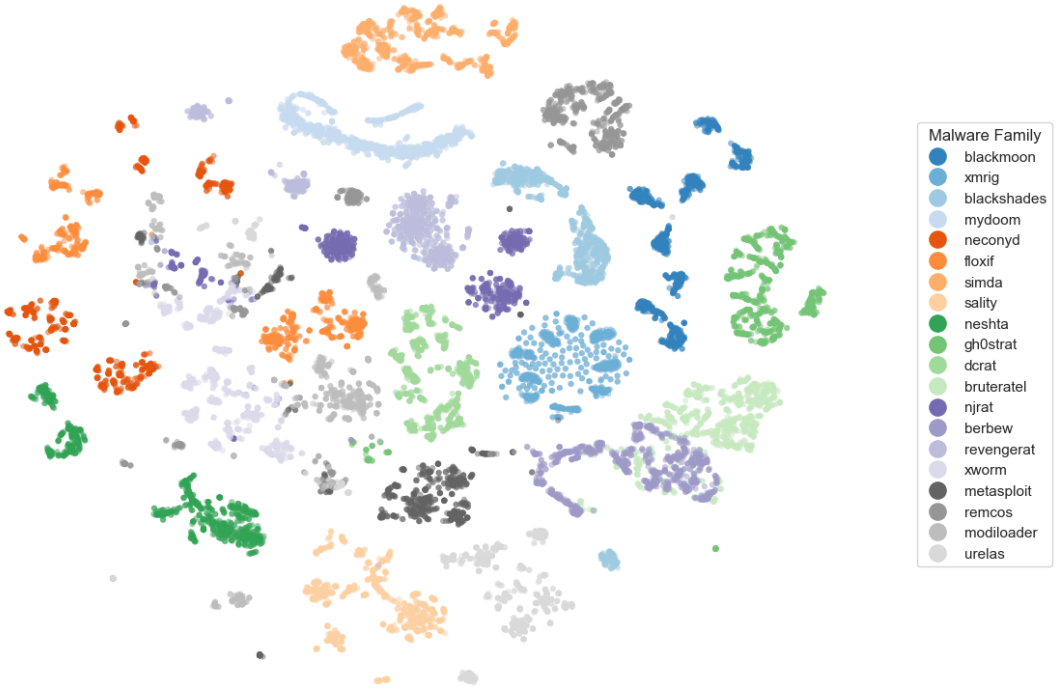


Don’t Judge a Malware by its Binary

Similarity-Based Embeddings for Efficient Malware Analysis

Master’s Thesis in Complex Adaptive Systems

AUGUST KLYNNE & MALTE ÅQVIST

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Don't Judge a Malware by its Binary

Similarity-Based Embeddings for Efficient Malware Analysis

August Klynne
Malte Åqvist

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2025

Don't Judge a Malware by its Binary
Similarity-Based Embeddings for Efficient Malware Analysis
August Klynne
Malte Åqvist

Supervisor: Anders Hansson, Recorded Future
Examiner: Mats Granath, Department of Physics

Master's Thesis 2025
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: t-SNE visualization of a similarity-based embedding of samples from 20 malware families, generated using BM25 analysis of behavioral reports.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Don't Judge a Malware by its Binary
Similarity-Based Embeddings for Efficient Malware Analysis
August Klynne
Malte Åqvist
Department of Physics
Chalmers University of Technology

Abstract

Cyberattacks are projected to cost the global economy more than $10 trillion annually by 2025, driven in large part by malware that remains difficult to detect, classify, and contain. Today, most malware classification still relies on manually engineered binary-level features, an expensive and brittle process. In this work, we ask wether it is possible to predict how a Windows executable will behave without first running it in a sandbox. By placing Windows PE samples into a continuous "behavior space", we aim to enable finer-grained distinctions than existing malware family labels provide. EMBER feature vectors were paired with dynamic behavior reports from the sandbox Recorded Future Triage. A deep metric learning model with triplet loss (FaceNet-style) was trained to project EMBER vectors into clusters defined by behavioral similarity. The model could create valuable embedding spaces for classifying malware by family. Text embeddings for the reports were computed with both BM25 combined with cosine similarity and a transformer encoder. When we projected sandbox reports into text-embedding space, both BM25 combined with cosine similarity and a transformer encoder revealed finer-grained behavioral structure. In contrast, static EMBER feature vectors showed almost no alignment with dynamic behavior, indicating that they carry insufficient behavioral features. Rich behavioral embeddings can be built directly from sandbox reports using transformer encoders, scaling more efficiently with corpus size than BM25 combined with cosine similarity.

Keywords: Malware, Metric Learning, Triplet Loss, EMBER, Text Embedding, Sandbox, BM25, Dynamic Malware Analysis.

Acknowledgements

We would like to express our sincere gratitude to the people at Recorded Future for their involvement and the many insightful discussions that contributed to the development of this thesis. A special thanks goes to the Malware Intelligence team for their guidance and support in a field that was previously unfamiliar to us. We are also grateful to our academic supervisor, Mats Granath, for his valuable feedback and input throughout the project. Finally, we extend our deepest appreciation to our supervisor at Recorded Future, Anders Hansson, for his continuous support, exceptional guidance, and for originally proposing the project idea. His expertise and encouragement have been instrumental to our work.

