2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 979-8-3503-7770-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/IROS58592.2024.10801765

ROBOGUARDZ: A Scalable Zero-Shot Framework for Detecting Zero-Day Malware in Robots

Upinder Kaur[1], Z. Berkay Celik[2], and Richard M. Voyles[3]

***Abstract*— The ubiquitous deployment of robots across diverse domains, from industrial automation to personal care, underscores their critical role in modern society. However, this growing dependence has also revealed security vulnerabilities. An attack vector involves the deployment of malicious software (malware) on robots, which can cause harm to robots themselves, users, and even the surrounding environment. Machine learning approaches, particularly supervised ones, have shown promise in malware detection by building intricate models to identify known malicious code patterns. However, these methods are inherently limited in detecting unseen or zero-day malware variants as they require regularly updated massive datasets that might be unavailable to robots. To address this challenge, we introduce ROBOGUARDZ, a novel malware detection framework based on zero-shot learning for robots. This approach allows ROBOGUARDZ to identify unseen malware by establishing relationships between known malicious code and benign behaviors, allowing detection even before the code executes on the robot. To ensure practical deployment in resource-constrained robotic hardware, we employ a unique parallel structured pruning and quantization strategy that compresses the ROBOGUARDZ detection model by 37.4% while maintaining its accuracy. This strategy reduces the size of the model and computational demands, making it suitable for real-world robotic systems. We evaluated ROBOGUARDZ on a recent dataset containing real-world binary executables from multi-sensor autonomous car controllers. The framework was deployed on two popular robot embedded hardware platforms. Our results demonstrate an average detection accuracy of 94.25% and a low false negative rate of 5.8% with a minimal latency of 20 ms, which demonstrates its effectiveness and practicality.**

I. INTRODUCTION

Robots, now ubiquitous, play an essential role in diverse sectors, from improving productivity on industrial assembly lines to providing personal care, thus proving to be indispensable in our daily lives. Their omnipresence in both domestic and professional environments has made them attractive targets for cyber attacks that threaten their core functionalities: confidentiality, integrity, and availability [1].

Recent studies have brought to light significant vulnerabilities, such as the potential for attackers to compromise user privacy through robot sensors or disrupt the operational integrity of robots by spoofing communications [2], [3], [4], [5]. The threat extends to manipulation of robot control software, which can lead to consequences, such as physical harm [6], [7], as illustrated in Fig. 1. Despite the growing sophistication of robot software, robust malware defense specifically for robotics is lacking, a situation worsened by vulnerabilities in the Robot Operating System (ROS) and the rapid progression of sophisticated malware [8], [9], [10].

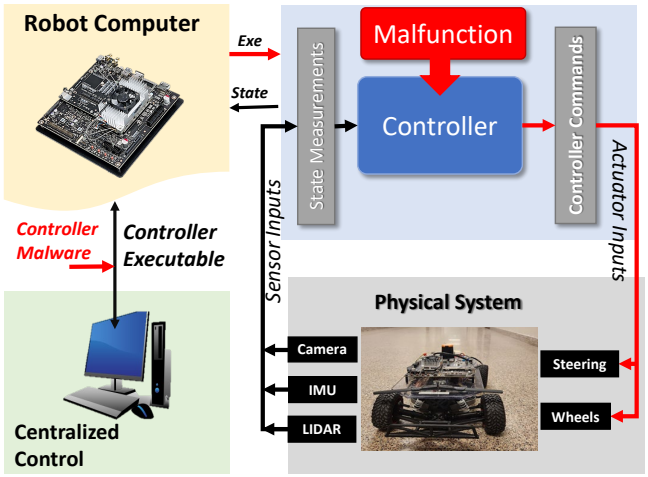

Fig. 1. Compromised robot controllers can result in major malfunctions, posing a risk to the robot's functionality and its surrounding environment.

Machine learning (ML) techniques are increasingly employed for malware detection due to their ability to identify emerging malware variants [11], [12]. These approaches, predominantly rooted in supervised learning paradigms, excel in detecting established malware patterns through intricate models. However, their reliance on extensive, prelabeled datasets poses a critical limitation: the incapacity to recognize novel or zero-day malware threats which are defined by their absence in existing datasets at the time of their discovery. This limitation is particularly critical in robotics, where the rapid co-evolution of software and hardware presents a significant challenge to the curation of comprehensive and up-to-date datasets. Although ML methods have been developed for runtime analysis [13], they have several limitations, including the need for isolated testing environments and the vulnerability to evasion by advanced malware that can mimic benign behavior [14], [12], [15].

