Classifying Malware using Tensor Decomposition

Maksim E. Eren[1,3], Boian S. Alexandrov[2], and Charles Nicholas[3]

[1] Advanced Research in Cyber Systems, Los Alamos National Laboratory
maksim@lanl.gov
maksimeren.com
[2] Theoretical Division, Los Alamos National Laboratory
boian@lanl.gov
[3] Computer Science and Electrical Engineering, University of Maryland Baltimore County
nicholas@umbc.edu

Abstract. Tensor decomposition is as a powerful unsupervised machine learning technique capable of modeling multi-dimensional data, including that related to malware. This chapter discusses a method that employs tensor decomposition for malware analysis. We introduce an innovative ensemble semi-supervised classification algorithm named Random Forest of Tensors (RFoT). RFoT leverages tensor decomposition to extract intricate latent patterns from the data. Our hybrid model combines multi-dimensional analysis with clustering to capture sample groupings within latent components, aiding in distinguishing between malware and benign-ware. The patterns extracted from malware data using tensor decomposition heavily rely on the configuration of the tensor, including dimension, entry, and rank selection. To encompass diverse perspectives offered by different tensor configurations, we adopt the 'wisdom of crowds' philosophy. This involves leveraging decisions made by the majority within a randomly generated ensemble of tensors, varying in dimensions, entries, and ranks. We illustrate RFoT's effectiveness in classifying Windows Portable Executable (PE) malware and benign-ware. To promote the utility of tensor decomposition for malware analysis and ensure the reproducibility of our results, we have made our code publicly available.

Keywords: malware analysis · tensor decomposition · semi-supervised

1 Introduction

1.1 Motivation for Automated Malware Detection

Malware is a broad term encompassing any unwanted software designed to steal personal or confidential information or cause harm to a system upon deployment. Recent cyber reports rank malware as one of the most costly and frequent cyber threats [Bissell and Ponemon(2019)]. Detection of cyber anomalies, including malware network traffic, continues to be a significant challenge for cyber defenders. In general, organizations report an annual cost of malware at $2.5 million [Bissell and Ponemon(2019)], while a single ransomware breach

totals nearly $4.62 million [IBM(2019)]. Simultaneously, the quantity of malware in circulation continues to escalate rapidly. Approximately 13.5 million new malware specimens and unwanted applications are reported monthly, accumulating to nearly 1.3 billion known malware specimens in 2022 [The Independent IT Security Institute(2022)]. This swift proliferation in malware volume is accompanied by increasing sophistication and threat capabilities, intensifying the challenge of defending against malware [Labs(2020), Verizon(2021)]. The escalating capabilities, sophistication, cost, and volume of malware, coupled with the shortage of experienced malware analysts to address the overwhelming number of malware attacks, drive the necessity to employ automated defense systems based on Machine Learning (ML) to combat malware. ML facilitates early detection and reduces response times, enabling automation to cut the cost of a security breach by 80% [IBM(2019), Bissell et al.(2020), Bissell and Ponemon(2019)].

ML-based malware detection utilizes two categories of features: Dynamic malware analysis-based features are gathered at runtime and frequently encompass system calls, file system events, network activities, and process behavior. In contrast, static malware analysis-based features are derived directly from the contents of an executable binary, such as the Portable Executable (PE) header, strings, code, and raw bytes. The static malware features can be extracted from a specimen without requiring binary execution. Dynamic analysis-based features typically offer a more detailed view of malware behavior and are less susceptible to potential obfuscations and packing techniques [Sikorski and Honig(2012)]. However, obtaining dynamic features presents several challenges. Dynamic features necessitate executing the malware within a resource-intensive isolated sandbox environment, often resulting in a slow feature collection process. Additionally, certain types of malware are capable of detecting the presence of a sandbox and modifying their behavior accordingly [Raff et al.(2018)]. Despite potential shortcomings, static malware features remain an effective means to detect and characterize malware. We refer the reader to [Sikorski and Honig(2012)] for more details on classical malware analysis. This chapter discusses a semi-supervised malware classification method applied to static malware analysis-based features, specifically focusing on Windows PE header features. More precisely, we present malware classification methods that utilize tensor decomposition, wherein malware data is represented within a multi-dimensional space.

1.2 Motivation on Using Tensor Decomposition for Malware Detection

Tensor decomposition is a powerful unsupervised ML method capable of extracting multifaceted latent patterns from large and complex datasets. In contrast to classical ML methods, which often operate as black-box systems, tensor decomposition yields interpretable results. This attribute makes it an effective tool for incident responders seeking to validate alerts generated by automated incident detection systems (IDS). Furthermore, given the continuous generation of new malware variants by malicious authors [Raff and Nicholas(2020)], ML models utilized in malware identification must effectively generalize well to

new threats. However, many prevalent ML models struggle to generalize to new data. Moreover, supervised ML models demand substantial quantities of labeled training data to achieve desired levels of performance in operation. This poses a significant challenge within the malware domain, where acquiring labeled malware data is both costly and time-consuming [Raff and Nicholas(2020)]. Semi-supervised methods, including the algorithm presented in this chapter, can mitigate these shortcomings. They hold the potential for enhanced generalizability to new data and the ability to achieve robust performance with limited labeled data quantities. Despite the potential advantages of semi-supervised solutions, the broader research community has not extensively explored their application in Windows malware detection [Raff and Nicholas(2020)]. Addressing this gap in malware research, we present a semi-supervised algorithm based on tensor decomposition tailored for classifying Windows malware.

1.3 Chapter Contributions

Malware data naturally exists in multiple dimensions, enabling the construction of a multi-dimensional representation using tensors. These representations can be analyzed using tensor decomposition methods. Within this chapter, we introduce an ensemble semi-supervised classification algorithm called Random Forest of Tensors (RFoT). RFoT effectively utilizes tensor decomposition's capability to extract significant patterns from multi-dimensional malware data. Our analysis reveals that both malware and benign samples form distinct clusters within and among the latent components identified through tensor decomposition. To capture these groupings, we employ clustering methods and execute semi-supervised class voting for each unknown specimen grouped within a cluster, using known samples from the same cluster as a reference point. We eliminate noisy clusters based on a cluster uniformity threshold calculated using the known specimens. The extracted latent patterns heavily rely on the tensor's configuration, including dimension, entry, and rank selection. RFoT operates on the principle of the 'wisdom of crowds.' The final class prediction is derived through a max-vote mechanism, aggregating votes received from each randomly generated ensemble of tensor configurations varying in dimensions, entries, and ranks.

In this chapter, we investigate the malware classification performance of RFoT using two distinct tensor decomposition algorithms: CANDECOMP/PARAFAC Alternating Least Squares (CP-ALS) [Battaglino et al.(2018),Bader and Kolda(2006), Kolda and Bader(2009)], and CANDECOMP/PARAFAC Alternating Poisson Regression (CP-APR) [Chi and Kolda(2012)] tensor decomposition. Additionally, we compare Mean Shift (MS) [Fukunaga and Hostetler(1975), Wu and Yang(2007), Jin and Han(2017), Pedregosa et al.(2011)] and Component Clustering algorithms utilized to capture patterns from latent factors. In our experiments, we employ Windows Portable Executable (PE) file features of malware and benign-ware sourced from the EMBER-2018 dataset [Anderson and Roth(2018)]. Specifically, we create 10 random subsets, each comprising 10,000 PE files, and conduct experiments within each random subset. This approach

demonstrates the statistical significance of our results using 95% confidence intervals. We benchmark our method against the tuned *XGBoost* [Chen and Guestrin(2016)] and *LightGBM* [Ke et al.(2017)] models, previously employed in research reporting high classification scores. Additionally, we compare our results against the *SelfTrain* extended *XGBoost* model, which establishes a semi-supervised learner [Yarowsky(1995)].

Our findings demonstrate that our semi-supervised method outperforms state-of-the-art supervised models, achieving an F1 score of 0.968 in classifying malware. This performance improvement comes with a trade-off involving a reduced number of predicted labels due to abstaining predictions (i.e., predicting *"we do not know what class this is"*). To the best of our knowledge, we are the first to formulate a tensor-based semi-supervised classifier within an ensemble learning framework for classifying Windows malware.

1.4 Code and Data Availability

To enable the reproducibility of our results and to provide an operationally-relevant tool, we have released RFoT on GitHub[4]. Our Python library follows the well-known Scikit-learn API to ease the usage of RFoT [Buitinck et al.(2013),Pedregosa et al.(2011)]. The RFoT Python library also includes the Python implementation of the CP-ALS algorithm which was originally released in MATLAB Tensor Toolbox [Bader et al.(2017),Harris et al.(2020)]. For the CP-APR algorithm, which was also originally introduced in MATLAB Tensor Toolbox [Bader et al.(2017),Hansen et al.(2015)], we use the latest Python implementation of the CP-APR algorithm with GPU capability, named pyCP_APR[5], as a dependency in the RFoT library [Eren et al.(2022),Eren et al.(2021)]. Further, to reduce the computation time, the RFoT library utilizes embarrassingly parallel decomposition of each random tensor member in the ensemble. Finally, the dataset used to demonstrate the capabilities of the method introduced in this chapter, namely the EMBER-2018 dataset[6] [Anderson and Roth(2018)], is publicly available.

2 Related Work

2.1 Machine Learning Based Malware Classification

ML-based automated detection and characterization of malware has been a longstanding area of research. Deep learning has proven to be an effective method for classifying malware in a supervised manner. Vinayakumar et al. utilized shallow neural networks to detect malware using PE header features [R. and K.P.(2018)]. Similarly, Fabian et al. employed neural networks, designing their method for use in environments with limited computational resources [Fonseca A et al.(2021)]. In contrast to these approaches that utilize a selected set

[4] RFoT is available at `https://github.com/MaksimEkin/RFoT`
[5] pyCP_APR is available at `https://github.com/lanl/pyCP_APR`
[6] EMBER-2018 dataset is available at `https://github.com/elastic/ember`

of static malware features, Raff et al. introduced a deep learning architecture named MalConv, aiming to classify malware directly based on the entire raw byte-sequences of the binary [Raff et al.(2018)]. These methods leverage static malware analysis-based features for malware classification. While static malware features can effectively identify malicious files, dynamic malware features can offer additional insights into the executable. Vinayakumar et al. adopt a multi-modular approach using Deep Neural Networks (DNN), performing classification using features from static analysis, dynamic analysis, and gray-scale malware images [Vinayakumar et al.(2019)].

Several prior studies have focused on malware detection using images generated from malware [Liu et al.(2017), Yan et al.(2018), Narayanan and Davuluru(2020), Yuan et al.(2020), Liu et al.(2020)]. This includes research on classifying malware visualizations utilizing an ensemble of random forests [Roseline et al.(2019)], as well as another study employing a semi-supervised approach to cluster gray-scale malware images [Abdelmonem et al.(2021)]. Additionally, Wang et al. utilized a semi-supervised approach with gray-scale malware images [Wang et al.(2021)]. In their research, byte n-grams were translated into fixed-size gray-scale image vectors as training features. Wang et al. perform classification with the gray-scale image vectors using a 1-dimensional Convolutional Neural Network (CNN), a supervised deep learning method, that is strengthened using a semi-supervised Generative Adversarial Network (SGAN).

Previous studies have also examined classical ML methods for malware classification. Kumar et al. demonstrated that XGBoost is an effective model for classifying Windows PE malware, achieved through low-resource feature selection [Kumar and Geetha(2020)]. Pham et al., also using the EMBER-2018 dataset, illustrated that statistical summaries of the original PE features can enhance detection results. They employed LightGBM, which surpassed the previously introduced deep learning solution MalConv while requiring fewer resources [Pham et al.(2018)].

The majority of the aforementioned work, which reported excellent malware detection capabilities, is based on supervised learning. However, supervised models often encounter performance degradation in production when confronted with specimens that do not conform to the same distribution observed during training. Qi et al. tackled this issue by integrating an unsupervised domain adaptation technique based on adversarial learning into LightGBM for static malware detection [Qi et al.(2021)]. Their approach extends LightGBM to learn domain-invariant features by using the predictions generated from each decision tree in the model as a feature space, subsequently employing these features as input into the adversarial learning framework.