August Klynne, Gothenburg, June 2025
Malte Åqvist, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ANN	Artificial Neural Networks
API	Application Programming Interface
AUC	Area Under the Curve
AV	Anti Virus
BC	Behavioral Cluster
CNN	Convolutional Neural Networks
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DML	Deep Metric Learning
EMBER	Endgame Malware BEnchmark for Research
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
kNN	k-Nearest Neighbors
Malware	Malicious Software
ML	Metric Learning
MLP	Multi Layer Perceptron
PCA	Principal Component Analysis
PE	Portable Executable
RFT	Recorded Future Triage
RNN	Recurrent Neural Networks
ROC	Receiver-Operating Characteristic Curve
SaaS	Software as a Service
SGD	Stochastic Gradient Descent
TF-IDF	Term Frequency-Inverse Document Frequency
t-SNE	t-Distributed Stochastic Neighbor

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i	Index for malware samples or picture (context dependent)

Parameters

x^a	Anchor sample in Triplet loss function
x^p	Positive sample in Triplet loss function
x^n	Negative sample in Triplet loss function
m	Margin parameter in the triplet loss function
N	Number of samples in the corpus
k_1	Controls term frequency scaling in BM25
b	Controls length normalization in BM25
k	Number of clusters desired in k-means algorithm

Variables

$\mathcal{L}$	Loss value from triplet loss function
t	A single term in the corpus
d	A single document in the corpus
D	The entire corpus of documents
V	Vocabulary size in the corpus
$f_{t,d}$	The number of times t appears in d
$tf_{t_i,d}$	The frequency of t_i in d

$avgdl$	The average length of a document in the corpus
S_c	Cosine similarity value
A, B	Vectors in embedding space of word embedding
θ	Angle between two vectors in embeddings space
e	Embedding space for `text-embedding-3-small`
c_i	Chunk of tokenized text data
$n_{families}$	Number of families in the dataset
y	Vector with labels for each data sample
$\mathcal{E}$	Embedding space created by the neural network
$\mathcal{D}$	Pairwise distance matrix
A	Adjacency matrix
ℓ	Loss in batch iteration

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

1

Introduction

Cyberattacks are projected to cost the global economy over $10 trillion annually by 2025, which would make cybercrime the world's third-largest economy if measured as a country [1]. Malicious software (malware) continues to be a significant component of cyberattacks, with 40% of global cyber incidents in 2023 involving some form of malware [2]. In this context, accurately classifying malware into distinct families is essential for understanding its behavior, developing effective mitigation strategies, and enhancing overall cybersecurity defenses.

Traditional approaches to malware classification rely heavily on handcrafted feature vectors based on their binary files, which can be time-consuming to construct and require deep domain expertise [3]. This is further complicated by the fact that there are thousands of different malware families, distributed with a long tail [4]. Additionally, the definition of malware families is often fluid, as they evolve over time, making the classification of malware families both ambiguous and imprecise [5].

Recent advancements in machine learning and computational power show promise in automating malware detection. In particular, convolutional neural networks (CNNs) demonstrate success in classifying raw binaries directly, without the need for manual feature extraction. While CNNs are promising for classifying software into benign and various malicious categories [6, 7, 8], challenges remain in further differentiating between distinct types of malware.

In addition to static analysis, there is a growing interest in examining malware based on its behavior. One advantage of behavioral analysis is that malware behavior tends to change more slowly than its code, as the fraudulent activities performed by the malware are often harder to conceal than the executable binary itself [9]. Analyses of malware behavior are performed in so called sandbox models, which are virtual machines where the malware is let loose. However, this approach presents its own challenges, particularly in terms of safety and efficiently disclosing malware behavior [10].

In addition to analyzing malware behavior, identifying similarities between different malware samples can provide valuable insights. After detecting a potential malware sample during an ongoing cyberattack, quickly gathering similar samples used in previous attacks can yield useful information for mitigation strategies. If the attacking malware is a novel strain, this information is highly valuable for under-

standing the nature of the threat [11]. Furthermore, since cyberattacks are often linked to specific ransomware groups, malware similarity could serve as a tool for tracking and identifying these groups [12].

This project aims to develop a tool for assessing malware similarity in terms of their behavior. As part of the work, an important question is whether information from the static binary malware files, leveraged through labels derived from dynamic behavior, could be sufficient to produce an embedding space that captures malware behavior and similarity. Specifically, the approach involves training a neural network model to take only the binary file data as input and use labels derived from dynamic behavior to generate a high dimensional embedding space. In this latent space, malware samples with similar behaviors are positioned closer together, while those with different behaviors are farther apart. This is a new application of the general approach used in FaceNet, a facial-recognition network [13].

1.1 Purpose and Aim

A tool for assessing malware similarity is valuable for tasks such as the early detection of specific malware types and the identification of novel and previously encountered samples during cyberattacks. By gathering several similar malware samples based on their behavior, it is possible to generate rules for fast detection of other similar threats, thereby enhancing protection.

While dynamic analysis of malware in sandbox environments is effective for understanding malware behavior, it is also time consuming and computationally expensive. For faster behavior-based similarity assessments, it would be advantageous if the binary files alone were sufficient to generate a similarity-based representation.

The aim of this project can be divided into two objectives:

- Create a nuanced embedding of malware similarity in terms of their behavior by using dynamic reports produced by a sandbox model.
- Investigate the potential of creating such an embedding using only binary files as input.

The first embedding is hereafter referred to as the Report embedding, as it is produced through analysis of dynamic behavior reports from the sandbox model. The second embedding is hereafter referred to as the Metric learning (ML) embedding, which relies solely on the binary file of the malware.

1.2 Limitations

In the process of fulfilling the aim, we need to define some limitations for the project. One such limitation is that we limit our investigation to Windows Portable Executable (PE) files, since this is the bulk of all uploads to the sandbox we have access to. Windows, with its dominant market share, is the operating system most commonly targeted by malware, which explains why the majority of uploaded files are

in PE format [14].

Since we had no prior experience in malware analysis, we relied on EMBER features extracted from PE files as input to our Metric learning model. These features are well-established in the field and capture relevant aspects of malware that would have been difficult for us to extract manually. However, we acknowledge that we do not fully understand the intricacies of all these features, which may affect our interpretation of the model's behavior and results.