In this paper, we present a novel static malware detection methodology tailored for robotic systems. This framework analyzes the byte structure of executables to proactively

*Kaur and Voyles acknowledge support from USDA Grant 2018-67007-28439 and NSF Grant CNS-1450342, while Celik acknowledges support from NSF Grants CNS-2144645 and IIS-2229876 in the fulfillment of this work.

[1] Upinder Kaur is with the Agricultural and Biological Engineering Department, Purdue University, 225 University Street, West Lafayette, Indiana, USA. kauru@purdue.edu

[2] Z. Berkay Celik is with the Department of Computer Science, Purdue University, 401 Grant Street, West Lafayette, Indiana, USA. zcelik@purdue.edu

[3] Richard M. Voyles is with the School of Engineering Technology, Purdue Polytechnic Institute, Purdue University, 401 Grant Street, West Lafayette, Indiana, USA. rvoyles@purdue.edu

identify malicious intent, crucial for robots in safety-critical environments. While static detection methods exist for traditional and mobile platforms [16], [17], [18], [19], research on robotic software stacks is limited [20], [21]. Moreover, deep learning-based approaches are often computationally prohibitive for resource-constrained robot embedded systems.

To address these challenges, we introduce RoboGuardZ, a malware detection framework that takes advantage of zero-shot learning principles [22], [23], [24], [25] to identify zero-day malware in robotic systems. RoboGuardZ eschews direct reliance on prior malware knowledge, instead employing a fine-grained byte-level feature extractor and similarity map to compare features from unseen malware to known classes, enabling proactive detection of novel threats. Furthermore, it uses model compression techniques such as pruning and quantization to minimize its hardware footprint, ensuring efficient operation without sacrificing detection accuracy.

- We introduce RoboGuardZ, a novel zero-shot learning framework for malware detection in robots. By integrating byte-level feature extraction with latent space similarity mapping, RoboGuardZ identifies unknown malware classes, achieving an average accuracy of 94% for malware detection in a robot vehicle.
- We present a feature mapping technique, which encodes known and novel malware classes into a shared feature space and enables knowledge transfer through relational byte-level embeddings. This facilitates an adaptive framework for unseen attacks.
- To address resource limitations in embedded platforms, we incorporate a parallel structured pruning and quantization algorithm to reduce the size of the RoboGuardZ model without compromising its detection capabilities.
- In testing on a robotic malware dataset, we show that RoboGuardZ demonstrates achieves 94.25% in zero-shot learning (ZSL) and 94.8% in generalized zero-shot learning (G-ZSL) scenarios. On a robot vehicle, it operates with an average runtime overhead of 20 ms and occupies only 7.38MB of disk space, which demonstrates its efficiency and practicality for real-world applications.

II. The RoboGuardZ Zero-Shot Learning Framework

A. Problem Statement

Our goal with RoboGuardZ to classify each robot software binary as benign or malicious. Formally, we consider a dataset $D = \{(x_1, y_1), (x_2, y_2), ...(x_i, y_i), ...(x_K, y_K)\}$, $x \in R^d$, where x_i are samples of benign code and malware with labels $y_i \in \{0, n\}$ for n classes. In supervised learning, the goal is to learn a model from the given dataset of benign and malware executables; however, in zero-shot learning, the goal is to detect an unseen class of malware by leveraging the knowledge gained from the seen classes. For a set of seen (s) and unseen (u) malware, zero-shot learning aims to infer $X^u -> Y^u$ without prior knowledge of Y^u. More specifically, the goal is to learn a classifier F that is trained in $D_{(train)} \in D$ and used to predict labels on unseen malware

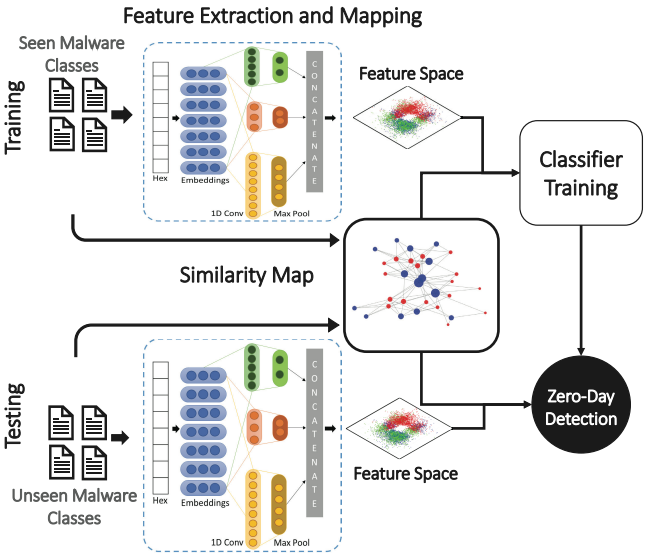

Fig. 2. RoboGuardZ's training and test loops. The feature maps and similarity maps created using the seen classes during training are leveraged to decide the label for the unseen classes with a supervised classifier.

