

# Efficient Malware Analysis Using Metric Embeddings

Ethan M. Rudd<sup>1</sup>, David Krisiloff<sup>1</sup>, Daniel Olszewski<sup>2</sup>, Edward Raff<sup>3,4</sup> and James Holt<sup>4</sup>

<sup>1</sup>Mandiant Inc.

<sup>2</sup>University of Florida

<sup>3</sup>Booz Allen Hamilton

<sup>4</sup>Laboratory for Physical Sciences, University of Maryland

## Abstract

Machine learning-based malware classification has become a key component of modern defense-in-depth strategies, with focus placed on the binary classification task of malware detection. These detection models are typically combined with other toolchains, which provide additional context necessary for triage and remediation, including detection names, capability, and type information. The resulting systems are often complex and interconnected, incurring significant technical debt, infrastructure costs, and inevitable errors.

In this paper, we examine the feasibility of using machine learning to streamline malware analysis pipelines in a manner which minimizes potential risks and costs while preserving flexibility and functionality. To this end, we explore the use of metric learning to embed malicious and benign samples in a low-dimensional vector space with enriched capability information for downstream use in a variety of applications, including detection, family classification, and malware attribute classification.

Specifically, we enrich labeling on malicious and benign PE files from the EMBER dataset using Mandiant's CAPA tool, an open-source toolchain which uses disassembly and subject matter expert (SME) derived rules and heuristics to determine malicious capabilities. Using these CAPA labels, we derive several different types of metric embeddings utilizing an embedding neural network trained via contrastive loss, Spearman rank correlation on malware similarity, and combinations thereof.

We then examine performance on a variety of transfer tasks performed on the EMBER and SOREL datasets. We show that for a variety of transfer tasks, we are able to utilize relatively low-dimensional metric embeddings with little decay in performance, which offers the potential to quickly retrain for a variety of transfer tasks. The low-dimensional representations offer added potential to significantly reduce training and storage overhead when performing retrains or transferring to additional downstream tasks.

## Keywords

Metric Embeddings, Machine Learning, Malware Analysis, Information Security

## 1. Introduction

Malware analysis is a complex process involving highly skilled experts and many person-hours. Given the number of new files seen each day (more than 500,000 on VirusTotal alone [1]) automation of malware analysis is a necessity. Development of new analysis tools provides an avenue for more efficient malware analysis teams.

Fortunately, malware analysis tasks are often amenable to machine learning (ML) solutions. The tasks (e.g., malware detection or malware family classification) are complex enough that traditional rules-based approaches remain brittle and require frequent updating. At the same time, it is possible to acquire and label large data sets to train ML models using threat feeds and crowdsourcing services, like VirusTotal or Reversing Labs.


One notable downside, however, is that ML models come with significant technical debt: they need to be retrained as malware evolves and often interdependencies between various model components can be hard to understand or predict [2]. Even with a consistent feature vector representation, when training on industry-scale datasets, feature stores may require tens of terabytes and model re-training can take multiple weeks.

---

CAMLIS'22: Conference on Applied Machine Learning in Information Security (CAMLIS), October 20–21, 2022, Arlington, VA  
✉ ethan.rudd@mandiant.com (E. M. Rudd); david.krisiloff@mandiant.com (D. Krisiloff); dolszewski@ufl.edu (D. Olszewski); raff\_edward@bah.com (E. Raff); holt@lps.umd.edu (J. Holt)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

This motivates the question: what if we can use ML to derive low-dimensional representations which capture semantic behavior of malware/goodware that can be used to speed up and reduce resource requirements for downstream tasks? This could significantly enhance capability for training classifiers for novel applications, performing rapid iteration/experimentation, and efficiently updating deployed models, all at reduced processing and storage requirements.

Since other applications of applied ML, including biometrics and information retrieval (IR) systems, have utilized metric learning to derive low-dimensional embeddings for similar downstream tasks, in this paper, we explore whether we can apply metric learning in a similar vein towards various malware analysis-oriented ML tasks. Using metric embeddings, we aim to simplify some of the engineering costs associated with running and maintaining a suite of different downstream ML tooling.

Contrary to other applications of ML classification, where data can trivially be assigned labels corresponding to one or more classes/attributes, labeling for malware analysis tasks can be more difficult [3]. Moreover, data for malware analysis often includes telemetry and metadata beyond hard labels which could ideally be used to enrich our metric embeddings. In this paper, we explore techniques to enrich our embeddings with complex semantic information provided by computationally-expensive tools (e.g., disassembly). This allows us to explore whether it is possible to approximate more expensive analysis with lower-overhead static representations.

When generating our embeddings, we utilize Mandiant’s CAPA tool; an open source tool, which utilizes rules and heuristics in conjunction with disassembly to yield capability labels (e.g., file read/write, registry key generation, process creation, data send/receive over networks, socket connection, base64/XOR encoding) associated with a given PE, ELF, and .NET file as well as shellcode snippets. We enrich samples from the EMBER dataset with capability labels, and using these labels generate different types of embeddings, including a Siamese embedding, which utilizes a contrastive loss over clusters of CAPA attributes, as well as a novel ranking embedding, which uses the Spearman Rank Correlation Coefficient as a loss and aims to embed ranked degree of similarity between different CAPA attribute clusters. We then perform comparisons of these different metric embedding loss functions across two different datasets: EMBER and SOREL-20M, on three different downstream transfer tasks: malware detection, malware family classification and malware attribute classification, making comparisons to original dataset benchmarks where applicable. Finally, we perform an analysis of adversarial robustness on the three respective embedding types.

## 2. Background and Related Work

Metric learning is a machine learning task that focuses on learning distances (i.e., metrics and/or measures) between objects that captures some semantically meaningful notion of similarity. These learned metric functions play an important role in fields including information retrieval, ranking, and recommendation systems [4]. The key property of a similarity metric/measure is that it maps similar objects close together and dissimilar objects far apart within the learned metric space. In practice, the objects are represented by a set of features, the metric function is the transformation of the features into a common metric space, and the learning process finds a transformation such that the similarity/dissimilarity behavior is correct with respect to the labels provided during training. Various learning architectures have been proposed, including Siamese networks that learn from the distances between pairs of objects, and triplet networks that use three samples to capture both similarity and dissimilarity to an anchor [5], [6]. There are also different loss functions for each architecture along with other subtle modifications of the learning process that can be applied to improve results (e.g., specialized algorithms for stochastic gradient descent [4]).

