutional Neural Networks (CNNs), Intrusion Detection System (IDS), Electric Vehicles (EVs), Internet of Vehicles (IoV), Vehicular Networks**

I. Introduction

The environmental advantages, technical developments, and rising public interest in Electric Vehicles (EVs) have all contributed to the industry's meteoric rise in recent years [1]. Electric vehicle (EV) integration with the IoT and Internet of Vehicles (IoV) [2] has the potential to revolutionize transportation and open the door to a host of new intelligent services [3]. However, since cyber-attacks may have serious ramifications for vehicle safety and functioning, guaranteeing the security of these networked systems has become more vital [4]. A distributed edge computing-based advanced intrusion detection system (IDS) framework is proposed in this paper to solve the issues with protecting electric vehicle networks.

A. Background & Motivation

Modern EVs have evolved into network-controlled vehicles, including Autonomous Electric Vehicles (AEVs) and Connected Electric Vehicles (CEVs) [5], thanks to the rapid development of IoT and IoV [2] technologies. Intra-vehicle networks (IVNs) and external networks make up the IoV [2] ecosystem, with IVNs using the Controller Area Network (CAN) bus to allow communication between electronic control units (ECUs) [6]. In contrast, external vehicular networks link diverse IoV [2] entities, such as roadside equipment, infrastructure, and road users, to smart electric cars.

The cyber-attack surface of contemporary electric cars has grown as a result of vehicular networks, notwithstanding the many advantages of such networks [7]. Due to their short length, CAN packets cannot accommodate authentication or encryption during processing [5]. Therefore, adversaries may launch Danial of Service (DoS), fuzzy, and spoofing attacks by injecting malicious messages into IVNs. In addition, the cellular links between EVs and external networks leave the cars vulnerable to the most common types of cyber attacks [8]. As a result, Intrusion Detection Systems (IDSs) are increasingly important for safeguarding IoV [2] infrastructure and smart electric cars against cyberattacks [9].

B. Problem Statement & Contributions

Researchers and automakers are interested in using recent developments in Machine Learning (ML) and Deep Learning (DL) to the fields of cybersecurity and electric car systems [10]. Classifier-based IDSs, which analyze network traffic data to distinguish between legitimate network activity and malicious cyberattacks, are often developed with the use of ML and DL methods [11]. To better protect IoV [2] infrastructure in the context of EVs, this research proposes a distributed Edge computing-based intelligent IDS framework that makes use of optimized Convolutional Neural Networks (CNNs), transfer learning, and ensemble learning [12]. To address those problems this paper presents several key contributions:

1) Our approach introduces a cutting-edge distributed edge framework specifically designed for detecting cyber-attacks in intra-vehicle and external networks of electric vehicles. This framework harnesses the power of Convolutional Neural Networks (CNNs), transfer learning, ensemble learning, and hyperparameter optimization techniques
2) A novel data transformation approach is proposed, adept at converting electric vehicle network traffic data into image form, thereby facilitating the discernment of various cyber-attack patterns more efficiently.
3) The proposed method's efficacy is scrutinized using one benchmark cybersecurity datasets, which encompass intra-vehicle and external network data for electric vehicles. The performance is then juxtaposed with other leading methods in the field.

C. Paper Organization

The structure of this paper is organized as follows: Section II reviews related studies on ML and DL in vehicular network intrusion detection. Section III gives a system overview. Section IV and Section V give information about our proposed distributed framework, including data transformation, CNN, and ensemble learning. In Section VI, experimental results are presented and discussed. Finally, Section VII offers a conclusion summarizing the paper.

Authorized licensed use limited to: Indian Institute of Technology - Jodhpur. Downloaded on January 09,2025 at 13:33:57 UTC from IEEE Xplore. Restrictions apply.

