Density Boosts Everything: A One-stop Strategy for Improving Performance, Robustness, and Sustainability of Malware Detectors

Jianwen Tian*, Wei Kong†, Debin Gao‡, Tong Wang*, Taotao Gu*, Kefan Qiu§, Zhi Wang¶, Xiaohui Kuang*,
*NKLSTISS, Institute of Systems Engineering, Academy of Military Sciences
†School of Information Science and Engineering, Zhejiang Sci-Tech University
‡School of Computing and Information Systems, Singapore Management University
§School of Cyberspace Science and Technology, Beijing Institute of Technology
¶DISSec, College of Cyber Science, Nankai University
Email: jianwentian1994@foxmail.com, kong_wei@ieee.org, dbgao@smu.edu.sg
tongwss@foxmail.com, gutaotao1995@qq.com, kfqiu@bit.edu.cn
zwang@nankai.edu.cn, xiaohui-kuang@163.com

***Abstract*—In the contemporary landscape of cybersecurity, AI-driven detectors have emerged as pivotal in the realm of malware detection. However, existing AI-driven detectors encounter a myriad of challenges, including poisoning attacks, evasion attacks, and concept drift, which stem from the inherent characteristics of AI methodologies. While numerous solutions have been proposed to address these issues, they often concentrate on isolated problems, neglecting the broader implications for other facets of malware detection.**

This paper diverges from the conventional approach by not targeting a singular issue but instead identifying one of the fundamental causes of these challenges, sparsity. Sparsity refers to a scenario where certain feature values occur with low frequency, being represented only a minimal number of times across the dataset. The authors elevate the significance of sparsity and link it to core challenges in the domain of malware detection, and then aim to improve performance, robustness, and sustainability simultaneously by solving sparsity problems. To address the sparsity problems, a novel compression technique is designed to effectively alleviate the sparsity. Concurrently, a density boosting training method is proposed to consistently fill sparse regions. The proposed strategies are applied to PE, Android and PDF datasets, respectively. Empirical results demonstrate that the proposed methodologies not only successfully bolster the model's resilience against different attacks but also enhance the performance and sustainability over time. For instance, on EMBER (PE) dataset, the backdoor attack success rate decreased from 99.99% to 23.71% while the F1 score increased from 99.301% to 99.488%; the AUT (a metric for evaluating sustainability) increased from 92.850% to 95.135% on SOREL-20M dataset (a larger and long spanning PE dataset). Moreover, the proposals are complementary to existing defensive technologies and successfully demonstrate practical classifiers with improved performance and robustness to attacks. At last, such observation is verified to be consistent on DREBIN (Android) and Contagio (PDF) datasets.

Xiaohui Kuang and Zhi Wang are corresponding authors.

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.240336
www.ndss-symposium.org

I. Introduction

As machine learning techniques continue to evolve, many Artificial Intelligence (AI) applications assume critical roles across diverse domains [44], [67], [89]. However, due to the inherent nature of machine learning, AI-based malware detectors confront with significant challenges such as adversarial attacks [26], [68] and concept drift [31]. Consequently, the development of robust models is of paramount importance, particularly in security-critical domains of malware detection [7].

Adversarial attacks, including evasion and poisoning, pose a considerable threat by introducing perturbations into the target malware to evade detection or by corrupting training sets with maliciously crafted samples [11], [26], [68]. Concept drift represents another critical challenge, undermining the model's performance over time as new paradigms of malware emerge and the incoming test distribution diverges from the original training distribution [22], [31], [87], [91].

In response to these challenges, researchers have endeavored to fortify malware classifiers against adversarial attacks [42], [45], [79] and to mitigate effects of concept drift [22], [31], [60], [91]. The battle between attacks and defenses has led to a proliferation of strategies [42] including ensemble learning [9], weight regularization [19], adversarial training [2], [41], verifiable learning [13], [30], robust features [80], input transformation [10], classification randomization [34] and sanitizing examples [11], [72]. Despite these efforts, a universal solution to the diverse attacks has yet to be identified [42]. Similarly, addressing concept drift involves three primary strategies: Model adaption through incremental retraining or online learning [55], [85], Rejection of samples that deviate from the training set's distribution [22], [31], [87], and Robust Features Designing to enhance intrinsic resilience to drift [91]. Despite these achievements, there remains considerable scope for improvement.

This paper diverges from the conventional focus on isolated issues by highlighting the pervasive problem of sparsity within security-related datasets. Sparsity refers to a scenario where certain feature values or feature sub-regions occur with low

frequency, being represented only a minimal number of times across the dataset. In such scenarios, a model can assign excessively large weights to these sparse values or values in sparse regions to reduce training loss without immediate repercussions on performance. This can cause performance degradation on the points with sparse regions represented or be manipulated to implement attacks.

Intuitively, input transformation is the top priority in solving the sparsity problem. Input transformation methods have been widely discussed in other aspects, such as image recognition [46], [58], [86]. However, existing methods in image recognition regions only narrow the value range to limit adversarial modifications [46], [86], but do not pay attention to sparse regions, leaving certain values or regions still underrepresented [86]. In the domain of malware detection, input transformation has not been extensively explored despite the more severe sparsity [66], [79], [80]. Tong et al. [80] employed a binarization mechanism for PDF datasets, converting nonzero entries to ones, thereby limiting the adversaries' capacity to induce perturbations. In addition, Rudd et al. [66] applied a logistic transformation to Windows PE datasets [5], mapping values onto a finite interval to enhance the efficacy of neural network training. However, both methods do not effectively eliminate the sparsity, see illustrative examples in Appendix. A. Tian et al. [79] are the first to introduce a pioneering approach to sparsity elimination through a subspace compression (SC) strategy. This method involves segmenting the feature space and iteratively merging subspaces until the density within each minimal subspace surpasses a predefined threshold. While the SC strategy has demonstrated effectiveness against backdoor attacks, it has been noted to result in significant performance degradation and does not comprehensively evaluate the broader implications of mitigating sparsity.

In this work, we introduce a novel subspace compression strategy with value bundling, referred to as SCB, and propose a density boosting strategy to consistently fill sparse regions during training. We validate the efficacy of our methods across three benchmark datasets: EMBER (PE) [5], DREBIN-2019 (Android) [22], and Contagio (PDF) [68]. Our experimental results indicate that mitigating sparsity significantly improves model performance and robustness. Specifically, on the EMBER testing dataset, the F1 scores increased from 99.301% to 99.488%, outperforming the state-of-the-art detector on EMBER testing set; and the AUT (a metric for evaluating sustainability) increased from 92.850% to 95.135% on SOREL-20M, which also outperforms other strategies. Meanwhile, the attack success rate of VR-based backdoors (poisoning 1% benign set) was substantially reduced from 99.99% to 23.710%. While our proposal solely offers marginal defenses against evasions, its combination with existing defensive methods demonstrates markedly improved performance and resistance to evasion attacks compared to the application of defensive methods in isolation [18], [73]. At last, such observation is verified to be consistent on DREBIN and Contagio datasets.

The contributions of this paper are as follows:

1) **Elevating the sparsity problem:** We associate sparsity with key issues in malware detection, such as clean performance, backdoor robustness, evasion robustness, and model sustainability, and provide an extensive evaluation demonstrating the severity of the issue in widely-adopted datasets.
2) **Novel subspace compression and density boosting robust training:** We propose a novel subspace compression strategy and a density-boosting training approach that effectively improve model robustness by mitigating sparsity while further enhancing performance and sustainability under concept drifts.
3) **Practical solution for malware detection:** We explore practical solutions by integrating our proposal with state-of-the-art defensive methods, showcasing its compatibility and the potential for improved robustness and superior clean performance compared to using defensive methods alone.

II. Background and Our Problem Space

A. Background

1) Malware Detectors: In the domain of learning-based malware detection, classifiers are primarily trained to discern between malicious and benign inputs, leveraging two distinct types of features: dynamic and static. Dynamic features encompass behavioral records indicative of suspicious activities, harvested through the execution of binary files within controlled virtual environments [4], [35], [83]. In contrast, static features are derived directly from the binary code or associated metadata, without the need for execution [7], [26], [57], [84]. Static feature-based methods can be further bifurcated into two subcategories: feature-based detectors and raw-binary detectors [5], [7], [15], [26], [57], [64]. Feature-based detectors [5], [7], [26], [57] extract a vector of salient characteristics from the binary, relying on expert knowledge to identify and quantify relevant features. Conversely, raw-binary detectors [15], [36], [64] process the raw binary data as input matrices, bypassing the need for feature extraction based on expert knowledge.

To build malware detection models, a feature vector, denoted as x, is extracted from either a raw binary or an Android application. After extraction, the input matrix, $\mathbf{X}$ and its associated labels $\mathbf{Y}$ are utilized to train the model. Within the conventional framework of malware detection, the objective is to ascertain the label $y \in \mathcal{C} = \{0, 1\}$ for a given input $x \in \mathbf{X}$. The input-label pairs are assumed to be independently and identically distributed (i.i.d.) samples drawn from an underlying distribution $\mathcal{D}$. The detection model is formalized as a function $F_{\boldsymbol{\theta}} : \mathbf{X} \to \mathcal{C}$, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$. The classifier's parameters, $\boldsymbol{\theta}$, are optimized by minimizing a loss function $L(x, y, \boldsymbol{\theta})$ across a training set $\widehat{\mathcal{D}} = \{(x_i, y_i)\}_{i=1}^n$ comprising labeled samples:

$$\begin{aligned}\arg\min_{\{\theta\}} L(x, y, \theta) = -[y * log(Prob(pred = y|x, \theta)) \\ +(1 - y) * log(Prob(pred = (1 - y)|x, \theta))]\end{aligned} \quad (1)$$

2) Adversarial Attacks: Adversarial machine learning attacks are divided into **evasion** and **poisoning**. **Evasion attacks** alter inputs to mislead classifiers at inference time [12], [18], [24], [26], [73], disguising malicious intent and posing a significant cybersecurity threat. Conversely, **poisoning attacks** taint the training dataset with crafted examples [54], [75]. They are categorized as: *availability* attacks reducing model efficacy [54]; *targeted* attacks misclassifying specific targets [69],

[75]; and *backdoor* attacks embed a covert trigger during training, which induces misclassification upon activation during inference [27], [68], [79].

Extensive research has explored adversarial attacks across various malware types, including Android [20], [26], [68], [79], PDF [50], [74], Windows (PE files) [2], [6], [28], [37], [73], [76], IoT [1], and Flash [51]. Unlike image processing, malware adversarial attacks require function-preserving modifications within file format constraints [73]. Minor changes could disrupt the file or its malicious function. Adversaries prioritize sample transformations that preserve functionality, such as altering instruction sequences to equivalent ones, appending redundant sections from other binaries, or adding benign content [18], [47], [61], [73], [88].

A multitude of defensive strategies have been proposed to counter evasion attacks, as classified by Li et al. [42]. These include *Ensemble Learning* for robustness through model diversity [9], *Weight Regularization* to limit sensitivity to adversarial examples [19], *Adversarial Training* to bolster resilience by incorporating such examples into training [2], [39], [41], *Verifiable Learning* ensuring consistent model decisions [13], [30], *Robust Feature Selection* based on inherently resistant features [80], *Input Transformation* to reduce adversarial impact [10], *Classifier Randomization* for unpredictability through diverse classifier sets [34], and *Sanitizing Examples* to refine samples using various metrics [43].

In contrast, defenses against poisoning attacks in malware detection, while less prevalent, are an area of growing research. Approaches include filtering based on similarity metrics or SHAP value analysis [11], [49], [82], employing AutoEncoders to neutralize backdoor poison attacks [56], and compressing sparse regions to counteract sparsity-based backdoors [79].