Another area that has garnered increasing interest is the application of ensemble learning to augment the predictive capabilities of malware classifiers. In pursuit of this, previous research has delved into an ensemble approach for Windows malware classification utilizing static features. Atluri demonstrated that various tree-based ensemble models, such as Random Forests, Bagging Decision Tree Classifier, and Gradient Boosting Classifier, among others, can be used

together in a single framework, named Voting Ensemble Classifier (VEC), to achieve enhanced detection of Windows PE malware [Atluri(2019)]. Similarly, Ramadhan et al. explored a comparable method by creating a voting-based ensemble model employing LightGBM, XGBoost, and Logistic Regression [Ramadhan et al.(2021)]. Their study showed that an ensemble comprising classifiers, each with its distinct inductive biases, could result in increased accuracy compared to any individual model alone, as each member of the ensemble complements the weaknesses of others. Additionally, the framework of ensemble learning has been applied in the realm of deep learning for malware detection by Dahl et al. [Dahl et al.(2013)]. The authors demonstrated that an ensemble of neural networks employing voting, alongside a novel feature selection method based on dimensionality reduction and random projections, significantly improves malware identification.

While the aforementioned prior work on ensemble learning utilized the voting method, Azeez et al. adopted a stacking approach employing an ensemble of CNNs to create a derived dataset based on the decisions made by the base models. This derived dataset was then used as input to the final prediction layer, incorporating an ExtraTrees classifier to enhance prediction accuracy [Azeez et al.(2021)]. Similarly, Gupta et al. also applied a stacking approach employing an ensemble of diverse supervised classifiers [Gupta and Rani(2020)]. However, in contrast, Gupta et al. meticulously selected the best-performing classifiers within the ensemble by initially ranking them based on their performance. Subsequently, the highest-ranking base models were integrated into the stacking ensemble layer, further enhancing the capability for malware detection.

Clustering has been integrated into ensemble learning frameworks for malware identification as well. Ye et al. presented a hybrid framework that constructs base clusters from an ensemble of clustering algorithms separately applied to TF-IDF, built from instruction frequency and instruction n-grams [Ye et al.(2010)]. This approach utilizes an ensemble of clustering algorithms with distinct characteristics, such as hierarchical clustering and weighted subspace K-medoids, to form the base clusters. These clusters are then utilized to extract the signatures that differentiate malware families. Similarly, Zhang et al. proposed a similar framework based on an ensemble of hybrid clustering algorithms [Zhang et al.(2017)].

Similar to clustering ensembles, distance metrics have been incorporated within an ensemble learning structure. Kong et al. utilized similarity metrics between pairs of malware to categorize malware families [Kong and Yan(2013)]. The authors derived similarity measurements using a distinct set of features such as opcodes, system calls, and file system activity. These new distance-based feature vectors, each derived from different malware features, were then employed to train an ensemble of classical ML models. Additionally, a similarity-based approach was previously employed by Raff et al., where they introduced the Burrows Wheeler Markov Distance (BWMD), an efficient similarity metric. This metric is based on embedding data into a fixed-size vector space, demonstrating its effectiveness in clustering malware [Raff et al.(2020)]. Finally, the malware

similarity for clustering IoT malware in an unsupervised manner was presented in [Bak et al.(2020)].

While several other works have examined supervised approaches [Huang and Stokes(2016), Jiang et al.(2019), Vinayakumar et al.(2019), Zhang et al.(2019), Loi et al.(2021), Sun et al.(2017), Ahmadi et al.(2016), Mohaisen et al.(2015), Hansen et al.(2016)], we draw inspiration from previous advancements and successes in ensemble learning and clustering methodologies. Our framework leverages ensemble learning and clustering techniques in multidimensional analysis through tensor decomposition. This approach combines the potency of tensor decomposition with ensemble learning. Furthermore, driven by the anticipated advantages, we structure our tensor decomposition-based solution within a semi-supervised methodology. The most analogous work to ours, concerning semi-supervised learning, was conducted by Irofti et al., who proposed a semi-supervised solution using Dictionary Learning (DL) for classifying Windows PE malware [Irofti and Băltoiu(2019)]. Their framework initially trains a dictionary in a supervised manner, facilitating the intermittent classification of new malware instances, subsequently updating the dictionary with signals from new malware in an unsupervised online fashion. However, DL is based on matrix factorization, which is constrained by the information conveyed in a two-dimensional space. Similar to [Irofti and Băltoiu(2019)], Non-negative Matrix Factorization, or NMF, has also been applied to the malware/benign-ware classification problem where Ling et al. derive similarity scores of structural patterns extracted with NMF to detect metamorphic malware (malware with the capability to modify its code during run-time) using static analysis features [Ling et al.(2019)].

In contrast, our approach employs tensors to model data in a higher-dimensional space. Each dimension allows for the inclusion of more intricate details about the nature of the data, resulting in the extraction of complex and multi-perspective information. Given the capacity for acquiring nuanced insights from data through multi-dimensional analysis, recent studies have employed tensors in addressing cybersecurity problems.

2.2 Tensor Decomposition and Cybersecurity

Data relevant to cybersecurity problems often exhibit a multi-dimensional nature, making tensors an ideal tool for analyzing cyber data. Several prior works have employed tensor decomposition to address cybersecurity issues in an unsupervised manner. The CANDECOMP/PARAFAC Decomposition (CPD) has emerged as a popular tool for identifying various types of outliers or anomalies in cyber data [Koutra et al.(2012), Maruhashi et al.(2011)].

Bruns et al. utilized non-negative tensor decompositions, particularly the CP-APR algorithm, to discern patterns of malicious network activity [Bruns-Smith et al.(2016)]. The authors leveraged the interpretability of tensor decomposition results, visually analyzing the latent factors, and successfully identified distinct stages of a cyber breach, including reconnaissance, brute-forcing, data exfiltration, and insider threats.

Subsequently, the integration of CP-APR with a statistical framework has demonstrated effectiveness in enhancing automatic anomaly detection capabilities. This method exhibited proficiency in identifying compromised user credentials, botnet network traffic, spam emails, and fraudulent credit card transactions [Eren et al.(2020), Eren et al.(2022)]. Moreover, CP-APR was employed to detect cyber anomalies by utilizing High-Performance Computing (HPC) resources for conducting embarrassingly parallel graph analytics within the latent components [Ezick et al.(2019)]. In [Eren et al.(2022)], botnet network traffic from the UGR'16 dataset [Maciá-Fernández et al.(2018)][7] was identified using non-negative tensor decomposition within a statistical framework based on user behavior analysis. Here, the patterns of normal or expected network traffic were modeled with CP-APR. During testing, this model, utilizing latent components extracted by factorizing the data from expected behavior, identified deviations from the norm via Poisson p-values, where lower p-values were used as an indicator of an anomaly or botnet network traffic. The analysis incorporated IP addresses of the network communication and temporal information as tensor dimensions. The study showcased how the unsupervised tensor decomposition-based method surpasses the anomaly detection capabilities of several state-of-the-art supervised and semi-supervised approaches.

The Tucker tensor decomposition has also emerged as a popular algorithm in addressing cybersecurity issues. Kanehara et al. utilized non-negative Tucker tensor decomposition with thresholding across latent factors to enable real-time botnet detection within the darknet [Kanehara et al.(2019)]. Similarly, Tork et al. employed Tucker tensor decomposition on a three-dimensional tensor to identify telecommunication anomalies [Fanaee-T et al.(2014)].

Xie et al. introduced a tensor truncating algorithm for rapid low-rank Tucker decomposition of tensors. They utilized reconstruction error as a metric for detecting network anomalies [Xie et al.(2017)]. Additionally, Sun et al. tackled the network anomaly detection problem by employing a dynamic Tucker tensor decomposition approach tailored for handling large-scale streaming data [Sun et al.(2006)].

In classical ML, ensemble learning has proven to significantly enhance individual model capabilities. Motivated by this, there has been growing exploration of ensemble learning in the tensor domain. Kisil et al. were among the first to apply an ensemble learning approach in multi-dimensional space [Kisil et al.(2018)]. They utilized an ensemble of latent factors extracted via tensor decomposition to train classical ML models, combining hypotheses from each tensor decomposition and ML model.

Similarly, Hou et al. introduced a framework employing tensor decomposition in an ensemble setting to classify Android malware [Hou et al.(2017)]. This involved utilizing application permissions, API calls, and hardware specifications as features to construct a tensor. Tensor Filter was then employed in a recursive boosting approach to generate an ensemble of base models. Their study demonstrated that the ensemble approach, coupled with tensor decompo-

[7] The UGR'16 dataset is available at https://nesg.ugr.es/nesg-ugr16/

sition, markedly enhances malware detection capabilities compared to classical ensemble models.

Another prior work also utilized an ensemble approach to demonstrate how anomaly detection capabilities can be enhanced, showcasing that an ensemble of tensors with varying ranks can augment anomaly detection capabilities using statistical p-value fusion techniques [Eren et al.(2022)].

The outcomes derived from the aforementioned prior studies heavily depend on the selection of tensor rank, dimensions, and entries. Crafting an appropriate tensor that yields favorable results upon decomposition presents a non-trivial challenge due to various factors. Firstly, determining the rank of a tensor is known to be NP-hard [Kolda and Bader(2009)]. Additionally, while constructing a tensor that carries intuitive significance might be feasible, choosing the most suitable features for tensor dimensions and entries often demands thorough investigation. This process typically involves trial and error, experimenting with different feature combinations.

For instance, several previous studies utilized source IP, destination IP, and temporal information as tensor dimensions for network traffic data (Netflow). However, Netflow data encompasses numerous other potential features like bytes transferred, packet counts, source and destination port numbers, and connection durations, among others. Simultaneously, the choice of tensor entry could involve binary values, counts, or other Netflow features. Thus, the process of selecting the optimal feature combinations to define a tensor's dimensions and entries from a multitude of possibilities poses an exponential scale problem.

In our forthcoming methodology, presented in this chapter, we propose that employing an ensemble of randomized tensor configurations eliminates the necessity of identifying a single tensor with optimal dimensions and entry. Before introducing our framework, we initiate the next chapter by summarizing tensor notation as preliminary information.

3 Tensor Decomposition Notation

Tensor decomposition is a potent data analysis method capable of extracting intricate patterns from data in an unsupervised manner. By utilizing tensors, data can be represented in a multi-dimensional space, allowing for the simultaneous exploration of natural relations among each dimension. This higher-dimensional and more complex representation facilitates the discovery and interpretation of hidden, multi-perspective information within the data. This section aims to summarize the tensor notations, which will be useful for understanding the later introduction of RFoT in this chapter. A summary of the notations utilized in this chapter can be found in Table 1.

Tensors represent a higher-order extension of matrices and enable the representation of multi-dimensional data. A first-order tensor corresponds to a vector, an order-2 tensor is referred to as a matrix, and any structure with dimensions ranging from 3 through D is denoted as a tensor. Specifically, a D-dimensional tensor is termed an order-D tensor, with each dimension also being named the

Table 1. Summary of the notations used in this chapter.

Notation	Description
x	Scalar
$\mathbf{x}$	Vector
$\mathbf{X}$	Matrix
$\boldsymbol{\mathcal{X}}$	Tensor
$\mathbf{x}_i$	ith element in the vector
$\mathbf{X}_{i,j}$	Entry located in row i and column j
$\mathbf{X}_{i:}$	ith row
$\mathbf{X}_{:j}$	jth column
$\boldsymbol{\mathcal{X}}^{(i)}$	Superscript (i) used to identify the ith random tensor
$\boldsymbol{\mathcal{X}}_{::j}$	jth slice of a tensor
$\circ$	Outer product

mode of the tensor. For instance, the first dimension of a tensor is also known as the first mode.

To illustrate the construction of a tensor, let's first consider matrices within a lower-dimensional space. For instance, a matrix $\mathbf{X}$ can represent a bag of words extracted from a collection of scientific papers, possessing dimensions *Documents* - *Words* with a shape of $N_{Documents} \times N_{Words}$. In this matrix, an entry $\mathbf{X}_{i,j}$ signifies the number of times word j appears in document i.

Utilizing tensors allows for the creation of a higher-dimensional representation of the data. Taking the example of scientific papers and including additional information such as the publication year for each paper, we can represent this data as an order-3 tensor $\boldsymbol{\mathcal{X}}$ with dimensions *Documents* - *Words* - *Time* and a shape of $N_{Documents} \times N_{Words} \times N_{Time}$. In this tensor, an entry $\boldsymbol{\mathcal{X}}_{i,j,k}$ denotes the frequency of word j appearing in document i at time k (for instance, by year).