II. Brief Literature Review

The rising interest in creating Intrusion Detection Systems (IDS) for EVs is a direct result of the rising popularity of EVs in recent years. Machine learning (ML) and deep learning (DL) models have found extensive use in this area. A Deep Learning (DL)-based IDS based on a Multi-Layer Perceptron (MLP) model was suggested by Rosay et al. [8]. The CICIDS2017 dataset [13] was used to test the model on an automotive microprocessor. Similarly, Yang et al. [6] proposed a tree-based stacking technique that showed excellent performance on the IoV [2] and CICIDS2017 datasets [13]for EV network traffic analysis. There have been a number of additional research that has concentrated on developing IDSs using Convolutional Neural Networks (CNNs) for EVs. For instance, the PLeNet approach for in-vehicle network intrusion detection was developed by Mehedi et al. [5], and it earned a high F1-score of 97.8 percent on the Car-Hacking dataset. Using the success of 1D-CNN models in time-series data analytics, Hossain et al. [9] presented a 1D-CNN based IDS for intra-vehicle intrusion detection. A deep convolutional neural network (DCNN) based intrusion detection system (IDS) model using reduced InceptionResnet was also suggested by Song et al. [14]. On the Car-Hacking dataset, the DCNN model performed very well.

While these approaches have shown promise in EV intrusion detection tasks, further work is needed to improve their accuracy and adaptability. Therefore, we offer an ideal IDS framework for EVs by combining distributed edge computing, ensemble learning [12], and hyperparameter optimization (HPO) [15] methodologies to fine-tune state-of-the-art CNN models. To further improve the effectiveness of model training, we also use transfer learning approaches. Our suggested approach is meant to enhance the reliability and precision of EV infiltration detection, making the roads safer for everyone.

III. System Overview

The purpose of this research is to develop a distributed edge computing-based advanced Intrusion Detection System (IDS) capable of detecting and mitigating a wide range of cyber-attacks in Electric Vehicle (EV) intra-vehicle and external vehicular networks. The proposed IDS framework is designed to address the vulnerability of Electric Vehicle Communication Standards (EVCS) interfaces to internal attacks, and wireless interfaces to external attacks, which can be used by cyber attackers to launch malicious traffic packets.

The proposed system, depicted in Fig. 1, revolves around an intelligent network that comprises a trio of Electric Vehicles (EV1, EV2, EV3), a pair of charging stations (CS1, CS2), and two Traffic Lights (TR1, TR2). These latter entities - the charging stations and traffic lights - represent Roadside Units (RSUs). The RSUs embody a distributed edge layer where a Convolutional Neural Network (CNN)-based pipeline is deployed to detect potential cyber attacks. The central function of the CNN-based pipeline is to classify, detect, and subsequently improve its detection capabilities based on accumulated data. To ensure a perpetual learning environment for the CNN, we employ a Data Generation Layer, embodied by the Electric Vehicles (EV1, EV2, EV3). This layer is responsible for producing new data used to refine the CNN model, thereby improving its attack detection capabilities. Inter-vehicle communication is facilitated by a Vehicle-to-Vehicle (V2V) communication protocol, which ensures seamless data transfer between the electric vehicles. In addition, we have a Vehicle-to-Infrastructure (V2I) communication system that allows for wireless communication between the EVs and RSUs. Both these communication modes, V2V and V2I, operate on the 802.11P wireless communication standard, ensuring robust, efficient, and high-speed communication within the system.

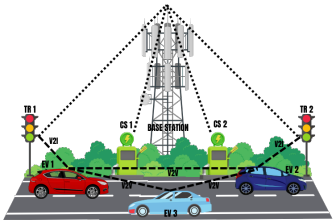


Fig. 1. Proposed System Overview

In a nutshell, the proposed system is a hybrid integration of various advanced components. It is designed to harness the benefits of distributed edge computing for cyber attack detection, using the power of CNN and intelligent data generation methods from Electric Vehicles. The specific details of our distributed CNN pipeline designed to detect cyber attacks are discussed in the next section.

IV. Proposed Distributed Framework

A unique layered Intrusion Detection System (IDS) based on distributed edge computing is provided in this paper's proposed system. Our CNN Pipeline resides on the distributed edge layer, one of the two primary components of the system. The whole of the proposed framework is seen in Fig. 2. In the Data Generation layer, information from the electric vehicle's On-Board Diagnostics (OBD) and the surrounding network is collected. The Message Queue Telemetry Transport (MQTT) protocol is used to deliver the gathered information to the Distributed Edge layer. Once the data reaches the Distributed Edge layer, it undergoes a multistep processing procedure. At first, the information received from the EVs is broken down