3) Concept Drift: Concept drift, a pervasive phenomenon in the domain of machine learning, also poses a significant challenge to the applicability of classifiers. It occurs when the joint distribution of inputs and outputs differs between training and test phases [62]. The primary types of shift include *covariate shift* (variation in feature distribution), *label shift* (variation in label distribution), and *concept shift* (evolution in class definitions). Given the complex and often simultaneous nature of these shifts, the term *concept drift* is commonly employed to encompass all such variations [21], [22], [31], [65], [70]. This paper adopts this widely accepted terminology.

Addressing concept drift, three predominant strategies have emerged. *Model Adapting* employs incremental retraining or online learning to refine the model [55], [85]. *Rejection* discards samples identified as being from a distribution that has shifted from the training data [22], [31], [87]. *Robust Features Designing* develops features that confer intrinsic resilience to the model or system against the effects of drift [91].

B. Our Problem Scope

The preceding discussion highlights a plethora of strategies proposed to address a spectrum of issues in the domain of machine learning and cybersecurity. Our work diverges from the conventional approach of targeting specific problems, instead delving into the intrinsic mechanisms of model vulnerability. While various elements can compromise model robustness [24], [32], [79], this paper zeroes in on "sparsity" as a key vulnerability factor. Despite recognizing sparsity's role in backdoor attacks [79] and evasion attack transferability [92], prior work has mainly exploited sparsity to boost attack potency. To date, no comprehensive evaluation has been conducted to assess the pervasive impact of sparsity across all dimensions. This paper posits that the detrimental effects of sparsity have been grossly underestimated, especially considering the ubiquity of sparse feature subspaces in security-related datasets, which can critically undermine detector efficacy.

In this study, we examine the correlation between sparsity and prevalent challenges in machine learning and malware detection, encompassing *performance loss*, *evasion attacks*, *poisoning attacks*, and *concept drift*. We demonstrate the effects of various input transformation strategies on these issues, specifically targeting low-density subspaces. These strategies include subspace compression (SC) [79], logistic transformation (LT) [66], binarization [80], and histogram [33], alongside our proposed method, subspace compression with bundling (SCB). Furthermore, we introduce a density boosting training strategy to further alleviate sparsity.

Our comprehensive evaluation concludes that addressing sparsity is essential for enhancing the reliability and robustness of malware detectors. Among the strategies considered, our proposal, SCB, exhibits superior detection performance on all datasets compared to other methods [5], [7], [79], [80]. Moreover, it demonstrates robust resilience against backdoor attacks [68], [79] and concept drift [22]. While the standalone robustness against evasion attacks is modest, our approach significantly enhances the efficacy of existing defensive techniques across the board [47], [73].

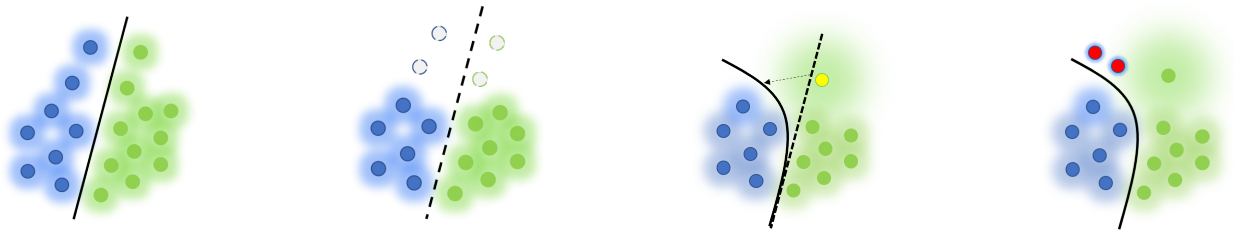
III. Related works

A. Feature compression

In the context of image classification, the feature space is considered excessively large. Studies have shown that shrunken representations still retain sufficient information for training models [8], [16], [16], [46], [52], [58], [86]. Xu et al. [86] implemented defenses against evasion attacks by reducing the color depth of images. Similarly, Lo et al. [46] leverage error diffusion halftoning methods to project images into a 1-bit space.

Despite widely mentioned in other domains, it is often overlooked in malware detection tasks. Tong et al. [80] improved evasion attack resilience with binarization. Narisada et al. [56] used AutoEncoders against backdoors, but Tian et al. [79] showed that this was ineffective for their attacks. Rudd et al. [66] enhanced PE dataset [5] accuracy with logistic transformation. Tree-based models like LightGBM [5] naturally compress features, which likely contributes to top EMBER dataset performance.

Beyond tree-based models, input transformations such as histogram [33], logistic [66], and binarization [80] focus on feature depth, not sparse regions. Tian et al. [79] introduced the first subspace compression (SC) strategy to eliminate sparse regions against backdoors. SC processes outliers, segments feature values, and merges subspaces based on KL-divergence until meeting a density threshold. While effective against backdoors, it impacts performance.



(a) Actual distribution of the two class data.

(b) Training data sampled from the overall distribution.

(c) A sample warp the decision boundary.

(d) testing samples are wrongly classified.

Fig. 1: A toy example of our motivation.

In response to these limitations, we introduce a novel mechanism that incorporates value bundling into the subspace compression process. Furthermore, we suggest merging subspaces based solely on density rather than label distribution to prevent unnecessary loss of classification capability.

B. Robust training

Robust training aims to strengthen machine learning models to improve generalization ability in disturbances or uncertainties [17], [24], [38], [59], [77], [78], [81], [83], [90] Robust training defenses have been proposed to strengthen malware classifiers [19], [63], often targeting specific attacks [2], [30], [63] or models [13], [19], [48]. Íncer et al. [30] use monotonic classifiers for evasion resistance, relying on features with non-simultaneous removable or addable properties and monotonic functions like linear models. Chen et al. [13] introduce a distance metric to bound robustness properties and train a verifiable classifier. Huang et al. [29] explore certified defenses for CNNs using random deletion inspired by random smoothing [17]. Daniel et al. [23] use (De) randomized smoothing to build a verifiable model based on binary segmentation and majority voting. Despite certified robustness, these defenses may fail against aggressive attacks like GAMMA [47].

Empirical adversarial training bolsters model robustness. Quiring et al. [63] combine heuristic semantic-gap detectors with an ensemble of classifiers to counter evasion attacks. Al-Dujaili et al. [2] use randomized rounds in adversarial training for API-based detectors, noting ineffectiveness against new attacks. Tong et al. [80] propose iterative adversarial training on PDF datasets. Li et al. [40] introduce AT-MaxMA, hardened by mixed adversarial training, proving robust against Android malware. PAD-SMA [39] furthers this with a convex outer bound and a stepwise attack mixture, optimizing the worst-case loss for provable robustness to any norm-bound attacks. PAD-SMA shows the ability for defending against unseen evasion attacks. Lucas et al. [48] show that using real adversarial PE examples in training, combined with multiple attacks, results in more robust models.

Our training approach diverges from the conventional focus on hardening models against specific attacks. Instead, we try to mitigate the pervasive issue of sparsity in malware detection datasets, ameliorating the associated problems by filling in sparse regions. By mitigating sparsity, we enhance both the robustness and performance of models. More importantly, our strategy is complementary to existing training methods.

IV. How Does Sparsity Affect Models?

A. Motivation

Low-density subspaces are deemed detrimental as they represent regions of sparse data, potentially with minimal or no points. A model can assign excessively large weights to these sparse subspaces to reduce training loss without immediate repercussions on performance. This could lead the model to learn superficial classification rules, which are not indicative of the underlying data distribution. To elucidate this motivation, we present a hypothetical example in Figure 1.

Suppose Figure 1(a) illustrates the actual distribution of a two-class dataset, upon which a robust classifier could be trained where the classification mainly relies on feature X (we refer to horizontal direction as feature X and vertical direction as feature Y). However, in most practical cases, the observable training data are sampled independently and identically from this actual distribution, potentially resulting in significant sparsity as depicted in Figure 1(b). Consequently, a few samples within sparse regions, irrespective of whether they are maliciously inserted or naturally occurring, can severely distort the classifier's decision boundary, as shown in Figure 1(c). Ultimately, this susceptibility to noise can result in misclassification, as illustrated in Figure 1(d).

With this intuition, one can easily associate sparsity with multiple threats to the model:

Backdoor attacks: The yellow dot in Figure 1(c) is maliciously inserted as poison to warp the model boundary. As a result, points with the trigger (large value in feature Y) will be misclassified; see Figure 1(d).

Evasion attacks: The yellow dot in Figure 1(c) is naturally occurring and it causes a distorted boundary. Adversaries can increase the value of Y to achieve misclassification; see Figure 1(d). Moreover, feature Y is less relevant to the classification rule on actual distribution, which may be imperceptible.

Clean performance and **Model sustainability**: As long as there are such warped boundaries, model decisions are vulnerable in the sparse regions. Any examples present in such sparse regions may be wrongly classified, leading to degradation in clean performance. In particular, the scenarios with serious concept drift where many unrepresented areas in the training become densely filled in the testing stage can lead to a drastic model aging phenomenon (degraded performance on newly emerged points).

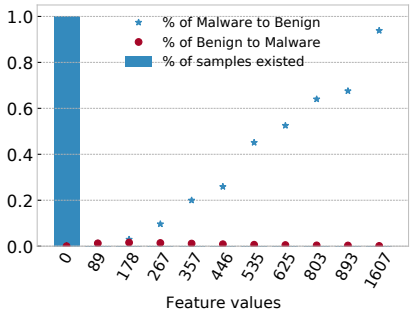


Fig. 2: The feature distribution of registry count in EMBER.

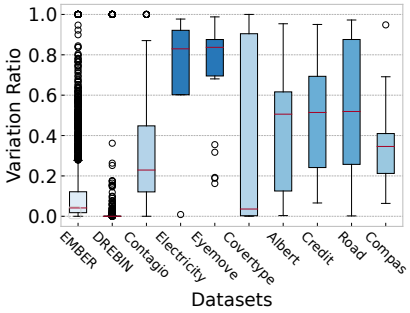


Fig. 4: The variation ratio distribution on different datasets.

Two real-world examples are provided to demonstrate the severity of sparsity threats, as shown in Figure 2. It displays the distribution of a feature (*registry count*) from the EMBER (PE) dataset [5], exhibiting a long-tail distribution where the x-axis represents the value space and the y-axis denotes the percentage of samples within that range (termed density). This indicates a pronounced sparsity. Moreover, when these values are introduced into the test set, they exert a significant influence on the model's predictions. Specifically, samples with more than 1,607 registries are highly likely to be misclassified as benign. This phenomenon is not confined to tabular datasets such as EMBER feature sets [5]. In a Support Vector Machine (SVM) trained on the DREBIN dataset [7], a prevalent Android feature set with binary features, similar issues arise where less frequent features in the dataset disproportionately influence the model's decisions, as shown in Figure 3.

In a more comprehensive way, we adopt variation ratio to quantify the sparsity of a dataset, which had been used by Tian et al. [79] to select backdoor triggers within sparse regions. Variation ratio evaluates the dispersal level of data (from 0 to 1). A low variation ratio indicates there is a value with predominant density and comes with a series of sparse regions. We calculate the variation ratio across all features on different datasets including EMBER, DREBIN, Contagio and other seven benchmark tabular datasets [25] (not for malware detection) to show the severe status quo of malware detection-related datasets, see Figure 4. As we can see, the datasets for malware detection tasks demonstrate more severe sparsity than other datasets, especially EMBER (PE) and DREBIN (Android). In addition, the characterization of binaries typically requires a larger number of features (e.g. Contagio (153), EMBER (2,381), and DREBIN (more than 630K)) than other seven datasets whose feature numbers are extremely tiny from 8 to 54. More importantly, malware detection tasks are in a highly adversarial scenario. All of this reinforces the necessity of mitigating the sparsity in malware detection tasks.