$D_{(test)}$. In this work, we accomplish this goal by building a rich understanding of the executables with both similarity maps and feature spaces, detailed below.

B. RoboGuardZ *Overview*

The architecture of RoboGuardZ is shown in Fig. 2. Using granular byte-level features and similarity mapping, it builds a rich representation of malware and benign software. It consists of three main components: (1) *Feature Space Mapping* extracts a set of fine-grained features from the binary. (2) *Similarity Mapping* maps the binaries to a common spatial space, and their pairwise distances are calculated. (3) *Fusion and Classification* combine the information extracted from the previous two stages and train a classifier to predict the label of the unseen malware.

C. Feature Space Mapping

This component extracts structural and pattern information directly from the raw bytes of the executable, eliminating the need for computationally expensive conversions to high-dimensional representations, such as images. Initially, each input binary executable is transformed into a string of hexadecimal values, with padding applied to ensure uniform length across the dataset. This string is then passed through an embedding layer with a vocabulary of 257 (representing 0 to 256 in hexadecimal). This embedding encodes each hexadecimal value into a fixed-length representation, facilitating finer-grained learning and capturing semantic relationships, irrespective of vocabulary size.

Subsequently, the refined inputs are processed by a stack of parallel one-dimensional convolutional neural network (CNN) layers. The varying filter sizes within these layers capture local mappings at different hierarchical levels. Moreover, the parallel architecture of CNNs enables RoboGuardZ to optimize both speed and performance compared to serial

structures. Finally, the layer outputs are pooled and concatenated to generate a latent feature space. The output of this component is as follows:

$$L_f = [f_m^{s1}(f_c^{k_1,w_1}(e^{d_w})), \\ f_m^{s_2}(f_c^{k_2,w_2}(e^{d_w})), \\ f_m^{s_3}(f_c^{k_3,w_3}(e^{d_w}))] \quad (1)$$

where, e^d and $f_c^{k,w}$ are the embedding layer with d_w dimensional input and convolutional layers with kernel size k and weights w. The output of the convolutional layers is pooled using the max-pooling layers f_m^s with stride s to generate a latent space mapping L_f. The latent space mapping is generated for all inputs to the network.

D. Similarity Mapping

To identify the malicious and benign classes of binaries, we create a higher-dimension feature space with the features extracted from the binaries. We define a function $F^d : X- > L_s$ that maps the inputs to a similarity space L_s using a distance metric d, as illustrated in Fig. 2. We use the χ^2 distance to calculate the distance between a pair of samples, which is defined as

$$k(x_i, x_j) = exp\left(-\gamma \sum_i \frac{(x_i - x_j)^2}{(x_i + x_j)}\right) \quad (2)$$

where x_i and x_j are two sets of binaries and γ is a hyperparameter. The set of k_i for all sample pairs builds a distance-based similarity mapping L_k for the input binaries for both seen and unseen classes of malware. Compared to other distance metrics, we show in our ablation study in Sec. IV that χ^2 distance yields superior results.

E. Fusion and Classification

We then combine the encoded latent feature space and similarity mapping to build a stack of features that can be easily used for training and, subsequently, at test time to characterize the unseen malware. A supervised classifier SC is used to decide the final label y for the input executable as,

$$y_i = SC([L_f, L_{k_i}]) \quad (3)$$

where the output label $y_i \in 0, n$, where n is the number of malware classes with 0 being the label for benign executables.

The model is trained concurrently on both the seen set of malware and benign executables, using the mini-batch stochastic gradient descent algorithm with binary cross-entropy loss as the objective function,

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)] \quad (4)$$

where N is the total number of samples, c is the samples of each class, y_i is the true label and $\hat{y}_i$ is the predicted label.

III. The RoboGuardZ Model Scaling

Robots often utilize embedded circuits with limited computational resources, which imposes space and time constraints. To address this, we aim to minimize the malware detection model's footprint on the robot's edge device, thereby reducing computation time and energy consumption. Crucially, this minimization must be achieved while preserving detection performance, specifically accuracy and sensitivity. To this end, RoboGuardZ employs a series of operations to reduce network complexity, optimizing space and time utilization without compromising its effectiveness in identifying threats.