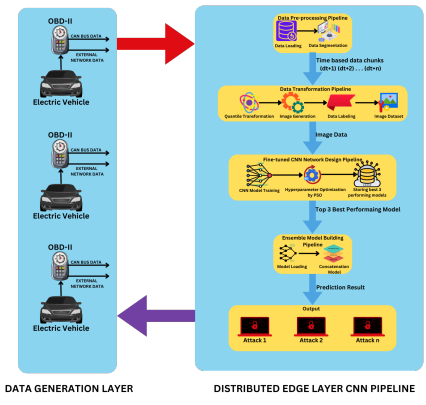


Fig. 2. Proposed Distributed IDS Framework

Authorized licensed use limited to: Indian Institute of Technology - Jodhpur. Downloaded on January 09,2025 at 13:33:57 UTC from IEEE Xplore. Restrictions apply.

978-1-7281-7254-5/21/$31.00 ©2021 IEEE

into time-based chunks. The quantile transform technique is then used to convert these blocks into finished pictures. Afterward, the resulting picture sets are used to train five state-of-the-art CNN models (VGG16 [16], VGG19 [17], Xception [18], Inception [19], and InceptionResNet [19]) that act as base learners. Particle swarm optimization (PSO) is used to improve these models' functionality. To improve productivity and precision, this hyper-parameter optimization (HPO) [15] strategy automates the tuning of model hyper-parameters. The best 3 CNN models are chosen after refinement. To improve the system's detection capabilities, these models are used as a foundation for creating ensemble learning models. In the last phase, the ensemble model is built using the concatenation technique. This model is the last line of defense against cyber-attacks on EVs that originate in a vehicle.

The Distributed Edge layer encompasses four key pipelines, namely the Data Pre-processing Pipeline, Data Transformation Pipeline, Fine-Tuned CNN Network Design Pipeline, and Ensemble Model Building Pipeline. These pipelines run concurrently on the RSU's inbuilt edge computing units, deploying containerization to ensure effective segregation, orchestration, and execution. Using edge computing and CNN models optimized by PSO, our proposed framework provides a distributed, intelligent, and resilient approach for detecting and combating cyber assaults on Electric Vehicles.

A. Data Pre-processing Pipeline

The Data Pre-processing Pipeline forms the initial, critical phase of our Distributed Edge Computing-based Intrusion Detection System (IDS). It is tasked with the initial cleaning, structuring, and reformatting of raw data that is received from the Data Generation Layer. Initially, this pipeline is responsible for filtering out any irrelevant noise from the data, thereby refining the quality of the data. This is a significant step in enhancing the accuracy of future model predictions. Furthermore, the pipeline manages missing values within the data. Utilizing various techniques such as interpolation, imputation, or deletion, missing data points are handled effectively, ensuring the data set's completeness and integrity. Lastly, the raw data is segmented into time-based chunks. The data is divided into a series of data points, represented as d_{t+1}, d_{t+2}, ..., d_{t+n}. This segmentation is essential for preparing the data for the next stage, which is the image generation process in the Data Transformation Pipeline. The formatted time-based chunks are especially compatible with Convolutional Neural Networks (CNNs) cause we can easily convert these time-based chunks into image sets, given that CNN models have been found to perform optimally on image sets. Since vehicular network traffic datasets typically come in tabular formats, transforming them into time-based chunks enhances their compatibility with the CNN-based IDS. Thus, the Data Preprocessing Pipeline is crucial in successfully detecting cyber attacks on Electric Vehicles.

B. Data Transformation Pipeline

The initiation of the data transformation procedure is marked by data normalization. Given that the range of image pixel values falls between 0 and 255, it is necessary to normalize network data within this same scale. Min-max and quantile normalization techniques are frequently employed methods that facilitate this conversion of data values into a uniform range. However, the min-max normalization technique's ability to handle outliers is somewhat limited, potentially resulting in the majority of data samples possessing exceptionally small values. Due to this limitation, our proposed framework utilizes the quantile normalization method instead. Quantile normalization operates by reshaping the distribution of feature values to match a normal distribution. It then recalculates all feature values based on this normal distribution. Consequently, the majority of the variable values align closely with the median values. This approach proves to be particularly effective when dealing with outliers, ensuring a more accurate and representative normalization of the data. The Quantile Normalization equation used in this method is:

$$X'_{ij} = Q_{ij} = \frac{1}{N}\sum_{k=1}^{N} R_{kj}$$

where X'_{ij} represents the quantile normalized value for sample i in dataset j, Q_{ij} is the quantile value (e.g., median) of sample i in dataset j, and N is the total number of samples in each dataset. The Algorithm 1 outlines the process of transforming network data into image sets suitable for use in subsequent Convolutional Neural Network (CNN) model pipeline building. The Car-Hacking dataset [14] used for this process contains four types of attacks - DoS Attack, Fuzzy Attack, RPM Spoofing Attack, and Gear Spoofing Attack - and is denoted as df in the algorithm. The Algorithm 1 commences with the extraction of numeric features from the dataset (line 1). Following this, a QuantileTransformer, denoted as $scaler$, is initialized for data normalization (line 2). The numeric features of the dataset are then scaled using this transformer (line 3), effectively normalizing the data within a range of 0-255 to match image pixel values (line 4). A count variable is initialized at zero (line 5), and an empty set ims is prepared for storing image data (line 6). An image directory "train/0/" is then created (lines 7-8). A loop is initiated to iterate through the dataset (line 9). For each iteration, the count variable is incremented (line 10), and if the count is less than or equal to 27, the corresponding data row is added to the ims set (lines 11-13). When the count exceeds 27 (line 14), the accumulated data in ims is reshaped into a 9x9x3 array (line 15).

This array is then converted into an array of unsigned 8-bit integers, essentially creating a color image (line 16). The image is saved with an appropriate filename in the created directory (lines 17-18), after which the count is reset and ims is emptied (lines 19-20). The loop continues until all data rows in the dataset have been processed, resulting in a complete set of color images representing the network data, which is then ready for the next pipeline of CNN model building.

Following the comprehensive data pre-processing steps detailed above, we generate the final transformed image set. This serves as the input for the ensuing pipeline. To illustrate the characteristics of each attack type in the Car-Hacking dataset, we provide representative samples in Fig. 3.

Authorized licensed use limited to: Indian Institute of Technology - Jodhpur. Downloaded on January 09,2025 at 13:33:57 UTC from IEEE Xplore. Restrictions apply.

978-1-7281-7254-5/21/$31.00 ©2021 IEEE

Algorithm 1 Generate 9x9x3 Color Images for Every Attack Class with Feature Scaling

$numeric_features \leftarrow df.dtypes \neq' object'$
Initialize $scaler$ as a QuantileTransformer
$df[numeric_features] \leftarrow scaler.fit_transform(df[numeric_features])$
$df[numeric_features] \leftarrow df[numeric_features] \times 255$
$count \leftarrow 0$
$ims \leftarrow \emptyset$
Define $image_path \leftarrow "train/"$
Create directory at $image_path$
for $i = 0$ to $len(df0)$ **do**
    $count \leftarrow count + 1$
    if $count \leq 27$ **then**
        $im \leftarrow df0[i]$
        $ims \leftarrow ims \cup \{im\}$
    else
        $ims \leftarrow ims$ reshaped to a $9 \times 9 \times 3$ array
        Convert ims to an array of unsigned 8-bit integers
        Create new_image from array
        Save new_image at $image_path$ with filename $i.png$
        $count \leftarrow 0$
        $ims \leftarrow \emptyset$
    end if
end for