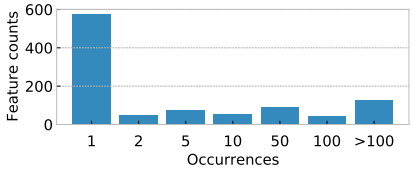


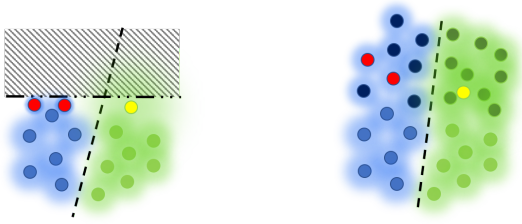
Fig. 3: Top 1000 features with highest weights in DREBIN.

B. *Strategies for Addressing Sparsity*

Intuitively, two principal methodologies present themselves as potential solutions to the sparsity problem:

- *Feature Compression*: This technique involves the reduction of the feature space, thereby mitigating the impact of low-density regions. This concept is illustrated in Figure 5(a).
- *Filling Sparse Regions*: Enhancing the density of sparse regions within the training set can effectively neutralize the influence of such spaces. This approach is demonstrated in Figure 5(b).

In light of these considerations, we propose a novel feature compression method designed to eradicate extremely sparse regions. Furthermore, we introduce a density-boosting training strategy. This training strategy consistently fills sparse regions, with the objective of encouraging models to discern more profound semantic patterns rather than relying on superficial associations within sparsely populated feature spaces. Through these interventions, our goal is to cultivate models that are not only resilient to adversarial perturbations but are also capable of making informed decisions based on a richer and more nuanced understanding of the data.



(a) Compressing the sparse regions. (b) Filling the sparse regions.

Fig. 5: Solving sparsity problems.

V. Solving the Sparsity Problem

In this section, we introduce our methods to solve the sparsity problems. We mainly propose two strategies in mitigating this problem, including a subspace compression with value bundling and a density-boosting robust training.

A. Subspace Compression with Bundling

We introduce a novel subspace compression that incorporates a value bundling mechanism. It merges subspaces based exclusively on density criteria rather than label distribution. Our experiments, detailed in Appendix B, demonstrate the redundancy of considering label distribution.

Step 1 - Outlier Processing: Our methodology commences with the processing of each feature in the training set using a box plot (See formal algorithm in Appendix H). The 25th and 75th quartile values, $Q1$ and $Q3$, are calculated, representing the points below which 25% and above which 75% of all values fall, respectively. The interquartile range (IQR), defined as $Q3-Q1$, is then determined. Values lying outside the range of $Q1-3\times$IQR and $Q3+3\times$IQR are adjusted to the boundary values. *To address the potential elimination of information-rich features, we implement a binarization mechanism for features where $Q3$ equals $Q1$. The value with the highest density is designated as 0, while all others are set to 1.*

Step 2 - Subspace Combination: Subsequently, the processed values are distributed into 100 bins using a histogram. Bins with a density below a predefined threshold, H, are iteratively merged with adjacent bins of lower density. This process continues until the minimum density exceeds H or *only two bins remain*, ensuring that critical information is retained and not eliminated. Figure 6 shows the processing of `num_write_sections` and `export_libs_hash66`. Formal algorithm can be found in Appendix H.

Step 3 - Value Bundling: The final step achieves the target density through value bundling across different features. Features with densities below the threshold are bundled with the sparsest values of other features that result in the fewest conflicts. We define "conflicts" between Feature A and B as occurrences where non-major values (values not with the highest density) of Feature A are accompanied by the non-major value of Feature B. For example, if a feature's non-major value always comes with another features' major value, they are "exclusive". See Figure 7 for illustration of bundling Feature A with Feature B. After bundling, the features are merged. see formal algorithm in Appendix H.

This systematic approach ensures that our model navigates the sparsity challenge by enhancing the density of feature space, fostering a more robust and semantically rich training process; see Figure 6. The initial distributions, depicted in the two figures on the left column, exhibit extreme sparsity. Those in the middle column represent the value distribution post-outlier removal (Step 1), resulting in a significant reduction in distinct values. Figures on the right column display the outcome of subspace combination (Step 2), where sparse values such as the value 5 in `num_write_sections` are amalgamated with the adjacent bin. `export_libs_hash66` is not further combined as it has only two values left.

Note that the original outlier processing (without binarization) and subspace combination [79] are directly applied, causing 73% sparse features to be removed. However, features of high sparsity may contain critical information for classification. Our proposal is different as it does not kill any features. The outlier processing method has a novel binarization mechanism which transforms dense values into 0 and sparse values into 1. The subspace combination does not kill any features either

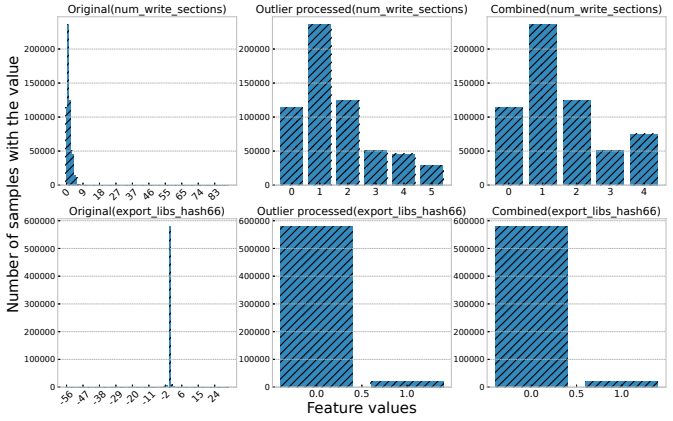


Fig. 6: Outlier processing and subspace combining

with early termination. After the processing and combination, features that do not meet the threshold requirement are bundled (not removed). In this way, our proposal is expected to retain more classification information while boosting the density. See Section VII-A for more details.

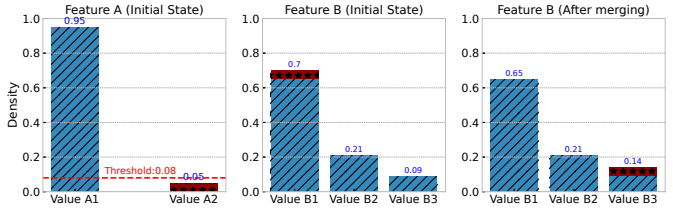


Fig. 7: Bundling values.

B. Filling the sparse regions

During training, we incorporate a density boosting strategy aimed at consistently filling the sparse regions to enhance the model's resilience against sparsity-induced influences.

Density Boosting (DB): An intuitive and efficient method is to replace existing feature values with the sparse ones. We consider the value distribution to facilitate the selection of sparse values in a manner that is inversely proportional to their existing density, thereby making the final distribution uniform, as illustrated in Figure 8. Specifically, for each sample in the training stage, a subset of features is randomly selected, and these features are all assigned sparse values which are determined based on a probability distribution q defined as:

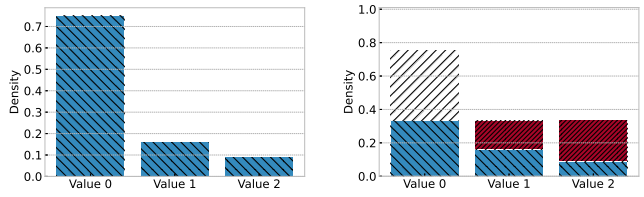
$$q = \left[\frac{\frac{1}{d_0}}{\sum_{i=1}^{K} \frac{1}{d_i}}, \frac{\frac{1}{d_1}}{\sum_{i=1}^{K} \frac{1}{d_i}}, \ldots, \frac{\frac{1}{d_K}}{\sum_{i=1}^{K} \frac{1}{d_i}} \right]$$

where d_i represents the density of the i-th value of the feature.

The optimization problem is formalized as:

$$\min_{\{\theta\}} \mathbb{E}_{(x,y)\in\widehat{D}} \left[L(x, y, \theta) + L(db(x), y, \theta) \right] \tag{2}$$

where x represents the input sample, y is the corresponding ground truth label, and $db(x)$ denotes the sample x with the



(a) The original distribution of values. (b) The distribution after density boosting.

Fig. 8: An example of density boosting applied.

augmented sparse values. This approach trains the model on both the original and augmented data.

Furthermore, we also experimented with alternative filling strategies, such as assigning zeros or uniformly random values to the features. However, these approaches led to diminished backdoor robustness and did not yield improvements in other performance metrics, see Appendix E for more details.

VI. Experimental Evaluation

A. Preliminaries

1) Datasets: **EMBER-v2 [5]** is a labeled benchmark dataset of Windows portable executable files, comprising 2,381-feature vectors extracted from 800K binary files: 600K training samples (300K malicious, 300K benign) and 200K testing samples (100K benign and 100K malicious). A sample is labeled benign if no engines flag it as malicious, and is labeled malicious if more than 40 engines flag it as such. The dataset includes five groups of parsed features: *General file information*, *Header information*, *Imported functions*, *Exported functions*, and *Section information*, and three groups of format-agnostic features: *Byte histogram*, *Byte-entropy histogram*, *String information*, and *Data directory*. Additionally, EMBER-v2 gives consideration to temporal consistency [60]; for example, all training samples are primarily collected between January 2017 and October 2017, while all testing samples are collected in November and December 2017.

SOREL-20M is the largest public PE dataset with the same features as EMBER-v2. It contains 12,699,013 training samples (7,596,407 malware and 5,102,606 goodware), 2,495,822 validation samples (962,222 malware and 1,533,579 goodware), and 4,195,042 test samples (1,360,622 malware and 2,834,441 goodware), where the labeling criteria are based on 1 or fewer flags for benign and 5 or more for malicious. The data was collected from January 1, 2017, to April 10, 2019. We utilize SOREL samples from January 1, 2018, onwards for evaluating the effects of concept drift because it has a large scale and offers a realistic, yet distinct, distribution compared to the EMBER-v2 dataset.

2) Models: We consider 10 models for evaluation as below:

VanillaNN [68]: In this case, we do not apply any compressions and directly normalize the features to have a mean of 0 and a variance of 1.

LightGBM [5]: We adopt LightGBM as it was originally selected for the EMBER dataset and demonstrates the best performance on it. Also, tree-based models naturally come with a strong "subspace compression" effect.

LTNN [66]: Logistic Transformation was previously used by ALOHA [66] and has demonstrated good performance on SOREL-20M. We hence adopt it for preprocessing datasets. Refer to the Equation below for details.

$$X_{\text{processed}} = \begin{cases} -\ln(1-X), & \text{if } X < 0 \\ \ln(1+X), & \text{if } X > 0 \end{cases} \tag{3}$$

BinarizedNN [80]: It has been verified to improve the robustness against PDF detectors; the operation replaces all non-zero values with 1.

HistogramNN [33]: The histogram-based algorithm was introduced to speed up the training of LightGBM, and its ability to discretize continuous floating-point features into bins may also help improve models' robustness. We use a small number of bins, specifically 6.

SCNN [79]: Subspace compression was proposed to defend against backdoor attacks, although it degrades the models' clean performance. This can also be seen as the counterpart of SCB without binarization and value bundling.

SCBNN: This is our proposal in this paper and is used to compare with the other strategies.

SCBNN-DB: We consider a varied perturbation to explore more varieties by perturbing 1%-15% of random features during training. For details of other settings, see Appendix E.

