Addressing Malware Family Concept Drift with Triplet Autoencoder

Numan Halit Guldemir
Centre for Secure Information Technologies
Queen's University Belfast
United Kingdom
nguldemir01@qub.ac.uk

Oluwafemi Olukoya
Centre for Secure Information Technologies
Queen's University Belfast
United Kingdom
o.olukoya@qub.ac.uk

Jesús Martínez-del-Rincón
Centre for Secure Information Technologies
Queen's University Belfast
United Kingdom
j.martinez-del-rincon@qub.ac.uk

***Abstract*—Machine learning is increasingly vital in cybersecurity, especially in malware detection. However, concept drift —where the characteristics of malware change over time— poses a challenge for maintaining the efficacy of these detection systems. Concept drift can occur in two forms: the emergence of entirely new malware families and the evolution of existing ones. This paper proposes an innovative method to address the former, focusing on effectively identifying new malware families. Our approach leverages a supervised autoencoder combined with triplet loss to differentiate between known and new malware families. We create clear and robust clusters that enhance the accuracy and resilience of malware family classification by utilizing this metric learning technique and the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. The effectiveness of our method is validated using an Android malware dataset and a Windows Portable Executable (PE) malware dataset, showcasing its capability to sustain model performance within the dynamic landscape of emerging malware threats. Our results demonstrate a significant improvement in detecting new malware families, offering a reliable solution for ongoing cybersecurity challenges.**

***Keywords*-Concept drift; Windows PE malware; temporal analysis; triplet loss; autoencoder; metric learning.**

I. Introduction

Machine learning has become a key tool in cybersecurity, particularly for detecting malware. These systems, when well-trained, are highly effective at identifying threats. However, the effectiveness of these systems is continually challenged by the dynamic nature of malware. As cyber threats rapidly evolve, machine learning models that were once effective can quickly become obsolete — a phenomenon known as concept drift [1]. Concept drift in malware detection is often driven by two major factors: developing entirely new malware and modifying existing malware to evade detection systems. A report by AV-Test indicates approximately 320,000 new malware samples emerge daily, underscoring the need for continuous model adaptation to unseen threats [2].

Retraining models frequently is one common strategy to fight against concept drift, but it has problems. Firstly, in cybersecurity, accurately labeling data is essential but costly, as it often requires experts to examine and classify new threats [3]. Moreover, determining the precise timing for updating or retraining a model is not straightforward [4]. Frequently retrained models might fail to keep up with the latest malware without a reliable method to decide when updates are necessary, resulting in security gaps.

Labeling unknown samples is crucial to ensure that the model does not misclassify them and can be accurately analyzed by experts. Machine learning models typically generate a probability distribution over the known class labels, always selecting the most likely class. Ideally, for an unknown input, all classes should exhibit low probabilities, and setting a threshold based on uncertainty should reject these unknown classes. However, recent studies have shown that even inputs far from any known class can produce high probability/confidence scores [5]. This leads to misleadingly high confidence scores even when the model's predictions are incorrect. Neural networks, in particular, tend to produce overly confident predictions in such scenarios, creating a false sense of reliability [6].

This paper addresses the challenges associated with concept drift in malware family detection by proposing a novel neural architecture and training paradigm tailored to this issue. It is important to clarify that this work focuses solely on analyzing and differentiating between malware samples. Thus, we assume that all input samples are malware, and the goal is to identify and address variations within malware families rather than distinguishing between malicious and benign software. To achieve this, we leverage metric learning to map input samples to their respective families based on their proximity to the centroids of known classes. This method enhances the model's generalization ability by constructing a feature space that accurately reflects the similarities and differences between samples. Additionally, we incorporate triplet loss to refine this feature space further, forming distinct, compact clusters that improve the accuracy of assignments. This approach ensures that samples from the same family are closely grouped, while those from different families are pushed apart. Importantly, our work specifically addresses concept drift in detecting new, unknown malware families, rather than focusing on the evolution of existing malware families. This distinction is critical because a related, yet underexplored, aspect of concept drift involves the automated detection of emerging malware families for multi-class classification purposes. While existing approaches may employ drift signaling techniques to determine when to retrain binary classification models, the automatic identification of new malware families —those that significantly deviate from historical data— poses a more intriguing challenge [7].

Our experiments demonstrate the effectiveness of the pro-

posed approach in detecting newly emerging malware families, offering valuable insights into maintaining robust defense mechanisms against rapidly evolving cyber threats. The results show a significant improvement in detection performance for new malware, providing a reliable and adaptive solution to keep pace with the dynamic landscape of cyber threats.

The main contributions of this paper are as follows:

- We present a method that leverages metric learning and DBSCAN for family clustering to address concept drift in malware analysis.
- We propose a neural network architecture that utilizes an autoencoder and triplet loss for robust malware family detection.
- We extensively evaluate our approach using two relevant benchmark datasets for Android and Windows PE malware detection, as commonly used in the literature.

The rest of the paper is organized as follows: Section II covers related work, discussing existing approaches in malware detection, metric learning, and concept drift. Section III explains our proposed method, detailing how we effectively utilize an autoencoder with triplet loss to differentiate between known and new malware families. Section IV details the datasets we used and the experimental setup. Section V presents our experiments' results and evaluates our approach's effectiveness. Section VI discusses the limitations of our study, highlighting potential areas for improvement. Finally, Section VII concludes the paper and discusses potential future work.