A. Parallel-Structured Model Pruning and Quantization

Neural networks excel at capturing intricate patterns within data, but this capability often comes with an increased model complexity due to the numerous trainable weights in layers such as embeddings and convolutions [26]. In RoboGuardZ, not all these weights contribute equally to the performance [27], [28]. Pruning addresses this by strategically removing neurons (and their associated weights) based on their magnitude, effectively simplifying the model [29]. We achieve this by setting the weights to zero, controlled by a sparsity hyperparameter η (0

Quantization, on the other hand, reduces the precision of numerical representations (e.g., from 64-bit floating point to 8-bit integer) [30]. This saves memory and can improve the inference speed, as some operations are optimized for lower-precision data types. Although pruning and quantization are applied independently, we adopt a combined approach.

Our approach to pruning and quantization is parallel-structured. We iterate through each convolutional layer, setting the smallest weights to zero. Since our feature extraction module uses parallel convolutional layers, pruning occurs simultaneously across these layers, coupled by their shared network output. Unlike many pruning algorithms that rely on the model's own output for evaluation, we let the performance of the downstream supervised classifier guide the pruning process. After each pruning iteration, we assess the average detection accuracy (See Algorithm 1). We incrementally increase the number of pruned weights until a predefined threshold of change in generalized accuracy is reached. Finally, we quantize the dense layer of the feature extraction module to further reduce memory usage, without impacting the feature mapping established by the pruned CNN layers.

.

IV. Experiments and Results

A. Dataset, Platforms, and Metric

We evaluated the performance of RoboGuardZ within the simulation environments of both an F1-tenth car and a robot rally car, as shown in Fig. 3. The RoboMal dataset [20], comprising benign and malicious robot executables, served as the source of binaries for these platforms. This dataset was generated using the source code of a scaled mobile robot, with the malware validated within a gazebo-based simulator for a wall-following scenario. RoboMal consists of 450 executables, with 218 classified as benign and 232 as

Algorithm 1 RoboGuardZ's Pruning and Quantization

Require: $\mathcal{D}_{train}, M\{Conv(W_c), SC(SC_W)\}$
  for $t \leq T$ **do**
    $\mathcal{P}rune(W_{ci} \leftarrow 0), W_{ci} \in \{1, 2, ...W_c\} \forall f_c$
    $f_{c_i} \leftarrow (W_{ci}, D_{train})$
    $M \leftarrow f_{c_i}, i = \{1, 2, 3\}$
    $train(SC) \leftarrow M(D)$
    if $\delta_{acc} \geq acc_{th}$ **then**
      $reset$
    else
      $buffer \leftarrow W_{ci}$
    end if
  end for
  $Quantize(W_d) \leftarrow I$

TABLE I
THE CLASS-WISE ATTACK DISTRIBUTION OF ROBOMAL DATASET

Attacks	Attack Motives	Samples
—	Benign	218
Attack Class 1	Parameter Attacks	24
Attack Class 2	Servo-targeting Attacks	29
Attack Class 3	Velocity-targeting Attacks	60
Attack Class 4	Steering-Targeting Attacks	81
Attack Class 5	Safety Attacks	38

malicious. As detailed in Table I, the malware is categorized into five classes based on the nature of the attacks they embody. To validate performance, we used a leave-one-out cross-validation approach, systematically excluding one class from the training examples.

We evaluated the performance of RoboGuardZ through standard metrics used in zero-shot learning: accuracy, recall, F1 score, false negative rate (FNR), and zero-day accuracy (ZD-Acc). The ZD-Acc is defined as

$$ZD - Acc = \frac{TP_z + TN_z}{TP_z + TN_z + FP_z + FN_z} * 100 \quad (5)$$

where TP_z, TN_z, FP_z, and FN_z represent the true positives, true negatives, false positives, and false negatives, respectively, computed exclusively during the evaluation of the framework on unseen malware classes. These metrics collectively capture the overall performance of the proposed framework.

B. Hyperparameter Settings

The RoboGuardZ has two critical sets of hyperparameters, one for the feature extraction module and the other for the supervised classifier. In the feature extraction module, we choose a vocabulary size of 257 for the 2500 bytes of input (input strings are padded for consistency). The three parallel 1D-CNN layers have filter size 22 with kernels of 3, 5, and 8. The subsequent max-pooling layers have a pool size of two and one stride. Each of these layers has LeakyReLU as its activation function. We set the batch size to five, and the layers are trained for 22 epochs and use the root-mean-square propagation (rmsprop) optimizer with a learning rate of 0.001 in training. For ZSL and G-ZSl, we use sparse categorical cross-entropy loss and binary cross-entropy loss.