1.3 Disposition of Thesis

This section provides an overview of the structure of the report and the content of each chapter.

Introduction

Puts our research question into context and presents the aims of the project.

Background

Gives a general introduction into the field of malware analysis, with further explanation of static and dynamic analysis. In this section Recorded Future is put into the context of the cybersecurity landscape, along with the sandbox model Recorded Future Triage.

Related Work

A selection of relevant previous studies and work are presented that inspired this thesis. This related work builds on the concepts introduced in the background section, now explored through more concrete examples. In particular, the use of EMBER features is highlighted.

Theory

Offers a theoretical background to the concepts used in our work. Starting with an introduction to artificial neural networks, a deeper explanation of metric learning and the triplet loss function is followed. Information retrieval techniques which are used for text document comparisons are also explained.

Method

The methodology used to produce our results is explained in this section. It is divided into two main parts. First, we describe how the embedding that captures malware behavior is created using the dynamic reports generated by the sandbox model. Second, we describe how a ML model is trained to recreate this embedding using only EMBER-features as input data.

Results

The result chapter is also divided into two parts. First, we present the results centered on the behavioral embedding space. Second, we present the results produced by the ML model.

Discussion

The results are discussed in relation to the aims of the project. Problems encountered during the work are also analyzed and explained. Suggestions for future work are provided, along with ethical considerations related to the project and its findings.

Conclusion

The conclusion of the project is summarized along with the main insights.

2
Background

The field of malware analysis is commonly divided into two subfields: (i) detecting whether software is benign or malicious, and (ii) assigning identified malware to families based on activity. While the former is often the more critical task in ensuring cybersecurity, the latter is important for understanding intentions of the malware. However, it has consistently proven more challenging [15, 16].

2.1 Static Malware Analysis

A common technique is rule-based search for potentially harmful patterns in executable code for malware classification. However, to identify patterns and develop rules for this sort of classification demands profound expertise in malware and is very time consuming for the scarce number of individuals who can perform it [17] [18]. As malware development has become easier and more straightforward, manual identification alone cannot meet demand for malware analysis [19].

Efforts continue to automate the process of determining which kind of malware that is posing a threat to the computer systems, network or data. This is often performed using static reports, which focus on examining the binary file of a Windows Portable Executable (PE) [20]. From the static report, information such as ASCII-range strings can be interpreted. Antivirus programs can use both the binary file and static reports to classify the program into if, and what kind of malware it is. The label given by such antivirus programs for the malware is commonly known as the AV label [21].

Antivirus companies have made use of recent rapid advancements in machine learning. For example, machine learning has shown great results in phishing detection, network intrusion detection, cryptography and more [22]. One thing that limits machine learning models is the vast amount of well-organized data required for the training process. Although the availability of such data is greater than ever for cybersecurity companies, academia often struggles with a lack of data provided from industry which hinders the research community [7].

2.2 Dynamic Malware Analysis

A common challenge with machine learning models in malware detection and classification is their struggle to generalize to unseen data. Malicious actors are constantly

improving and evolving their malware to avoid detection, commonly by altering the PE files to avoid patterns recognized by anti-malware software and rule-based searches. While the core behavior of malware often remains consistent, changes in PE files to make identification of malware more difficult cause data sets used to train machine learning models to become outdated quickly [23].

Since the behavior of the malware does not change as quickly as the binary file, a lot can be learned from studying how the malware acts in a Windows environment [24]. One can draw a parallel between an individual's genotype, which encodes all the information necessary to produce a person (the binary file), and the phenotype, which represents the actual outcome in reality (the malware's behavior). To study the malware's behavior, one uses sandbox models. These are isolated virtual environments where the malware will exhibit its true behavior [25]. However, it can sometimes be fairly complicated to disclose the malware's intention. They can detect that they are inside a sandbox environment, and it can take weeks before malware begins to do any harm [10].

2.3 Recorded Future

Recorded Future is the world's largest intelligence company. Recorded Future uses machine learning and natural language processing methods to continuously collect and organize data from the open web, dark web, and technical sources [26]. With customer organizations ranging from the private sector to governments, Recorded Future analyzes the data and provides actionable insights for threat management. The resulting information is displayed within a software-as-a-service (SaaS) portal.

As part of its offerings, Recorded Future is providing a state-of-the-art malware analysis sandbox, called Recorded Future Triage (RFT). Every day, about 100,000 suspicious files are uploaded to the sandbox for analysis. Files that exhibit malicious behavior get a high risk score, but more than half of these malware binaries are not attributed to a specific malware strain. The ability to quickly pin down the correct family of a malware sample can be critically important during a cyber-attack. Not only does family classification provide immediate insights about the characteristics and behaviors of a malware sample, but it is a core part of the triage, remediation, and attribution efforts. In addition, RFT automatically produces a comprehensive, JSON-formatted behavioral report for every uploaded sample; Listing 2.1 shows a condensed snippet of that output.