We delineate the specific metrics employed in this paper:

AUT (Area Under Time): AUT is a metric proposed by Tesseract [60], which defines the area under the curve in each figure to represent the model's sustainability over time as shown in Equation 4, where f in the performance metric (e.g. F1 score, Precision, Recal, etc.), N is the number of test slots, and $f(k)$ is performance metric evaluated at the time k, and in our case the final metric is the $AUT(F_1, M)$ where M denotes the test samples' spanning months. An AUT metric that is closer to 1 means better performance over time.

$$\text{AUT}(f, N) = \frac{1}{N-1} \sum_{k=1}^{N-1} \frac{[f(k+1) + f(k)]}{2} \tag{4}$$

ASR (Attack Success Rate): The ASR is defined as the proportion of malware instances that were previously classified correctly but are misidentified as benign by the target model under adversarial conditions.

Research Questions: The experimental design is structured to address the following research questions:

1) **RQ1:** *Analysis of Sparsity Effect on Performance and Sustainability.* Does addressing the sparsity in data enhance the performance and sustainability of malware detectors? (see Section VI-B)
2) **RQ2:** *Analysis of Sparsity and Backdoor Attacks.* Does mitigating sparsity in feature spaces improve the robustness of malware detectors against backdoor attacks? (see Section VI-C)

3) **RQ3:** *Analysis of Sparsity and Evasion Attacks.* Does addressing sparsity bolster the resilience of malware detectors against evasion attacks? (see Section VI-D)

B. Effect of Sparsity on Performance and Sustainability

This section presents an evaluation of model performance under clean conditions, devoid of adversarial interference.

Performance on the EMBER Testing Set Our analysis begins with evaluating model performance at different compression rates on the EMBER test set, shown in Figure 9. Results show our compression preserves performance, with SCBNN outperforming VanillaNN and original SC in F1 scores. The benefit of value bundling is clear, when comparing SCBNN (without bundling). SCBNN still yield excellent performance under high compression rates, reaching 99.456% F1 score at 8% density. Thus, we chose 8% as the main compression rate for further experiments.

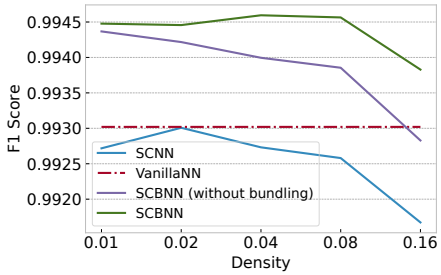


Fig. 9: Clean performance under different compression rates.

Table I compares final model performances, showing LTNN, HistogramNN, and SCBNN generally outperform vanilla NN. SCBNN excels over other compression methods, supporting our hypothesis that handling sparse regions improves performance. Density boosting further enhances this, with SCBNN-DB reaching an F1 score of 99.488%. LightGBM surpasses other NN models without density boosting due to tree-based models' inherent feature compression, aligning with the prevalent performance superiority of tree-based models on tabular datasets [25].

Performance on SOREL-20M: We then evaluated model performance under concept drift using the SOREL-20M dataset, with monthly F1 scores leading to the final AUT (F1,16m) shown in Table I. Except for BinarizedNN, compression improves performance, with SCBNN leading at 94.444% among NN models without density boosting. Density boosting strategies further enhance performance, achieving an AUT of 95.135%, outperforming even LightGBM.

Our strategies were also tested on the seven dense datasets from Section IV-A. Results varied, with improvements in six datasets and slight decline in one (that with the largest amount of data). Sparsity mitigation may be less critical for such dense datasets without attack considerations; see Appendix C.

TABLE I: Performance of different models

Model	F1 score	FP rate	FN rate	AUT (F1,16m) on SOREL
VanillaNN	0.99302	0.00442	0.00958	0.92850
LTNN	0.99311	0.00400	0.00977	0.93312
BinarizedNN	0.98942	0.00776	0.01339	0.91887
HistogramNN	0.99390	0.00323	0.00852	0.94148
SCNN	0.99225	0.00397	0.01148	0.93387
LightGBM	0.99470	0.00258	0.00799	0.94651
SCBNN	0.99456	0.00363	0.00721	0.94444
SCBNN-DB	**0.99488**	0.00381	0.00642	**0.95135**

Answer to RQ1:

Addressing sparse regions does improve performance and sustainability. Our strategies present a promising approach for training on tabular malware datasets.

C. Analysis of Sparsity and Backdoor Attacks

Here, we assess defense against two backdoor attacks: VR-based backdoors [79] and EG-based backdoors [68].

1) Implementing backdoor attacks: Creating a model backdoor involves: 1) Choosing a feature combination as the trigger, and 2) Selecting benign samples, known as "poisoning seeds", to mark with the trigger. These poisoned instances, with the embedded trigger, are then merged into the training set, training a model with a backdoor.

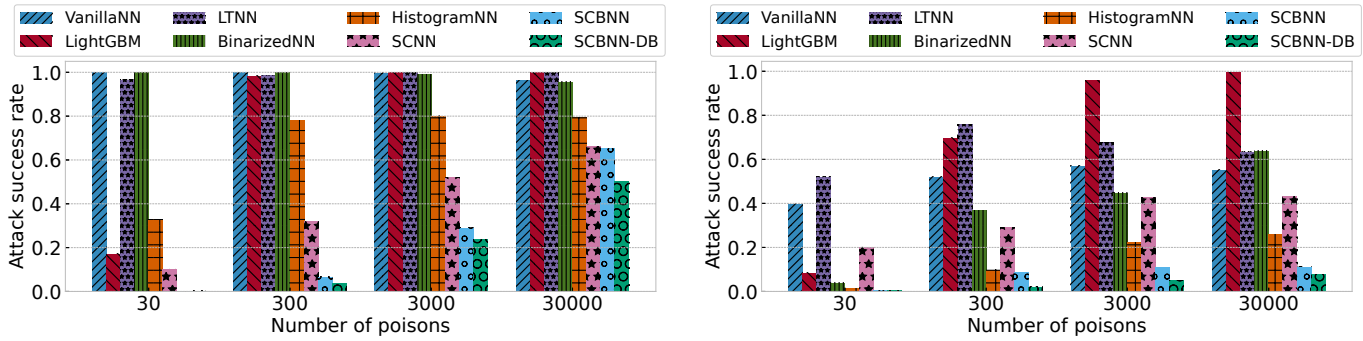
VR-based Trigger [79]: VR-based triggers excel in attacking malware detectors by targeting sparse feature-value regions, allowing substantial trigger weights without harming clean data performance.

EG-based Trigger [68]: EG-based triggers encompass three sub-variants, from which we select the greedy strategy. This strategy prioritizes the selection of feature-value combinations that are more aligned with benign orientations, as determined by SHAP values.

P-value-based Poison [79]: P-value-based poisons are generated based on samples difficult to be classified, forcing the model to rely on the trigger for classification. This approach is used for both VR- and EG-based triggers.

2) Settings: Given that knowing the feature set is quite important for backdoor attacks, we assume strong, adaptive attackers aware of the victim detectors' *feature set* and *compression strategies* for a fair comparison. Following the original papers [68], [79], we consider 16 controllable, non-hashed features, detailed in the Appendix F. In addition, adversaries can access detector outputs for p-value computation, common as vendors allow sample submission for detection. For EG-based backdoors, we assume direct model access for calculating SHAP values. The adversary then introduces "poisons" (watermarked benign samples) into the training set to implant a backdoor in the model.

3) Experimental Results: We incrementally added poisoned samples to evaluate our strategies' backdoor attack efficacy, with results in Figure 10(a) (VR-based) and Figure 10(b) (EG-based). SCB strategy excels in enhancing robustness against both backdoor attacks. With 3,000 poisons, the attack success rates are only 29.139% (VR-based) and 11.002% (EG-based)



(a) VR-based Backdoor Attacks

(b) EG-based Backdoor Attacks

Fig. 10: Evaluation of backdoor attacks on different models.

on SCBNN. Even at 10% poisoning, VR-based backdoors' attack success rate is only 65.226%, proving subspace compression's effectiveness. Other strategies, except original SC, show less robustness, with ASR for VR attacks reaching 80% at 300 poisons. Vanilla models are especially vulnerable, with 99.99% ASR for VR-based backdoors at just 30 poisons. SCB's slightly better robustness over SC is likely due to more retained features, causing the trigger to be relatively smaller. Density boosting strategies also moderately improve backdoor robustness. For SCBNN-DB, VR attack rates are 23.710% (1% poisons) and 50.352% (10% poisons). The results underscore that addressing sparsity is key to enhancing malware detectors' resilience against backdoor attacks.

4) Advanced attackers: Here, we examine SCBNN-DB's resilience against advanced adversaries capable of arbitrarily manipulating feature values to assess the robustness under the worst conditions. The quantity of poisons introduced is fixed at 3,000 (1% of the benign set). Results are in Figure 11.

With arbitrary feature changes, both backdoor types show increased efficacy. Yet, SCBNN-DB retains robustness at trigger sizes below 64. For instance, VR-based backdoors, utilizing 16 features, and EG-based backdoors demonstrate attack success rates of 35.268% and 47.003% on SCBNN-DB, respectively. Furthermore, as the trigger size expands to 128, the attack success rates increase to 99.952% for EG-based backdoors and 93.646% for VR-based backdoors. Notably, despite potent attacks, these features are all hash-based with interrelated changes, complicating control.

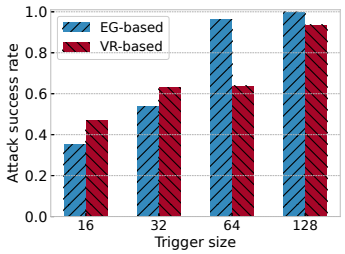


Fig. 11: Backdoor attacks with arbitrary feature modification.

Answer to RQ2:

Mitigating sparsity largely enhances models' robustness against backdoor attacks. Within all strategies, our proposal demonstrates the best robustness against backdoors.

D. Analysis of Sparsity and Evasion Attacks

This section evaluates model robustness against evasion attacks, focusing on two potent black-box, query-based techniques: GAMMA [29], [47] and MAB [73].

1) Implementing Evasion Attacks: We delineate the foundational principles underlying the evasion attacks as follows.

GAMMA: Demetrio et al. [47] introduce a genetic programming method for creating evasive PE malware examples. The approach frames the evasion as a constrained minimization task, balancing the evasion likelihood against payload size, The process iteratively (i) extracting payloads from goodwares, (ii) injecting them into target malware, and (iii) computing and selecting variants based on the objective function.

MAB-malware: MAB-malware [73] employs Reinforcement Learning (RL) for black-box evasion attacks under hard labels using a one-state Markov decision process. It selects from candidate actions like OA/SA (appending benign content/section), SP (appending random bytes), RC/RD/BC (clearing cert/debug/checksum), SR (section rename), and CR (code randomization) to explore and exploit reward likelihoods. After finding an evasive sample, it refines actions to the minimized version (details in Appendix D). MAB-malware aims to find a set of influential actions.

Settings: We download 1,000 malwares from Virusshare[1] to implement both attacks. To guarantee the quality of the malware, they are all labeled malicious by 40+ engines. We adopts the settings in the original paper where the number of queries is 60 for MAB and 100 for GAMMA, and therefore GAMMA's population size is 10 while the iteration of generation is 10.

2) Experimental Results: GAMMA attack results are shown in Figure 12. GAMMA is highly effective, surpassing 60% ASR on all models. Most compression strategies

[1]https://virusshare.com/about

showed limited defense, except LightGBM and SCNN. Notably, SCBNN provided no improvement, as SCB integrates sparse features to others, keeping their influence on model. Instead, the removal of these sparse features, as seen in the SCNN approach, does enhance robustness. Additionally, Density boosting slightly improves resilience. To further prove the effect of density boosting, we then combined density boosting with SCNN, and GAMMA only demonstrates 61.8% with F1 scores at 99.322%. Thus, mitigating sparsity does enhance robustness against GAMMA.