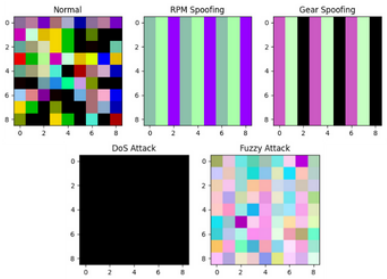


Fig. 3. illustrative sample images from each category in the Car-Hacking dataset

C. Fine-Tuned CNN Network Design Pipeline

For building this pipeline, we choose M_1 (VGG16 [16]), M_2 (VGG19 [17]), M_3 (Xception [18]), M_4 (Inception [19]), and M_5 (InceptionResnet [19]) as the foundational Convolutional Neural Network (CNN) models. This selection is rooted in their proven track record in successfully resolving a broad range of image classification challenges. These models have been pretrained on the renowned ImageNet dataset, denoted as D_{ImageNet}, a benchmark resource in image processing with over a million images spanning 1,000 classes. Their training on D_{ImageNet} equips them with robust capabilities in general image classification tasks. The architecture of M_1 (VGG16) encompasses five blocks of convolutional layers and three fully connected layers. M_2 (VGG19), on the other hand, includes an additional three convolutional layers. M_4 (Inception), another model incorporated into our framework, is designed with convolutional feature extractors that amalgamate different contexts to derive diverse feature patterns, aiding in reducing computational costs through efficient dimensionality reduction. As an extension of M_4 (Inception), M_3 (Xception) deploys depthwise separable convolutions as a substitute for standard network convolutions, which marginally reduces its memory requirements compared to M_4 (Inception). M_5 (InceptionResnet), another Inception extension, infuses residual connections from Resnet into the Inception network, delivering superior performance in image classification challenges, albeit at the expense of increased computational operations and memory usage. After employing transfer learning and fine-tuning to adapt M_1, M_2, M_3, M_4, and M_5 on the vehicular network datasets, denoted as $D_{\text{Vehicular}}$, we proceed to select the top three performers. Let M'_{Top1}, M'_{Top2}, and M'_{Top3} be the top three performing models, respectively.

V. Ensemble Model Building Pipeline

In the construction of our ensemble model building pipeline, we employed a concatenation technique, as outlined in Listing 1. This powerful method combines the strengths of multiple base models, making it particularly effective in the context of deep learning, where different models may capture different aspects of the data. Listing 1. describes the process of building the ensemble model using the last dense layer of each base Convolutional Neural Network (CNN) model. Specifically, we utilized the Xception [18], VGG16 [16], VGG19 [17], Inception [19], and InceptionResNet [19] models. Each of these models was defined with their respective base CNN models and dense layers.

```
# Ensemble Model Building and Training
def ConstructEnsembleModel(model1, model2, model3)
    :
    # Initialize data generators
    train_datagen = DataGenerator(rescale_factor
    =1/255)
    test_datagen = DataGenerator(rescale_factor
    =1/255)

    # Define train and validation generators
    train_generator = train_datagen.
    create_generator(train_dir, target_size,
    batch_size)
    validation_generator = test_datagen.
    create_generator(validation_dir, target_size,
    batch_size)

    # Define learning rate decay function
    def lr_decay(epoch):
        return initial_learning_rate * 0.1 ** (
    epoch // learning_rate_drop)

    auto_lr = ReduceLROnPlateau(monitor='val_loss'
    , factor=0.1, patience=patience, verbose=1,
    mode='auto', cooldown=0, min_lr=0)

    max_epochs = 20
    batch_size = 32
    num_classes = 5

    # Create ensemble model
    ensemble_model = CreateEnsembleModel(model1,
    model2, model3)

    for epoch in range(1, max_epochs + 1):
        current_lr = lr_decay(epoch)
        update_learning_rate(ensemble_model,
    current_lr)

        train_loss, train_acc, train_steps = 0, 0,
     total_training_samples // batch_size
```

Authorized licensed use limited to: Indian Institute of Technology - Jodhpur. Downloaded on January 09,2025 at 13:33:57 UTC from IEEE Xplore. Restrictions apply.