Listing 2.1 Condensed snippet of a RFT behavioral report.

```
{
  "dynamic": {
    "dumped": [
      { "name": "0x0008...12119-2.dat", "path":
      ↪  "...\\icedid_ramnitmgr.exe", "size": 90112.0 },
      { "name": "0x0001...69e2-27.dat", "path":
      ↪  "C:\\Windows\\System32\\catroot2\\dberr.txt", "size": 196192.0 }
    ],
    "extracted": [
      {
        "config": { "family": "ramnit", "rule": "Ramnit" },
        "dumped_file": "memory/...-memory.dmp"
      }
    ],
    "network": {
      "dns": [
        {
          "request_domain": ["api.bing.com"],
          "response_value": ["e-0001.e-msedge.net", "13.107.5.80"]
        }
      ],
      "flows": [
        { "dst_ip": "8.8.8.8", "dst_port": 53, "proto": "udp", "layer_7":
        ↪  ["dns"] },
        { "dst_ip": "204.79.197.200", "dst_port": 443, "proto": "tcp",
        ↪  "layer_7": ["tls"] }
      ]
    },
    "registry": {
      "create": ["...\\Recovery\\PendingRecovery", "...\\SearchScopes",
      ↪  "...\\Main"],
      "read": ["...\\NLS\\Language"],
      "write": ["...\\Main\\CompatibilityFlags",
      ↪  "...\\PendingRecovery\\AdminActive"]
    },
    "signatures": [
      {
        "label": "ramnit",
        "tags": ["trojan", "spyware", "worm"]
      }
    ]
  }
}

```

3
Related Work

Recent studies in malware classification using machine learning models have made use of CNNs, given their successful use in image classification. A CNN model processing only the binary PE files as input was created, achieving malware detection on par with hand-crafted rule based search [7]. Using EMBER features as input, ML models have also been trained to create embeddings of malware, capturing intra-family similarity and inter-family dissimilarity in the embedding space, which is useful in classification tasks [27].

3.1 Static Malware Analysis and the EMBER Dataset

Static malware analysis examines a program without execution, typically analyzing file structure, headers, imports, and byte patterns. It is fast, scalable, and less resource-intensive than dynamic methods. To facilitate research in this area, the EMBER dataset [28] was introduced, providing structured feature vectors extracted from PE files through static analysis. These features include metadata, entropy, imported APIs, and byte histograms, grouped by semantic categories to support robust machine learning models [29].

MalConv [30] showed that raw-byte CNNs could detect malware without any hand-crafted features, offering robustness to many packing and obfuscation tricks. EMBER soon followed, releasing a large, open dataset with carefully engineered PE-file features and labels; classification models trained on these EMBER features matched or surpassed MalConv's accuracy on clean binaries while remaining far faster to train and deploy. These works sparked a new line of work such as the contrastive-embedding approach that seeks the best of both worlds: deep-learned representations with EMBER-level performance and greater resilience to evasion[27].

3.2 Dynamic Malware Analysis and Behavior Clustering

Dynamic analysis complements static methods by executing malware in sandbox environments and recording its runtime behavior such as system calls, file operations or network communication. This approach is more resilient to obfuscation because

it focuses on what the malware does rather than how it is written [31].

Bayer *et al.* pioneered scalable clustering of behavior profiles from sandbox logs [32], and Rieck *et al.* developed Malheur, a framework for clustering malware based on system call traces [33]. Such behavior-based grouping allows analysts to discover new families or variants based on shared operational traits. However, dynamic analysis is resource-intensive and may fail to trigger certain behaviors during a single execution trace.

3.3 Bridging Static and Dynamic Analysis

Hybrid analysis methods aim to combine the strengths of static and dynamic techniques. Anderson *et al.* used multikernel learning to fuse binary and behavioral features for malware classification [34], and DUET [35] combined both views for clustering. More recently, deep learning models have incorporated both static and dynamic features. For instance, Xia *et al.* used RNNs to encode sandbox API traces and CNNs for static features in a joint classifier [36]. These approaches show that combining both modalities yields stronger models but often require runtime analysis during deployment, limiting scalability.

3.4 Metric Learning and Malware Embeddings

Metric learning has emerged as a useful approach for malware representation. Instead of predicting labels directly, models are trained to embed samples such that similar malware is close and dissimilar malware is distant in the vector space. Rudd *et al.* trained models with contrastive loss and Spearman correlation to create static embeddings that captured shared capabilities and family structure [27].

4

Theory

This section provides the theoretical background for our work, beginning with an introduction to artificial neural networks, followed by a detailed explanation of metric learning and the triplet loss function. It also covers information retrieval techniques used for comparing text documents.

4.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are models inspired by the structure and function of biological neural networks. Since the introduction of ANNs, they have evolved significantly, largely thanks to recent hardware development, which have provided greater computation power, thus enabling deeper and more complex networks. Today, ANNs are foundational in modern machine learning tasks, including image recognition, natural language processing, and similarity learning [37].

4.1.1 Multilayer Perceptrons and Supervised Learning

A common ANN architecture is a Multilayer Perceptron (MLP), a type of feedforward neural network composed of an input layer, one or more hidden layers, and an output layer. Each neuron in the network receives input, applies a weighted sum followed by a non-linear activation function (such as ReLU or Sigmoid), and passes the result forward. These nonlinearities enable the network to capture complex patterns in high-dimensional data [38].

During supervised training, the network learns to minimize the difference between its predictions and the true labels using a loss function, commonly cross-entropy for classification tasks. The weights are updated using gradient descent, where the error is propagated backward through the network via backpropagation, applying the chain rule to compute gradients for each layer. Repeated over many training epochs, this process gradually improves the network's performance on the task [39].

Although MLPs are often used for classification, they are also well suited for learning embeddings in tasks such as metric learning, where the goal is not to classify directly but to learn a space where similar inputs are closer together [40]. This is the role MLPs serve in the model described in this thesis.

4.2 Metric learning

Metric learning (ML) is a field in machine learning focused on capturing similarity between data points. ML models learn a transformation of input data into a latent space where similar samples are closer and dissimilar ones are further apart. In cases when data features are not easily separated, ML offers a powerful tool for data transformation, thus enhancing and preparing it for classification and clustering tasks [41].

Deep metric learning (DML) is a subfield of ML where neural networks are employed in the mapping function. With the inherent ability of neural networks to capture complex non-linear relations in data, the main objective in this field of research is to develop and optimize loss functions enabling efficient training and learning [42]. DML has been applied successfully in many domains, including computer vision, video, audio, and text recognition tasks. [43].