We can extend this example from a 3-dimensional tensor to a D-dimensional tensor $\boldsymbol{\mathcal{X}} \in \mathrm{IR}^{\mathrm{N_1 x N_2 x \cdots x N_D}}$, where an entry in the tensor is denoted as $\boldsymbol{\mathcal{X}} i_1, i_2, \cdots, i_D$ and the indexing ranges of each mode are $i_1, i_2, \cdots, i_D \in [0 \leq i_1 < N_1, 0 \leq i_2 < N_2, \cdots, 0 \leq i_D < N_D]$. To facilitate notation, we adopt the *multi-index* notation and use $\mathbf{i}$ to represent the indexing of D modes, such that $\mathbf{i} = i_1, i_2, \cdots, i_D$ and $\boldsymbol{\mathcal{X}}\mathbf{i}$ denotes the tensor entry [Chi and Kolda(2012), Hong et al.(2020)].

Let $nnz(\boldsymbol{\mathcal{X}})$ represent the set of all non-zero entries in the tensor $\boldsymbol{\mathcal{X}}$, and let Ω denote the set of all entries, including the zeros, indicating that we have a sparse tensor when $nnz(\boldsymbol{\mathcal{X}}) < |\Omega|$ [Chi and Kolda(2012)]. Here, $|\Omega|$ denotes the size of the tensor and is calculated as follows:

$$|\Omega| = \prod_{d=1}^{D} N_d \tag{1}$$

We can use the number of non-zeros and the size of the tensor $\boldsymbol{\mathcal{X}}$ to calculate the sparsity of the tensor as follows:

$$\eta = \frac{nnz(\boldsymbol{\mathcal{X}})}{|\Omega|} \tag{2}$$

Tensors generated from cyber data often exhibit both extreme sparsity and substantial size. For instance, tensors derived from cyber Netflow data can demonstrate sparsity as low as $\eta = 10^{-8}$ [Eren et al.(2020)]. In the context of Netflow data, the tensor's dimensions might represent the source and destination devices and the timing of network communication events. As illustrated in Figure 1, we depict a tensor with dimensions *User-Source-Destination*, where each entry signifies a *user* engaging in an authentication activity from a *source device* to a *destination device*. Due to the limited communication between most devices within a network, tensors derived from Netflow data often display sparsity (as demonstrated in Figure 1). Similarly, diverse malware features, such as file size, number of sections, and timestamps, can function as dimensions in the tensor. Each specimen's feature space aligns with a single index along each tensor dimension, akin to the Netflow example, leading to a sparse tensor.

Leveraging the sparsity of tensors offers an opportunity to circumvent storing the entire tensor in memory, a challenge posed by their substantial size. Instead, the tensor can be stored in a Coordinate (*COO*) format, which comprises a list of non-zero coordinates and their corresponding non-zero values. In the COO format, each coordinate represents the indexing $\mathbf{i}$, while each non-zero value corresponds to the entry $\boldsymbol{\mathcal{X}}_{\mathbf{i}}$.

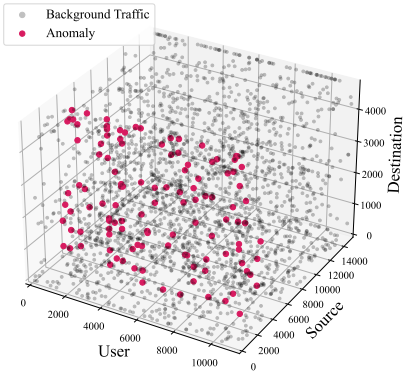

Fig. 1. Binary tensor with the dimensions *User - Source - Destination*. The background traffic is shown with gray and anomalies are highlighted in red.

The higher-order representation of data using tensors allows for the analysis of concealed information within the data by considering interactions simultaneously across each dimension, thus facilitating the extraction of intricate and multifaceted details. Two widely used tensor decomposition algorithms for factorizing tensors are the Tucker and CANDECOMP/PARAFAC Tensor Decomposition (CPD) [Kolda and Bader(2009)]. In this chapter, we utilize CPD to extract latent patterns from malware data. CPD compresses the D-dimensional tensor $\boldsymbol{\mathcal{X}}$ into lower-dimensional R rank-1 tensors, also referred to as components, aiming to approximate the original tensor's sum with the R rank-1 tensors:

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \lambda_r \cdot \mathbf{a}_r^{(1)} \circ \mathbf{a}_r^{(2)} \circ \cdots \circ \mathbf{a}_r^{(D)} \tag{3}$$

Here, $\circ$ denotes the outer product. The latent factors $\mathbf{a}^{(d)}r$ correspond to each dimension d, where $1 \leq d \leq D$, and the rth component, where $1 \leq r \leq R$, describes the latent information for the given dimension. Each $\mathbf{a}^{(d)}r$ is normalized to sum up to 1, and the weight is absorbed by each λ_r. An illustration of CPD is provided in Figure 2 for a 3-dimensional tensor. The tensor rank R

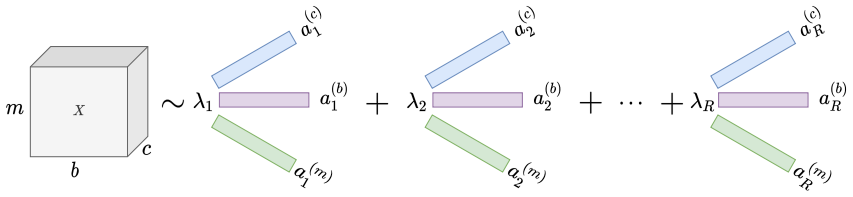

Fig. 2. Illustration of CPD on a 3-dimensional tensor

is a hyperparameter selected by the user. Determining the rank R of a tensor is known to be NP-Hard [Kolda and Bader(2009)]. CPD can be expressed in a more concise format using the KRUSKAL notation as follows:

$$\mathfrak{X} \approx \mathfrak{M} \equiv [\![\lambda\ ;\ \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \cdots, \mathbf{A}^{(D)}]\!] \tag{4}$$

Here the KRUSKAL tensor $\mathfrak{M}$ is the low-rank approximation of $\mathfrak{X}$. Each $\mathbf{A}^{(d)}$ is a matrix of latent factors for dimension d. $\mathbf{A}^{(d)}_{:r}$ is the rth latent factor for dimension d with size N_d such that we can write $\mathbf{A}^{(d)}$ as follows:

$$\mathbf{A}^{(d)} = [\mathbf{a}^{(d)}_1, \mathbf{a}^{(d)}_2, \ldots, \mathbf{a}^{(d)}_R] \tag{5}$$

Within each latent factor matrix $\mathfrak{M}{::}\ d-1 = \mathbf{A}^{(d)}$ for dimension d, linearly dependent columns $\mathbf{A}^{(d)}{:}\ r$ can be present for each r [Kolda and Bader(2009)]. However, when considering all latent factor matrices $\mathbf{A}^{(1,2,\cdots,D)}$ together, the CPD solution is almost always unique [Qi et al.(2016), Kolda and Bader(2009)]. The uniqueness of CPD allows each component to represent distinct events or characteristics of the data. Therefore, the results obtained from CPD provide interpretable outcomes when examining each component individually.

In this chapter, we employ two popular tensor decomposition algorithms with different properties to heuristically test RFoT's malware classification capability. The first one is the CANDECOMP/PARAFAC Alternating Least Squares (CP-ALS) tensor decomposition algorithm [Battaglino et al.(2018), Bader and Kolda(2006), Kolda and Bader(2009)]. To fit the tensor $\mathfrak{X}$, CP-ALS performs updates using least squares by alternating between each latent factor matrix $\mathbf{A}^{(d)}$, while fixing the remaining $\mathbf{A}^{(d-1)}$ factor matrices until convergence to solve the following optimization function:

$$\min_{\mathbf{A}^{(d)}} ||\mathfrak{X} - \mathfrak{M}||^2 \tag{6}$$

The second tensor decomposition algorithm used in our studies is CANDECOMP/PARAFAC Alternating Poisson Regression (CP-APR), a non-negative tensor decomposition method that minimizes Kullback-Leibler (KL) divergence via an updated Multiplicative Update (MU) algorithm [Chi and Kolda(2012)]. The CP-APR algorithm includes a non-negativity constraint, which allows the

latent factors to be additive parts of the original data, resulting in improved interpretability. In CP-APR, the tensor is modeled under a Poisson distribution with the Poisson rate parameter $\gamma > 0$ as follows:

$$\mathcal{X}_{\mathbf{i}} \sim \text{Poisson}(\gamma_{\mathbf{i}}) \tag{7}$$

The CP-ALS algorithm was initially included in the widely used MATLAB Tensor Toolbox [Bader et al.(2017)]. With RFoT, we introduce a Python implementation of CP-ALS [8], utilized in our experiments. For the CP-APR algorithm, we utilize an existing Python implementation with GPU capabilities that was previously introduced [Eren et al.(2022), Eren et al.(2021)]. For further information on tensors, we recommend [Kolda and Bader(2009), Rabanser et al.(2017)]. More details regarding the CP-APR algorithm can be found in [Chi and Kolda(2012)], and additional information about CP-ALS is available in [Battaglino et al.(2018)].

4 Semi-Supervised and Ensemble Methodology

We next present the RFoT methodology.

4.1 Clustering Specimens Over the Latent Components

We find that malware and benign-ware samples can be separated in an unsupervised manner using tensor decomposition. This section begins by delineating the necessary tensor configuration essential for extracting sample groupings from a tensor decomposition. Moreover, we present concrete instances of clusters representing malware and benign-ware within the latent factors. We then summarize the clustering methods used in RFoT to capture the patterns formed in the latent factors.

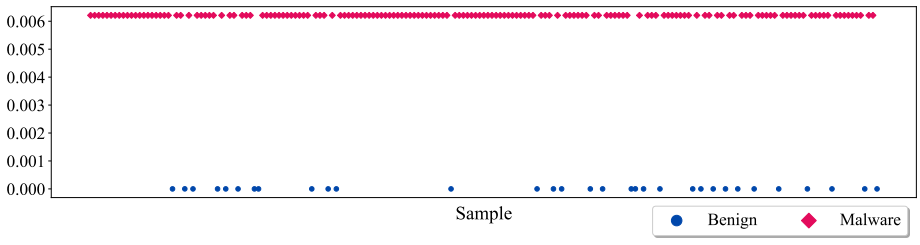

Fig. 3. Clean malware and benign-ware clusters found by tensor decomposition

[8] CP-ALS is available at https://github.com/MaksimEkin/RFoT

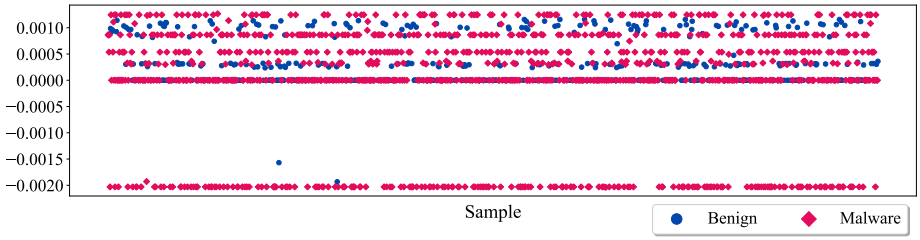

Fig. 4. Clean malware clusters and noisy benign-ware clusters found by tensor decomposition

Malware Patterns in the Latent Factors: In order to extract latent factors with the capability of describing malware and benign-ware patterns based on each sample individually, we set our first dimension of the tensor $\mathcal{X}$ to represent each malware sample while selecting the remaining of the $D-1$ dimensions and the tensor entry from static-malware-analysis based features using the PE header. We give a detailed description of how the remaining $D-1$ is configured to build the tensor from PE features in Section 4.2. In this tensor configuration, the shape of $\mathcal{X}$ is $N_1 \times N_2 \times \cdots \times N_D$, where N_1 is the total number of malware and benign-ware files from our dataset. For example, to access the tensor entry of the first specimen from the dataset, for features that are indexing at $i_2, \cdots, i_D$, we would index the tensor as $\mathcal{X}_{0,i_2,\cdots,i_D}$. Because the first dimension of $\mathcal{X}$ represents the samples, the obtained latent factor matrix for mode-1 is $\mathcal{M}_{::0} = \mathbf{A}^{(1)} \in \mathbb{R}^{N_1 \times R}$, where R is the tensor rank and $\mathbf{A}^{(1)}$ carries latent information regarding the samples in our data. Using $\mathbf{A}^{(1)}$, we can access each individual latent factor $\mathbf{A}^{(1)}_{:r} \in \mathbb{R}^{N_1 \times 1}$ in which the N_1 malware and benign-ware samples would form clusters. In Figure 3, we provide an example latent factor $\mathbf{A}^{(1)}_{:r}$ obtained by factorizing $N_1 = 10,000$ malware and benign-ware from the EMBER-2018 dataset using CP-ALS. Here, we can see that CP-ALS was able to cleanly separate malware and benign-ware instances within the latent factor. We also provide a second example with more noisy clusters in Figure 4. In Figure 4, around 7 lines forming clusters can be seen. Although the lines that have the majority of the samples from the malware class form cleaner clusters, there are other clusters where benign and malware samples are included within the same cluster. In Section 4.3, we will describe how we handle the more noisy clusters, or the clusters with poor uniformity where the majority of the cluster is not represented by a single class, using the *Cluster Uniformity Score*.