978-1-7281-7254-5/21/$31.00 ©2021 IEEE

```
        for step in range(1, train_steps + 1):
            batch_train_data, batch_train_labels =
     train_generator.next_batch()
            train_batch_loss, train_batch_acc =
    ensemble_model.train_on_batch(batch_train_data
    , batch_train_labels)
            train_loss += train_batch_loss
            train_acc += train_batch_acc

        train_loss /= train_steps
        train_acc /= train_steps

        val_loss, val_acc, val_steps = 0, 0,
    total_validation_samples // batch_size

        for step in range(1, val_steps + 1):
            batch_val_data, batch_val_labels =
    validation_generator.next_batch()
            val_batch_loss, val_batch_acc =
    ensemble_model.evaluate(batch_val_data,
    batch_val_labels, batch_size, verbose=0)
            val_loss += val_batch_loss
            val_acc += val_batch_acc

        val_loss /= val_steps
        val_acc /= val_steps

        print(epoch, current_lr, train_loss,
    train_acc, val_loss, val_acc)

    return ensemble_model

def CreateEnsembleModel(model1, model2, model3):
    img = Input(shape=(224, 224, 3))
    feature1 = model1(img)
    feature2 = model2(img)
    feature3 = model3(img)
    x = Concatenate()([feature1, feature2,
    feature3])
    x = Dropout(0.5)(x)
    x = Dense(512, activation='relu')(x)
    output = Dense(num_classes, activation='
    softmax')(x)
    ensemble_model = Model(inputs=img, outputs=
    output)
    ensemble_model.compile(optimizer='Adam', loss=
    'categorical_crossentropy', metrics=['accuracy
    '])
    return ensemble_model

# Call the function with your specific CNN models
    (M'_{Top1}, M'_{Top2}, M'_{Top3})
ensemble_model = ConstructEnsembleModel(model1,
    model2, model3)
```

Listing 1. Python based pseudo code for ensemble model building pipeline

The training and validation images were generated and normalized using an ImageDataGenerator, with a rescale factor of 1/255. The images were then fed into a train generator and a validation generator, which were set up with respective directories, target sizes, batch sizes, and class modes. A learning rate decay function was defined and used in conjunction with a ReduceLROnPlateau function for adjusting the learning rate during training. The model was compiled with an Adam optimizer, categorical cross-entropy loss, and accuracy metric. Early stopping and best model saving callbacks were also defined to prevent overfitting and to ensure the best model was saved. The model was trained using the train generator and the validation generator. The ensemble model was then defined with a number of classes equal to 5 and epochs equal to 20. The computational complexity of the ensemble model construction, as described in Listing 1., is $O(MZ)$, where M is the number of data samples, and Z is the total number of features extracted from the dense layers of the base CNN models. This complexity is a direct consequence of the concatenation operation, which involves combining the feature vectors from $M'_{\text{Top1}}(\mathbf{x})$, $M'_{\text{Top2}}(\mathbf{x})$, and $M'_{\text{Top3}}(\mathbf{x})$. Despite this complexity, the concatenation method offers a robust way to leverage the strengths of multiple models, leading to improved performance in many cases. The ensemble model $F(\mathbf{x})$ is constructed by combining the predictions of M'_{Top1}, M'_{Top2}, and M'_{Top3} using a fusion method, denoted as $\text{fusion}(\cdot)$, such as voting or weighted averaging:

$$F(\mathbf{x}) = \text{fusion}\left(M'_{\text{Top1}}(\mathbf{x}), M'_{\text{Top2}}(\mathbf{x}), M'_{\text{Top3}}(\mathbf{x})\right)$$

The ensemble model $F(\mathbf{x})$ is evaluated on a separate evaluation dataset to assess its performance and effectiveness in vehicular image classification tasks.

VI. Performance Assessment

A. Experimental Setup

The execution of our experiments leveraged Python's Scikit-learn and Keras libraries, mainly for Deep Learning (DL) tasks. Apache Airflow was employed for the creation of pipelines and their deployment via Docker [20] in our experimental test-bed. The training phase for the CNN Designing pipeline took place on a HP Victus machine. This machine, equipped with a Core i5-11400H processor, 16GB memory, and NVIDIA GeForce GTX 1650 Graphics, ensured robust processing capabilities for our training tasks. Post-training, testing was performed on our unique test-bed. Our testbed setup, as depicted in Fig. 4, comprises an array

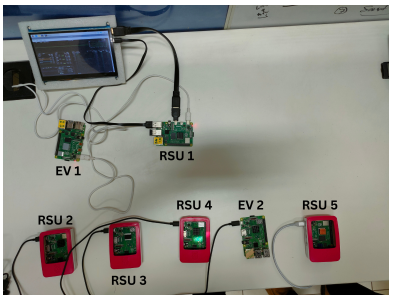


Fig. 4. Experimental Test-bed

of Raspberry Pi units - specifically, five Raspberry Pi 3 units and two Raspberry Pi 4 units. These units play distinct roles in our simulation. The five Raspberry Pi 3 units function as Roadside Units (RSUs), while the two Raspberry Pi 4 units operate as Electric Vehicles (EVs) simulating network attacks. This configuration effectively represents a realistic Internet of Vehicles (IoV) [2] scenario pertinent to Electric Vehicles. The RSUs, essential components of our proposed architecture, run the other pipelines necessary for the overall system's operation. The performance of our proposed framework is assessed utilizing the Car-Hacking datasets, as detailed in Section IV-B.