One common loss function in DML is the contrastive loss which works by processing paired input data through two identical neural networks (called Siamese networks). The high dimensional input data is projected down to a lower dimensional latent space. A distance metric, often Euclidean distance, is applied for measuring the similarity between the two embeddings. The loss used in training of the network is calculated by using this distance with respect to whether the samples belong to the same class or not [44]. This is the metric-learning approach used by Rudd et al. for creating a latent space for malware binaries.

4.2.1 Triplet loss

The Triplet loss was first introduced in the FaceNet paper [13], where the algorithm achieved state-of-the-art performance in face recognition, verification and clustering tasks. By applying a triplet loss in a MLP in a supervised learning setting, each malware sample x_i is mapped by the embedding function f to a vector $f(x_i) \in \mathcal{R}^d$, where $\mathcal{R}^d$ denotes the embedding space. Here, the samples (e.g., pictures of faces) x_i are projected down to a d-dimensional Euclidean space, creating a representation of faces in this space. Thus, Triplet loss falls under metric learning, where face images are ordered according to a distance-based metric that quantifies similarity. As a result, images of the same person will be closer together in the embedding space, while images of different people will be farther apart.

FaceNet's innovation lies in how the loss function is calculated, with training samples selected in groups of three: an anchor, a positive, and a negative. During training a sample x_i^a (anchor) is selected belonging to a certain person. After training, any sample x_i^p (positive) belonging to the same person should be closer in the Euclidean space than any sample x_i^n (negative) not belonging to that person (Figure 4.1).

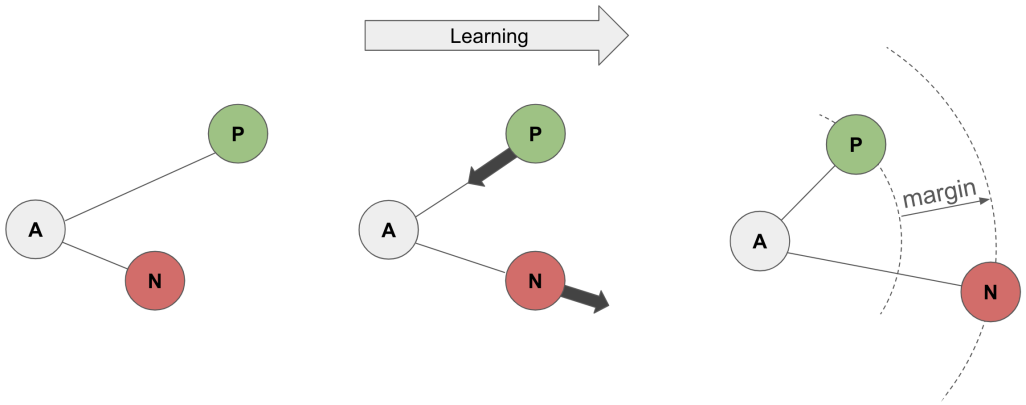


Figure 4.1: Illustration of the learning process in a triplet loss setup. A loss is incurred when a positive sample is embedded farther from the anchor than a negative sample. During training, the network adjusts its weights so that the positive sample is positioned closer to the anchor than any negative sample, by at least a specified margin.

One key difference from contrastive loss is that this should hold even with a certain margin m, also illustrated in Figure 4.1. The inclusion of a margin is beneficial because it prevents positive samples from collapsing to a single point in the embedding space and avoids negative samples being pushed infinitely apart; when the distance to the margin is satisfied the loss turns to zero and training stops. This ensures that some degree of variation among samples of the same person is preserved, preventing overfitting and improving generalization.

For efficient learning and faster convergence, the triplet loss algorithm relies on an adequate sampling technique of triplets. Instead of sampling at random, an alternative is to only select triplets such that the condition in Equation 4.1 is violated, that is only sampling so called hard triplets.

$$||x_i^a - x_i^p||_2^2 \leq ||x_i^a - x_i^n||_2^2 \tag{4.1}$$

However, this can lead the algorithm into local minima, where mislabeled or poor quality samples dominate, negatively impacting the training process. The solution developed by FaceNet is to sample so called "semi-hard negatives", sampling triplets by violating Equation 4.2. This entails selecting negative samples closer to the anchor compared to positive samples, but still not closer than the margin m. This selection of negatives includes enough information to continue learning, but avoids early convergence to local minima.

$$||x_i^a - x_i^p||_2^2 + m \leq ||x_i^a - x_i^n||_2^2 \tag{4.2}$$

Regarding the selection of positives, all are considered as the anchor one at a time and the loss is averaged over all combinations of triplets created (Equation 4.3). It is

also important to strike a balance between smaller batch sizes (better for stochastic gradient descent) but batches large enough to form meaningful triplets.

$$\mathcal{L} = \frac{1}{N}\sum_{i}^{N} max(0, ||x_i^a - x_i^p||_2^2 + m - ||x_i^a - x_i^n||_2^2) \tag{4.3}$$

4.3 Information Retrieval Metrics

This section introduces key metrics TF-IDF, BM25, and cosine similarity for assessing document relevance, alongside clustering methods for grouping similar documents.

4.3.1 The Term Frequency-Inverse Document Frequency (TF-IDF)

The TF-IDF concept, first introduced by [45], is used as a weighting factor for feature extraction. It quantifies the significance of a word within a single document in relation to an entire collection of documents. In this measure, TF represents the number of times a word occurs in a document, while IDF represents how common a word is in the entire collection of documents.
Mathematically, TF-IDF is defined as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \times \text{IDF}(t, D) \tag{4.4}$$

where:

- t denotes a term,
- d denotes a single document,
- D denotes the entire collection of documents.