II. Related work

A. Machine Learning in Malware Detection

Numerous studies have leveraged machine learning techniques to enhance the detection of malware and its various families by utilizing a range of features. These include static features, which are extracted from the malware without executing it to reveal the underlying code structure, such as Application Programming Interface (API) function calls [8][9], bytes [10][11] and opcodes [12]–[14]. Additionally, some research has focused on using a set of these static features [15]–[17]. Dynamic features, on the other hand, capture the behavior of malware during its execution and include data, such as API call traces [18][19], instruction traces [20], and network traffic [21]. Some studies have furthered this approach by converting these inputs into visual formats, aiding in recognizing malicious patterns [8][22]. Machine learning thus plays a crucial role in strengthening cybersecurity measures against diverse threats.

B. Metric Learning

Recent advancements in metric learning have demonstrated their efficacy in learning high-quality data representations for tasks involving object differentiation and semantic similarity across various fields, such as computer vision [23]–[25], audio processing [26]–[28], and bioinformatics [29][30]. Metric learning has also garnered significant attention from security researchers due to its potential to enhance malware detection capabilities. Wu *et al.* [31] introduce IFDroid, a system that leverages contrastive learning to enhance the robustness and accuracy of Android malware family classification. IFDroid trains an encoder to extract features resilient to code obfuscation by converting function call graphs into images. Similarly, Jureček and Lórencz [32] employ Particle Swarm Optimization to optimize feature weights for a weighted Euclidean distance metric, improving the accuracy of k-Nearest Neighbors (k-NN) classification. Building on these concepts, Liu *et al.* [33] model the execution behavior of malware as heterogeneous graphs, capturing interactions between entities, such as APIs, processes, and files. This approach uses data augmentations and graph attention networks to generate robust positive and negative samples, enabling effective few-shot detection of malware variants without extensive labeled data. Andresini *et al.* [34] present a method for network intrusion detection that combines autoencoders and triplet networks to improve predictive accuracy by addressing data imbalance and enhancing the separation of normal and malicious network traffic.

C. Concept Drift

In recent years, various approaches have been proposed to address concept drift in different domains, particularly in malware and intrusion detection systems. Singh *et al.* [35] explored concept drift in malware detection by introducing new tracking methods and examining different types of malware evolution. Building on this, Jordaney *et al.* [4] developed the TRANSCEND framework, which identifies concept drift in classification models through statistical metrics and a conformal evaluator to assess model credibility and confidence. The framework was further refined in TRANSCENDENT [36], which formalized the theoretical foundation of conformal evaluation and introduced new evaluators to enhance robustness. Additionally, CADE [37] employed contrastive learning to detect outliers and explain concept drift by mapping data samples into a low-dimensional space. In the realm of intrusion detection, Andresini *et al.* [38] proposed the INSOMNIA framework, which integrates incremental, active, and transfer learning to adapt to non-uniform data distribution over time, thus maintaining the efficacy of intrusion detection models through continuous updates and active learning strategies. Moreover, Zola *et al.* [39] conducted a temporal analysis of distribution shifts in malware classification, proposing a three-step forensic exploration approach to understand model failures caused by concept drift. These diverse methodologies highlight the evolving landscape of concept drift management in cybersecurity applications.

Many approaches do not adequately consider the issue of concept drift, which can lead to a decline in detection accuracy over time [8][13][31]. Additionally, some methods overlook the complexity of dealing with multiple clusters within a single malware family and fail to address the presence of outliers effectively [4][37]. These gaps highlight the need for more robust and adaptive solutions in malware family detection.

III. METHOD

Our method enhances the robustness and performance of detection models through a structured approach to address the challenges posed by concept drift in malware family classification. Initially, high-dimensional feature vectors are extracted from the malware samples. These vectors are then processed using an autoencoder, trained with triplet loss to reduce dimensionality while ensuring that similar points are closer together and dissimilar points are further apart in the latent space. Subsequently, clustering in the latent space using the DBSCAN algorithm is performed to identify sub-clusters within malware families and exclude outliers. Finally, centroids of these clusters are calculated to determine the classification of new samples based on their proximity to the closest centroid, compared against a pre-calculated threshold for cluster membership. The high-level process of the method is illustrated in Figure 1. This section outlines the approach and techniques we employed to maintain reliable classification performance in the face of the constantly evolving nature of malware.

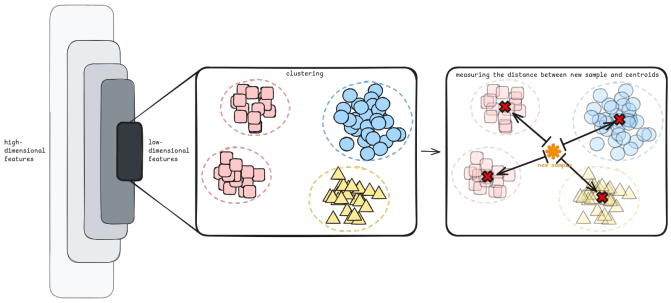


Figure 1. An overview of the method.