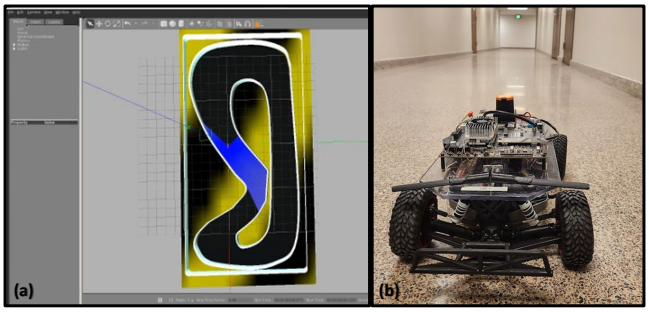

Fig. 3. The (a) simulation and (b) robot vehicle used in the validation of the RoboGuardZ framework.

We optimize the hyperparameters of the classifiers through a grid search to improve their generalization capabilities. Specifically, we employ the following configurations: a nearest-neighbor classifier with two neighbors and Manhattan distance, a random forest classifier with 100 estimators, an SVM classifier with a radial basis function (RBF) kernel, an AdaBoost classifier with 200 estimators, and a decision tree classifier with a maximum depth of five.

C. Performance Evaluation of RoboGuardZ

We evaluated the performance of the RoboGuardZ framework under two distinct conditions: a zero-shot learning (ZSL) setting, where only unseen classes are included in the evaluation, and a generalized zero-shot learning (G-ZSL) setting, where both seen and unseen classes are part of the test set. In both scenarios, we used a leave-one-out strategy, excluding one class from the training set to serve as the unseen class. We trained the remaining malware and benign classes using 10-fold cross-validation. The framework classifies each sample as benign (0) or malware (1). To ensure independence from initial conditions, we repeated training and testing for 10 runs. We calculated the previously discussed metrics for each setup, and computed their mean and standard deviation in all 10 runs for the five classes with the selected supervised classifiers, as shown in Table II.

The supervised classifiers chosen for this evaluation comprise nearest neighbors (k-NN), random forests (RF), AdaBoost, XGBoost, and LightGBM. We leverage XGBoost, SVM, and AdaBoost as final classifiers in conjunction with the RoboGuardZ feature extraction framework. The results in Table II show that classifiers employing boosting techniques outperform standard classifiers. Notably, the AdaBoost classifier paired with RoboGuardZ achieves the highest performance, boasting an average zero-day accuracy of 94.25%, with SVM and XGBoost demonstrating comparable performance in the ZSL approach. These findings demonstrate the RoboGuardZ's effectiveness in constructing a feature space rich enough for robust classifier performance.

A closer examination of the classwise zero-day average accuracy, illustrated in Fig. 4 for a suite of classifiers using RoboGuardZ for initial processing and feature extraction (including decision trees (DT) and Gaussian Naive Bayes (GB) classifiers), reveals a notable drop in accuracy when class 3

TABLE II

THE PERFORMANCE OF ROBOGUARDZ FRAMEWORK WITH THE SELECTED CLASSIFIERS ON THE ROBOMAL DATASET.

Model	ZSL				G-ZSL			
	Zero-day Accuracy	Precision	F1-Score	FNR	Accuracy	Precision	F1-Score	FNR
k-NN	82.62	82.62	89.4	17.31	88.81	85.82	91.9	5.02
RF	83.25	83.25	89.99	16.74	86.74	91.13	99.04	7.8
AdaBoost	82.51	82.52	89.51	17.4	80.93	80.59	80.68	19.95
XGBoost	81.9	82.1	89.72	18	89.13	86.96	91.90	8.3
LightGBM	84.12	82.45	84.74	13.54	79.48	81.47	78.41	23.15
RoboGuardZ(XGBoost)	87.89	87.89	93.32	11.11	91.38	92.88	95.99	7.9
RoboGuardZ(SVM)	87.28	87.63	92.18	12.5	83.80	84.9	89.40	11.8
RoboGuardZ(AdaBoost)	**94.25**	**94.25**	**97.12**	**5.8**	**94.8**	**92.39**	**96.58**	**4.38**

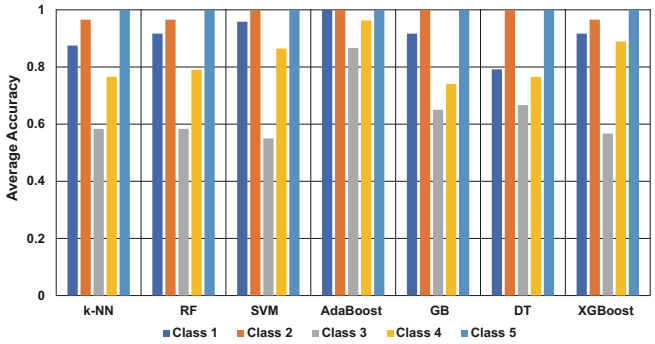

Fig. 4. The class-wise performance of the classifiers with the ROBOGUARDZ framework for pre-processing and feature extraction.

is excluded as the unseen class. This performance decline across all classifiers may be attributed to the unbalanced class sizes and unique perturbations associated with this particular class. Despite this, AdaBoost consistently demonstrates the most stable performance across all classes.