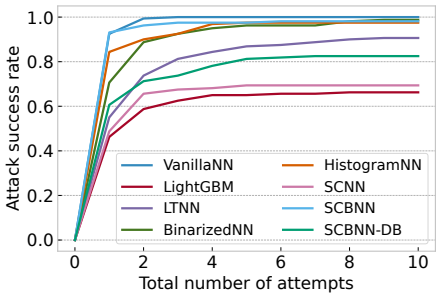


Fig. 12: Attack success rate of GAMMA.

MAB-malware demonstrates greater attack potency than GAMMA (see Figure 13) due to a wider range of actions. While models like LTNN and SCBNN-DB show slower increases in attack success, MAB-malware achieves over 90% for most models, with LTNN reaching a maximum of 80%.

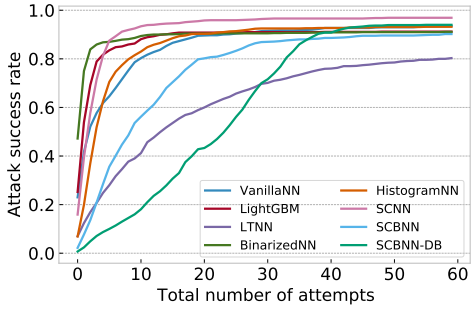


Fig. 13: Attack success rate of MAB malware.

Despite potent attacks, model behaviors vary, with Figure 14 showing successful evasion action sets' percentages. Models like SCBNN-DB require more actions for evasion; a single OA action achieves 37.44% evasion on SCBNN-DB. LightGBM is highly vulnerable, with 80% of malware undetectable by applying only one OA action. Notably, LTNN and BinarizedNN are more susceptible to SR (Section Rename) actions, with over 82% mislabeled benign by altering section names. This reveals LTNN and BinarizedNN's fragile decision-making. Our observations also highlight a limitation within the MAB-malware framework where the MAB-malware framework disproportionately increases the likelihood of selecting SR actions due to their high efficacy, leading to all subsequent operations being derived from the SR set and hindering higher evasion rates on LTNN models.

Unlike GAMMA's sequential attacks, MAB-malware evaluates actions in parallel across samples, finding strongly evasive sets. With limited queries and robust models, MAB-malware might not identify sufficient actions for high efficacy. As Table II shows, at 6,000 queries (100 samples, each with 60 queries), MAB-malware's average success rate is 37.18%, ranging from 8.53% to 85.621% on SCBNN-DB. This suggests that the decision of SCBNN-DB is somewhat more complex and robust to more actions than other strategies.

TABLE II: MAB-malware attacks at 6,000 queries.

Model	Attack success rate
VanillaNN	1.0
LightGBM	0.9588
LTNN	0.8454
BinarizedNN	0.8980
HistogramNN	0.9375
SCNN	0.8854
SCBNN	0.8351
SCBNN-DB	**0.3718**

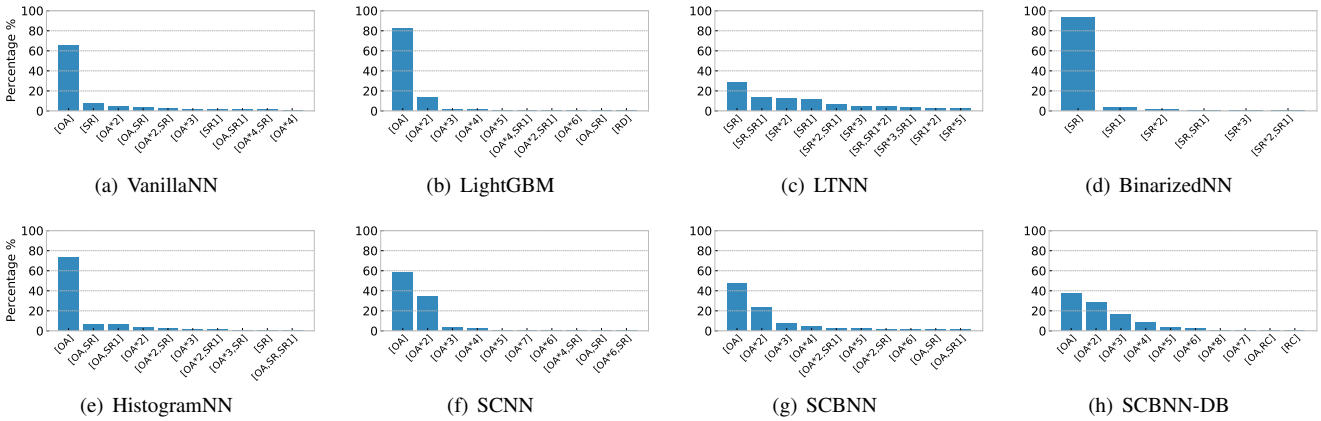
We see that both evasion attacks demonstrate strong attack effectiveness. This is because that they are not one-off but rather apply a range of actions persistently until the "malicious signal" predominates. Mitigating sparsity, albeit to a limited extent, does indicate an improvement in robustness against such practical evasions. It is noteworthy that our proposed method is complementary to other defensive strategies and can be effectively integrated with them.

3) Combining with other defensive methods: This part presents the integration of our methods with state-of-the-art defensive techniques to demonstrate improvements in model robustness. We primarily focus on PAD-SMA [39]. PAD-SMA (Principled Adversarial Detection) constructs a convex outer bound realized by **a DNN-based malware detector** and **an adversary detector**, both enhanced by adversarial training incorporating the Stepwise Mixture of Attacks. SMA selects the strongest variant from all generated adversarial examples in *every attack step*. Therefore, PAD-SMA computes and optimizes the worst-case loss within the convex outer bound, training a model that is provably robust to any norm-bounded adversarial attacks [39]. PAD-SMA was originally verified on an Android dataset with DREBIN features and demonstrated state-of-the-art defensive effects. We adopted the same settings as in the original paper. The logic is as follows: when a sample arrives, it is classified by a malware detector. If classified as benign, the adversary detector is then applied to check if adversarial. If identified as benign and non-adversarial, then returns a benign label (0); otherwise, returns 1 (detected as malicious) or undetectable (detected as adversarial).

We evaluated three models, VanillaNN+PAD[2], SCBNN+PAD, and SCBNN-DB+PAD, on the clean F1 scores, attack success rates for VR-based and EG-based backdoor attacks (3,000 poisons with 16 practical features), and both GAMMA and MAB-malware evasion attacks. The results are presented in Table III.

From the results, we draw several conclusions: firstly, *addressing sparsity remains crucial for EMBER datasets*; without compression, models exhibit significantly inferior clean

[2]The original feature values are standardized into continuous values between 0 and 1, and the trained model without PAD achieves 99.301% F1 score on the testing set.



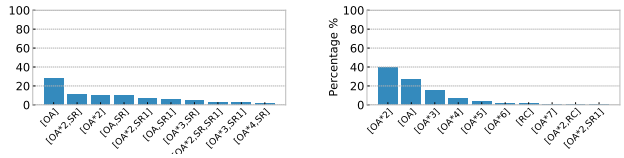
(a) VanillaNN (b) LightGBM (c) LTNN (d) BinarizedNN

(e) HistogramNN (f) SCNN (g) SCBNN (h) SCBNN-DB

Fig. 14: Action combinations that cause evasions.

TABLE III: Evaluation on models with PAD.

Metric	VanillaNN+PAD	SCBNN+PAD	SCBNN-DB+PAD
F1 score	0.97136	0.99240	**0.99362**
Rejected ratio	0.05	0.0235	0.0305
ASR on VRB	0.99021	0.13640	**0.01883**
ASR on EGB	0.63251	0.05210	**0.00875**
ASR on GAMMA	0.868	0.563	**0.256**
ASR on MAB	0.968	0.873	**0.806**



(a) SCBNN+PAD (b) SCBNN-DB+PAD

Fig. 15: Evasions on models with PAD.

performance, with PAD offering no robustness improvement. Secondly, *density boosting consistently enhances performance across different aspects*. Thirdly, *PAD further strengthens robustness against backdoor attacks* as PAD perturbs only malicious examples for adversarial training, preventing exacerbation of the backdoor effect in goodwares. Lastly, while MAB-malware consistently achieves high attack success rates across models, SCBNN+PAD and SCBNN-DB+PAD demonstrate robust to more actions, and therefore MAB-malware have to explore stronger attack budgets, as shown in Figure 15.

Oblivious Attacks: In prior scenarios, the adversary could adjust perturbations based on feedback from detectors to evade both the malware detector and the adversary detector. In an oblivious attack, only the malware detector's prediction is returned, yet the adversary detector still influences the final decision. Under this, GAMMA and MAB-malware see reduced success rates (19.8% and 76.2%) on SCBNN-DB+PAD. Further exploration of defenses against MAB-malware is needed but is beyond this paper's scope.

Given that malware detectors often pay less attention to perturbation size compared to image recognition [24], [53], conventional robust and adversarial training may not suffice [29], [39]. Our observations show that without an adversary detector GAMMA achieves a 68.522% attack success rate on SCBNN-DB+PAD without the adversary detector. Thus, in malware detection, prioritizing adversary detection over robust or adversarial training alone is advisable.

Answer to RQ3:

Addressing sparsity bolsters the resilience (despite marginal) against query-based evasion attacks, enhances the decision, and increases attackers' query budgets. More importantly, our proposal is complementary to other defensive strategies.

E. Evaluation on PDF datasets

Our research expands to include an evaluation on the Contagio PDF dataset[3], which encompasses a collection of 9,109 benign and 11,106 malicious PDF files. Each PDF file is decomposed into a 135-dimensional feature vector, derived from the features outlined in PDFRate [71].

We considered six models for evaluation:

- **Random Forest**, configured with default settings as presented in Mimicus [74],
- **NN**, a neural network trained on the original dataset,
- **LTNN/BinarizedNN/HistogramNN**, a neural network trained on a dataset with the corresponding processing method,
- **SCNN/SCBNN**, a neural network trained on a processed dataset with minimum density threshold 16%,
- **SCBNN-DB**, a neural network trained with density boosting on the aforementioned processed dataset.

[3]https://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html

- **SCBNN+PAD** and **SCBNN-DB+PAD**, SCBNN and SCBNN-DB integrated with PAD-SMA.

The dataset is divided into 70% training set and 30% test set, with the training set further split into 90% training set and 10% validation set. As shown in Table IV, SCBNN and SCBNN-DB show no significant gains due to high performance saturation. Models with PAD further improve performance by rejecting about 4% of inputs.

TABLE IV: Attack success rate on PDF datasets.

	F1 score	VRB	EGB	Mimicry ×1	Mimicry ×10	Mimicry ×30
Random Forest	0.99863	0.99370	0.99201	0.743	1.0	1.0
NN	0.99852	0.99119	0.98690	0.575	0.995	1.0
LTNN	0.99874	0.74498	0.18952	0.475	0.910	0.970
BinarizedNN	0.99849	0.32736	0.56511	0.230	0.765	0.935
HistogramNN	0.99892	0.40286	0.25122	0.320	0.950	0.990
SCNN	0.99849	0.40023	0.14560	0.205	0.76	0.9750
SCBNN	0.99897	0.22615	0.14880	0.281	0.862	0.977
SCBNN-DB	0.99882	0.01061	0.00068	0.334	0.893	1.0
SCBNN+PAD	0.99904	0.02105	0.02280	0.02	**0.053**	**0.075**
SCBNN-DB+PAD	**0.99939**	**0.00520**	**0.0**	**0.01**	0.06	0.085

Backdoor Attacks: We consider an adaptive adversary with access to the training set and model predictions, using 16 of 35 arbitrarily modifiable features and create 200 (1%) poisons for backdoor attacks. Table IV shows SCBNN-DB's near-perfect resilience. Moreover, the attack success rates for VR and EG backdoors minimal is at 8.136% and 5.136% even with 2,000 poisons. We also note that PAD integration does not compromise backdoor robustness.