We employ metric learning, a machine learning approach that focuses on defining a distance metric between data points to transform the data space. This method aims to bring similar points closer together while pushing dissimilar points further apart [40]. In our implementation, we utilize triplet loss, which simultaneously considers pairs of similar and dissimilar points, enhancing the model's capability to learn meaningful representations. Triplet loss operates by using triplets of data points: an anchor, a positive sample that is similar to the anchor, and a negative sample that is dissimilar [23]. The objective is to ensure that the distance between the anchor and the positive sample is smaller than the distance between the anchor and the negative sample by at least a specified margin. This margin enforces a separation between similar and dissimilar pairs, driving the model to learn a more discriminative feature space. Additionally, working with high-dimensional feature vectors introduces the challenge of the curse of dimensionality. In high-dimensional spaces, distances between data points become less informative, complicating the task of distinguishing between different classes [41]. To overcome this, we incorporated an autoencoder to reduce the dimensionality of the data, thereby mitigating the effects of the curse of dimensionality and making distance measures more reliable. The autoencoder compresses the high-dimensional data into a lower-dimensional latent space, preserving the most critical features while discarding redundant information. This compression not only enhances the efficiency of distance computations but also helps in capturing the underlying structure of the data, making it easier to identify and distinguish between different classes.

After samples are projected into the latent space, samples from the same malware family are grouped, facilitating easier classification. However, since our goal is to analyze new malware families, a more fine-grained analysis is required. To achieve this, clustering in the latent space is used to group samples belonging to subgroups of the same malware families. This approach allows us to project new samples during testing and not only classify them as belonging to an existing or new family but also measure their deviation from existing families. Building on this foundation, we employed the DBSCAN [42] clustering algorithm to identify multiple clusters within each class in the latent space obtained from the bottleneck of the autoencoder. Although we know the number of classes from the training data, DBSCAN helps us identify sub-clusters within these classes, which is crucial since a single class can contain multiple distinct clusters.

Ignoring these sub-clusters can result in misleadingly large distances between data points and their centroids. By employing DBSCAN, we can accurately detect these sub-clusters, leading to precise centroid calculations. This clustering process allows us to determine whether a new sample fits within existing classes or should be considered a new malware family or variant, pending expert verification. Additionally, DBSCAN effectively handles outliers, ensuring that anomalies do not distort the centroid calculations.

To further refine our classification approach, we calculate the centroids of each cluster. Given that we use the DBSCAN algorithm, which does not include every point in a cluster and filters out outliers, we compute the centroids more accurately. The centroids are calculated by taking the mean of the data points within each cluster. This method benefits from DBSCAN's ability to handle outliers effectively, ensuring that these outliers do not distort the centroid calculations.

Once the clusters and their centroids are established, we determine whether a new sample belongs to an existing family by locating the closest centroid in the latent space and noting the family to which this centroid belongs. The distance between the new sample and the closest centroid is then calculated and compared with a pre-calculated threshold for the identified family. Each family's threshold is determined by the distance from the centroid to the furthest point within the cluster (excluding outliers). This threshold acts as a boundary to decide cluster membership. If the calculated distance for the new sample is less than or equal to the threshold, the sample is classified as belonging to that family. Otherwise, it is considered as not belonging to any existing family and may be flagged as a potential new family or variant.

The threshold for each family is calculated during the clus-

tering process with DBSCAN. For each cluster, the threshold is defined as the distance from the centroid to the furthest point within the cluster. This is feasible because DBSCAN effectively excludes outliers, ensuring that the threshold represents the maximum distance within the core points of the cluster.

By employing DBSCAN and centroid calculation, our method can accurately determine whether a new sample fits within existing classes or should be considered a new malware family or variant, pending expert verification. This approach not only enhances detection performance but also provides a reliable mechanism to handle the dynamic nature of malware, maintaining the model's robustness over time.

Impact of triplet loss

The triplet loss function is a powerful technique used in machine learning to enhance the discriminative ability of models [23]. It operates by optimizing the distance between samples in such a way that similar samples (belonging to the same class) are brought closer together, while dissimilar samples (belonging to different classes) are pushed farther apart.

Given an anchor sample x_a, a positive sample x_p of the same class, and a negative sample x_n of a different class, the triplet loss function aims to ensure that the distance between the anchor and positive samples is less than the distance between the anchor and negative samples by at least a margin α. The triplet loss function L can be defined as:

$$L(x_a, x_p, x_n) = \max\{0, D(x_a, x_p) - D(x_a, x_n) + \alpha\} \quad (1)$$

where:

$$D(x_i, x_j) = \|f(x_i) - f(x_j)\|^2 \quad (2)$$

represents the squared Euclidean distance between the embedding vectors of two samples, produced by the embedding function f. The margin α defines the desired separation between positive and negative pairs.