The Spearman embedding technique that we introduce is not the first to incorporate ranking information into a metric Embedding. The triplet based approach to learning was originally proposed as a method to learn a function for object ranking [7] and used an anchor  $a$  with a positive and negative pair  $a_p$ , and  $a_n$  respectively. This work has been extended and become popular in deep learning, with most works focusing on the embedding constraints (e.g., L2 normalized or unconstrained) and the approach

for finding triplet pairs [8, 9, 10, 11, 12, 13, 14], however recent work has shown that most of these gains may be attributed to learning algorithm improvements and model architecture (e.g., invention of batch-norm), rather than improvements in the triplet learning procedure [15]. In contrast our work comes full-circle, developing a ranking based loss based on the Spearman coefficient of correlation to try and capture more fine-grained information than the standard triplet loss.

To perform metric learning, we require a dataset of objects along with their similarities, which act as labels in a supervised ML setting. Since our objects are binary portable executables (PEs) we need to define a meaningful notion of similarity among binaries. Binary similarity gauges the likeness between two binary files and can be defined in several different ways. Perhaps the cleanest definition is that two binaries are similar if they were compiled from the same source code or contain a large fraction of the same source code [16], [17]. This definition is particularly useful from a malware reverse engineering standpoint: knowing that a file contains source code from a known piece of malware can significantly speed up analysis. However, this definition introduces problems when labeling a dataset of binary files built from unknown source code. More broadly, binary similarity can be defined as two files whose structures are similar. This definition is the one often used by those developing fuzzy or locality sensitive hashing algorithms [18, 19, 20, 21]. While less precise, these functions are computationally easier to compute and some are amenable to fast ( $\mathcal{O}(n \log n)$  or faster) database lookups. These hashes are routinely used in the malware analysis space, lacking any better option. Critically, recent work where ground-truth (as assessed by manual reverse engineering) that hashes of this nature can produce true-positive rates near 99.8%, at the cost of many false negatives [22, 23]. Given a sample  $x$ , this allows selecting a similar pair  $x_p$  with high confidence. Selecting a negative pair  $x_n$  can still be done with high confidence by random selection, since only a minority of samples will be similar to any given  $x$ .

To the best of our knowledge, ours is the first work to pursue a discriminative style triplet loss as our method of learning a general purpose feature representation. Prior methods generally fall into the category of being byte-based static processes or heavy lifting processes to extract higher level code representations. In both cases, these methods are not amenable to being feature vectors that can adjust to population change over time (i.e., quarterly retraining) in our deployment scenario (i.e., low overhead).

Prior byte-based and compression based approaches [24, 25] to constructing a feature vector have been proposed, which allow for similarity search in a computationally efficient manner. Similarly digital forensic hashes, which produce a hash-code of fixed or variable length that can be used to calculate similarities [26, 27, 28, 29, 30, 31, 26, 32] are another approach that works on raw byte content. These approaches are often fast and low-overhead. However, no learning step occurs for either of these types of approaches, which prevents the method from being able to adapt to changes in the population of malware.

The other primary approach are code similarity measures. These tend to work at either the disassembly [33, 34, 35] or disassembly call-graph [36, 37, 38], and use a neural network to train an auto-regressive model that can produce a fixed-length representation. This allows adapting the model over time, but the reliance on at least disassembling the given executable limits our ability to deploy such representations. Disassembly itself is computationally demanding, and often error-prone, as many times a file will not yield accurate disassembly without unpacking or deobfuscation. These processes may need to be done manually. Combined this makes the approach undesirable for our goals.

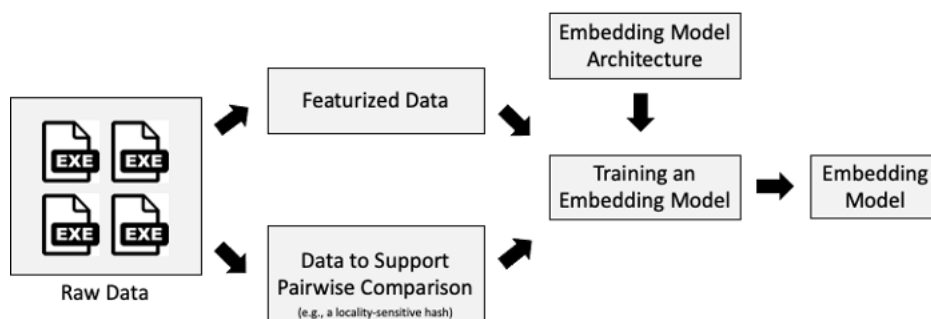
## 3. Approach

### 3.1. Overview

In this paper, we focus on building a model that produces embeddings of Windows PE files. The goal is to learn a representation that can be used for multiple downstream malware analysis tasks. The methodology can be separated into two phases. Figure 1 represents the first phase where an embedding model is trained. The starting point is the raw data, which in our case is the EMBER 2018 dataset [39], a

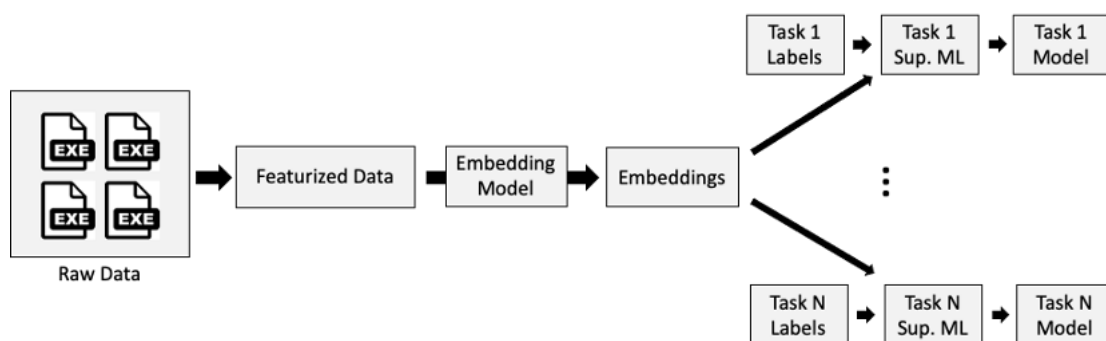
set of 1 million PE files seen in or before 2018. The raw data goes through two steps: (1) a featurization step, and (2) a step to compute file information that aids in determining the pairwise similarity between any two files. For featurization, we focus on subject matter expert (SME)-derived, static features that are efficient to compute and which capture a broad range of malicious signals. These features include things such as the APIs imported, parsing errors, entropy, and byte distributions.