B. Analysis and Interpretation of Experimental Outcomes

The experimental results and analysis of our study are based on the performance metrics of various models leveraging the Car-Hacking dataset, as detailed in the Table I. Our proposed Concatenation method demonstrated superior performance across all key metrics. It achieved an accuracy

Authorized licensed use limited to: Indian Institute of Technology - Jodhpur. Downloaded on January 09,2025 at 13:33:57 UTC from IEEE Xplore. Restrictions apply.

978-1-7281-7254-5/21/$31.00 ©2021 IEEE

TABLE I
PERFORMANCE METRICS OF MODELS

Model Names	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	Training Time(s)	Test Time Per Packet(ms)
K-Nearest Neighbors [13]	95.9	96.4	93.9	96.5	15250.6	0.3
Random Forest [13]	98.85	98.9	99.960	98.9	1850.3	0.4
Multilayer Perceptron [8]	99.48	99.54	99.42	99.48	-	1.2
VGG16(PSO)	98.930	98.635	98.730	99.680	440.5	0.2
VGG19(PSO)	99.859	99.860	99.859	99.860	690.1	0.2
Xception(PSO)	98.909	99.710	99.709	99.707	660.5	0.3
Inception(PSO)	99.760	99.735	99.760	99.739	785.8	0.4
InceptionResnet(PSO)	99.859	99.860	99.859	99.860	1190.2	0.8
Proposed Model	**100.0**	**100.0**	**100.0**	**100.0**	**2480.4**	**3.2**

of 100.0%, precision of 100.0%, recall of 100%, and an F1-score of 100%. These results are notably higher than those of other methods, such as K-Nearest Neighbors (KNN) [13], Random Forest (RF) [13], Multilayer perception (MLP) [8] , and various PSO-optimized models, which underscores the effectiveness of our approach.

However, the superior performance of the Concatenation method comes with a trade-off in terms of computational time. The training time for our model was 2480.4 seconds, which is significantly longer than that of other models. For instance, the training time for the VGG16-PSO model was only 440.5 seconds. Similarly, the test time per packet for our model was 3.2 milliseconds, which is also higher than that of other models. The longer training and testing times can be attributed to the complexity of the Concatenation method and the limited resources of our testbed. This method involves combining the feature vectors from each base model, which increases the computational complexity. The Concatenation method exhibits a computational complexity of O(MZ), where M denotes the quantity of data samples and Z represents the aggregate number of features derived from the dense layers of the base Convolutional Neural Network models. Moreover, our testbed was composed of an array of Raspberry Pi units, which, while cost-effective and versatile, do not possess the same processing power as more advanced computing systems. This limitation in computational resources contributed to the slightly higher testing time. Despite the higher computational complexity and limited resources, the overall system performance was satisfactory. This is because the superior performance of the Concatenation method in terms of accuracy, precision, recall, and F1-scores outweighs the longer training and testing times. In other words, the trade-off between performance and computational time is justified in this case.

VII. CONCLUSION

As digitization proliferates in electric vehicles (EVs), potential cyber threats within IoV systems concurrently rise. In response, we've developed an innovative Intrusion Detection System (IDS), based on distributed edge computing. Our IDS uses transfer learning, ensemble learning, and optimally-tuned Convolutional Neural Networks (CNNs) to identify various attacks in EV's IoV systems. A standout feature is our chunk-based data transformation approach that effectively converts EV network traffic data into image data suitable for CNN models. Comprehensive testing on the Car-Hacking dataset revealed our IDS's ability to identify a wide variety of attack types with exceptional F1-scores of 100%, surpassing other state-of-the-art methods. Test-bed evaluations highlight the real-world applicability of our IDS in EV networks. Our future goal is to evolve the IDS into an online adaptive model, enhancing its capability to manage concept drift in time-series vehicular network data, and ultimately improving its adaptability and predictive accuracy in dynamic IoV systems.