In practice, the loss is computed over a batch of triplets, an anchor input (input pair), a positive input (similar pair), and a negative input (dissimilar pair), and the objective is to minimize the total loss across the batch. In triplet loss, a margin enforces a distinct separation, where distances smaller than the margin do not contribute to the loss function. Assuming we have a set of triplets $\{(x_{a_i}, x_{p_i}, x_{n_i})\}_{i=1}^{N}$, the overall triplet loss can be defined as:

$$L_{\text{batch}} = \frac{1}{N} \sum_{i=1}^{N} L(x_{a_i}, x_{p_i}, x_{n_i}) \quad (3)$$

This formulation forces the network to learn an embedding space where positive pairs (anchor and positive samples) are closer than negative pairs (anchor and negative samples), with a margin α separating them.

Impact of DBSCAN

Selecting an appropriate clustering algorithm is crucial for accurate identification and analysis. Popular clustering methods like K-means were not suitable for our needs due to several reasons. Firstly, K-means require the number of clusters to be specified beforehand, which is challenging in our scenario because the number of clusters is unknown and can vary significantly. A single class could potentially have multiple clusters due to intraclass variants. Secondly, K-means tries to include all data points in a cluster, which is not ideal for real-world data where outliers, such as incorrectly labeled data or genuine anomalies, are present. These outliers can distort the clustering results and reduce the overall accuracy. Additionally, K-means assumes that clusters are spherical and equally sized, which is rarely the case in malware datasets [43]. Malware families can exhibit diverse and complex structures that do not fit the spherical assumption.

Given these limitations, we considered DBSCAN [44] as a potential solution. DBSCAN is effective in identifying clusters of arbitrary shapes and sizes and is robust in detecting outliers. It operates on the principle that regions of high data density are separated by regions of low density. DBSCAN requires two parameters: epsilon ε, which defines the radius for neighborhood search, and minPts, the minimum number of points required to form a dense region. The challenge with DBSCAN lies in setting these parameters correctly, especially for datasets with varying densities. To address this, we employed several heuristics and techniques discussed in the literature. Firstly, we set the minPts parameter based on the dimensionality of the data. A common heuristic is to set minPts to twice the number of dimensions, i.e., $minPts = 2 \times dim$ [42]. Choosing the appropriate value for ε is more challenging. One effective method is to use the k-distance plot, where we plot the distance to the k-th nearest neighbor for each point in the dataset. A sharp bend in this plot, known as the "knee" or "elbow," often indicates a good choice for ε. In practice, the value of ε should be chosen as small as possible to capture the most relevant clusters without merging distinct ones.

In our case, there is another advantage of using DBSCAN. One crucial parameter to decide is the distance threshold metric, which helps us determine whether samples exceed or fall within the distance threshold. We utilized the DBSCAN algorithm to establish this distance threshold. The DBSCAN algorithm identifies clusters in the provided data, finding one or multiple clusters if they exist. Importantly, it does not force every point into a cluster; instead, it marks points that do not belong to any cluster as noise or outliers. This feature is advantageous for determining a robust threshold that accounts for outliers. We set the highest distance between a point and a centroid within a cluster as the threshold.

To demonstrate the effectiveness of DBSCAN, we present Figure 2 where multiple clusters within a single class are shown. Specifically, using DBSCAN, we detected two clusters in one of the classes, FakeInstaller. This approach decreased the mean distance between a sample and its centroid by

26%, from 1.46 to 1.07. This significant reduction in mean distance highlights the effectiveness of DBSCAN in accurately separating clusters within a class, ultimately improving the clustering performance.

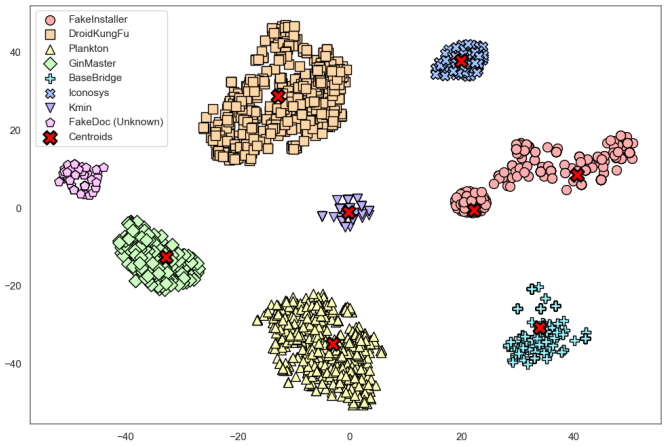


Figure 2. Clustering results using DBSCAN algorithm.

IV. Dataset and Setup

A. Drebin dataset

We utilized the Drebin dataset, an Android malware dataset containing 5,560 malware samples and 123,453 benign applications compiled between August 2010 and October 2012 [15]. Drebin extracts features from the Android Manifest file, such as hardware components, requested permissions, app components, and filtered intents, as well as features from the app's disassembled code, including restricted API calls, used permissions, suspicious API calls, and network addresses.

For our analysis, we selected families with a minimum of 100 malware samples, reducing the dataset to 8 families and a total of 3,317 samples (Table I). We followed a previous study's strategy of splitting the dataset into training and testing sets using an 80:20 ratio, based on the malware creation timestamps [37]. This temporal split ensures that our evaluation reflects real-world scenarios, where models encounter newer malware after being trained on older data, thus providing a robust evaluation against established benchmarks.

TABLE I. Sample Distribution of Drebin Dataset.