To determine pairwise similarity between two PE files, one can use a tool like CAPA [40] that detects capabilities of executable files (i.e., two files with overlapping capabilities are similar). Notably, we hypothesize that using more complex similarity information than what is available naturally in the features (e.g., disassembly-based similarity vs. static features) will imbue the learned metric space with additional information without the added overhead of the more complex analysis techniques. Once the data and labels are defined, we specify an architecture for the embedding, which we instantiate with a multi-layered neural network. An algorithm then takes those components as input and trains an embedding model. The training algorithm can use a metric embedding network, e.g., a Siamese network or a triplet loss network to learn the model parameters that effectively cause similar PE files to be near one another in the embedding space.



**Figure 1:** A system diagram depicting the upstream training of an embedding model. Training is conducted using a selected embedding model architecture, feature vectors from executable binaries, and additional data to support pairwise sample comparisons (e.g., CAPA labels).

During the second phase, as shown in Figure 1, we measure the transferability of the embedding space to various malware analysis tasks. Concretely, we embed our training data and use that representation to train new models for each downstream task. By keeping the models used for the transfer process constant and only varying our embedding process we can make precise measurements of the utility of various similarity information, loss function, binary representation, or other parameters of our embedding network. The following sections detail our modeling setup.



**Figure 2:** A system diagram depicting the downstream use of a trained embedding model for multiple tasks. Features are extracted from executable binaries and fed as inputs to a trained embedding model, which generates a low-dimensional embedding representation of each of the input binaries. These embeddings along with auxiliary label information can be used for different downstream malware analysis tasks.

### 3.2. Network architecture

Our embedding neural network architecture is shown in Figure 3. The network takes as input 2381-dimensional static features from [39]. The features are first normalized via standard scaling with respect to mean and variance on the EMBER 2018 training set, then fed to an embedding network, which is comprised of four dense layers of dimension 4,000, 1,024, 512, and 512. Each layer uses a sigmoid activation. Between the layers we include both a BatchNorm and a Dropout layer with a dropout probability of 10%. Following those layers is the final embedding layer using a linear activation with specified output dimension; in our experiments, we utilize an output dimension of 32. During training the network outputs are then optionally normalized and losses, which compare CAPA attribute information are evaluated and minimized via backpropagation.

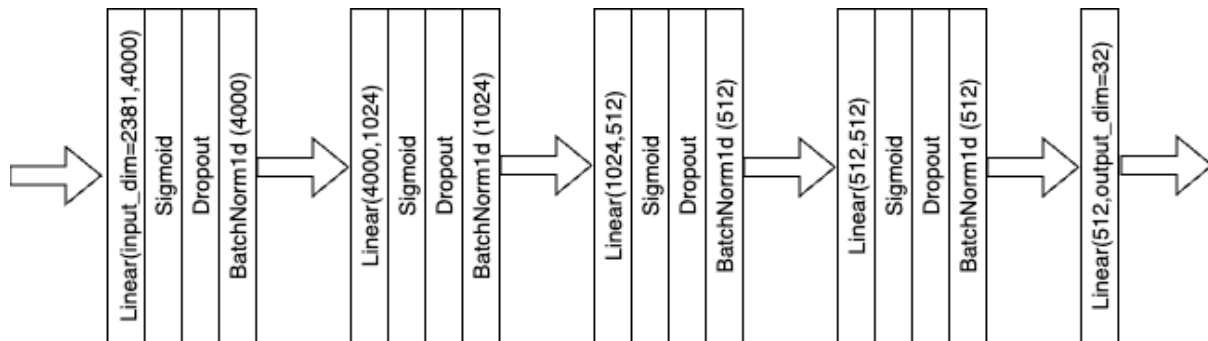


Figure 3: Architectural schematic of our embedding network.

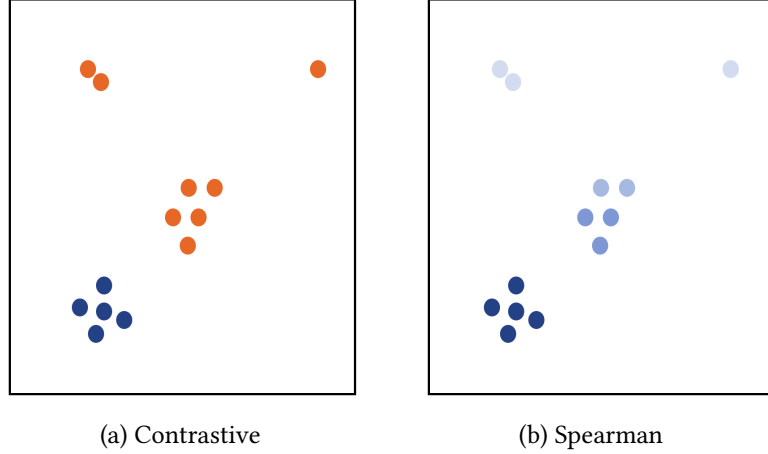
### 3.3. Enriching Metric Embeddings with CAPA Labels

The CAPA system detects various capabilities of a binary file based using both the static analysis and disassembly of the file, and yields a set of capabilities for each file. These capabilities are categorical and non-mutually exclusive and are labeled with short text snippets, e.g., “read file”. We incorporate these generated sets of capabilities to enrich our embeddings via two different loss functions, which we apply both solo and in tandem (via summation) in our experiments.

```
$ capa Lab01-01.dll_
+-----+-----+
| md5          | 290934c61de9176ad682ffdd65f0a669 |
| path         | Lab01-01.dll_                       |
+-----+-----+

+-----+-----+
| CAPABILITY   | NAMESPACE                           |
+-----+-----+
| receive data | communication                         |
| send data    | communication                         |
| initialize Winsock library | communication/socket                |
| receive data on socket | communication/socket/receive        |
| send data on socket | communication/socket/send           |
| connect TCP socket | communication/socket/tcp            |
| create TCP socket | communication/socket/tcp            |
| act as TCP client | communication/tcp/client            |
| check mutex    | host-interaction/mutex               |
| create mutex   | host-interaction/mutex               |
| resolve DNS   | host-interaction/network/dns/resolve |
| create process | host-interaction/process/create      |
+-----+-----+
```

Figure 4: A Sample CAPA report for a PE file Lab01-01.dll\_ from [40].