Evasion Attacks: We adopt Mimicus [74] as evasion attacks, altering PDFs by inserting string patterns between the CRT and trailer, simulating benign behavior in all 68 modifiable features (see Appendix G). Mimicry$\times N$ denotes creating N adversarial instances from distinct N benign PDFs. Table IV shows that BinarizedNN, SCNN, SCBNN and SCBNN-DB have slightly better robustness (below 30% at Mimicry$\times$ 1) at lower mimicry levels, but all models fail beyond 30 instances. After all, PAD integration maintains significant robustness, confirming our method's compatibility with PAD techniques.

F. Evaluation on Android datasets

For evaluation on Android dataset, we use Federico et al. [22] provided dataset, which comprises 232,848 benign and 26,387 malicious apps sourced from AndroZoo [3]. The apps cover a time span of 5 years, ranging from January 2014 to December 2018. Specifically, training and validation are conducted using apps from the year 2014, while testing is performed over the remaining period. It is based on the same feature rule as DREBIN [7] as it is widely-adopted and demonstrates the best performance [22], [41], [60], [79]. DREBIN uses a binary feature space to abstract an app where components (activities, permissions, URLs, etc) are represented as present (1) or absent (0). The DREBIN feature is with binary values and extremely sparse (only 0.1% features appeared in at least 1% samples). While we are unable to apply the subspace compression strategies, but the feature selection and value bundling is still available. Therefore, we consider 1) a feature selection strategy which selects features whose density is larger than 1% and 2) a value bundle strategies: we first remove the features with occurrences less than 50, and bundling feature values whose density is smaller than 4%.

We conducted a comparative analysis of seven models:

- **SVM**, a support vector machine leveraging the complete set of 640,008 features for training [22],
- **NN-Selected**, a neural network trained on the 1,000 most "important" features identified through L1 regularization [14], [68], [79],
- **NN-Dense**, a neural network trained on 682 selected features exceeding 1% density threshold,
- **NN-Bundle**, a neural network trained on the remaining 558 features post-bundling to 4% density,
- **NN-DB**, a neural network trained with density boosting strategy on the remaining 558 bundled features,
- **NN-Bundle+PAD** and **NN-DB+PAD**, SCBNN and SCBNN-DB integrated with PAD-SMA.

The evaluation framework encompasses three key metrics: AUT(F1, 48m), backdoor robustness, and evasion robustness. We also consider evaluation on advanced APIGraph features [91], see Section VII-C.

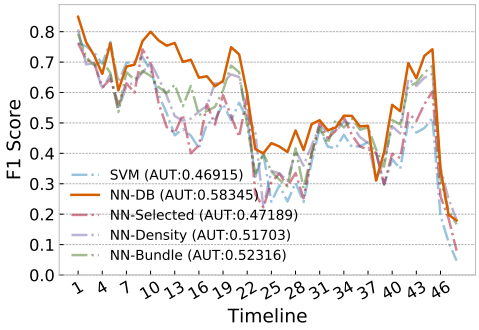


Fig. 16: Performance under concept drift scenarios.

AUT Analysis: Figure 16 shows performance profiles with **NN-DB** largely outperforming others, reaching an AUT (F1, 48m) of 58.345%. Models trained on dense features as **NN-Dense** and **NN-Bundle** outperformed **NN-Selected** on L1-regularized features and full-feature **SVM**. This supports the hypothesis that sparsity hinders performance. Notably, density boosting also enhances performance under PAD; see Table V.

Backdoor Attacks: Given DREBIN features' malleability [68], [79], we hypothesized an adaptive adversary with full feature manipulation and access to dataset/model. The backdoor is based on a 16-feature trigger and 1% benign dataset poisoning. Table V shows that backdoors significantly affected most models (even **NN-Bundle** and **NN-DB** showing limited resilience) with VR backdoors at a 91.523% success rate. A higher bundle ratio (16%) on **NN-DB** reduced the feature set to 176, lowering AUT to 52.253% and marginally improving backdoor resistance to 81.240% for VR attacks. This vulnerability stems from DREBIN's fully binarized and extremely sparse features. Bundling alone provided marginal defense, but combined with PAD, it mitigated backdoors substantially. Notably, dense features remain essential for DREBIN; **NN-Selected** with PAD achieved AUT of 49.852% and 74.812% ASR for VR backdoors, indicating insufficient defense.

TABLE V: Attack success rate on Android datasets.

	AUT(F1,48m)	VRB	EGB	Mimicry ×1	Mimicry ×10	Mimicry ×30
SVM	0.46915	0.83671	0.00102	0.316	0.721	0.840
NN-Selected	0.47189	0.99075	0.98690	0.491	0.788	0.858
NN-Dense	0.51703	0.96760	0.97770	0.634	0.939	0.977
NN-Bundle	0.52316	0.85580	0.80251	0.374	0.848	0.966
NN-DB	**0.58345**	0.91523	0.85532	0.290	0.784	0.962
NN-Bundle+PAD	0.52709	**0.13820**	0.0535	0.086	**0.182**	**0.256**
NN-DB+PAD	0.55567	0.16556	**0.0512**	**0.04**	0.204	0.316

Evasion Attacks: We also examined mimicry attacks, known for effectiveness against Android detectors [39]. Mimicry attackers [74] modify malware to mimic benign apps without internal model knowledge, querying models instead. Using N_{ben} benign examples, the attacker creates N_{ben} perturbed samples. Table V shows that while **NN-Bundle** and **NN-DB** show some robustness, all models fail against mimicry attacks at 30 steps (Mimicry×30). Combining with PAD-SMA still markedly enhanced robustness against mimicry evasions.

VII. Discussion

A. Ablation Study on Subspace Compression with Bundling

We delve into an ablation analysis of our strategy on Subspace Compression with Bundling (SCB) by meticulously examining each component's removal and its impact; see Figure 17. Notably, after incorporating outlier processing (Step 1), we observed a marked improvement in performance with the F1 score on EMBER testing set rising from 99.302% to 99.417% and the AUT on Sorel-20M soaring from 92.850% to 94.003%. This enhancement underscores the detrimental effect of extreme sparsity on performance. Our Step 1 incorporates a binarization mechanism designed to preserve classification information, the absence of which (as original SC's outlier processing) results in a lower F1 score of 99.228% and an AUT of 93.188%.

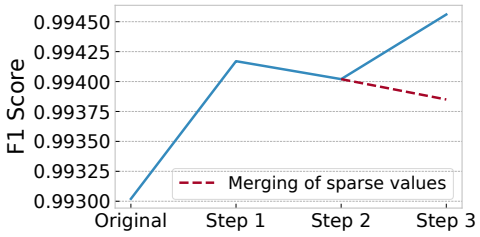


Fig. 17: Ablation study on Subspace Compression with Bundling.

Subsequently, we explored the impact of subspace combination (Step 2) which led to a minor fluctuation in performance, with the F1 score dipping to 99.402% and the AUT climbing to 94.422%. This outcome is attributed to the merging of non-binary feature values to a density just above 8%, leaving many binary features with sparse regions intact.

Ultimately, upon bundling all sparse features (Step 3), we observed a further enhancement in performance, with the F1 score reaching 99.456% and the AUT increasing to 94.444%. This outcome signifies that our SCB strategy effectively retains classification information even at high compression rates. We also tried directly merging sparse values instead of bundling, which results in the F1 scores dropping to 99.385% and the AUT decreasing to 93.788%.

B. Sparse Feature Elimination

We considered an alternative solution of removing features with sparse regions and evaluated the performance. Specifically, we referred to the VR-based backdoor strategy to assess sparsity, which involves dividing the feature value space into five subsections and calculating the variation ratio. A low variation ratio indicates that most values are concentrated within a subspace while others are extremely sparse. In this manner, we attempted to remove features with a variation ratio lower than a threshold (e.g., 0.01, where 99% of samples belong to a main subspace and only 1% of values exist in other subspaces). The results showed that after removing features with a variation ratio below 0.01, only 287 features remained, and the F1 score decreased to 99.041%, while the AUT on Sorel-20M decreased to 92.235%. We also considered removing features with variation ratio lower than 0.1, with only 64 features left and the F1 scores decreased to 98.547% and AUT decreased to 90.645%, see Table VI.

TABLE VI: Performance under Sparse feature elimination.

	feture kept	F1 score	AUT(F1,16m)
Original	2,381	0.99302	0.92850
VR≥0.01	287	0.99041	0.92235
VR≥0.1	64	0.98547	0.90645
SCB (8% density)	1,239	0.99456	0.94444

More importantly, sparse subspaces still exist; see Figure 18 for illustration where x-axis indicates the corresponding values, and we further verified that VR-based backdoors can still achieve a 96.237% attack success rate with 4 features and 30 poisons. Therefore, directly dropping features with large sparsity is not an effective solution.

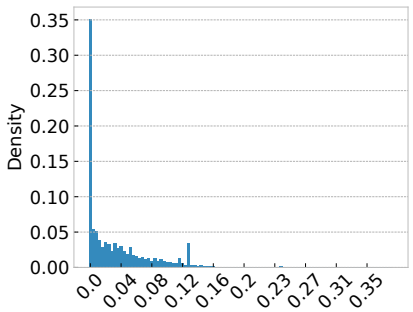


Fig. 18: Value distribution of ByteEntropyHistogram208.

C. Advanced API Features

In addition to processing the raw DREBIN feature set, we also consider some advanced API features, namely APIGraph [91], which associates and bundles all semantically-equivalent or similar APIs to stabilize the feature spaces, thus naturally slowing down classifier aging. We applied APIGraph to DREBIN and retrieved 148 APIGraph features.

We considered three models: 1) **NN-APIGraph-Selected**, trained on APIGraph-based features combined with another 1,000 important features selected using L1 regularization; 2) **NN-APIGraph-Dense**, trained on APIGraph features combined with 735 other features above 1% density; 3) **NN-APIGraph-Bundle**, trained on a dataset with APIGraph and our bundling applied sequentially (above 4% density) with 509 features left. Results show that NN-APIGraph-Selected shows an improved AUT of 51.549%, NN-APIGraph-Dense demonstrates an AUT of 52.963%, and NN-APIGraph-Bundle achieves 54.039%. We see that mitigating sparsity achieves better performance over APIGraph features, while our bundling strategy is orthogonal to APIGraph and does not diminish the performance gains provided by APIGraph. At last, we tried density boosting on the dataset used by NN-APIGraph-Bundle and the final AUT performance increased to 60.156%.

Given that APIGraph affects only API-based features, it did not enhance robustness against attacks. For example, VR-based backdoor still achieves 88.46% ASR, while Mimicry× 30 has 95.8% ASR.

VIII. Conclusion

This paper elucidates the detrimental effects of sparsity on model performance and underscores its pervasiveness, and introduces a novel subspace compression technique coupled with density-boosting training, demonstrating that mitigating sparsity can enhances model performance, robustness, and sustainability simultaneously. The proposal is verified compatible with existing defensive strategies.

Acknowledgment

This research / project is supported by the National Research Foundation, Singapore, and the Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme (Proposal ID: NCR25-DeSCEmT-SMU). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore, and the Cyber Security Agency of Singapore.