In addition to observing clusters that separate malware and benign-ware within each latent factor $\mathbf{A}^{(1)}_{:r}$, we also find that malware and benign instances cluster among components in $\mathbf{A}^{(1)}$, such that a single component r represents samples from a single class. For instance, in Figure 5 we again show a latent factor obtained by factorizing $N_1 = 10,000$ malware and benign-ware from the EMBER-2018 dataset using CP-ALS. This time, it can be seen that CP-ALS was able to cluster benign instances within a single factor from the component r.

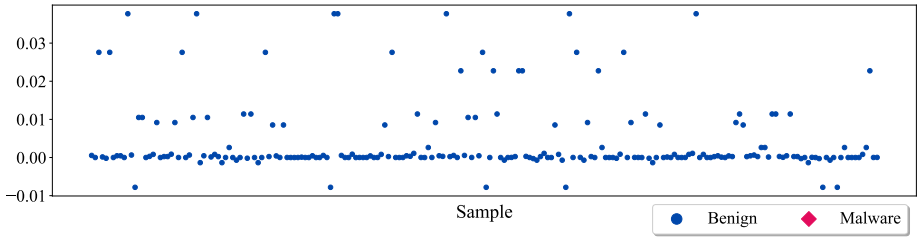

Fig. 5. Tensor decomposition placing benign samples into a single latent factor

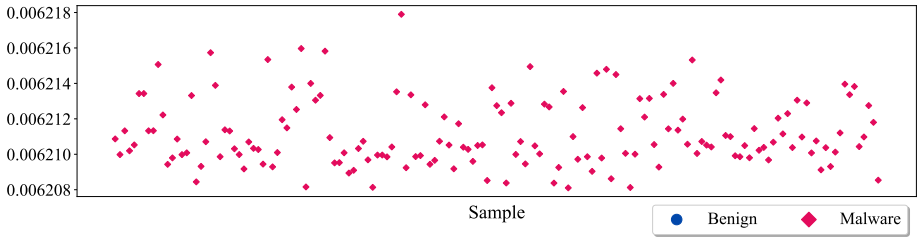

Fig. 6. Tensor decomposition placing malware specimens into a single latent factor

Similarly, in Figure 6, it can be seen that the latent factor r only contains malware specimens. Motivated by the fact that we can acquire meaningful patterns that distinguish malware and benign-ware using tensor decomposition, we next look at how these patterns can be captured to enable building a semi-supervised classifier.

Capturing the Latent Patterns via Clustering: The performance of RFoT depends on the success in capturing the patterns found by tensor decomposition into clusters. Therefore, in this section we compare two different clustering methods. The first clustering algorithm we use to capture the patterns is called Mean Shift (MS) [Fukunaga and Hostetler(1975), Wu and Yang(2007), Jin and Han(2017)]. Specifically, we use the Scikit-learn implementation of this algorithm [Pedregosa et al.(2011)]. MS uses centroids to be the mean of clusters and updates the location of the clusters in a hill-climbing fashion to locate the maxima of a given density function, making it a good fit to perform clustering in a 1-dimensional space [Pedregosa et al.(2011)]. The window length, or the furthest point from the centroid of a cluster, is selected via the bandwidth. We use Scikit-learn's *estimate_bandwith*[9] API to automatically determine the number of clusters. RFoT applies MS to each component within the latent factor for the first dimension $\mathbf{A}^{(1)}_{:r}$ from a given tensor decomposition. We extract a total of G_r clusters from each latent factor $\mathbf{A}^{(1)}_{:r}$, adding up to total of $G = G_0 + G_1 + \cdots + G_{R-1}$

[9] We heuristically set the *quantile* hyper-parameter to be 0.1 for estimating the bandwidth.

clusters for a single tensor decomposition. We let the $\mathbf{g}_{j,r}$ represent a cluster with a set of samples from the rth component and jth cluster, where $0 \leq j \leq G_r - 1$.

In addition to the MS clustering, we use *Component* clustering. The motivation behind the Component clustering comes from our observation that we can obtain class-based groupings among components, rather than within individual latent factor, obtained from tensor decomposition. We discussed this in Section 4.1, where figures 5 (component with only benign samples) and 6 (component with only malware specimens) showed an example of clean clustering within a single latent factor from the component r. Formally, when using the *Component* clustering, we will let each $\mathbf{A}^{(1)}_{:r}$ to define a single cluster such that $\mathbf{g}_{r,r} = \mathbf{A}^{(1)}_{:r}$. The total number of clusters from the component r in this case is $G_r = 1$, and the total number of clusters for the decomposition is $G = R$, where R is the tensor rank. Recall that figures 3 and 4 showed example latent factors where we had mix of both malware and benign-ware clusters. We will also use the *Cluster Uniformity Score*, introduced below at Section 4.3, to filter out the cases where we have more than one class describing a single latent factor.

After each tensor decomposition, we apply pre-processing to each latent factor $\mathbf{A}^{(1)}_{:r}$ to keep the samples with signals, or samples with a value that is not near 0 within the latent factor. To this end, prior to applying MS or Component clustering, we mask out (or remove) the points that are close to zero, where the distance to 0 is controlled with the hyper-parameter *zero_tol* in RFoT. In our experiments, we set $zero\_tol = 1e-08$.

4.2 Ensemble of Random Tensor Configurations

Notation for an Ensemble of Tensors: Patterns extracted with tensor decomposition depends on the configuration of the tensor including the selection of the dimensions, tensor entry, and tensor rank. RFoT uses the *"wisdom of crowds"* philosophy by utilizing the patterns found from an ensemble of tensor configurations with randomly selected dimensions, entries, and ranks. We use the variable *n_estimators* to represent the number of randomly generated tensor configurations. Let $\boldsymbol{\mathcal{X}}^{(i)}$ be one of the randomly generated tensors where i is in range $1 \leq i \leq n\_estimators$. To describe the random tensor configuration members of an ensemble, we re-formulate the notations introduced for tensor decomposition in Section 3. We begin with re-writing the CPD formula with sum of rank-1 tensors:

$$\boldsymbol{\mathcal{X}}^{(i)} \approx \sum_{r=1}^{R_i} \lambda_r \cdot \mathbf{a}_r^{(i,1)} \circ \mathbf{a}_r^{(i,2)} \circ \cdots \circ \mathbf{a}_r^{(i,D_i)} \tag{8}$$

Here we have the rank R_i CPD for the ith random tensor $\boldsymbol{\mathcal{X}}^{(i)}$ with D_i dimensions, and each $\mathbf{a}_r^{(i,d)}$ represents the rth latent factor for dimension d, where r is in range $1 \leq r \leq R_i$ and d is in range of $1 \leq d \leq D_i$. Following the KRUSKAL format we re-write the low-rank approximation as follows:

$$\boldsymbol{\mathcal{X}}^{(i)} \approx \boldsymbol{\mathcal{M}}^{(i)} \equiv [\![\lambda \; ; \; \mathbf{A}^{(i,1)}, \mathbf{A}^{(i,2)}, \cdots, \mathbf{A}^{(i,D_i)}]\!] \tag{9}$$

Here $\mathcal{M}^{(i)}$ is the low-rank estimation for the ith random tensor, and $\mathcal{M}^{(i)}_{::d-1} = \mathbf{A}^{(i,d)}$ is the latent factors matrix for dimension d. Each $\mathbf{A}^{(i,d)} \in \mathbb{R}^{\mathrm{N_d \times R_i}}$ is a collection of latent factors as follows:

$$\mathbf{A}^{(i,d)} = [\mathbf{a}_1^{(i,d)}, \mathbf{a}_2^{(i,d)}, \ldots, \mathbf{a}_{R_i}^{(i,d)}] \tag{10}$$

As explained in Section 4.1, to capture the sample groupings, RFoT fixes the first dimension to represent each sample from our dataset. In an ensemble of random tensor configurations setting, $\mathcal{M}^{(i)}_{::0} = \mathbf{A}^{(i,1)} \in \mathbb{R}^{\mathrm{N_1 \times R_i}}$ is the latent factors matrix for the first dimension representing the N_1 malware and benign instances for the ith random tensor. MS clustering is applied to each $\mathbf{A}^{(i,1)}_{:r} = \mathbf{a}_r^{(i,1)}$, to capture $G_r^{(i)}$ number of clusters from component r, such that the total number of clusters found from ith tensor is $G^{(i)} = G_0^{(i)} + G_1^{(i)} + \cdots + G_{R_i-1}^{(i)}$. In an ensemble notation, we will let each cluster with a set of samples to be denoted with $\mathbf{g}_{j,r}^{(i)}$, where $0 \leq j \leq G_r^{(i)}$, for the rth component of ith tensor decomposition. With Component clustering, each cluster is $\mathbf{g}_{r,r}^{(i)} = \mathbf{A}^{(i,1)}_{:r}$, and the total number of clusters $G^{(i)} = R_i$ for the ith tensor decomposed to rank R_i.

Random Tensor Configuration Sampling: Our random tensor sampling includes a random selection of the number of dimensions, the features to represent each dimension, tensor entry, and random or fixed tensor rank. For each i random tensor, we first randomly choose the number of dimensions D_i with replacement, such that the range of D_i is $3 \leq D_i \leq \beta - 1$, where β is the total number of features from the original matrix $\mathbf{X} \in \mathbb{R}^{\mathrm{N_1 \times \beta}}$. The minimum and maximum number of dimensions a random tensor configuration can have is controlled using the RFoT hyper-parameters (*min_dims*, *max_dims*), where *dims* is short for dimensions. Here *min_dims* >= 3 since tensors have at least 3 dimensions, and *max_dims* <= β since we need one of the features to be the tensor entry. The first dimension is size N_1 representing each malware and benign-ware sample, and the features representing the remaining $D_i - 1$ dimensions are randomly selected from β features without replacement. Next, from the remaining $\beta - D_i$ features, which represent the feature(s) that are not selected to be a tensor dimension, RFoT randomly selects the feature to be used as the tensor entry with replacement. Finally, rank R_i is selected randomly, also with replacement, or each random tensor is assigned a user-defined fixed rank $R_i = rank$.

Tensor rank determines the number of hidden features, or latent components, that the tensor decomposition should extract. If we choose the rank to be too low, then we may miss vital information (*under-fitting*), while if the rank is chosen to be too high, then we might include noise in our solution (*over-fitting*) [Vangara et al.(2020)]. If we under-fit or over-fit the solution, then the latent factors might not have meaningful patterns to cluster benign and malware instances. By randomly selecting the rank, we attempt to avoid the need to correctly determine the rank of the tensor. Using the philosophy *"wisdom of crowds"*, we hope that an ensemble of tensor decomposition collectively can reach a consensus in

the extracted patterns and allow precise malware detection. By doing so, we let the ensemble members, random tensors, complement each others' weaknesses. Specifically, we want to cancel out the impact of the cases where we obtain poor tensor decomposition results by deriving a decision based on the majority of the population. This hypothesis assumes that the cases with impure clusters do not represent the majority of the population.

As we sample each random tensor configuration, we do not check if the same configuration was already used. This check is avoided to ensure that the random sampling process remains fast as the size of the ensemble grows. Therefore, the aforementioned four steps for sampling random tensor configurations for building the ensemble of *n_estimators* tensors can result in repeated tensor configurations, which would need to be discarded after the sampling. Inspired from the previously introduced technique for fast sampling of zero tensor indices [Hong et al.(2020)], we over-sample the tensor configurations to lower the probability of the repeated tensor configurations. Therefore, we set the ensemble size to be $n\_estimators + (n\_estimators \cdot 0.1)$ random tensors. We then perform post-processing to keep the unique tensor configurations and under-sample the ensemble to be the size of at most *n_estimators*.