D. Ablation Study

We conduct an ablation study to assess the impact of fundamental design choices in ROBOGUARDZ, specifically examining its overall performance with the best-performing classifiers while systematically varying two key features.

1) Embedding Layer: The ROBOGUARDZ framework incorporates an embedding layer to enable individual characterization of each byte within the executable. In this representation, we employ a vocabulary of 257 words, corresponding to each unique hexadecimal value. To assess the impact of this design choice, we conducted an experiment in which the embedding layer was removed and AdaBoost and SVM classifiers were evaluated in the same set of 5 classes using 10-fold cross-validation and 10 runs.

Under the ZLS setup, the average zero-day accuracy for both classifiers, with and without the embedding layer, was plotted and is presented in Table III. The results clearly demonstrate that removal of the embedding layer leads to an approximate 17% decrease in classification accuracy.

2) Similarity Metric: The choice of the similarity metric is crucial in ROBOGUARDZ, as it determines the mapping between the learned and unseen classes, along with their respective characteristics. We opt for the χ-squared similarity mapping to efficiently capture pairwise similarities in a statistically sound manner. However, cosine similarity, often used in conjunction with Laplacian kernels, is also prevalent in the literature to map pairwise similarities [31]. Therefore, we investigated the impact of χ-squared, cosine, and Laplacian similarity mappings on the overall performance of ROBOGUARDZ.

TABLE III

ABLATION STUDY: IMPACT OF THE EMBEDDINGS LAYER ON ROBOGUARDZ PERFORMANCE

Model	No Embeddings	With Embeddings
RoboGuardZ(SVM)	0.70	0.87
RoboGuardZ(AdaBoost)	0.79	0.97

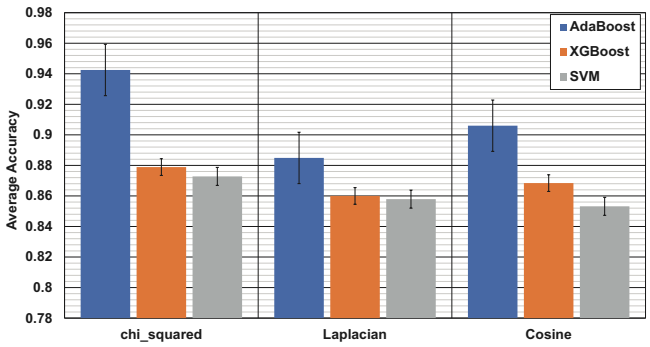

Fig. 5. The variation in average zero-day detection accuracy achieved by Adaboost, SVM, and XGBoost classifiers in ROBOGUARDZ with χ^2, Laplacian, and cosine similarity metrics.

As depicted in Fig. 5, the χ-squared similarity metric outperforms both cosine and Laplacian for the AdaBoost, XGBoost, and SVM classifiers within the ROBOGUARDZ framework. Although cosine similarity achieves an average zero-day accuracy of 91% for AdaBoost, its performance with SVM and XGBoost is significantly worse, indicating its inability to meet our specific application.

E. Hardware Performance Analysis

We employed our parallel-structured pruning and quantization strategy to compress ROBOGUARDZ for deployment on embedded hardware. To evaluate its compressed performance, we selected two representative robot vehicles equipped with Raspberry Pi 4B and Jetson TX2 as their embedded computing

TABLE IV
SPECIFICATIONS OF ROBOT PLATFORMS

Specs	Robot Vehicle	Robot Rally Car
CPU	Quad-core Cortex-A72 (ARM v8) 64-bit SoC@1.5GHz	Dual-Core Denver 64-bit quad-core ARM Cortex-A57
System	Raspberry Pi-4B	Jetson TX2
RAM	4GB	4GB
OS	Raspbian + ROS	Ubuntu 18.04 + ROS
NVM	64GB	64GB

platforms. Table IV provides a detailed overview of the hardware specifications for these boards. Subsequently, we used TensorFlow Lite and TinyML [32] to facilitate the deployment of ROBOGUARDZ on these platforms.