**Figure 5:** A visual representation of the difference between the Contrastive and Spearman losses. The Contrastive loss considers all entities either in-cluster (blue) or out-of-cluster (orange). The Spearman loss captures the relative similarity of objects, shown in shades of blue, regardless of cluster status.

### 3.3.1. Contrastive Loss

Contrastive loss is defined as:

$$L_{contrastive} = Y_{true}D + (1 - Y_{true}) \max(\text{margin} - D, 0) \quad (1)$$

where  $D$  is the distance between a pair of points,  $Y_{true}$  is 1 if the pair contains similar objects or 0 if the objects are dissimilar, and  $\text{margin}$  is the desired separation between dissimilar objects and is a tunable hyperparameter. Equation 1 requires that samples either belong to the same group or not, meaning that we cannot include more fine-grained similarity (e.g., these two samples are 75% similar), in the loss function. Consequently, in this case we convert the CAPA detection sets into hard clusters with a locality sensitive hash. Employing a MinHash with one band and 64 permutations, we compute a single hash (cluster) for each binary file, where two files are similar if they lie in the same cluster ( $Y_{true} = 1$ ) and different otherwise ( $Y_{true} = 0$ ).

During our experiments we employ a contrastive loss with a margin of 10 based on the Euclidean distance of the embeddings for each pair of samples in the batch. Note that we have tried a few tuning experiments for the margin hyperparameter but did not see significant improvements over this default.

### 3.3.2. Spearman Loss

While our contrastive approach assesses similarity based on CAPA clusters, it coarsely embeds binaries as “similar” or “not similar”, when in reality some sets of CAPA labels are more similar than others. To account for finer-grained similarities, we employ a novel approach based on the Spearman Rank Correlation Coefficient.

Specifically, advances in approximate differentiable sorting and ranking [41] allow us to optimize Spearman’s rank correlation coefficient with stochastic gradient descent. This allows us to compute the loss between a ground truth ranking and a predicted ranking from our model. This is desirable as it allows inserting *more nuanced information into the loss function based on finer grained degree*, rather than a simple binary similar/dissimilar decision. In our experiments the ranking is based on similarity, from most similar to least. Given integer ranks we can define Spearman’s rank correlation coefficient as

$$r = 1 - \frac{6 \sum_i (R(X_i) - R(Y_i))^2}{n(n^2 - 1)} \quad (2)$$

where  $X_i$  and  $Y_i$  are the ground truth similarity and predicted similarity of data point  $i$  and  $R(X_i)$  and  $R(Y_i)$  are the corresponding ranks. We assess ground truth similarity between two CAPA capability sets as their Jaccard similarity, and use the soft rank implementation from [41] to compare predicted and

ground truth ranks. For a given batch, we use these ranks to establish the Spearman rank correlation coefficient – the loss for that batch.

### 3.4. Training Process

All the layers of our networks are initialized by the Xavier algorithm [42] and trained with stochastic gradient descent (SGD) to a maximum of 30 epochs with a learning rate of 0.001. The batching algorithm used for SGD training was modified to better support our metric learning loss functions. Ordinarily, each batch contains  $C$  randomly sampled (with replacement) clusters and  $M$  randomly sampled PE files from each cluster (again, sampled with replacement). For our binary similarity problem, we are confronted with two complications. First, each cluster can potentially contain both goodware and malware unlike typical metric learning problems where clusters are homogeneous. Second, we have an extremely large number of clusters ( $O(10^5)$ ). To address the goodware and malware heterogeneity concern, we split each cluster  $C$  into two clusters,  $C$ -goodware and  $C$ -malware. When we sample  $C$  clusters, we do so from the combination of all  $C$ -goodware and  $C$ -malware clusters. To address the second concern, we sample  $C$  clusters without replacement and define the end of the epoch when the model has processed examples from every cluster. This algorithm ensures we cover the full space of goodware, malware, and clusters in each epoch while maintaining a balance between positive and negative pairs in each batch. For these experiments, we set  $C = 20$  and  $M = 4$ .

### 3.5. Transfer Process

After training the embedding, we measure the embedding’s usability on various malware classification tasks. For our experiments, we train an embedding network using the EMBER 2018 training partition and extracted CAPA labels. Once we have a trained embedding network, we can use this to extract embeddings from any dataset with EMBER features. Using extracted embeddings for a given dataset, we can then fit a lightweight classifier over the embeddings and corresponding labels to make predictions for arbitrary different tasks.

The choice of the best final classifier for each task is not obvious. Typically, generalization-based learning using ensemble methods (e.g., random forests or gradient boosted trees) provide state of the art performance on malware tasks. However, our feature space is unique in that distances between two training points have meaning and decision tree methods that rely on splitting individual features may have difficulty capturing that geometry. Notably, SVMs are a generalization-based method that could take our metric space into account, but we ignore it here due to the computational cost of training an SVM on very large datasets. An alternative would be an instance-based learning algorithm (e.g.,  $k$ -nearest neighbors), which explicitly considers distances between training data points. As we will show in the following evaluation, we consider both instance and generalization-based classifiers, and the best classifier can vary based on the transfer task.