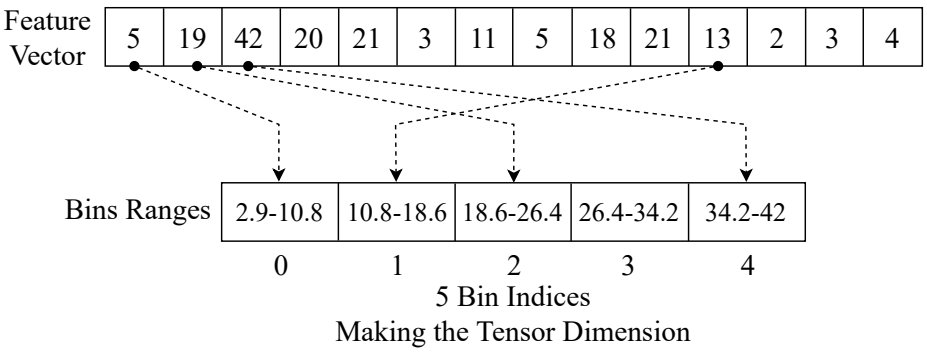

Fig. 7. Example of 4 numerical feature values being mapped to 5 bins to form a tensor dimension.

Feature to Tensor Dimension Mapping and Tensor Entry: Categorical features can easily be mapped to an index in the tensor dimension. For example, take the EMBER-2018 feature *has_signature*, a binary feature that can be a 0 or 1. If a tensor dimension represents this feature, the size of that dimension would be 2, where $has\_signature = 0$ would map to index 0, and $has\_signature = 1$ would map to index 1. This generalizes to any categorical feature where the labels can be encoded to retrieve features to dimension index mapping.

On the other hand, in order to use a numerical feature as a tensor dimension we need to utilize binning to map the given numerical value to a certain index

in the dimension. RFoT uses the *cut* API from the *Pandas* Python library to bin numerical values [pandas development team(2020), Wes McKinney(2010)]. The number of bins, or dimension size N_d, is determined by the RFoT hyper-parameter *bin_scale*, where the number of bins is $N_d = \textit{bin\_scale} \cdot \text{num\_unique}(\mathbf{f})$. Here num_unique($\mathbf{f}$) gives the total number of unique elements present in a given feature vector $\mathbf{f}$, which is one of β features. We provide an example in Figure 7 which shows how numerical features are mapped to 5 bins to from a tensor dimension.

We can utilize a real example to further explain how a tensor for malware data can be built using numerical feature binning and categorical feature mapping. For example, let ith random tensor $\mathcal{X}^{(i)}$ have the dimensions *Sample - Number of Strings - Has Signature*, and entry *Number of Sections*. *Sample* dimension represents each N_1 malware and benign-ware sample. The dimension *Number of Strings* represents the number of printable strings present in a given malware or benign instance, with size $N_2 = \textit{bin\_scale} \cdot \text{num\_unique}(\mathbf{f})$, such that the EMBER-2018 features for the number of strings will map to an index between 0 and $N_2 - 1$. The categorical dimension *Has Signature* identifies if a given specimen has a signature or not, thus the size of the last dimension is $N_3 = 2$. Finally, the tensor entry *Number of Sections* determines the number of sections present in the PE header of a given file. An entry $\mathcal{X}^{(i)}_{n,s,f}$ in this tensor represents the number sections that a specimen $n \in [0, 1, \cdots, N_1 - 1]$, with number of strings that bins to an index $s \in [0, 1, \cdots, N_2 - 1]$, and with the signature flag $f \in [0, 1]$ has.

4.3 Semi-supervised Classification with RFoT

Tensor decomposition extracts latent patterns from multi-dimensional data in an unsupervised fashion, and we capture these patterns for malware and benign samples using clustering techniques as described in Section 4.1. Using the captured clusters, we formulate a semi-supervised classifier that utilizes the information found by tensor decomposition. In this section, we first describe how the semi-supervised voting over the clusters is performed. This include the cases where the model is unable to make a decision for a given sample, and thus *abstaining vote* is given. We then introduce the *Cluster Uniformity Score* that is used as a threshold to filter out noisy, or non-uniform clusters.

RFoT takes a dataset $\mathbf{X} \in \mathbb{R}^{n \times \beta}$, where n is the number of samples and β is the number of features, and a vector $\mathbf{y}$ that represents the labels for each n samples such that $y_n \in [-1, 0, 1, \cdots, C - 1]$. Note that -1 is used for the unknown specimens, and C is the number of classes. In this chapter, we have $C = 2$ for malware and benign-ware, such that $y_n \in [-1, 0, 1]$ where 0 labels the benign-ware and 1 labels the malware. When we obtain a cluster, we use the known samples (samples with labels) as a reference to help us make a decision against the unknown samples (samples without labels, or -1) within that cluster. Specifically, the class vote of the given unknown samples that are in the cluster $\mathbf{g}^{(i)}_{j,r}$ is determined by the majority class of the known samples that are in the

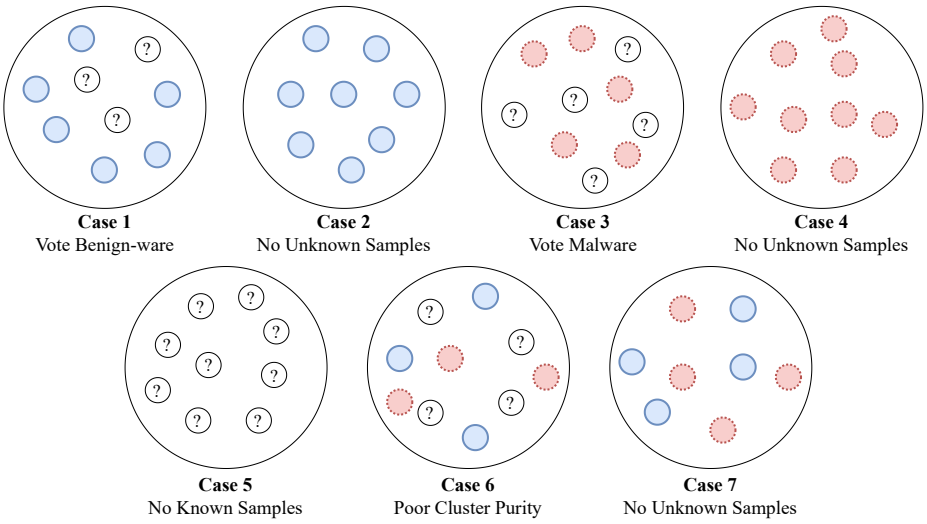

Fig. 8. Possible cases of clusters that can be seen

same cluster $\mathbf{g}_{j,r}^{(i)}$. There are 7 possible cases of cluster characteristics that we can obtain from the latent components, which are shown in Figure 8. In **Case 1**, we may have a cluster containing a set of unknown specimens and a set of known benign-ware. In this case, we would vote the unknown specimens as benign files. Similarly, we can vote the unknown specimens as malware if the majority of the known specimens are malware in the same cluster, as shown in **Case 3**. It is also possible to come across clusters where no unknown specimens are present, as shown in **Case 2**, **Case 4**, and **Case 7**. If there are no unknown samples in a given cluster, we disregard the cluster since we do not need to perform any voting.

This semi-supervised setup for classifying unknown specimens via clustering allows us to perform abstaining predictions (i.e. predict *"I do not know"*) due to not being able to obtain a class vote for a given sample. For instance, if a cluster consists of only a set of unknown specimens, as shown in **Case 5**, we cannot take a vote for these samples since we do not have any labeled instances to inform us regarding the class vote. We also cannot take a vote for the samples that are masked out due to the lack of signals (samples that are close to 0 with a certain threshold), as described in Section 4.1, since these instances would not be used in clustering. If a given sample always falls in a cluster without any known samples, as in **Case 5**, for each random tensor $\mathcal{X}^{(i)}$ and its latent factor for the first dimension $\mathbf{A}^{(i,1)}$ obtained by the decomposition, then this sample is predicted to be abstaining, or its label is kept as unknown (-1). Similarly, if a given sample n is consistently masked out due to being near zero in each $\mathbf{A}^{(i,1)}$, then it is predicted to be abstaining.

For the samples that do get class vote(s), we perform max-vote to determine the final class prediction. That is, if a given specimen n has its majority of the votes (over 50%) representing one of the C classes, the instance n is predicted to be that class.

Cluster Uniformity Score: It is possible to encounter a cluster that is not uniform in representing a single class (cluster have known instances from multiple classes). We have already shown an example of a latent factor with non-uniform clusters in Figure 4, where noisy clusters occur. In Figure 8, **Case 6** demonstrates a cluster where we have a mix of known malware and benign-ware specimens. In such cases, we cannot obtain an accurate class vote from the cluster. To filter out these clusters, we use the *Cluster Uniformity Score* which is calculated based on the fraction of the most dominant known class in the given cluster $\mathbf{g}_{j,r}^{(i)}$. We have previously used cluster uniformity score in our prior work for determining the uniformity of the clusters based on known specimens [Eren et al.(2023)]. In this chapter, we utilize the same metric, but re-formulate it to match with our ensemble of tensors notation as follows:

$$U^{\mathbf{g}_{j,r}^{(i)}} = \frac{|\max(\mathbf{g}_{j,r}^{(i)^{known}})|}{|\mathbf{g}_{j,r}^{(i)^{known}}|} \tag{11}$$

Here $U^{\mathbf{g}_{j,r}^{(i)}}$ is the cluster uniformity score for jth cluster obtained from rth component of tensor decomposition of ith tensor, $\mathbf{g}_{j,r}^{(i)}$. $|\max(\mathbf{g}_{j,r}^{(i)^{known}})|$ is the number of samples that belongs to the most dominant class with known samples in the cluster $\mathbf{g}_{j,r}^{(i)}$, while $|\mathbf{g}_{j,r}^{(i)^{known}}|$ is the total number of known samples in the cluster. The clusters where $U^{\mathbf{g}_{j,r}^{(i)}}$ is below the specified uniformity threshold t are removed from consideration, and thus no class vote is obtained from these clusters. If a given specimen n continuously falls in the clusters that are removed due to poor purity, it is also predicted to be abstaining at the end.

4.4 Putting it Together: the RFoT Algorithm

We summarize the RFoT methodology in Figure 9, and with pseudo-code in Algorithm 1. We first randomly sample tensor configurations (1). Then each tensor configuration is factorized to obtain the latent components (2). Within each latent component, we look at the latent factor representing each malware and benign-ware sample (2). Clustering is applied to capture the groupings within each of these latent factors (2). We filter out the noisy clusters using the cluster uniformity score. In the cases where we were able to acquire clean clusters, we take a class vote in a semi-supervised fashion (2). After each tensor is factorized, and class votes are obtained from each latent factor for the first dimension, we get the final class prediction via max-vote (3). The specimens are predicted to

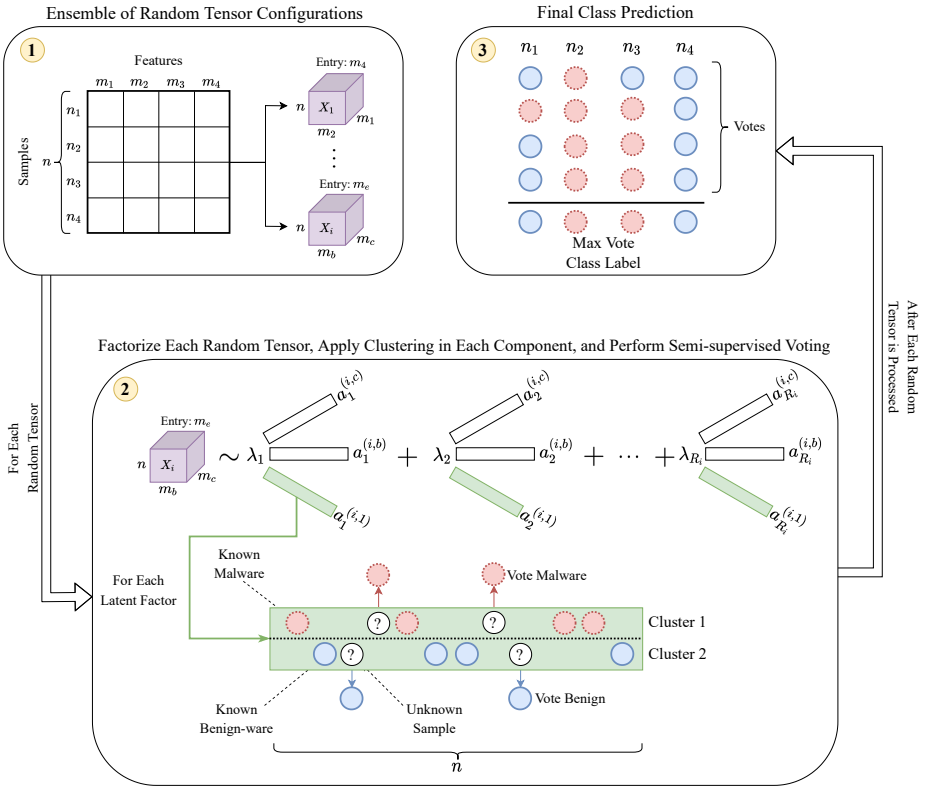

Fig. 9. RFoT methodology overview

be abstaining if they did not get any class vote due to either being part of clusters that were not uniform, not falling in a cluster that had known samples, or because they were masked out due to not having a signal.