References

[1] A. Abusnaina, A. Khormali, H. Alasmary, J. Park, A. Anwar, U. Meteriz, and A. Mohaisen, "Examining adversarial learning against graph-based iot malware detection systems," *arXiv*, 2019.

[2] A. Al-Dujaili, A. Huang, E. Hemberg, and U. M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *Proc. of SPW*, 2018.

[3] K. Allix, T. Bissyandé, J. Klein, and Y. Traon, "Androzoo: collecting millions of android apps for the research community," in *Proc. of MSR*, 2016.

[4] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *Proc. of IWCMC*, 2013.

[5] H. Anderson and P. Roth, "EMBER: an open dataset for training static PE malware machine learning models," *CoRR*, 2018.

[6] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," *arXiv*, 2018.

[7] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proc. of NDSS*, 2014.

[8] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, "Enhancing robustness of machine learning systems via data transformations," in *Proc. of CISS*, 2018.

[9] B. Biggio, I. Corona, Z. He, P. P. K. Chan, G. Giacinto, D. S. Yeung, and F. Roli, "One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time," in *Proc. of MCS*, 2015.

[10] L. Chen, S. Hou, Y. Ye, and S. Xu, "Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks," in *Proc. of ASONAM*, 2018.

[11] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *Computers and Security*, 2018.

[12] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE Trans. Inf. Forensics Secur.*, 2020.

[13] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust PDF malware classifiers," in *Proc. of USENIX Security*, 2020.

[14] Z. Chen, Z. Zhang, Z. Kan, L. Yang, J. Cortellazzi, F. Pendlebury, F. Pierazzi, L. Cavallaro, and G. Wang, "Is It Overkill? Analyzing Feature-Space Concept Drift in Malware Detectors," *CoRR*, 2023.

[15] Z. Chua, S. Shen, P. Saxena, and Z. Liang, "Neural nets can learn function type signatures from binaries," in *Proc. of USENIX Security*, 2017.

[16] G. Chuan, R. Mayank, C. Moustapha, and V. Laurens, "Countering adversarial images using input transformations," *Proc. of ICLR*, 2018.

[17] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proc. of ICML*, 2019.

[18] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, and F. Roli, "Adversarial exemples: A survey and experimental evaluation of practical attacks on machine learning for windows malware detection," *ACM Trans. Priv. and Secu.*, 2021.

[19] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on android malware detection," *CoRR*, 2017.

[20] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks," in *Proc. of USENIX Security*, 2019.

[21] A. Deo, S. K. Dash, G. Suarez-Tangil, V. Vovk, and L. Cavallaro, "Prescience: Probabilistic guidance on the retraining conundrum for malware detection," in *Proc. of AISec*, 2016.

[22] B. Federico, P. Feargus, P. Fabio, and C. Lorenzo, "Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift," *Proc. of Usenic Security*, 2020.

[23] D. Gibert, G. Zizzo, Q. Le, and J. Planes, "Adversarial robustness of deep learning-based malware detectors via (de)randomized smoothing," *IEEE Access*, 2024.

[24] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. of ICLR*, 2015.

[25] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Proc. of NeurIPS*, 2022.

[26] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *CoRR*, 2016.

[27] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *CoRR*, 2017.

[28] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," in *Proc. of ICDM*, 2022.

[29] Z. Huang, N. G. Marchant, K. Lucas, L. Bauer, O. Ohrimenko, and B. I. P. Rubinstein, "Rs-del: Edit distance robustness certificates for sequence classifiers via randomized deletion," *Proc. of NeurIPS*, 2023.

[30] I. Incer, M. Theodorides, S. Afroz, and D. A. Wagner, "Adversarially robust malware detection using monotonic classification," in *Proc. of IWSPA*, 2018.

[31] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. of USENIX Security*, 2017.

[32] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Investigating labelless drift adaptation for malware detection," in *Proc. of AISec*, 2021.

[33] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proc. of NeurIPS*, 2017.

[34] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "Rhmd: Evasion-resilient hardware malware detectors," in *Proc. of MICRO*, 2017.

[35] D. Kirat and G. Vigna, "Malgene: Automatic extraction of malware analysis evasion signature," in *Proc. of CCS*, 2015.

[36] M. Krcál, O. Svec, M. Bálek, and O. Jasek, "Deep convolutional malware classifiers can learn from raw executables and labels only," *Proc. of ICLR*, 2018.

[37] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv*, 2018.

[38] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *Proc. of SP*, 2019.

[39] D. Li, S. Cui, Y. Li, J. Xu, F. Xiao, and S. Xu, "PAD: Towards Principled Adversarial Malware Detection Against Evasion Attacks," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[40] D. Li and Q. Li, "Adversarial deep ensemble: Evasion attacks and defenses for malware detection," *IEEE Transactions on Information Forensics and Security*, 2020.

[41] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge," *arXiv*, 2018.

[42] D. Li, Q. Li, Y. F. Ye, and S. Xu, "Arms Race in Adversarial Malware Detection: A Survey," *ACM Computing Surveys*, 2023.

[43] H. Li, S. Zhou, W. Yuan, X. Luo, C. Gao, and S. Chen, "Robust android malware detection against adversarial example attacks," *Proc. of WWW*, 2021.

[44] J. Li *et al.*, "Recent advances in end-to-end automatic speech recognition," *APSIPA Transactions on Signal and Information Processing*, 2022.

[45] X. Ling, L. Wu, J. Zhang, Z. Qu, W. Deng, X. Chen, Y. Qian, C. Wu, S. Ji, T. Luo *et al.*, "Adversarial attacks against windows pe malware detection: A survey of the state-of-the-art," *Computers & Security*, 2023.

[46] S. Lo and V. M. Patel, "Error diffusion halftoning against adversarial examples," in *Proc. of ICIP*, 2021.

[47] D. Luca, B. Battista, L. Giovanni, R. Fabio, and A. Alessandro, "Functionality-preserving black-box optimization of adversarial windows malware," *IEEE Trans. Inf. Fore. Secu.*, 2021.

[48] K. Lucas, S. Pai, W. Lin, L. Bauer, M. K. Reiter, and M. Sharif, "Adversarial training for raw-binary malware classifiers," in *Proc. of USENIX Security*, 2023.

[49] S. Lundberg and S. Lee, "A unified approach to interpreting model predictions," in *Proc. of NeurIPS*, 2017.

[50] D. Maiorca, B. Biggio, and G. Giacinto, "Towards robust detection of adversarial infection vectors: Lessons learned in pdf malware," *arXiv*, 2018.

[51] D. Maiorca, A. Demontis, B. Biggio, F. Roli, and G. Giacinto, "Adversarial detection of flash malware: Limitations and open issues," *Computers & Security*, 2020.

[52] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *Proc. of CCS*, 2017.

[53] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proc. of CVPR*, 2016.

[54] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proc. of AISec*, 2017.

[55] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "Context-aware, adaptive, and scalable android malware detection through online learning," *IEEE Trans. Emerg. Top. Comput. Intell.*, 2017.

[56] S. Narisada, Y. Matsumoto, S. Hidano, T. Uchibayashi, T. Suganuma, M. Hiji, and S. Kiyomoto, "Countermeasures against backdoor attacks towards malware detectors," in *Proc. of CANs*, 2021.

[57] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. J. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version)," *ACM Trans. Priv. Secur.*, 2019.

[58] M. Osadchy, J. C. Hernandez-Castro, S. J. Gibson, O. Dunkelman, and D. Pérez-Cabo, "No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to CAPTCHA generation," *IEEE Trans. Inf. Forensics Secur.*, 2017.

[59] M. Paknezhad, C. P. Ngo, A. A. Winarto, A. Cheong, C. Y. Beh, J. Wu, and H. K. Lee, "Explaining adversarial vulnerability with a data sparsity hypothesis," *Neurocomputing*, 2022.

[60] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "Tesseract: Eliminating experimental bias in malware classification across space and time," in *Proc. of Usenix Security*, 2019.

[61] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ML attacks in the problem space," in *Proc. of SP*, 2020.

[62] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset shift in machine learning*. Mit Press, 2008.

[63] E. Quiring, L. Pirch, M. Reimsbach, D. Arp, and K. Rieck, "Against All Odds: Winning the Defense Challenge in an Evasion Competition with Diversification," *CoRR*, 2020.

[64] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole EXE," in *Proc. of AAAI*, 2018.

[65] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *CoRR*, 2018.

[66] E. M. Rudd, F. N. Ducau, C. Wild, K. Berlin, and R. Harang, "Aloha: Auxiliary loss optimization for hypothesis augmentation," in *Proc. of USENIX Security*, 2019.

[67] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *computer vision*, 2015.

[68] G. Severi, J. Meyer, S. E. Coull, and A. Oprea, "Explanation-guided backdoor poisoning attacks against malware classifiers," in *Proc. of USENIX Security*, 2021.

[69] T. Shapira, D. Berend, I. Rosenberg, Y. Liu, A. Shabtai, and Y. Elovici, "Being single has benefits. instance poisoning to deceive malware classifiers," *CoRR*, 2020.

[70] A. Singh, A. Walenstein, and A. Lakhotia, "Tracking concept drift in malware families," in *Proc. of AISec*, 2012.

[71] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *Proc. of ACSAC*, 2012.

[72] ——, "When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors." in *Proc. of NDSS*, 2016.

[73] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "MAB-Malware: A Reinforcement Learning Framework for Blackbox Generation of Adversarial Malware," *Proc. of Asia CCS*, 2022.

[74] N. Srndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study," in *Proc. of SP*, 2014.

[75] O. Suciu, R. Marginean, Y. Kaya, H. D. III, and T. Dumitras, "When does machine learning fail? generalized transferability for evasion and poisoning attacks," in *Proc. of USENIX Security*, 2018.

[76] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *Proc. of SPW*, 2019.

[77] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *Proc. of ICLR*, 2014.

[78] G. Tao, Y. Liu, G. Shen, Q. Xu, S. An, Z. Zhang, and X. Zhang, "Model Orthogonalization: Class Distance Hardening in Neural Networks for Better Security," *Proc. of SP*, 2022.

[79] J. Tian, K. Qiu, D. Gao, Z. Wang, X. Kuang, and G. Zhao, "Sparsity brings vulnerabilities: Exploring new metrics in backdoor attacks," in *Proc. of USENIX Security*, 2023.

TABLE VII: Description of Action set — Types and Abbreviations

Type	Abbr	Name	Description
Macro	OA	Overlay Append	Appends benign contents at the end of a binary
	SP	Section Append	Appends random bytes to the unused space between sections.
	SA	Section Add	Adds a new section with benign contents.
	SR	Section Rename	Change the section name to a name in benign binaries.
	RC	Remove Certificate	Zero out the signed certificate of a binary.
	RD	RD Remove Debug	Zero out the debug information in a binary.
	BC	Break Checksum	Zero out the checksum value in the optional header.
	CR	Code Randomization	Replace instruction sequence with semantically equivalent one.
Micro	OA1	Overlay Append 1 Byte	Appends 1 byte at the end of a binary
	SP1	Section Append 1 Byte	Appends 1 byte to the unused space at the end of a section.
	SA1	Section Add 1 Byte	Adds a new section with 1 byte content.
	SR1	SR1 Section Rename 1 Byte	Change 1 byte of a section name.
	CP1	Code Section Append 1 Byte	Appends 1 byte to the unused space at the end of the code section.

[80] L. Tong, B. Li, C. Hajaj, C. Xiao, N. Zhang, and Y. Vorobeychik, "Improving robustness of ml classifiers against realizable evasion attacks using conserved features." in *Proc. of USENIX Security*, 2019.

[81] D. A. Van Dyk and X.-L. Meng, "The art of data augmentation," *Journal of Computational and Graphical Statistics*, 2001.