## 4. Experiments

### 4.1. Embedding Networks

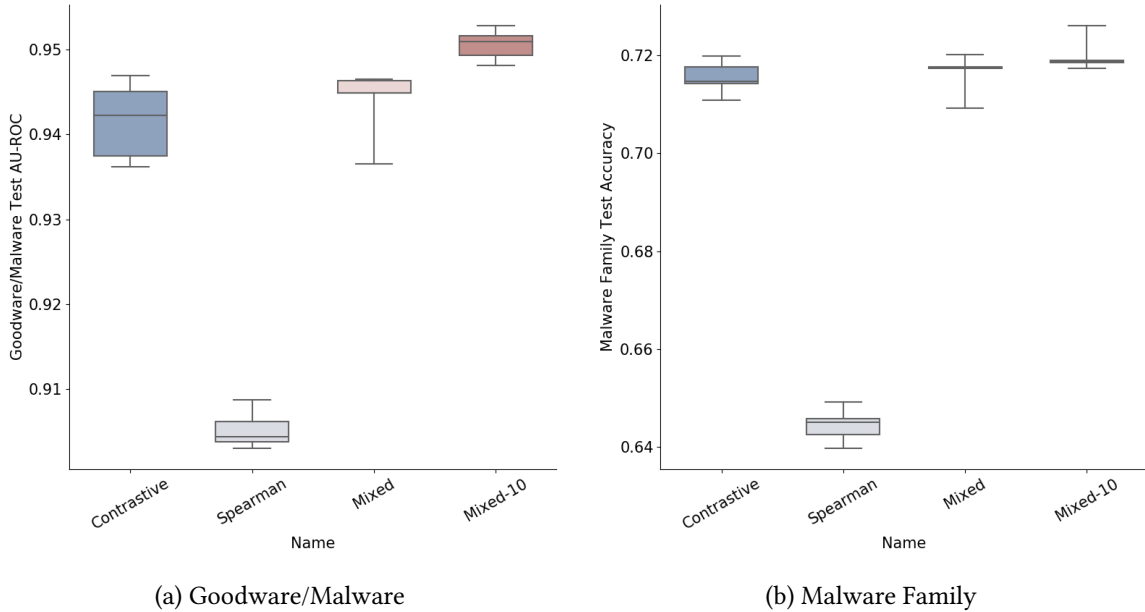
We trained various embedding networks using EMBER feature vectors and CAPA labeling extracted from PEs in the EMBER 2018 train partition. These consist of:

- Contrastive loss on CAPA clusters.
- Spearman loss on Jaccard similarities between CAPA attribute sets.
- Mixed objective Spearman and Contrastive loss, where the net loss term is the sum of the losses. In practice, we employ a weighting of 10X on the Spearman loss term to bring the contributions from each constituent loss term to roughly the same order of magnitude.

We trained each embedding network according to the procedure discussed in Section 3.4. Since deep learning models are not amenable to convex optimization (i.e., no global minimum guarantee), we trained five different instantiations of each model in order to assess variance in performance. When performing transfer task experiments, we then aggregated mean and standard deviation statistics across embeddings from all five networks of a given type.

## 4.2. Transfer Experiments on EMBER

As an initial evaluation of our embeddings, we performed two transfer tasks on the EMBER dataset: malware detection and malware family classification.



**Figure 6:** Transfer Experiments on EMBER. In (a), results of the Goodware/Malware transfer experiment are reported in terms of the area under the ROC curve (AU-ROC). In (b), results of the malware family transfer experiment are reported in terms of accuracy. For both experiments, classifiers trained on Spearman embeddings underperformed those trained on Contrastive embeddings while classifiers trained on embeddings derived from our weighted mixed-objective loss were the top performers.

The malware detection transfer task aims to detect malware by fitting a transfer classifier on embeddings of the EMBER dataset along with malicious/benign labels. For this task, we extracted embeddings across both train and test partitions of EMBER 2018. We then fit a lightGBM ensemble with 1000 trees and otherwise default parameters over the embeddings extracted from the training set, and evaluated using embeddings extracted from the test set. The results of this experiment are shown in Figure 6a in terms of the area under the ROC curve (AU-ROC) on the test set.

In this experimental setting, we tried different weightings of the mixed objective loss, with the Spearman component both unweighted and up-weighted by a factor of 10 to be on the same scale as the contrastive component. We notice that the transfer performance on the contrastive loss embedding significantly outperforms the transfer performance on the Spearman loss embedding. However, both embeddings which use a combination of the two losses offer better classification performance than strictly either of the embeddings trained on a solo loss (Spearman or Contrastive), with the weighted mixed objective loss outperforming the unweighted. Note that none of the transfer malicious/benign classifiers on EMBER 2018 perform comparably to the baseline model from [39].

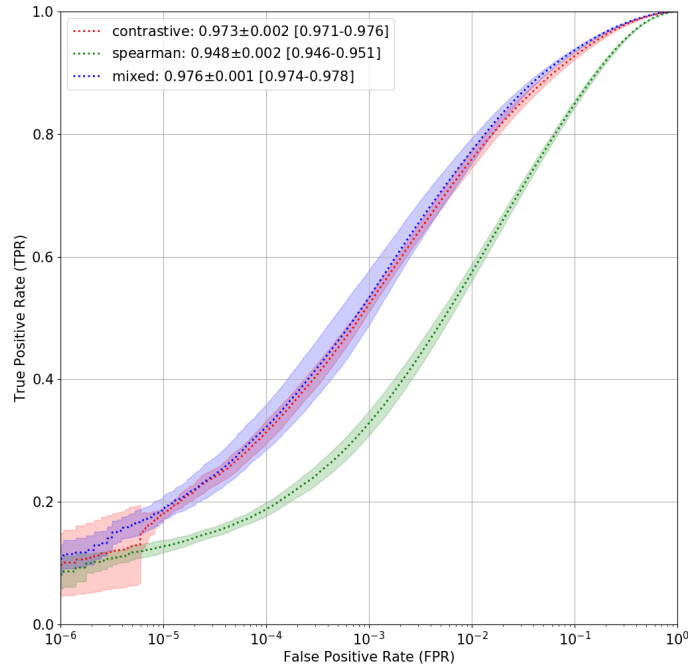
The second transfer task is a malware family recognition task, which utilizes a nearest neighbor classifier in the embedding space in conjunction with the EMBER 2018 malware family labels (derived via AVClass [43]) to attribute malware family, for only the malicious samples. These results are shown



in Figure 6b. While here the performance evaluation is in terms of accuracy not AU-ROC, we notice the same performance trend across embedding types as for the detection transfer task.