Family	Number of Samples
FakeInstaller	925
DroidKungFu	667
Plankton	625
GingerMaster	339
BaseBridge	330
Iconosys	152
Kmin	147
FakeDoc	132

B. BODMAS dataset

We utilized the BODMAS dataset, originally consisting of 57,293 malware samples and 77,142 benign samples, for a total of 134,435 samples [17]. Each sample is represented by a 2,381-dimensional feature vector, extracted through static analysis. These feature vectors include elements such as general file information (file size, imported/exported functions, and section data like relocations, resources, and signatures), header information (machine type, subsystem, and image versions), imported and exported functions, and section information (section names, entropy, and virtual size). Additionally, byte histograms, byte entropy histograms, and string information (e.g., URLs, registry keys, and string entropy) are included to capture statistical properties. The dataset also records timestamp metadata, indicating when each sample was first seen on VirusTotal.

In our preprocessing, we performed several preprocessing steps to refine the subset used in our analysis. We excluded packed malware samples, as packers encrypt and compress the code, making it difficult to carry out accurate drift detection [35][39][45]. Additionally, we focused only on malware families with more than 1,500 samples, resulting in the inclusion of seven families for our study (Table II).

We split the dataset into training and testing sets in an 80:20 ratio based on the first-seen timestamps to simulate real-world settings and evaluate the model's performance under more realistic conditions. The training data consists of samples from August 2019 to July 2020, while the testing data includes samples from July 2020 to September 2020. This temporal split helps minimize experimental bias and aligns with recommendations from previous work [3].

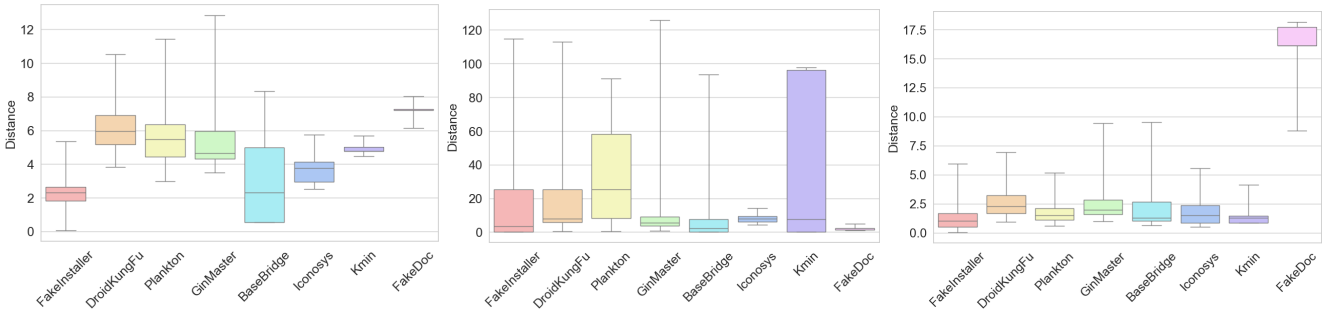
TABLE II. Sample Distribution of BODMAS Dataset.

Family	Number of Samples
berbew	1741
dinwod	1942
ganelp	1413
mira	1526
sfone	3218
sillyp2p	3012
small	3606

C. Experimental Setup

To evaluate the robustness and accuracy of our proposed method, we designed a comprehensive experimental setup involving preprocessing, model training, and validation stages. This section outlines the procedures and configurations employed to ensure the credibility and reproducibility of our results. The experiments were conducted on two prominent malware datasets, Drebin and BODMAS, each requiring tailored preprocessing steps to handle their specific characteristics.

We began by implementing a variance threshold to filter out features with low variance in both datasets. Data was then split into training and testing sets based on timestamps, utilizing the malware creation time for Drebin and the first-seen date (based



(a) Original feature space. (b) Vanilla AE feature space. (c) Triplet autoencoder feature space.

Figure 3. Boxplot diagrams showing the distances between samples and their family centroids for three feature representations: original features, vanilla autoencoder features, and triplet autoencoder features.

on VirusTotal) for BODMAS, ensuring a temporal allocation. The Drebin dataset was modeled using neural network layers consisting of 1376 input neurons, followed by 1024, 256, and 32 neurons. The BODMAS dataset's model architecture included layers with 2381, 1024, 256, and 32 neurons. For training, the vanilla autoencoder utilized mean squared error as its loss function, whereas the triplet autoencoder employed a combination of triplet loss and reconstruction loss. Triplets were selected by including one sample from a random class, one sample from the same class, and one from a different class.

To validate our model's performance, we used an 80:20 split for both datasets, with the training set comprising older samples and the testing set consisting of more recent samples. This temporal split simulates real-world scenarios where models are deployed and subsequently encounter new malware.

In our experiments, we simulate the presence of drifting samples by systematically excluding one malware family from the training data in each iteration. For example, if we exclude the FakeInstaller family, the remaining families are used to train the model, and during testing, the previously excluded family (e.g., FakeInstaller) is reintroduced alongside the test sets of the other families. This approach creates a scenario where the model encounters 'unknown' or 'drifting' families, allowing us to evaluate its ability to manage concept drift. This technique aligns with approaches used in previous studies [4][37]. Although this method effectively labels drifted samples (since they are excluded from training), it has a limitation: newer samples from known families in the test set might exhibit drift or initiate drifting, but without specific labels to denote this finer level of drift, they will not be classified as such.