REFERENCES

[1] H. Al-Alwash and E. Borcoci, "Optimal charging scheme for electric vehicles (evs) in sdn-based vehicular network, scheduling, and routing," in *14th International Conference on Communications, COMM 2022, Bucharest, Romania, June 16-18, 2022*. IEEE, 2022, pp. 1–8. [Online]. Available: https://doi.org/10.1109/COMM54429.2022.9817280

[2] X. Li, Z. Hu, M. Xu, Y. Wang, and J. Ma, "Transfer learning based intrusion detection scheme for internet of vehicles," *Information Sciences*, vol. 547, pp. 119–135, 2021.

[3] K. K. Mondal and D. Das, "Flamenet: A real-time, lightweight fire & smoke detection solution for internet of vehicles," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2023, pp. 539–544.

[4] K. K. Mondal and D. Guha Roy, "Iot data security with machine learning blckchain: Risks and countermeasures," in *Deep Learning for Security and Privacy Preservation in IoT*. Springer, 2022, pp. 49–81.

[5] S. T. Mehedi, A. Anwar, Z. Rahman, and K. Ahmed, "Deep transfer learning based intrusion detection system for electric vehicular networks," *Sensors*, vol. 21, no. 14, p. 4736, 2021.

[6] L. Yang, A. Moubayed, and A. Shami, "Mth-ids: A multitiered hybrid intrusion detection system for internet of vehicles," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 616–632, 2021.

[7] L. Yang, A. Moubayed, I. Hamieh, and A. Shami, "Tree-based intelligent intrusion detection system in internet of vehicles," in *2019 IEEE global communications conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[8] A. Rosay, F. Carlier, and P. Leroux, "Feed-forward neural network for network intrusion detection," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–6.

[9] M. D. Hossain, H. Inoue, H. Ochiai, D. Fall, and Y. Kadobayashi, "An effective in-vehicle can bus intrusion detection system using cnn deep learning approach," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[10] L. Yang, A. Moubayed, A. Shami, P. Heidari, A. Boukhtouta, A. Larabi, R. Brunner, S. Preda, and D. Migault, "Multi-perspective content delivery networks security framework using optimized unsupervised anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 686–705, 2021.

[11] L. Yang, D. M. Manias, and A. Shami, "Pwpae: An ensemble framework for concept drift adaptation in iot data streams," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 01–06.

[12] S. Moloudi, M. Lentmaier, and A. G. i Amat, "A unified ensemble of concatenated convolutional codes," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 26–30.

[13] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSp*, vol. 1, pp. 108–116, 2018.

[14] H. M. Song, J. Woo, and H. K. Kim, "In-vehicle network intrusion detection using deep convolutional neural network," *Vehicular Communications*, vol. 21, p. 100198, 2020.

[15] W.-C. Yeh, Y.-P. Lin, Y.-C. Liang, C.-M. Lai, and C.-L. Huang, "Simplified swarm optimization for hyperparameters of convolutional neural networks," *Computers & Industrial Engineering*, vol. 177, p. 109076, 2023.

[16] H. Qassim, A. Verma, and D. Feinzimer, "Compressed residual-vgg16 cnn model for big data places image recognition," in *2018 IEEE 8th annual computing and communication workshop and conference (CCWC)*. IEEE, 2018, pp. 169–175.

[17] M. Masum and H. Shahriar, "A transfer learning with deep neural network approach for network intrusion detection," *International journal of intelligent computing research*, vol. 12, no. 1, 2021.

[18] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[19] A. K. Desta, S. Ohira, I. Arai, and K. Fujikawa, "Rec-cnn: In-vehicle networks intrusion detection using convolutional neural networks trained on recurrence plots," *Vehicular Communications*, vol. 35, p. 100470, 2022.

[20] M. Openja, F. Majidi, F. Khomh, B. Chembakottu, and H. Li, "Studying the practices of deploying machine learning projects on docker," in *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, 2022, pp. 190–200.

Authorized licensed use limited to: Indian Institute of Technology - Jodhpur. Downloaded on January 09,2025 at 13:33:57 UTC from IEEE Xplore. Restrictions apply.

978-1-7281-7254-5/21/$31.00 ©2021 IEEE