### 4.3. Transfer Experiments on SOREL-20M

We additionally evaluated the performance of our embeddings for different tasks on the SOREL-20M dataset. SOREL is a large industry-scale dataset with publicly available labeling telemetry beyond just malicious/benign detection; it also contains public labeling telemetry for 11 distinct malware attributes, namely: Adware, Crypto Miner, Downloader, Dropper, File Infector, Flooder, Installer, Packed, Ransomware, Spyware, and Worm. Note that these attributes are non-mutually exclusive across samples, meaning that a given malware sample can have multiple malware attributes.



**Figure 7:** Results of the Detection Transfer Experiment on SOREL-20M. Classifiers trained on the Contrastive and Mixed embeddings have the highest AU-ROCs of  $0.973 \pm 0.002$  and  $0.976 \pm 0.001$ , respectively. Classifiers trained on the Spearman embedding have an AU-ROC of  $0.948 \pm 0.002$ .

SOREL also contains different data with a different data distribution than EMBER (on which the embeddings were extracted). This suggests any strong performance over the EMBER-to-SOREL transition is an indication of the robustness of our approach to producing general purpose features for downstream tasks.

We extracted 32-dimensional embeddings for all of the SOREL samples apriori, and trained second-stage task-specific lightGBM classifiers on the extracted embeddings. We assessed embedding performance/quality for two distinct tasks: malware detection and malware attribute labeling.

Results from the malware detection task are shown in Figure 7. Embedding extraction and lightGBM fit was performed five different times to obtain error bars. Consistent with our transfer experiments on EMBER, the mixed objective weighted Spearman+Contrastive embedding yields the highest AU-ROC, slightly outperforming the Contrastive embedding and significantly outperforming the Spearman embedding. For reference, the lightGBM baseline trained on full 2381-dimensional features has an AU-ROC of  $0.981 \pm 0.002$  [44], a relative average improvement over the top-performing Spearman+Contrastive model of 0.05%. However, storing the full 2381-dimensional feature vectors requires 74.4 times the

amount of storage of as that of the 32-dimensional embeddings, indicating that in practice, the top-performing embeddings significantly reduce the storage burden at a slight reduction in net performance.

Results from the malware attribute labeling task are shown in Table 1. For this task, we fit lightGBM classifiers across each of the malware tags, using 1-hit for each tag as a criterion for presence of the attribute, consistent with [44]. Notably, we see a similar pattern with the *Mixed-10* loss on average outperforming the Contrastive loss and both losses on average, outperforming the Spearman loss. Note also, that the Mixed-10 loss under-performs the Contrastive loss where Spearman performs poorly. On average, the results for tagging under-perform the baseline provided with SOREL-20M benchmark, though this is a somewhat invalid comparison, as the attribute baselines from [44] utilized a large multi-target network, factoring in number of vendor hits, malicious/benign classification, and simultaneous attribute predictions; thus some of the performance discrepancy is likely due to limitations of single-target classifiers.

	Contrastive	Spearman	Mixed-10
Adware	<b>0.917 ± 0.005</b>	0.883 ± 0.005	<b>0.917 ± 0.002</b>
Crypto Miner	<b>0.976 ± 0.004</b>	0.962 ± 0.001	<b>0.976 ± 0.003</b>
Downloader	0.832 ± 0.007	0.798 ± 0.005	<b>0.835 ± 0.004</b>
Dropper	0.819 ± 0.009	0.773 ± 0.005	<b>0.824 ± 0.011</b>
File Infector	0.878 ± 0.003	0.834 ± 0.005	<b>0.885 ± 0.007</b>
Flooder	<b>0.982 ± 0.006</b>	0.981 ± 0.003	0.979 ± 0.003
Installer	0.957 ± 0.003	0.929 ± 0.002	<b>0.962 ± 0.002</b>
Packed	<b>0.783 ± 0.003</b>	0.742 ± 0.004	0.779 ± 0.013
Ransomware	0.977 ± 0.003	0.959 ± 0.002	<b>0.978 ± 0.003</b>
Spyware	<b>0.848 ± 0.010</b>	0.776 ± 0.003	0.846 ± 0.014
Worm	<b>0.877 ± 0.014</b>	0.804 ± 0.014	<b>0.877 ± 0.014</b>

**Table 1**

Results from the malware attribute transfer experiment on SOREL-20M. Mean AU-ROC and AU-ROC standard deviation are reported, with results aggregated over five runs. Best results are shown in bold.

## 5. Discussion

We have introduced two different approaches to enrich metric embeddings with static disassembly capabilities information and performed evaluations thereof on multiple downstream tasks. These approaches consist of a fine-grained Spearman embedding approach and a coarse-grained Contrastive embedding approach. In the vast majority of our experiments, the coarse-grained Contrastive approach exhibited superior performance to the finer-grained Spearman approach. In some respects, this is not surprising, as Contrastive loss inherently forces separability in a way that the Spearman loss does not. An in-depth examination of similarity distributions and adoption of additional similarity measures other than Jaccard similarity could be helpful in improving the Spearman embedding. Consistent with other literature in the ML security/applied ML space, we found that combining both Spearman and Contrastive embedding losses generally improved performance as did balancing loss contributions to a similar order of magnitude [45, 46, 47].

While our embeddings performed comparably to classifiers trained on raw features for certain tasks, they did not work so well for others. Among a variety of factors, this may be due to including semantic information inherent to the CAPA embeddings, over/under-fitting, and hyperparameter selection, all of which are logical areas of follow-up research. Generally, we surmise that improving performance of metric embeddings for additional tasks is trivially feasible by utilizing additional training objectives. These may include malicious/benign labels, malware attribute tags, MITRE ATT&CK tactics (notably, CAPA outputs these as well as attributes) and additional metadata. Moreover, examining how embedding performance scales when training on larger more heterogeneous groups of samples and evaluating on substantially concept-drifted data could offer further insight into embedding performance and design

(e.g., [48]).

Notably, a low-dimensional embedding which performs well for a variety of classification and/or information retrieval tasks could yield significant computational and storage savings over utilizing raw features or binaries. Such embeddings could be utilized in both academic contexts, where compute resources are often limited or in commercial contexts for rapid prototyping. As a reference, the SOREL-20M dataset is an order of magnitude smaller than industry datasets which are typically used to train commercial PE malware detectors, yet it still comes with a warning about potentially incurring bandwidth fees or exhausting disk space. Even in featurized format as 32-bit floating point, the SOREL-20M dataset requires 172 GB of storage. Using the embeddings introduced in this paper, this can be compressed to roughly 2.3 GB, which is small enough to fit in memory, even for most laptops.