[82] X. Wang, C. Liu, X. Hu, Z. Wang, J. Yin, and X. Cui, "Make data reliable: An explanation-powered cleaning on malware dataset against backdoor poisoning attacks," in *Proc. of ACSAC*, 2022.

[83] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proc. of ICML*, 2018.

[84] K. Xu, Y. Li, R. H. Deng, and K. Chen, "Deeprefiner: Multi-layer android malware detection system applying deep neural networks," in *Proc. of EuroSP*, 2018.

[85] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *Proc. of EuroSP*, 2019.

[86] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Proc. of NDSS*, 2018.

[87] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," *Proc. of USENIX Security*, 2021.

[88] L. Yang, Z. Chen, J. Cortellazzi, F. Pendlebury, K. Tu, F. Pierazzi, L. Cavallaro, and G. Wang, "Jigsaw Puzzle: Selective Backdoor Attack to Subvert Malware Classifiers," *Proc. of SP*, 2023.

[89] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, "A survey of modern deep learning based object detection models," *Digital Signal Processing*, 2022.

[90] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," *arXiv*, 2019.

[91] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proc. of CCS*, 2020.

[92] Y. Zhu, J. Sun, and Z. Li, "Rethinking adversarial transferability from a data distribution perspective," in *Proc. of ICLR*, 2022.

APPENDIX

A. Illustration of previous processing methods

Here we provide illustrative examples to explain why previous processing methods (including feature squeezing with 4-bit depth [86], logistic transformation [66], histogram [33], and binarization [80]) did not effectively eliminate sparse regions, see Figure 19. We apply these processing methods on the EMBER feature, `num_write_sections`. As observed, only binarization mitigates the sparsity, and the others still exhibit a clearly sparse area. However, binarization is not effective for all features if a feature contains an extremely large or small number of zeros; see `export_libs_hash66` in Figure 6.

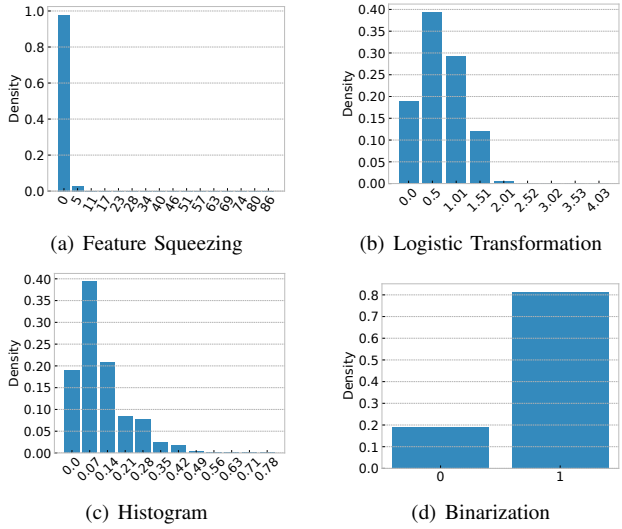


Fig. 19: Examples of applying different processing methods.

B. Performance Evaluation on Varied Level of Label Distribution

Here, we demonstrate that considering label distribution is unnecessary for subspace compression. Specifically, we enforce a different combination rule that spaces whose density lower than threshold 0.08 are iteratively combined with one of its neighbour sections based on Jenson's Shannon (JS) divergence (calculated based on the label distribution) and the density of the target section. We use the JS divergence instead of KL divergence because JS divergence has a confined value space (0-1), and therefore we can incorporates the density of the subspace into consideration. In detail, the line 21 in Algorithm 3, $y_l \leq y_r$, is replaced with $\alpha\times$ JSDiv(y_m,y_l) $+(1-\alpha)\times\frac{y_l}{y_l+y_r} \leq \alpha\times$ JSDiv(y_m,y_r) $+(1-\alpha)\times\frac{y_r}{y_l+y_r}$. In this way, we can adjust α from 0 (density only) to 1 (label distribution only) to control the significance of label distribution during the combination. The results can be found in Figure 20 which shows that varied α demonstrate minimal effect on the performance. To be noted, the model is trained on dataset before value bundle stage.

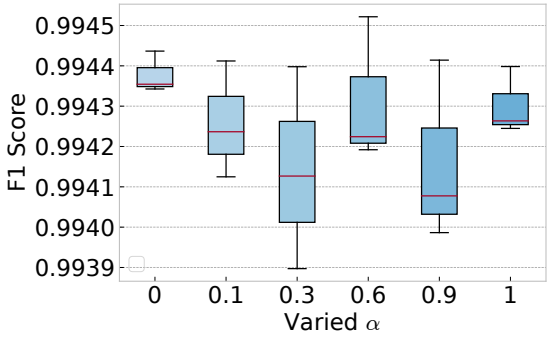


Fig. 20: Clean performance under varied α.

C. Applying our strategies on dense datasets

Given that our strategies are theoretically feasible to all tabular datasets, we then apply them to the seven dense datasets mentioned in Section sec:motivation. We consider a small density (1%) since there is no attack consideration. The results can be found in Table VIII. We can see that mitigating sparsity does not consistently improve the performance on such dense datasets. In detail, Covertype contains more than 420K samples, indicating a higher density. Therefore, mitigating sparsity may not be so important anymore especially when there is no attack consideration.

	VanillaNN	SCBNN	SCBNN-DB
Electricity	0.85728	0.88112	**0.88832**
Evemove	0.62529	**0.65331**	0.64433
Covertype	**0.96525**	0.96346	0.96186
Albert	0.67737	0.68006	**0.69436**
Credit	0.70210	**0.70921**	0.70731
Road	0.80214	0.79889	**0.80235**
Compas	0.67490	0.66728	**0.68170**

TABLE VIII: F1 scores on the seven dense datasets.

D. The action sets of MAB-malware framework

Table VII shows the action sets used in MAB-malware framework.

E. Density Boosting with different settings

This section presents an evaluation of various density boosting configurations based on the EMBER dataset. We examined four distinct settings for comparison:

1) Zero-filled: Selected features are filled with zeros instead of sparse values.
2) Random-filled: Selected features are filled with uniformly random values.
3) Fixed-filled: A fixed percentage, such as 10%, of features are filled with sparse values.
4) Varied-filled: A variable percentage, ranging from 1% to 15%, of features are filled with sparse values.

Our assessment was based on three metrics: F1 scores, Attack Success Rate (ASR) for VR-based backdoors with 3,000 poisons utilizing a 16-feature practical trigger, and GAMMA evasions. The results are summarized in Table IX, leading to several conclusions:

TABLE IX: Evaluation on density boosting strategies with different settings.

	F1	VRB	GAMMA
Zero-filled	0.99481	0.52178	0.921
Random-filled	0.99476	0.33249	0.872
Fixed-filled	0.99479	0.22785	0.905
Varied-filled	**0.99488**	0.23710	0.869
More-filled	0.99458	**0.12235**	**0.835**

1) Zero-filled settings improved F1 scores but exhibited reduced robustness against backdoor and evasion attacks, as detailed in Table IX.
2) Random-filled settings resulted in marginally lower F1 scores and diminished robustness against backdoors, as illustrated in Table IX.
3) Fixed-filled settings matched the robustness against backdoors comparable to our current approach but demonstrated slightly inferior performance and lacked defense against GAMMA evasion, as observed in Table IX. Moreover, when the poisoning size increased to 10%, Fixed-filled underperformed compared to Varied-filled, with VR-based backdoors achieving an ASR of 60.124% for Fixed-filled versus 50.352% for Varied-filled.

Consequently, we selected the varied (1%-15%) perturbation with sparse values as our final configuration. We also experimented with filling a larger proportion of features (1%-30%) with sparse values; however, while this approach slightly enhanced robustness against attacks, it led to a decrease in F1 scores to 99.458%, as indicated in Table IX.

F. The practical features under problem space consideration

Table X lists the 16 practical features used for implementing backdoors.

TABLE X: The practical features under problem space consideration.

Features
path_count
url_count
registry_count
MZ_count
timestamp
num_write_section
num_execute_section
num_zero_size_sections
num_unnamed_sections
major_image_version
minor_image_version
major_linker_version
minor_linker_version
major_operating_system_version
minor_operating_system_version
minor_subsystem_version

G. The modifiable features for PDFRate features

Within the 135 disclosed features, Mimicus [74] allow for modification on 68 features, where 33 is increment-only while the other 35 are all editable. Table G lists the 68 modifiable features in PDFRate [71] feature set.

Features with IncrementOnly Values (33 in total)	Features with Editable Values (35 in total)
count_acroform	author_dot
count_image_xlarge	author_lc
count_acroform_obs	author_num
count_image_xsmall	author_oth
count_action	author_uc
count_javascript	createdate_ts
count_js	createdate_tz
count_objstm	creator_dot
count_objstm_obs	creator_lc
count_page	creator_num
count_stream	creator_oth
count_trailer	creator_uc
count_xref	producer_dot
size	producer_lc
count_box_a4	producer_num
count_box_legal	producer_oth
count_box_letter	producer_uc
count_box_other	version
count_box_overlap	
count_endobj	
count_endstream	
count_eof	
count_font	
count_font_obs	
count_image_large	
count_image_med	
count_image_small	

TABLE XI: Supported features in the MIMICUS framework for modification.

H. Algorithms for Subspace Compression with Bundling

We present algorithms for the outlier processing and subspace combination and value bundling, see Algorithm 1, 3 and 2.

Algorithm 1 Process outliers with binarization

Input: Feature values x;

$p_{25}, p_{50}, p_{75} = \text{Percentile}(x, [25, 50, 75])$
if $p_{25} \neq p_{75}$ **then**
  $iqr = p_{75} - p_{25}$
  $lb = p_{25} - 3 \times iqr$
  $ub = p_{75} + 3 \times iqr$
  $x[x < lb] = lb$
  $x[x > ub] = ub$
else
  $indicies = x == p_{25}$
  $x[indicies] = 0$
  $x[\sim indicies] = 1$
end if

Algorithm 2 Value Bundling Algorithm

Input: Dataset matrix X; Threshold H;

while True **do**
  Find the feature with the lowest density, F_{min}, in the dataset
  if The lowest density in feature F_{min} is greater than the threshold H **then**
    Terminate the loop
  end if
  Find the feature F_{target} that has the least conflict with feature F_{min}
  Bind feature F_{min} to F_{target}
  Delete feature F_{min}
end while

Algorithm 3 Subspace Compression with Early Stop

Input: Feature values x; Labels Y; Threshold H;

$x = \text{ProcessOutliers}(x)$
if len(set(x))> 100 **then**
  $x = \text{Histogram}(x, 100)$
end if
$valueset = \text{sorted(set}(x))$
for v in $valueset$ **do**
  $densities[m] = \text{len}(x[x{==}v])/\text{len}(x)$
end for
md = min($densities$)
$index$ = $valueset$.index(md)
while $md < H$ and len($densities$)> 2 **do**
  v = $valueset[index]$
  if $index == 0$ **then**
    $target = index + 1$
  else if $index ==$ len($densities$) $- 1$ **then**
    $target = index - 1$
  else
    $y_l = Y[\text{x} == \text{valueset}[index - 1]]$
    $y_m = Y[\text{x} == \text{valueset}[index]]$
    $y_r = Y[\text{x} == \text{valueset}[index + 1]]$
    if len(y_l) $\leq$ len(y_r) **then**
      $target = index - 1$
    else
      $target = index + 1$
    end if
    $x[x == \text{valueset}[index]] = \text{valueset}[target]$
  end if
  $densities[target] += densities[index]$
  del $densities[index]$
  del $valueset[index]$
  md = min($densities$)
  $index$ = $valueset$.index(md)
end while