V. Evaluation

A. Experiment on Android malware dataset (Drebin)

In Figure 3, we present the distance boxplot diagrams of the distance between every sample and its family centroid for three different feature representations: the original features (1376-dimensional features), the vanilla autoencoder (32-dimensional features), and the triplet autoencoder (32-dimensional features). We observe that the distances from centroids are significantly smaller by comparing the original feature space to the triplet autoencoder feature space. This indicates that the samples of the same class are more tightly clustered and closer to each other in the triplet autoencoder feature space.

When comparing the vanilla autoencoder to the triplet autoencoder, we also find that the distances in the triplet autoencoder are more compact. Additionally, we conducted an experiment where one class was designated as an unknown class by excluding any samples from that class during the training phase for both the vanilla autoencoder and the triplet autoencoder. This setup allows us to test the models' ability to handle out-of-distribution samples.

The results show that the distances between the unknown class samples and the closest centroid are much larger, indicating that the unknown class is effectively separated from the known classes. This separation suggests that the triplet autoencoder successfully differentiates between known and unknown classes, enhancing its robustness to data drift and unseen samples.

To further illustrate the separation of classes, we also provide t-distributed Stochastic Neighbor Embedding (t-SNE) graphs of the same data used to generate the boxplots in Figure 4. These t-SNE visualizations offer a clearer picture of how the different feature representations perform in terms of clustering [46]. By comparing the t-SNE plots of the vanilla autoencoder and the triplet autoencoder, it becomes evident that the clusters in the triplet autoencoder are more distinct and tightly packed. This demonstrates the superior capability of the triplet autoencoder in creating a well-defined and separated feature space.

Table III provides comprehensive details about our experiments. The first two columns list the labels of the malware families, along with the family names that were excluded from the training set and reserved solely for testing. The table also presents the F1 scores, which measure the performance of the models in terms of both precision and recall. Three methods are compared in our evaluation, CADE [37], triplet autoencoder with Median Absolute Deviation (MAD) threshold, and

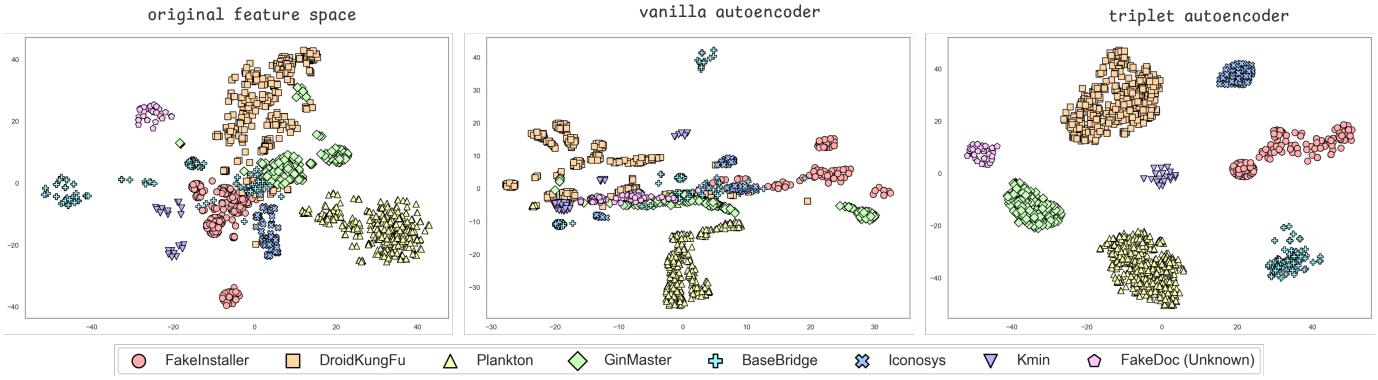


Figure 4. t-SNE diagrams of original features space, vanilla autoencoder and triplet autoencoder.

our proposed approach. MAD, calculated using the formula $MAD = median(|X_i - median(X)|)$, serves as a measure of statistical dispersion, and is used to determine the distance threshold. Including MAD allows us to compare the performance of the threshold using DBSCAN. Since the training model structure is the same for both the MAD and DBSCAN-based methods —both using the triplet autoencoder— the only difference lies in how the threshold is determined to decide whether each sample belongs to existing families. The results in the table demonstrate that the DBSCAN-based method consistently outperforms MAD method in terms of the overall F1 score. Even though in some cases MAD's performance is closer to that of DBSCAN, it is important to note that MAD requires a coefficient determined empirically. In contrast, DBSCAN calculates the threshold automatically without requiring a manual coefficient. When comparing CADE, which uses contrastive loss and MAD threshold, and our method, which employs triplet loss along with the DBSCAN threshold, our method demonstrates superior performance overall.

B. Experiment on Windows PE malware dataset (BODMAS)

We also used the BODMAS dataset to measure the performance of our method. This dataset brings two different advantages: it is a more recent dataset, and it is a Windows PE dataset, representing a different operating system, which helps us measure the generalizability of our method.