## References

- [1] VirusTotal, Virustotal - stats, 2022. URL: <https://www.virustotal.com/gui/stats>, accessed: 2022-08-04.
- [2] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, Machine learning: The high interest credit card of technical debt, SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop) (2014).
- [3] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, G. Wang, Measuring and modeling the label dynamics of online Anti-Malware engines, in: 29th USENIX Security Symposium (USENIX Security 20), USENIX Association, 2020, pp. 2361–2378. URL: <https://www.usenix.org/conference/usenixsecurity20/presentation/zhu>.
- [4] M. Kaya, H. Ş. Bilge, Deep metric learning: A survey, *Symmetry* 11 (2019) 1066.
- [5] G. Koch, R. Zemel, R. Salakhutdinov, et al., Siamese neural networks for one-shot image recognition, in: *ICML deep learning workshop*, volume 2, Lille, 2015, p. 0.
- [6] E. Hoffer, N. Ailon, Deep metric learning using triplet network, in: *International workshop on similarity-based pattern recognition*, Springer, 2015, pp. 84–92.
- [7] G. Chechik, V. Sharma, U. Shalit, S. Bengio, Large Scale Online Learning of Image Similarity Through Ranking, *J. Mach. Learn. Res.* 11 (2010) 1109–1135. URL: <http://dl.acm.org/citation.cfm?id=1756006.1756042>.
- [8] Y. Zhao, Z. Jin, G.-J. Qi, H. Lu, X.-S. Hua, An Adversarial Approach to Hard Triplet Generation, in: V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss (Eds.), *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part IX*, volume 11213 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 508–524. URL: [https://doi.org/10.1007/978-3-030-01240-3\\_31](https://doi.org/10.1007/978-3-030-01240-3_31). doi:10.1007/978-3-030-01240-3\_31.
- [9] E. Hoffer, N. Ailon, Deep Metric Learning Using Triplet Network, in: A. Feragen, M. Pelillo, M. Loog (Eds.), *SIMBAD 2015: Similarity-Based Pattern Recognition*, Springer International Publishing, Cham, 2015, pp. 84–92. URL: [https://doi.org/10.1007/978-3-319-24261-3\\_7](https://doi.org/10.1007/978-3-319-24261-3_7). doi:10.1007/978-3-319-24261-3\_7.
- [10] F. Schroff, D. Kalenichenko, J. Philbin, FaceNet: A unified embedding for face recognition and clustering, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 815–823. URL: <http://ieeexplore.ieee.org/document/7298682/>. doi:10.1109/CVPR.2015.7298682.
- [11] Y. Zhai, X. Guo, Y. Lu, H. Li, In Defense of the Triplet Loss for Person Re-Identification, *ArXiv e-prints* (2018). URL: <http://arxiv.org/abs/1809.05864>. arXiv:1809.05864.
- [12] S. Kim, M. Seo, I. Laptev, M. Cho, S. Kwak, Deep Metric Learning Beyond Binary Supervision, in: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2288–2297. URL: <http://arxiv.org/abs/1904.09626>. arXiv:1904.09626.
- [13] W. Zheng, Z. Chen, J. Lu, J. Zhou, Hardness-Aware Deep Metric Learning, in: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 72–81.
- [14] F. Xu, W. Zhang, Y. Cheng, W. Chu, Metric Learning with Equidistant and Equidistributed

- Triplet-Based Loss for Product Image Search, in: Proceedings of The Web Conference 2020, WWW '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 57–65. URL: <https://doi.org/10.1145/3366423.3380094>. doi:10.1145/3366423.3380094.
- [15] K. Musgrave, S. Belongie, S.-N. Lim, A Metric Learning Reality Check, in: ECCV, 2020. URL: <http://arxiv.org/abs/2003.08505>. arXiv:2003.08505.
- [16] E. C. R. Shin, D. Song, R. Moazzezi, Recognizing functions in binaries with neural networks, in: 24th USENIX security symposium (USENIX Security 15), 2015, pp. 611–626.
- [17] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, D. Song, Neural network-based graph embedding for cross-platform binary code similarity detection, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 363–376.
- [18] J. Oliver, C. Cheng, Y. Chen, Tlsh—a locality sensitive hash, in: 2013 Fourth Cybercrime and Trustworthy Computing Workshop, IEEE, 2013, pp. 7–13.
- [19] E. Raff, C. Nicholas, An alternative to ncd for large sequences, lempel-ziv jaccard distance, in: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, 2017, pp. 1007–1015.
- [20] J. Jang, D. Brumley, S. Venkataraman, Bitshred: feature hashing malware for scalable triage and semantic analysis, in: Proceedings of the 18th ACM conference on Computer and communications security, 2011, pp. 309–320.
- [21] A. Lee, T. Atkison, A comparison of fuzzy hashes: evaluation, guidelines, and future suggestions, in: Proceedings of the SouthEast Conference, 2017, pp. 18–25.
- [22] R. J. Joyce, D. Amlani, C. Nicholas, E. Raff, MOTIF: A Large Malware Reference Dataset with Ground Truth Family Labels, in: The AAAI-22 Workshop on Artificial Intelligence for Cyber Security (AICS), 2022. URL: <https://github.com/boozallen/MOTIF>. doi:10.48550/arXiv.2111.15031. arXiv:arXiv:2111.15031v1.
- [23] R. J. Joyce, E. Raff, C. Nicholas, A Framework for Cluster and Classifier Evaluation in the Absence of Reference Labels, in: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security (AISec '21), Association for Computing Machinery, 2021. doi:10.1145/3474369.3486867. arXiv:arXiv:2109.11126v1.
- [24] E. Raff, C. Nicholas, M. McLean, A New Burrows Wheeler Transform Markov Distance, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, 2020, pp. 5444–5453. URL: <http://arxiv.org/abs/1912.13046>. doi:10.1609/aaai.v34i04.5994. arXiv:1912.13046.
- [25] E. Raff, C. Nicholas, Malware Classification and Class Imbalance via Stochastic Hashed LZJD, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec '17, ACM, New York, NY, USA, 2017, pp. 111–120. URL: <http://doi.acm.org/10.1145/3128572.3140446>. doi:10.1145/3128572.3140446.
- [26] C. Winter, M. Schneider, Y. Yannikos, F2S2: Fast forensic similarity search through indexing piecewise hash signatures, *Digital Investigation* 10 (2013) 361–371. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1742287613000789>. doi:10.1016/j.diin.2013.08.003.
- [27] F. Breiting, K. P. Astebol, H. Baier, C. Busch, mvHash-B - A New Approach for Similarity Preserving Hashing, in: Proceedings of the 2013 Seventh International Conference on IT Security Incident Management and IT Forensics, IMF '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 33–44. URL: <http://dx.doi.org/10.1109/IMF.2013.18>. doi:10.1109/IMF.2013.18.
- [28] F. Breiting, H. Baier, D. White, On the database lookup problem of approximate matching, *Digital Investigation* 11 (2014) S1–S9. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1742287614000061>. doi:10.1016/j.diin.2014.03.001.
- [29] F. Breiting, C. Rathgeb, H. Baier, An Efficient Similarity Digests Database Lookup - A Logarithmic Divide & Conquer Approach, *The Journal of Digital Forensics, Security and Law (JDFSL)* 9 (2014) 155–166. URL: <http://ojs.jdfsl.org/index.php/jdfsl/article/view/276>.
- [30] F. Breiting, H. Baier, Similarity Preserving Hashing: Eligible Properties and a New Algorithm MRSH-v2, in: *Digital Forensics and Cyber Crime*, Springer, 2013, pp. 167–182. URL: [http://link.springer.com/10.1007/978-3-642-39891-9\\_11](http://link.springer.com/10.1007/978-3-642-39891-9_11). doi:10.1007/978-3-642-39891-9\_11.