The term frequency is typically calculated as:

$$\text{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}, \tag{4.5}$$

where $f_{t,d}$ is the number of times term t appears in document d.
The inverse document frequency is given by:

$$\text{IDF}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}, \tag{4.6}$$

where N is the total number of documents in the corpus and $|\{d \in D : t \in d\}|$ is the number of documents containing the term t.
This weighting approach is valuable because it identifies words that are significant in a single document yet rare in the overall dataset. By diminishing the influence of common terms and emphasizing those that are unique, TF-IDF can improve the accuracy of tasks like document classification, clustering, and search relevance.

4.3.2 BM25

BM25 is a ranking function derived from the probabilistic relevance framework that measures a document's relevance to a query by combining several nuanced factors. Instead of a linear relationship, the term frequency component is modeled with a saturation function, ensuring that after a certain threshold, additional occurrences of a term yield diminishing returns in relevance. At the same time, an inverse document frequency factor is incorporated, assigning greater importance to rarer terms that are more discriminative within the corpus. Moreover, BM25 normalized for document length by comparing each document's length to the average document length in the corpus, thereby mitigating the bias towards longer texts. The complete formulation is given by

$$BM25(t,d) = \sum_{t_i \in t} IDF(t_i) \cdot \frac{tf_{t_i,d}(k_1+1)}{tf_{t_i,d}(1-b+b\frac{|d|}{avgdl})} \tag{4.7}$$

where $tf_{t_i,d}$ represents the frequency of a term t_i in document d, $|d|$ is the length of the document, $avgdl$ denotes the average document length in the corpus, k_1 along with b are parameters that control the influence of term frequency scaling and length normalization [46]. This formulation refines earlier models like TF-IDF by more effectively capturing the complexities of term distribution and document structure.

4.3.3 Cosine similarity

Cosine similarity is a metric used to describe how similar two vectors are by measuring the cosine of the angle between them. This approach normalizes the vectors, focusing only on the orientation of the vectors and not on the magnitude [47].
For example, consider comparing two vectors that encode the phrases "Hello, world!" and "Hello!" in a two-dimensional space. In this representation, each axis corresponds to a word, and the more frequently a word appears, the further along its axis the vector extends (Figure 4.2). The cosine similarity is then calculated as the cosine of the angle between the two vectors, which can be efficiently computed using the formula:

$$S_c(\mathbf{A},\mathbf{B}) = cos(\theta) = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||\quad||\mathbf{B}||}$$

4.3.4 Transformers

Transformer models, first introduced by [48], rely on a self-attention mechanism that lets every token in a sequence simultaneously weigh the importance of every token, capturing long-range relationships in a single, efficient pass.

A token is raw text split into sub-word with a reversible byte-pair-encoding tokenizer such as OpenAI's fast `tiktoken` [49].

During pretraining, the model learns context dependent token representations. After training these internal activations can be projected to a fixed length embedding

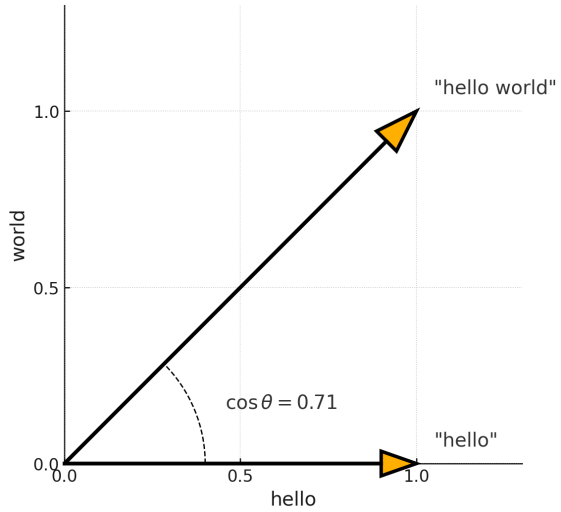


Figure 4.2: Illustration of cosine similarity between two text inputs: "Hello, world!" and "Hello!", showing how semantic similarity is measured in embedding space.

vector that captures meaning. Modern APIs (e.g. OpenAI's text-embedding-3-small [50]) expose these vectors directly. Empirically, transformer-based embeddings outperform classic bag-of-words or static word2vec vectors on clustering and nearest neighbor tasks [51].

4.3.5 Clustering

Clustering is the task of partitioning a collection of objects into subsets called clusters so that objects in the same subset are more similar to each other than to those in different subsets. In metric learning, the triplet-loss objective relies on a label for each sample [13]. When no other ground truth labels are available, cluster assigned labels can serve as pseudo-labels, providing practical but imperfect labels for metric-learning.

K-means partitions the data into a user-specified number of clusters k by minimizing the sum of squared Euclidean distances between points and their assigned centroids [52]. Given n data points and a desired number of groups k, the algorithm proceeds as follows:

1. Seed k centers
2. Assign every data point to the nearest center using Euclidean distance
3. Update each center to the average of the points just assigned to it
4. Repeat steps 2 and 3 until no assignments change

The loop steadily lowers the total square distance between points and their centers.

Because k-means assumes spherical, equally sized clusters and optimizes a single global objective, it is fast and easy to implement but less flexible when the underlying clusters differ in density, size, or shape.

4.3.6 t-SNE (t-Distributed Stochastic Neighbor Embedding)

t-SNE is a non-linear dimensionality reduction technique commonly used for visualizing high-dimensional data in two or three dimensions. It works by converting similarities between data points into joint probabilities and aims to preserve the local structure of the data. In high-dimensional space, it calculates the probability that a point would choose another point as its neighbor. It then seeks a low-dimensional embedding that reflects these probabilities as closely as possible. Unlike linear methods like PCA, t-SNE is particularly effective at revealing clusters or patterns in complex datasets [53].