The results, summarized in Table IV, show that our method performs well on the Windows PE malware dataset. This performance demonstrates the robustness and generalizability of our method across different types of malware and operating systems. By using the BODMAS dataset, we validated that our approach is not limited to a specific type of malware or operating system but can be effectively applied to various scenarios. This enhances the overall reliability and applicability of our malware detection method, providing a practical and robust solution for cybersecurity challenges across different platforms.

VI. Limitations

While our study provides valuable insights, it is important to acknowledge several limitations. Our analysis considers only one family as the unknown family. However, in real-world scenarios, there can be multiple unknown families, which could affect the clustering results and the interpretation of the data. Moreover, although we automatically determined the parameters of the DBSCAN algorithm based on the method outlined in [42], parameters, such as ε and the minimum number of points (minPts) could be further fine-tuned depending on the specific context and dataset characteristics. Fine-tuning these parameters might lead to more accurate and meaningful clustering results. Additionally, our study did not include any packed samples because the encryption and compression of code by packers make accurate drift detection challenging.

VII. Conclusion and Future Work

This paper presents an approach to addressing concept drift in malware family detection, specifically focusing on the emergence of new malware families. Our method effectively differentiates between known and new malware families by leveraging a triplet autoencoder and the DBSCAN clustering algorithm. We validated our method using two prominent datasets, Drebin and BODMAS, representing Android and Windows PE malware. The results demonstrate that our approach significantly improves the detection performance for new malware families. This robust and reliable solution addresses the dynamic nature of cyber threats, ensuring that detection models remain effective over time. Our contributions include applying metric learning and clustering techniques to improve malware family classification in the face of concept drift, providing a practical framework for ongoing cybersecurity efforts.

Moving forward, we aim to explore retraining strategies to adapt our model to evolving malware behaviors more effectively. This will involve identifying the features and factors contributing to malware drift, allowing us to understand the dynamics of malware evolution better. By incorporating these insights, we aim to develop more robust detection mechanisms to maintain high performance even as malware tactics change.

Acknowledgments

Numan Halit Guldemir is supported by the Republic of Türkiye Ministry of National Education (MoNE-1416/YLSY).

TABLE III. PERFORMANCE OF THE MODELS. THE FAMILY COLUMN INDICATES WHICH MALWARE FAMILY WAS EXCLUDED FROM TRAINING AND KEPT FOR TESTING TO SIMULATE DRIFTING SAMPLES.

Family	No. of known samples	No. of unknown samples	F1 score CADE [37]	F1 score MAD	F1 score DBSCAN
FakeInstaller	478	925	0.86	0.95	**0.95**
DroidKungFu	529	667	0.87	0.89	**0.90**
Plankton	538	625	0.77	0.90	**0.87**
GinMaster	595	339	0.63	0.84	**0.85**
BaseBridge	597	330	0.59	0.97	**0.98**
Iconosys	632	152	0.42	0.46	**0.65**
Kmin	633	147	0.40	0.63	**0.62**
FakeDoc	636	132	0.38	0.56	**0.66**
Overall	**4638**	**3317**	**0.62**	**0.78**	**0.81**

TABLE IV. PERFORMANCE OF THE MODELS. THE FAMILY COLUMN INDICATES WHICH MALWARE FAMILY WAS EXCLUDED FROM TRAINING AND KEPT FOR TESTING TO SIMULATE DRIFTING SAMPLES.

Family	No. of known samples	No. of unknown samples	F1 score
berbew	2817	1741	0.99
dinwod	2634	1942	0.96
ganelp	2636	1413	0.97
mira	2470	1526	0.58
sfone	2231	3218	0.51
sillyp2p	2737	3012	0.83
small	1473	3606	0.96

REFERENCES

[1] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[2] AV-Test, "Malware statistics & trends report," Accessed: 14 July 2024, 2023, [Online]. Available: https://www.av-test.org/en/statistics/malware/.

[3] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "Tesseract: Eliminating experimental bias in malware classification across space and time," in *28th USENIX security symposium (USENIX Security 19)*, 2019, pp. 729–746.

[4] R. Jordaney *et al.*, "Transcend: Detecting concept drift in malware classification models," in *26th USENIX security symposium (USENIX security 17)*, 2017, pp. 625–642.

[5] A. Bendale and T. E. Boult, "Towards open set deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1563–1572.

[6] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in neural information processing systems*, vol. 31, 2018.

[7] A. Guerra-Manzanares, "Android malware detection: Mission accomplished? a review of open challenges and future perspectives," *Computers & Security*, p. 103 654, 2023.

[8] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," in *Proceedings of the sixth ACM conference on data and application security and privacy*, 2016, pp. 183–194.

[9] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls," in *Proceedings of the 2010 ACM symposium on applied computing*, 2010, pp. 1020–1025.

[10] S. Jain and Y. K. Meena, "Byte level n–gram analysis for malware detection," in *Computer Networks and Intelligent Computing: 5th International Conference on Information Processing, ICIP 2011, Bangalore, India, August 5-7, 2011. Proceedings*, Springer, 2011, pp. 51–59.

[11] Z. Fuyong and Z. Tiezhu, "Malware detection and classification based on n-grams attribute similarity," in *2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, IEEE, vol. 1, 2017, pp. 793–796.