We finally note that our implementation of RFoT computes the decomposition of the ensemble of random tensor configurations in a parallel fashion, since they are independent of one another. The parallel computation of the members of ensemble allows us to reduce the total time needed for prediction. Specifically, in Algorithm 1, lines 3 through 15 are executed in parallel based on the number of jobs that the user wants to run.

Now that we have introduced our methodology, we will next showcase the experiment results from a case-study where we classified malware and benign samples from the EMBER-2018 dataset using RFoT.

5 Experiments

In this section, we begin by introducing the dataset and the features utilized in our experiments. Following that, we provide a summary of the performance

Algorithm 1 RFoT($\mathbf{X}$, $\mathbf{y}$, *n_estimators*, *bin_scale*, t, R, *min_dims*, *max_dims*)

tensor_configs = sample_tensors($\mathbf{X}$, *n_estimators*, R, *min_dims*, *max_dims*)
class_votes = []
for config in tensor_configs **do** ▷ Start the parallel execution
  $\boldsymbol{\mathcal{X}}^{(i)}$ = build_tensor(bin_features($\mathbf{X}$, config), config) ▷ COO format
  $\boldsymbol{\mathcal{M}}^{(i)}$ = decompose($\boldsymbol{\mathcal{X}}^{(i)}$, R_i) ▷ CP-ALS or CP-APR
  $\mathbf{A}^{(i,1)}$ = get_signals($\boldsymbol{\mathcal{M}}^{(i)}_{::0}$) ▷ Mask out near zero elements for the mode-1
  clusters = cluster_latent_factor($\mathbf{A}^{(i,1)}_{:r}$) ▷ For each R_i, MS or Component
  for $\mathbf{g}^{(i)}_{j,r}$ in clusters **do**
    if $\mathbf{g}^{(i)}_{j,r}$ in [**Case** 4,5,6 or 7] **then** ▷ See Figure 8 for the cases
      continue ▷ Abstaining votes
    else
      class_votes.append(vote($\mathbf{g}^{(i)}_{j,r}$, $\mathbf{y}$)) ▷ Semi-supervised voting
    end if
  end for
end for ▷ End the parallel execution
$\mathbf{y}_{\text{pred}}$ = max_vote(class_votes, $\mathbf{y}$) ▷ Final class prediction
return $\mathbf{y}_{\text{pred}}$

evaluation metrics applied in our studies. Subsequently, we compare our results to the baseline models and assess the performance of RFoT alongside the baseline models as the labeled data percentage decreases.

5.1 EMBER-2018 Dataset and Experiment Setup

The availability of publicly accessible malware datasets for benchmarking ML methods has historically been limited. This constraint stems from challenges related to acquiring labeled malware data, encompassing issues such as copyright concerns for benign data and the time-consuming, expensive nature of labeling malware data [Raff and Nicholas(2020)]. To mitigate this problem, Anderson et al. introduced the EMBER-2018 dataset [Anderson and Roth(2018)]. Since its release, EMBER-2018 has emerged as a popular dataset employed by researchers to assess the efficacy of their ML-based techniques in the malware domain. Consequently, we utilize the EMBER-2018 dataset in our study to evaluate the capabilities of our introduced algorithm, RFoT.

EMBER-2018 consist of PE header and file meta-data features drawn from 1.1 million Windows malware and benign-ware, out of which 800,000 of them has labels. In this study we use 9 PE header features to construct our tensors, and train the baseline models. Specifically, following features are used:

1. *Number of Strings:* the number of printable strings in the file
2. *Strings Entropy:* randomness measurement for the printable strings
3. *File Size:* size of the executable in bytes

4. *Number of Exports:* number of functions being exported by the binary
5. *Number of Imports:* number of functions being imported by the binary
6. *Size of Code:* the size of the code section (.txt) in PE header
7. *Number of Sections:* the total number of sections in the PE header
8. *Has Debug:* flag to indicate if the debug value is on/off
9. *Has Signature:* flag to indicate if the file has a signature

To conduct our experiments, we build 10 smaller subsets out of all the 800,000 instances in EMBER-2018, where each subset contains 10,000 balanced amounts of malware and benign-ware. We apply our experiments to every 10 subset data to show that the reported results are statistically significant. Our final results are reported with a 95% Confidence Interval (CI). In Section 5.3, where the several evaluation metrics used to report the performance of our method as compared to the baseline models, we make the test set size to be 30% of 10,000 malware and benign instances for each 10 subsets. For RFoT, 30% test size gives us the number of samples without labels (unknown set).

5.2 Performance Evaluation Metrics

Precision, Recall, and F1 Measure: To evaluate the performance of our method and the baseline models, we use Precision, Recall, and F1 score. Precision score measures the ability of the model's correctly identify the positive class and can be calculated as follows:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{12}$$

where TP is the number of true positive predictions, FP is the false positives. The Recall metric measures the extent to which model can detect the positive class, malware in our case, and it is calculated as follows:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{13}$$

where FN is the number of false negatives. F1 score is calculated using both the Precision and Recall scores together; therefore, F1 score is only high when both Precision and Recall are high. F1 score can be calculated as follows:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{14}$$

More specifically, F1 can be calculated as follows:

$$F_1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{15}$$

Execution Time and Abstaining Predictions: The other two metrics used in our results are the execution time for both RFoT and baseline models, and the percent of abstaining predictions for RFoT. The execution time for the baseline models *XGBoost*, *LightGBM*, and *XGBoost+SelfTrain* (the semi-supervised extension of *XGBoost*) include the time it took to train the models and make predictions. Finally, the percentage of the abstaining predictions gives tells us the percent of unknown samples that remained unknown (or with label -1) after the prediction.

5.3 RFoT Performance and Baseline Comparisons

Table 2. Baseline comparisons

Model	Method	F1	Precision	Recall	Abstaining (%)	Time (sec)
RFoT (Component, CP-ALS)	Semi-supervised	**0.968** (+-0.005)	**0.968** (+-0.005)	**0.968** (+-0.006)	75.703 (+- 0.863)	536.151 (+- 5.132)
RFoT (MS, CP-ALS)	Semi-supervised	0.913 (+-0.005)	0.915 (+-0.004)	0.913 (+-0.005)	58.158 (+- 0.399)	554.831 (+- 5.263)
RFoT (Component, CP-APR)	Semi-supervised	0.940 (+-0.016)	0.941 (+-0.016)	0.940 (+-0.016)	93.220 (+- 1.534)	880.700 (+- 21.192)
RFoT (MS, CP-APR)	Semi-supervised	0.793 (+-0.008)	0.805 (+-0.007)	0.797 (+-0.008)	54.218 (+- 1.646)	1582.156 (+- 20.056)
LightGBM	Supervised	0.871 (+-0.005)	0.871 (+-0.005)	0.871 (+-0.005)	NA	**78.595** (+- 6.040)
XGBoost	Supervised	0.873 (+-0.005)	0.874 (+-0.005)	0.873 (+-0.006)	NA	93.805 (+- 2.752)
XGBoost+SelfTrain	Semi-supervised	0.872 (+-0.006)	0.873 (+-0.006)	0.873 (+-0.006)	NA	87.410 (+- 6.813)

We compare RFoT with CP-ALS and CP-APR decomposition, using MS and Component clustering, against baseline models *XGBoost*, *LightGBM*, and *XGBoost+SelfTrain*. For CP-APR we use 16 parallel jobs to decompose each random tensor using GPUs, while for CP-ALS is decomposed with 50 parallel jobs on CPUs. We tune the baseline models using a popular Python package *Optuna* [Akiba et al.(2019)]. *XGBoost* and *LightGBM* tuned with 3-fold stratified cross-validation and 50 trials to identify the optimal hyper-parameters. The tuning settings, or search space for the optimal hyper-parameters, listed below are the same from our prior work on semi-supervised malware family classification [Eren et al.(2021)].

For *LightGBM*, we used 250 maximum number of iterations, *gbdt* boosting type, and objective function *binary_logloss*. The following hyper-parameters were tuned (ranges are shown in parenthesis): *min_data_in_leaf* (5-100 in log scale), *max_depth* (2-7), *bagging_freq* (0-5), *bagging_fraction* (.5-1.0), *learning_rate* (.001-.1 in log scale), and *feature_fraction* (.1-.7).

As for *XGBoost*, we set the maximum boosting rounds to 250 and use the *binary-hinge* objective function. The following hyper-parameters were tuned, with ranges again shown in parentheses: *max_depth* (2-10), *eta* (.003-0.5 in log scale), *subsample* (.2-.7), *rounds* (10-300), *colsample_bytree* (.3-1.0), *colsample_bylevel* (.5-1.0), and *lambda* (.1-2.0). We use the same tuned hyper-parameters for *XGBoost* for the *XGBoost+SelfTrain* baseline model.

In Table 2, we compare RFoT with baseline models based on F1 score, Precision, Recall, and computation time in seconds. Additionally, we display the percentage of abstaining predictions for RFoT. From the table, it's evident that

each RFoT model outperforms every other baseline model. The RFoT model employing Component clustering and CP-ALS tensor decomposition achieves the highest F1 score of 0.968. However, this model also generates a significantly high abstaining prediction rate of 75.70%. Thus, an ideal model choice emerges with RFoT utilizing MS clustering and CP-ALS tensor decomposition, boasting a commendable F1 score of 0.91 and a lower abstaining prediction rate of 58%.

CP-APR with component clustering also demonstrates high performance with an F1 score of 0.94. However, it registers the highest abstaining prediction rate at 93.22%. In contrast, RFoT with CP-APR decomposition and MS clustering yields the lowest F1 score of 0.79. Among the models compared, the fastest performer is *LightGBM* clocking in at 78.59 seconds.

These results underscore that RFoT is an ideal model for precise malware detection, although it might predict a lower number of samples due to abstaining predictions. Notably, as a semi-supervised solution, RFoT surpasses supervised models, potentially offering better generalizability to novel malware.

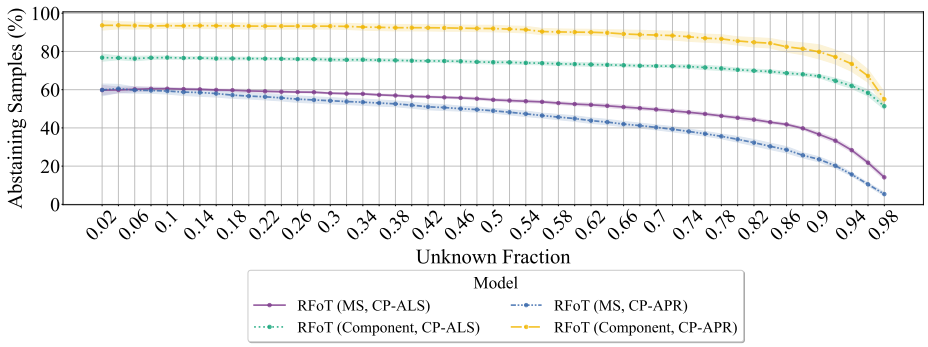

Fig. 10. Abstaining percentage is shown for different values of the unknown fraction.

Labeled Malware Data Scarcity Experiment: Compared to other ML fields, obtaining labeled malware data is time-consuming and expensive [Raff and Nicholas(2020)]. This issue is particularly problematic because popular supervised ML solutions for malware detection often require a large quantity of labeled data to achieve good performance. Additionally, Raff et al. emphasize that semi-supervised solutions in the realm of Windows malware classification have not received sufficient attention, despite their potential benefits such as improved generalizability to novel malware and achieving higher performance even with a limited quantity of labeled data [Raff and Nicholas(2020)].