5
Method

Our methodology centered around three distinct areas of work (Figure 5.1). The first cornerstone was the analysis of dynamic reports provided by the sandbox model RFT for different types of malware, which we used to create a Report embedding reflecting malware behavior. Clustering was also performed on this embedding to define new malware groups as Behavioral Clusters (BCs). The second step involved developing a Metric learning (ML) model that aimed to replicate this embedding using only input features from the malware's PE files. The clusters created in the previous step were used as the malware's labels. The final step was the evaluation of both the Report embedding and the embedding produced by the ML model.

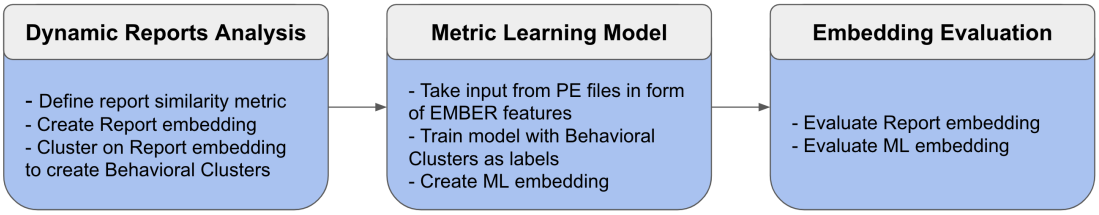


Figure 5.1: The methodology consists of three main steps: (1) generating the Report embedding, (2) developing the ML model and producing its corresponding embedding, and (3) evaluating the resulting embeddings.

5.1 Dynamic Reports Analysis

The goal of this part of the methodology was to create a Report embedding containing information of malware behavior, following the workflow illustrated in Figure 5.2. In order to investigate a broad variety of malware, 2000 samples each of the 20 prevalent malware families were collected. Their dynamic reports were manipulated with regards to structure and information to set up for further analysis on them. Different tools, such as BM25 and Transformers, were then used to investigate relations between all reports and create a metric for report similarity. This similarity metric between all reports was used to create an embedding based on malware behavior. This embedding provided a way to cluster which malware samples that belong together based on similar behavior, which is not entirely coupled to AV labels.

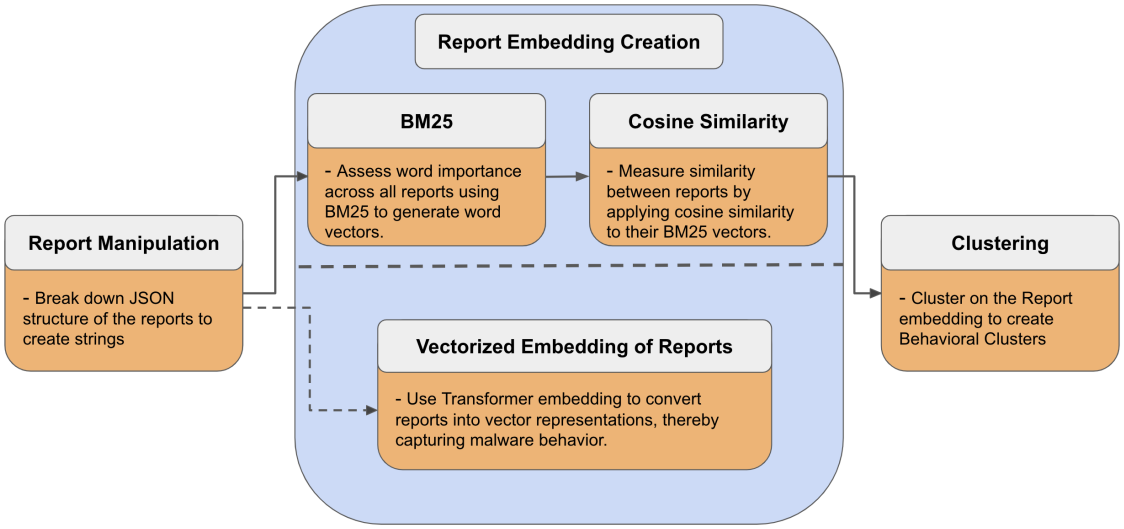


Figure 5.2: The structure of the workflow for the Report similarity investigation centered around three main components: (1) report manipulation, (2) creation of the Report embedding, and (3) clustering, specifically for the embedding generated using BM25.

5.1.1 Report Manipulation

The dynamic analysis reports from RFT were provided in JSON format, with each element representing an aspect of the observed behavior (Listing 2.1). To enable comparisons between different reports, the data had to be restructured. For each top-level category in the JSON files, all combinations of their sub-elements were concatenated into a series of strings (Listing 5.1). As we had limited prior knowledge in the field of malware intelligence beforehand, we opted for a simple and straightforward approach to restructure the reports.