- [31] D. Lillis, F. Breitingner, M. Scanlon, Expediting MRSN-v2 Approximate Matching with Hierarchical Bloom Filter Trees, in: 9th EAI International Conference on Digital Forensics and Cyber Crime (ICDF2C 2017), Springer, Prague, Czechia, 2017.
- [32] J. Oliver, C. Cheng, Y. Chen, TLSH – A Locality Sensitive Hash, in: 2013 Fourth Cybercrime and Trustworthy Computing Workshop, IEEE, 2013, pp. 7–13. URL: <http://ieeexplore.ieee.org/document/6754635/>. doi:10.1109/CTC.2013.9.
- [33] S. H. H. Ding, B. C. M. Fung, P. Charland, Kam1N0: MapReduce-based Assembly Clone Search for Reverse Engineering, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, ACM, New York, NY, USA, 2016, pp. 461–470. URL: <http://doi.acm.org/10.1145/2939672.2939719>. doi:10.1145/2939672.2939719.
- [34] S. H. H. Ding, B. C. M. Fung, P. Charland, Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization, in: 2019 IEEE Symposium on Security and Privacy (SP), 2019. doi:10.1109/SP.2019.000003.
- [35] L. Massarelli, G. A. Di Luna, F. Petroni, L. Querzoni, R. Baldoni, SAFE: Self-Attentive Function Embeddings for Binary Similarity, in: Detection of Intrusions and Malware, and Vulnerability Assessment, 2019, pp. 309–329. URL: <http://arxiv.org/abs/1811.05296>. arXiv:1811.05296.
- [36] X. Li, Y. Qu, H. Yin, PalmTree : Learning an Assembly Language Model for Instruction Embedding, in: CCS, 2021. arXiv:arXiv:2103.03809v3.
- [37] S. Yang, L. Cheng, Y. Zeng, Z. Lang, H. Zhu, Z. Shi, Asteria: Deep Learning-based AST-Encoding for Cross-platform Binary Code Similarity Detection, in: 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2021, pp. 224–236. URL: <https://ieeexplore.ieee.org/document/9505086/>. doi:10.1109/DSN48987.2021.00036.
- [38] M. Chandramohan, Y. Xue, Z. Xu, Y. Liu, C. Y. Cho, H. B. K. Tan, BinGo: Cross-architecture cross-OS Binary Search, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, ACM, New York, NY, USA, 2016, pp. 678–689. URL: <http://doi.acm.org/10.1145/2950290.2950350>. doi:10.1145/2950290.2950350.
- [39] H. S. Anderson, P. Roth, Ember: an open dataset for training static pe malware machine learning models, arXiv preprint arXiv:1804.04637 (2018).
- [40] W. Ballenthin, M. Raabe, capa: Automatically identify malware capabilities (2020). URL: <https://www.mandiant.com/resources/capa-automatically-identify-malware-capabilities>, accessed: 2022-08-05.
- [41] M. Blondel, O. Teboul, Q. Berthet, J. Djolonga, Fast differentiable sorting and ranking, in: International Conference on Machine Learning, PMLR, 2020, pp. 950–959.
- [42] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
- [43] M. Sebastián, R. Rivera, P. Kotzias, J. Caballero, Avclass: A tool for massive malware labeling, in: International symposium on research in attacks, intrusions, and defenses, Springer, 2016, pp. 230–253.
- [44] R. Harang, E. M. Rudd, Sorel-20m: A large scale benchmark dataset for malicious pe detection, arXiv preprint arXiv:2012.07634 (2020).
- [45] E. M. Rudd, F. N. Ducau, C. Wild, K. Berlin, R. Harang, {ALOHA}: Auxiliary loss optimization for hypothesis augmentation, in: 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 303–320.
- [46] E. M. Rudd, M. S. Rahman, P. Tully, Transformers for end-to-end infosec tasks: A feasibility study, in: Proceedings of the 1st Workshop on Robust Malware Analysis, 2022, pp. 21–31.
- [47] E. M. Rudd, M. Günther, T. E. Boult, Moon: A mixed objective optimization network for the recognition of facial attributes, in: European Conference on Computer Vision, Springer, 2016, pp. 19–35.
- [48] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, G. Wang, Bodmas: An open dataset for learning based temporal analysis of pe malware, in: 4th Deep Learning and Security Workshop, 2021.