Therefore, we conducted tests on the performance of our semi-supervised solution with a decreasing quantity of labeled data. We then compared its performance with that of the supervised and semi-supervised baseline models.

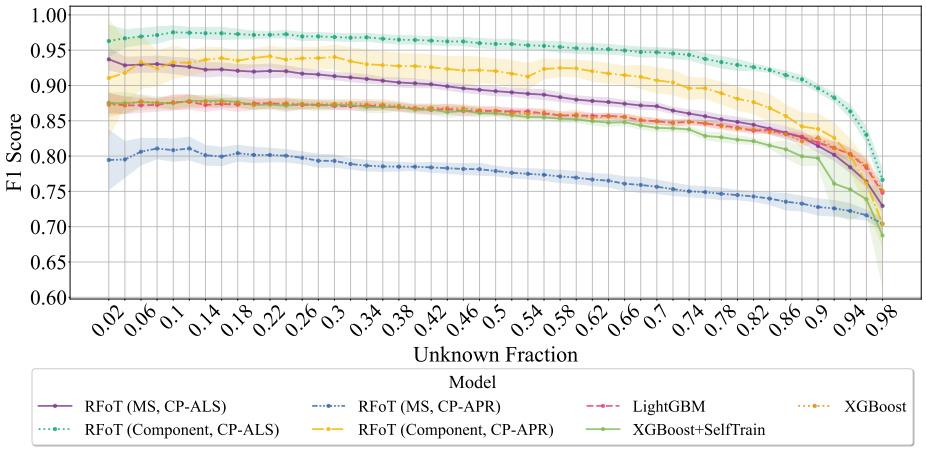

Fig. 11. F1 score is shown for different values of the unknown fraction.

We range the fraction of unknown specimens between 0.02 and 0.98 with the step size of 0.02. The fraction of unknown samples θ means that the proportion of the known samples would be $1 - \theta$. For the supervised baseline models, the fraction of the unknown samples θ is equivalent to the size of the test set, while the fraction of the known samples determines the training set size. The baseline models are also tuned in this experiment as described in Section 5.3.

In Figure 10, the percentage of abstaining predictions for RFoT with CP-ALS and CP-APR, along with MS and Component clustering, is illustrated. CP-APR with Component clustering exhibits the highest number of abstaining predictions, while CP-APR with MS clustering shows the lowest. Meanwhile, Figure 11 indicates that CP-APR with MS clustering demonstrates the lowest performance. This suggests that CP-APR struggles to identify meaningful patterns that differentiate the classes within each latent factor. However, CP-APR with Component clustering achieves high-performance results, albeit with a trade-off of a high number of abstaining predictions.

CP-ALS with MS and Component clustering presents a lower percentage of abstaining predictions along with higher F1 scores. As the fraction of unknown specimens increases, the percentage of abstaining predictions initially remains stable and then rapidly decreases, particularly after an unknown fraction of approximately 0.7, as depicted in Figure 10. This decline could be attributed to the cluster uniformity score's inability to filter out noisy clusters, as they are now represented by unknown specimens from the same class. Consequently, the other known specimens, which initially revealed the poor uniformity, are lost as the unknown fraction increases.

In Figure 10, it's noticeable that our baseline models generally demonstrate similar performance trends as the fraction of unknown specimens increases. However, there's a significant performance drop observed for *XGBoost+SleftTrain* after an unknown fraction of 0.74.

RFoT with CP-ALS and MS clustering, along with CP-APR with Component clustering, outperforms each baseline model until *XGBoost* and *LightGBM* start to surpass RFoT with CP-ALS and MS clustering after the unknown fraction reaches 0.86. Similarly, *XGBoost* and *LightGBM* outperform RFoT with CP-APR and Component clustering after an unknown fraction of 0.94.

Considering that abstaining predictions contribute to maintaining model performance, it's noteworthy that RFoT based on CP-ALS with Component clustering consistently outperforms each of the baseline models, irrespective of the fraction of unknown specimens.

We demonstrated the performance of RFoT and compared it to the tuned baseline models that prior studies have used to report state-of-the-art malware detection results. Our findings unveiled that RFoT, as a semi-supervised solution, exhibits superior capabilities in detecting malware compared to the baseline models, including the supervised algorithms. Furthermore, our experiments highlighted that RFoT can achieve higher accuracy in detecting malware even as the percentage of known samples decreases.

6 Conclusions

We introduced RFoT, a semi-supervised method that employs tensor decomposition to classify malware and benign ware. Tensor decomposition reveals significant latent patterns that clustering methods can capture, effectively distinguishing between malware and benign-ware. This approach allows us to utilize known samples as a reference point for voting on class labels of unknown specimens.

As the information extracted via tensor decomposition relies on the tensor's configuration, we devised a model that generates an ensemble of random configurations. Using a max-vote system based on the majority decisions within the ensemble population, we make the final class predictions.

Our experiments demonstrated that our semi-supervised method delivers more precise classification results compared to baseline supervised and semi-supervised ML models. However, this comes with the trade-off of predicting a lower number of samples due to abstaining predictions. Additionally, we have made our code publicly available to support the utility of tensor decomposition in malware analysis and to ensure the reproducibility of our results.

7 Acknowledgement

This manuscript has been assigned LA-UR-24-20205.

References

Abdelmonem et al.(2021). Salma Abdelmonem, Shahd Seddik, Rania El-Sayed, and Ahmed S. Kaseb. 2021. Enhancing Image-Based Malware Classification Using

Semi-Supervised Learning. In *2021 3rd Novel Intelligent and Leading Emerging Sciences Conference (NILES)*. Institute of Electrical and Electronics Engineers, 125–128. https://doi.org/10.1109/NILES53778.2021.9600511

Ahmadi et al.(2016). Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. 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*. ACM, 183–194.

Akiba et al.(2019). Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. ACM, 2623–2631.

Anderson and Roth(2018). H. S. Anderson and P. Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *ArXiv e-prints* (April 2018), 8. arXiv:1804.04637 [cs.CR]

Atluri(2019). Venkata Atluri. 2019. Malware Classification of Portable Executables using Tree-Based Ensemble Machine Learning. In *2019 SoutheastCon*. IEEE, IEEE, 1–6.

Azeez et al.(2021). Nureni Ayofe Azeez, Oluwanifise Ebunoluwa Odufuwa, Sanjay Misra, Jonathan Oluranti, and Robertas Damaševičius. 2021. Windows PE malware detection using ensemble learning. In *Informatics*. Multidisciplinary Digital Publishing Institute, Multidisciplinary Digital Publishing Institute, 10.

Bader and Kolda(2006). Brett W. Bader and Tamara G. Kolda. 2006. Algorithm 862: MATLAB Tensor Classes for Fast Algorithm Prototyping. *ACM Trans. Math. Software* 32, 4 (December 2006), 635–653. https://doi.org/10.1145/1186785.1186794

Bader et al.(2017). Brett W. Bader, Tamara G. Kolda, et al. 2017. MATLAB Tensor Toolbox Version 3.0-dev. Available online. https://gitlab.com/tensors/tensor%5Ftoolbox

Bak et al.(2020). Márton Bak, Dorottya Papp, Csongor Tamás, and Levente Buttyán. 2020. Clustering IoT Malware based on Binary Similarity. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, IEEE, 1–6.

Battaglino et al.(2018). Casey Battaglino, G. Ballard, and T. Kolda. 2018. A Practical Randomized CP Tensor Decomposition. *SIAM J. Matrix Anal. Appl.* 39 (2018), 876–901.

Bissell et al.(2020). K Bissell, R LaSalle, and P.D. Cin. 2020. *Innovate for Cyber Resilience*. Technical Report. Accenture, Ponemon Institute.

Bissell and Ponemon(2019). K. Bissell and L. Ponemon. 2019. *The Cost of Cybercrime*. Technical Report. Accenture, Ponemon Institute. https://www.accenture.com/%5Facnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf

Bruns-Smith et al.(2016). David Bruns-Smith, Muthu Manikandan Baskaran, James R. Ezick, Thomas Henretty, and Richard A. Lethin. 2016. Cyber Security through Multidimensional Data Decompositions. *2016 Cybersecurity Symposium (CYBERSEC)* (2016), 59–67.

Buitinck et al.(2013). Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

Chen and Guestrin(2016). Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. ACM, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

Chi and Kolda(2012). Eric C. Chi and Tamara G. Kolda. 2012. On Tensors, Sparsity, and Nonnegative Factorizations. *SIAM J. Matrix Anal. Appl.* 33 (2012), 1272–1299.

Dahl et al.(2013). George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 3422–3426. https://doi.org/10.1109/ICASSP.2013.6638293

Eren et al.(2021). M. E. Eren, M. Bhattarai, R. J. Joyce, E. Raff, C. Nicholas, and B. Alexandrov. 2021. *Semi-supervised Classification of Malware Families via Hierarchical Non-negative Matrix Factorization with Automatic Model Determination*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States). LA-UR-21-29919.

Eren et al.(2023). Maksim E. Eren, Manish Bhattarai, Robert J. Joyce, Edward Raff, Charles Nicholas, and Boian S. Alexandrov. 2023. Semi-Supervised Classification of Malware Families Under Extreme Class Imbalance via Hierarchical Non-Negative Matrix Factorization with Automatic Model Selection. *ACM Trans. Priv. Secur.* (sep 2023). https://doi.org/10.1145/3624567 Just Accepted.

Eren et al.(2020). M. E. Eren, J. S. Moore, and B. S. Alexandrov. 2020. Multi-Dimensional Anomalous Entity Detection via Poisson Tensor Factorization. In *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 1–6. https://doi.org/10.1109/ISI49825.2020.9280524

Eren et al.(2021). M. E. Eren, J. S. Moore, E. Skau, M. Bhattarai, G. Chennupati, and B. S. Alexandrov. 2021. pyCP_APR. https://github.com/lanl/pyCP_APR. https://doi.org/10.5281/zenodo.4840598

Eren et al.(2022). M. E. Eren, J. S. Moore, E. W. Skau, E. A. Moore, M. Bhattarai, G. Chennupati, and B. S. Alexandrov. 2022. General-Purpose Unsupervised Cyber Anomaly Detection via Non-Negative Tensor Factorization. *Digital Threats: Research and Practice* (2022), 28 pages. https://doi.org/10.1145/3519602

Ezick et al.(2019). James Ezick, Tom Henretty, Muthu Baskaran, Richard Lethin, John Feo, Tai-Ching Tuan, Christopher Coley, Leslie Leonard, Rajeev Agrawal, Ben Parsons, and William Glodek. 2019. Combining Tensor Decompositions and Graph Analytics to Provide Cyber Situational Awareness at HPC Scale. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7. https://doi.org/10.1109/HPEC.2019.8916559

Fanaee-T et al.(2014). Hadi Fanaee-T, Márcia D. B. Oliveira, João Gama, Simon Malinowski, and Ricardo Morla. 2014. Event and Anomaly Detection Using Tucker3 Decomposition. *ArXiv* abs/1406.3266 (2014).

Fonseca A et al.(2021). Fabian H. Fonseca A, Serena Ferracci, Federico Palmaro, Luca Iocchi, Daniele Nardi, and Luisa Franchina. 2021. Static Analysis of PE files Using Neural Network Techniques for a Pocket Tool. In *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. IEEE, 01–06. https://doi.org/10.1109/ICECCME52200.2021.9590958

Fukunaga and Hostetler(1975). K. Fukunaga and L. Hostetler. 1975. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory* 21, 1 (1975), 32–40. https://doi.org/10.1109/TIT.1975.1055330

Gupta and Rani(2020). Deepak Gupta and Rinkle Rani. 2020. Improving malware detection using big data and ensemble learning. *Computers & Electrical Engineering* 86 (2020), 106729.

Hansen et al.(2015). Samantha Hansen, Todd Plantenga, and Tamara G. Kolda. 2015. Newton-Based Optimization for Kullback-Leibler Nonnegative Tensor Factorizations. *Optimization Methods and Software* 30, 5 (April 2015), 1002–1029. https://doi.org/10.1080/10556788.2015.1009977

Hansen et al.(2016). Steven Strandlund Hansen, Thor Mark Tampus Larsen, Matija Stevanovic, and Jens Myrup Pedersen. 2016. An approach for detection and family classification of malware based on behavioral analysis. In *2016 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 1–5. https://doi.org/10.1109/ICCNC.2016.7440587

Harris et al.(2020). Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hong et al.(2020). David Hong, Tamara G. Kolda, and Jed A. Duersch. 2020. Generalized Canonical Polyadic Tensor Decomposition. *ArXiv* abs/1808.07452 (2020).