The neurons within ROBOGUARDZ are pruned based on a sparsity factor, defined as the percentage of total trainable parameters and classifier weights to be removed. We evaluated the impact of pruning on the average accuracy in 10 executions, varying the sparsity in the weights of the feature extraction model from 0% to 60%. As illustrated in Fig. 6, we did not observe a significant decrease in accuracy up to a sparsity level of 30%. However, increasing the sparsity beyond 40% resulted in a decrease in accuracy exceeding 1%. Furthermore, we quantized the dense layer weights per algorithm, reducing their representation to 8-bit integers. This combined pruning and quantization strategy reduced the framework's memory footprint from 11.8MB to 7.38MB, achieving a 37.4% reduction in memory size.

TABLE V
PERFORMANCE OF SCALED ROBOGUARDZ FRAMEWORK ON EMBEDDED HARDWARE

	Robot Vehicle (R-Pi)	Robot Rally Car (Jetson TX2)
Memory	7.38MB	7.38MB
Execution	$175ms \pm 8.78$	$20ms \pm 3.14$
Accuracy	94.67 ± 2.1	94.49 ± 3.87

The ROBOGUARDZ framework was successfully deployed on the aforementioned robot platforms. Table V presents a comparative analysis that focuses on detection latency, required storage space on robot onboard memory, and average zero-day detection accuracy. We observed only a slight increase in latency on both platforms, 175 ms and 20 ms, respectively, attributed to the integration of ROBOGUARDZ. Furthermore, it can be seamlessly integrated as a preliminary check within the ROS controller node or as an essential verification process executed before each operation cycle. These results highlight the feasibility of deploying it on a wide range of robotic platforms, particularly those with embedded computing and limited memory resources, affirming its practicality and efficiency in real-world applications.

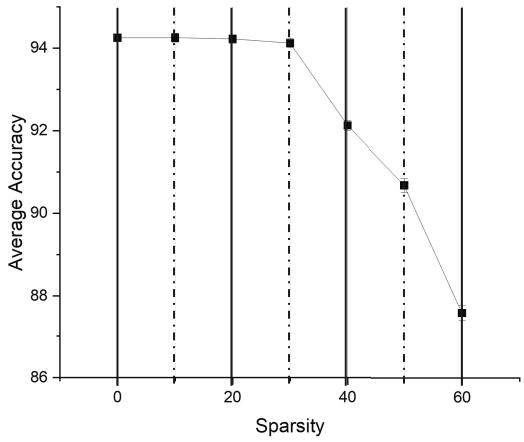

Fig. 6. The change in average accuracy with increasing sparsity in weights (0% to 60%) for ROBOGUARDZ with pruning.

V. CONCLUSION AND DISCUSSION

We introduce ROBOGUARDZ, a novel zero-shot learning framework designed to address the critical challenge of zero-day malware detection in robotic systems. By leveraging byte-level features and similarity mapping, ROBOGUARDZ establishes a comprehensive understanding of executable feature distributions, enabling it to identify previously unseen malware threats.

Evaluation in a dedicated robot malware dataset shows that ROBOGUARDZ achieves a zero-day detection accuracy of 94. 25% and a precision of 94. 8%, with a false negative rate of 5. 8%. Performance is robust across malware classes. Ablation studies validate our embedded layer and similarity metric choices.

We successfully adapted the model for deployment on common embedded hardware in robotic vehicles. Our custom pruning and quantization reduced the model size by 37.4% with minimal accuracy loss. On a Raspberry Pi 4B, the model exhibits 20 ms latency (Jetson TX2) and occupies 7.38MB of storage, demonstrating its practicality for real-time zero-day malware detection in robotic networks.

Future work will explore the integration of ROBOGUARDZ into distributed robotic systems for collaborative threat detection and response. In addition, our goal is to develop advanced compression techniques for further latency reduction in time-critical robotic applications.

REFERENCES

[1] S. Geris and H. Karimipour, "A feature selection-based approach for joint cyber-attack detection and state estimation," in *IEEE International Conference on Smart Energy Grid Engineering (SEGE)*, 2019, pp. 1–5.

[2] H. M. Rouzbahani, H. Karimipour, A. Rahimnejad, A. Dehghantanha, and G. Srivastava, "Anomaly detection in cyber-physical systems using machine learning," in *Handbook of Big Data Privacy*. Springer, 2020, pp. 219–235.

[3] T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno, and H. J. Chizeck, "To make a robot secure: An experimental analysis of cyber security threats against teleoperated surgical robots," *arXiv preprint arXiv:1504.04339*, 2015.

[4] J.-P. A. Yaacoub, O. Salman, H. N. Noura, N. Kaaniche, A. Chehab, and M. Malli, "Cyber-physical systems security: Limitations, issues and future trends," *Microprocessors and Microsystems*, 2020.