[12] N. McLaughlin *et al.*, "Deep android malware detection," in *Proceedings of the seventh ACM on conference on data and application security and privacy*, 2017, pp. 301–308.

[13] A. G. Kakisim, S. Gulmez, and I. Sogukpinar, "Sequential opcode embedding-based malware detection method," *Computers & Electrical Engineering*, vol. 98, p. 107 703, 2022.

[14] D. Yuxin and Z. Siyi, "Malware detection based on deep learning algorithm," *Neural Computing and Applications*, vol. 31, pp. 461–472, 2019.

[15] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Ndss*, vol. 14, 2014, pp. 23–26.

[16] H. S. Anderson and P. Roth, "Ember: An open dataset for training static pe malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[17] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "Bodmas: An open dataset for learning based temporal analysis of pe malware," in *2021 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2021, pp. 78–84.

[18] Z. Salehi, A. Sami, and M. Ghiasi, "Maar: Robust features to detect malicious activity based on api calls, their arguments and return values," *Engineering Applications of Artificial Intelligence*, vol. 59, pp. 93–102, 2017.

[19] D. Uppal, R. Sinha, V. Mehra, and V. Jain, "Malware detection and classification based on extraction of api sequences," in *2014 International conference on advances in computing, communications and informatics (ICACCI)*, IEEE, 2014, pp. 2337–2342.

[20] P. O'kane, S. Sezer, and K. McLaughlin, "Detecting obfuscated malware using reduced opcode set and optimised runtime trace," *Security Informatics*, vol. 5, pp. 1–12, 2016.

[21] A. Boukhtouta, S. A. Mokhov, N.-E. Lakhdari, M. Debbabi, and J. Paquet, "Network malware classification comparison using dpi and flow packet headers," *Journal of Computer Virology and Hacking Techniques*, vol. 12, pp. 69–100, 2016.

[22] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, "Malicious software classification using transfer learning of resnet-50 deep neural network," in *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, IEEE, 2017, pp. 1011–1014.

[23] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *arXiv preprint arXiv:1703.07737*, 2017.

[24] X. Dong and J. Shen, "Triplet loss in siamese network for object tracking," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 459–474.

[25] B. McCartney, B. Devereux, and J. Martinez-del-Rincon, "A zero-shot deep metric learning approach to brain–computer interfaces for image retrieval," *Knowledge-Based Systems*, vol. 246, p. 108 556, 2022.

[26] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[27] E. Ghaleb, M. Popa, and S. Asteriadis, "Metric learning-based multimodal audio-visual emotion recognition," *IEEE Multimedia*, vol. 27, no. 1, pp. 37–48, 2019.

[28] J. S. Chung *et al.*, "In defence of metric learning for speaker recognition," *arXiv preprint arXiv:2003.11982*, 2020.

[29] Y. Xu, H. Min, H. Song, and Q. Wu, "Multi-instance multi-label distance metric learning for genome-wide protein function prediction," *Computational biology and chemistry*, vol. 63, pp. 30–40, 2016.

[30] H. Luo, J. Wang, C. Yan, M. Li, F.-X. Wu, and Y. Pan, "A novel drug repositioning approach based on collaborative metric learning," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 18, no. 2, pp. 463–471, 2019.

[31] Y. Wu, S. Dou, D. Zou, W. Yang, W. Qiang, and H. Jin, "Contrastive learning for robust android malware familial classification," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[32] M. Jureček and R. Lórencz, "Application of distance metric learning to automated malware detection," *IEEE Access*, vol. 9, pp. 96 151–96 165, 2021.

[33] C. Liu, B. Li, J. Zhao, Z. Zhen, X. Liu, and Q. Zhang, "Fewm-hgcl: Few-shot malware variants detection via heterogeneous graph contrastive learning," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[34] G. Andresini, A. Appice, and D. Malerba, "Autoencoder-based deep metric learning for network intrusion detection," *Information Sciences*, vol. 569, pp. 706–727, 2021.

[35] A. Singh, A. Walenstein, and A. Lakhotia, "Tracking concept drift in malware families," in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, 2012, pp. 81–92.

[36] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending transcend: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 805–823.

[37] L. Yang *et al.*, "Cade: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2327–2344.

[38] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "Insomnia: Towards concept-drift robustness in network intrusion detection," in *Proceedings of the 14th ACM workshop on artificial intelligence and security*, 2021, pp. 111–122.

[39] F. Zola, J. L. Bruse, and M. Galar, "Temporal analysis of distribution shifts in malware classification for digital forensics," in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2023, pp. 439–450.

[40] M. Kaya and H. Ş. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, no. 9, p. 1066, 2019.

[41] N. Altman and M. Krzywinski, "The curse (s) of dimensionality," *Nat Methods*, vol. 15, no. 6, pp. 399–400, 2018.

[42] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[43] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, p. 1295, 2020.

[44] M. Ester *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, vol. 96, 1996, pp. 226–231.

[45] D. W. Fernando and N. Komninos, "Fesa: Feature selection architecture for ransomware detection under concept drift," *Computers & Security*, vol. 116, p. 102 659, 2022.

[46] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.