Hou et al.(2017). Jieqiong Hou, Minhui Xue, and Haifeng Qian. 2017. Unleash the Power for Tensor: A Hybrid Malware Detection System Using Ensemble Classifiers. In *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*. IEEE, 1130–1137. https://doi.org/10.1109/ISPA/IUCC.2017.00170

Huang and Stokes(2016). Wenyi Huang and Jay Stokes. 2016. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In *Proceedings of 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2016)* (proceedings of 13th international conference on detection of intrusions and malware, and vulnerability assessment (dimva 2016) ed.). Springer, 399–418. https://www.microsoft.com/en-us/research/publication/mtnet-multi-task-neural-network-dynamic-malware-classification/

IBM(2019). IBM. 2019. Cost of a Data Breach Report. , 23 pages.

Irofti and Băltoiu(2019). Paul Irofti and Andra Băltoiu. 2019. Malware Identification with Dictionary Learning. In *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE, 1–5. https://doi.org/10.23919/EUSIPCO.2019.8903043

Jiang et al.(2019). Jianguo Jiang, Song Li, Min Yu, Gang Li, Chao Liu, Kai Chen, Hui Liu, and Weiqing Huang. 2019. Android malware family classification based on sensitive opcode sequence. In *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, IEEE, 1–7.

Jin and Han(2017). Xin Jin and Jiawei Han. 2017. *Mean Shift*. Springer US, Boston, MA, 806–808. https://doi.org/10.1007/978-1-4899-7687-1_532

Kanehara et al.(2019). Hideaki Kanehara, Yuma Murakami, Jumpei Shimamura, Takeshi Takahashi, Daisuke Inoue, and Noboru Murata. 2019. Real-Time Botnet Detection Using Nonnegative Tucker Decomposition. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing* (Limassol, Cyprus) *(SAC*

'19). Association for Computing Machinery, New York, NY, USA, 1337–1344. https://doi.org/10.1145/3297280.3297415

Ke et al.(2017). Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

Kisil et al.(2018). Ilia Kisil, Ahmad Moniri, and Danilo P. Mandic. 2018. TENSOR ENSEMBLE LEARNING FOR MULTIDIMENSIONAL DATA. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 1358–1362. https://doi.org/10.1109/GlobalSIP.2018.8646694

Kolda and Bader(2009). Tamara G. Kolda and Brett W. Bader. 2009. Tensor Decompositions and Applications. *SIAM Rev.* 51, 3 (September 2009), 455–500. https://doi.org/10.1137/07070111X

Kong and Yan(2013). Deguang Kong and Guanhua Yan. 2013. Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 1357–1365. https://doi.org/10.1145/2487575.2488219

Koutra et al.(2012). Danai Koutra, Evangelos E. Papalexakis, and Christos Faloutsos. 2012. TensorSplat: Spotting Latent Anomalies in Time. In *2012 16th Panhellenic Conference on Informatics*. IEEE, 144–149. https://doi.org/10.1109/PCi.2012.60

Kumar and Geetha(2020). Rajesh Kumar and S Geetha. 2020. Malware classification using XGboost-Gradient boosted decision tree. *Advances in Science, Technology and Engineering Systems Journal* 5, 5 (2020).

Labs(2020). Malwarebytes Labs. 2020. *State of Malware Report*. Technical Report. Malwarebytes Labs. 57 pages.

Ling et al.(2019). Yeong Tyng Ling, Nor Fazlida Mohd Sani, Mohd Taufik Abdullah, and Nor Asilah Wati Abdul Hamid. 2019. Nonnegative matrix factorization and metamorphic malware detection. *Journal of Computer Virology and Hacking Techniques* 15, 3 (2019), 195–208.

Liu et al.(2017). Liu Liu, Bao-sheng Wang, Bo Yu, and Qiu-xi Zhong. 2017. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering* 18, 9 (2017), 1336–1347.

Liu et al.(2020). Xinbo Liu, Yaping Lin, He Li, and Jiliang Zhang. 2020. A novel method for malware detection on ML-based visualization technique. *Computers & Security* 89 (2020), 101682. https://doi.org/10.1016/j.cose.2019.101682

Loi et al.(2021). Nicola Loi, Claudio Borile, and Daniele Ucci. 2021. Towards an Automated Pipeline for Detecting and Classifying Malware through Machine Learning. *arXiv preprint arXiv:2106.05625* (2021).

Maciá-Fernández et al.(2018). Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. 2018. UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs. *Comput. Secur.* 73 (2018), 411–424. https://doi.org/10.1016/j.cose.2017.11.004

Maruhashi et al.(2011). Koji Maruhashi, Fan Guo, and Christos Faloutsos. 2011. MultiAspectForensics: Pattern Mining on Large-Scale Heterogeneous Networks with

Tensor Analysis. In *2011 International Conference on Advances in Social Networks Analysis and Mining*. IEEE, 203–210. https://doi.org/10.1109/ASONAM.2011.80

Mohaisen et al.(2015). Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. 2015. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Computers & Security* 52 (2015), 251–266. https://doi.org/10.1016/j.cose.2015.04.001

Narayanan and Davuluru(2020). Barath Narayanan Narayanan and Venkata Salini Priyamvada Davuluru. 2020. Ensemble Malware Classification System Using Deep Neural Networks. *Electronics* 9, 5 (2020). https://doi.org/10.3390/electronics9050721

pandas development team(2020). The pandas development team. 2020. *pandas-dev/pandas: Pandas*. Zenodo. https://doi.org/10.5281/zenodo.3509134

Pedregosa et al.(2011). F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

Pham et al.(2018). Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. 2018. Static PE Malware Detection Using Gradient Boosting Decision Trees Algorithm. In *Future Data and Security Engineering*, Tran Khanh Dang, Josef Küng, Roland Wagner, Nam Thoai, and Makoto Takizawa (Eds.). Springer International Publishing, Cham, 228–236.

Qi et al.(2021). Panpan Qi, Wei Wang, Lei Zhu, and See Kiong Ng. 2021. *Unsupervised Domain Adaptation for Static Malware Detection Based on Gradient Boosting Trees*. Association for Computing Machinery, New York, NY, USA, 1457–1466. https://doi.org/10.1145/3459637.3482400

Qi et al.(2016). Yang Qi, Pierre Comon, and Lek-Heng Lim. 2016. Semialgebraic Geometry of Nonnegative Tensor Rank. *SIAM J. Matrix Anal. Appl.* 37 (2016), 1556–1580.

R. and K.P.(2018). Vinayakumar R. and Soman K.P. 2018. DeepMalNet: Evaluating shallow and deep networks for static PE malware detection. *ICT Express* 4, 4 (2018), 255–258. https://doi.org/10.1016/j.icte.2018.10.006

Rabanser et al.(2017). Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. 2017. Introduction to Tensor Decompositions and their Applications in Machine Learning. *ArXiv* abs/1711.10781 (2017).

Raff et al.(2018). Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K. Nicholas. 2018. Malware Detection by Eating a Whole EXE. In *AAAI Workshops*. AAAI.

Raff and Nicholas(2020). Edward Raff and C. Nicholas. 2020. A Survey of Machine Learning Methods and Challenges for Windows Malware Classification. *ArXiv* abs/2006.09271 (2020).

Raff et al.(2020). Edward Raff, Charles K. Nicholas, and Mark McLean. 2020. A New Burrows Wheeler Transform Markov Distance. In *AAAI*. AAAI.

Ramadhan et al.(2021). Fauzan Hikmah Ramadhan, Vera Suryani, and Satria Mandala. 2021. Analysis Study of Malware Classification Portable Executable Using Hybrid Machine Learning. In *2021 International Conference on Intelligent Cybernetics Technology Applications (ICICyTA)*. IEEE, 86–91. https://doi.org/10.1109/ICICyTA53712.2021.9689130

Roseline et al.(2019). S. Abijah Roseline, A. D. Sasisri, S. Geetha, and C. Balasubramanian. 2019. Towards Efficient Malware Detection and Classification using Multilayered Random Forest Ensemble Technique. In *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 1–6. https://doi.org/10.1109/CCST.2019.8888406

Sikorski and Honig(2012). Michael Sikorski and Andrew Honig. 2012. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press.

Sun et al.(2017). Bowen Sun, Qi Li, Yanhui Guo, Qiaokun Wen, Xiaoxi Lin, and Wenhan Liu. 2017. Malware family classification method based on static feature extraction. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 507–513. https://doi.org/10.1109/CompComm.2017.8322598

Sun et al.(2006). Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *KDD '06*. ACM.

The Independent IT Security Institute(2022). The Independent IT Security Institute. 2022. Malware Statistics & Trends Report: AV-TEST. https://www.av-test.org/en/statistics/malware/

Vangara et al.(2020). R. Vangara, E. Skau, G. Chennupati, H. Djidjev, T. Tierney, J. P. Smith, M. Bhattarai, V. G. Stanev, and B. S. Alexandrov. 2020. Semantic Nonnegative Matrix Factorization with Automatic Model Determination for Topic Modeling. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 328–335. https://doi.org/10.1109/ICMLA51294.2020.00060

Verizon(2021). Verizon. 2021. *Data Breach Investigations Report 2021*. Technical Report. Verizon. 119 pages. https://enterprise.verizon.com/resources/reports/dbir/

Vinayakumar et al.(2019). R. Vinayakumar, Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. 2019. Robust Intelligent Malware Detection Using Deep Learning. *IEEE Access* 7 (2019), 46717–46738. https://doi.org/10.1109/ACCESS.2019.2906934

Wang et al.(2021). Shuwei Wang, Qiuyun Wang, Zhengwei Jiang, Xuren Wang, and Rongqi Jing. 2021. A Weak Coupling of Semi-Supervised Learning with Generative Adversarial Networks for Malware Classification. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 3775–3782. https://doi.org/10.1109/ICPR48806.2021.9412832

Wes McKinney(2010). Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). SciPy, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a

Wu and Yang(2007). Kuo-Lung Wu and Miin-Shen Yang. 2007. Mean Shift-Based Clustering. *Pattern Recogn.* 40, 11 (nov 2007), 3035–3052. https://doi.org/10.1016/j.patcog.2007.02.006

Xie et al.(2017). Kun Xie, Xiaocan Li, Xin Wang, Gaogang Xie, Jigang Wen, Jiannong Cao, and Dafang Zhang. 2017. Fast Tensor Factorization for Accurate Internet Anomaly Detection. *IEEE/ACM Transactions on Networking* 25, 6 (2017), 3794–3807. https://doi.org/10.1109/TNET.2017.2761704

Yan et al.(2018). Jinpei Yan, Yong Qi, and Qifan Rao. 2018. Detecting malware with an ensemble method based on deep neural network. *Security and Communication Networks* 2018 (2018).

Yarowsky(1995). David Yarowsky. 1995. Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics* (Cambridge, Massachusetts) *(ACL '95)*. Association for Computational Linguistics, USA, 189–196. https://doi.org/10.3115/981658.981684

Ye et al.(2010). Yanfang Ye, Tao Li, Yong Chen, and Qingshan Jiang. 2010. Automatic Malware Categorization Using Cluster Ensemble. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Washington, DC, USA) *(KDD '10)*. Association for Computing Machinery, New York, NY, USA, 95–104. https://doi.org/10.1145/1835804.1835820

Yuan et al.(2020). Baoguo Yuan, Junfeng Wang, Dong Liu, Wen Guo, Peng Wu, and Xuhua Bao. 2020. Byte-level malware classification based on markov images and deep learning. *Computers & Security* 92 (2020), 101740. https://doi.org/10.1016/j.cose.2020.101740

Zhang et al.(2019). Shao-Huai Zhang, Cheng-Chung Kuo, and Chu-Sing Yang. 2019. Static PE Malware Type Classification Using Machine Learning Techniques. In *2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA)*. IEEE, 81–86. https://doi.org/10.1109/ICEA.2019.8858297

Zhang et al.(2017). Yunan Zhang, Chenghao Rong, Qingjia Huang, Yang Wu, Zeming Yang, and Jianguo Jiang. 2017. Based on Multi-features and Clustering Ensemble Method for Automatic Malware Categorization. In *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 73–82. https://doi.org/10.1109/Trustcom/BigDataSE/ICESS.2017.222