[5] H. Kim, R. Bandyopadhyay, M. O. Ozmen, Z. B. Celik, A. Bianchi, Y. Kim, and D. Xu, "A systematic study of physical sensor attack hardness," in *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[6] K. Chung, X. Li, P. Tang, Z. Zhu, Z. T. Kalbarczyk, R. K. Iyer, and T. Kesavadas, "Smart malware that uses leaked control data of robotic applications: The case of raven-ii surgical robots," in *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2019, pp. 337–351.

[7] P. Dash, M. Karimibiuki, and K. Pattabiraman, "Out of control: stealthy attacks against robotic vehicles protected by control-based techniques," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 660–672.

[8] J.-P. A. Yaacoub, H. N. Noura, O. Salman, and A. Chehab, "Robotics cyber security: Vulnerabilities, attacks, countermeasures, and recommendations," *International Journal of Information Security*, vol. 21, no. 1, pp. 115–158, 2022.

[9] B. Dieber, R. White, S. Taurer, B. Breiling, G. Caiazza, H. Christensen, and A. Cortesi, "Penetration testing ros," *Robot Operating System (ROS) The Complete Reference (Volume 4)*, pp. 183–225, 2020.

[10] H. Choi, W.-C. Lee, Y. Aafer, F. Fei, Z. Tu, X. Zhang, D. Xu, and X. Deng, "Detecting attacks against robotic vehicles: A control invariant approach," in *ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 801–816.

[11] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," in *AAAI Workshop on Artificial Intelligence for Cyber Security*, 2018.

[12] B. Athiwaratkun and J. W. Stokes, "Malware classification with lstm and gru language models and a character-level cnn," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2482–2486.

[13] T. Raffetseder, C. Kruegel, and E. Kirda, "Detecting system emulators," in *International Conference on Information Security*, 2007, pp. 1–18.

[14] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning." *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/jcs/jcs19.html

[15] U. Kaur, H. Zhou, X. Shen, B.-C. Min, and R. M. Voyles, "Robomal: Malware detection for robot network systems," *arXiv preprint arXiv:2201.08470*, 2022.

[16] E. Raff, J. Sylvester, and C. Nicholas, "Learning the pe header, malware detection with minimal domain knowledge," in *ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 121–132.

[17] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in *2012 European Intelligence and Security Informatics Conference*, 2012, pp. 141–147.

[18] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21 235–21 245, 2019.

[19] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, 2015, pp. 11–20.

[20] U. Kaur, H. Zhou, X. Shen, B.-C. Min, and R. M. Voyles, "Robomal: Malware detection for robot network systems," in *IEEE International Conference on Robotic Computing (IRC)*, 2021, pp. 65–72.

[21] H. Kim, M. O. Ozmen, A. Bianchi, Z. B. Celik, and D. Xu, "Pgfuzz: Policy-guided fuzzing for robotic vehicles." in *Network and Distributed System Security (NDSS) Symposium*, 2021.

[22] M. Sarhan, S. Layeghy, M. Gallagher, and M. Portmann, "From zero-shot machine learning to zero-day attack detection," *arXiv preprint arXiv:2109.14868*, 2021.

[23] C. Patrício and J. C. Neves, "Zspeedl - evaluating the performance of zero-shot learning methods using low-power devices," in *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, 2021, pp. 1–8.

[24] M. K. Yucel, R. G. Cinbis, and P. Duygulu, "A deep dive into adversarial robustness in zero-shot learning," in *European Conference on Computer Vision*. Springer, 2020, pp. 3–21.

[25] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, pp. 83–102, 2018.

[26] F. Gomez-Bravo, R. J. Naharro, J. M. García, J. G. Galán, and M. Raya, "Hardware attacks on mobile robots: I2c clock attacking," in *Robot 2015: Second Iberian Robotics Conference*. Springer, 2016, pp. 147–159.

[27] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the granularity of sparsity in convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 13–20.

[28] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, and X. Hu, "Pruning from scratch," in *AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 12 273–12 280.

[29] H. Wang, C. Qin, Y. Zhang, and Y. Fu, "Emerging paradigms of neural network pruning," *arXiv preprint arXiv:2103.06460*, 2021.

[30] U. Kulkarni, S. Meena, S. V. Gurlahosur, and G. Bhogar, "Quantization friendly mobilenet (qf-mobilenet) architecture for vision based applications on embedded platforms," *Neural Networks*, vol. 136, pp. 28–39, 2021.

[31] P. H. Barros, E. T. Chagas, L. B. Oliveira, F. Queiroz, and H. S. Ramos, "Malware-smell: A zero-shot learning strategy for detecting zero-day vulnerabilities." *Computers & Security*, p. 102785, 2022.

[32] P. Warden and D. Situnayake, *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media, 2019.