Listing 5.1: Example of strings generated from a dynamic report shown as a JSON structure in Listing 2.1.

```
dynamic.dumped_count:15
dynamic.dumped[0].name:0x0008...12119-2.dat
dynamic.dumped[0].path:...\icedid_ramnitmgr.exe
dynamic.dumped[0].size:90112.0
dynamic.dumped[0].name:0x0001...69e2-27.dat
dynamic.dumped[0].path:C:\Windows\System32\catroot2\dberr.txt
dynamic.dumped[0].size:196192.0
dynamic.extracted[0].config.family:ramnit
dynamic.extracted[0].config.rule:Ramnit
dynamic.extracted[0].dumped_file:memory/...-memory.dmp
dynamic.network.dns_count:1
dynamic.network.dns[0].request_domain[0]:api.bing.com
dynamic.network.dns[0].response_value[0]:e-0001.e-msedge.net
dynamic.network.dns[0].response_value[0]:13.107.5.80
dynamic.network.flows_count:2
```

```
dynamic.network.flows[0].dst_ip:8.8.8.8
dynamic.network.flows[0].dst_port:53
dynamic.network.flows[0].proto:udp
dynamic.network.flows[0].layer_7[0]:dns
dynamic.network.flows[0].dst_ip:204.79.197.200
dynamic.network.flows[0].dst_port:443
dynamic.network.flows[0].proto:tcp
dynamic.network.flows[0].layer_7[0]:tls
dynamic.processes[0].cmd:...\icedid_ramnit.exe
dynamic.processes[0].procid:27
dynamic.processes[0].cmd:...\icedid_ramnitmgr.exe
dynamic.processes[0].procid:28
dynamic.registry_count:31
dynamic.registry.create[0]:...\Recovery\PendingRecovery
dynamic.registry.read[0]:...\NLS\Language
dynamic.registry.write[0]:...\Main\CompatibilityFlags
dynamic.signatures_count:13
dynamic.signatures[0].label:ramnit
dynamic.signatures[0].tags[0]:trojan
dynamic.signatures[0].tags[0]:spyware
dynamic.signatures[0].tags[0]:worm
dynamic.signatures[0].label:system_language_discovery
dynamic.signatures[0].tags[0]:discovery
dynamic.signatures[0].ttp[0]:T1614.001
dynamic.signatures[0].label:modifies_iexplore_settings
dynamic.signatures[0].tags[0]:adware
dynamic.signatures[0].tags[0]:spyware
dynamic.signatures[0].ttp[0]:T1112
```

5.1.2 BM25-based Report Embedding

Initially, more basic investigation methods were used to measure the degree of similarity between two and more reports. For example, TF-IDF confirmed a relation between malware families and dynamic reports. However, due to superior results, BM25 became the primary metric for report similarity. In accordance with theory, we chose to use $b = 0.75$ and $k = 2.0$ for our parameters in BM25 algorithm as the document length vary and unique words are important.

Using BM25, feature vectors were created for each report in the corpus, where the length of the vector is equal to the number of unique words in the corpus. Each element in the vector highlights the relevance of that particular word in the corpus. To determine how similar one report is to the next, we used the cosine similarity between two feature vectors to create a score between 0.0 and 1.0, where 1.0 would be the same report and 0.0 would share no similarities. When cosine similarity is computed between each possible pair, one has a matrix (Figure 5.3) showing the similarities from each report to every other report.

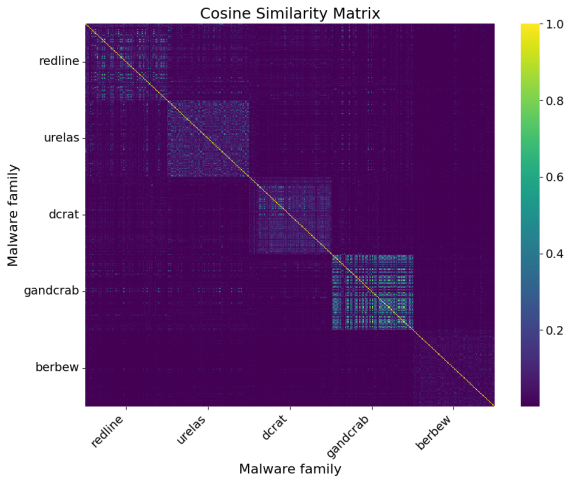


Figure 5.3: Heatmap showing the cosine similarity values between samples. This visual shows five malware families with 100 samples per family. High cosine similarity value for samples within the same AV label and less regular high values for samples from different AV label.

The hypothesis was that AV labels alone do not capture all aspects of malware behavior. However, we still expected a high degree of correlation between the dynamic reports from RFT and AV labels. Therefore, we expected to see cosine similarities higher within families than between families and with varying degree within the family, which was confirmed in Figure 5.3. AV labels were used as the starting point for behavior investigations using the reports, and some tendencies to correlation was seen as a validation of the investigation methods.

5.1.3 Transformer-based Report Embedding

We embedded each dynamic-analysis report with OpenAI's `text-embedding-3-small` model, which mapped an input sequence of up to 8 192 tokens into a 1536-dimensional vector $\mathbf{e} \in \mathbb{R}^{1536}$. Using the `tiktoken` tokenizer, every report was tokenized and split into contiguous chunks of at most 8 000 tokens, leaving a margin for the model's internal special tokens. All punctuation and special characters were retained because artefacts such as registry keys, mutex names, or obfuscated URLs were semantically informative for malware behaviour.

Each chunk c_i yielded an embedding $\mathbf{e}_i$. To obtain a single representation per report we applied *element-wise max-pooling*, which emphasized rare but high-importance features while remaining order-invariant. The resulting matrix $\mathbf{E} \in \mathbb{R}^{n_{\text{reports}} \times 1536}$ was compact, supported efficient nearest-neighbor retrieval and could be used for downstream tasks.

5.1.4 Cluster creation

To replace the malware family label with semantically richer classes, we benchmarked three unsupervised algorithms; k-means, DBSCAN and HDBSCAN on the high-dimensional embedding space of the samples.

Because visual interpretation beyond three dimensions is infeasible, cluster quality was quantified with the silhouette coefficient [54] and corroborated by qualitative inspection of two-dimensional t-SNE projections.

Across all candidate methods, k-means (initialized with k-means++) provided the best balance of simplicity, parameter robustness, and computational scalability. In contrast, DBSCAN and HDBSCAN required narrow parameter tuning: small deviations from the optimal settings led either to fragmented clusters or to a single dominant cluster.

The Silhouette analysis (Figure 5.4) suggested an optimal k close to 10, the number of malware families (n_{families}), which would merely reproduce the original labels. To obtain finer grained groupings, we therefore set $k = 3 \cdot n_{\text{families}}$ as any more would introduce unnecessary confusion for downstream tasks.

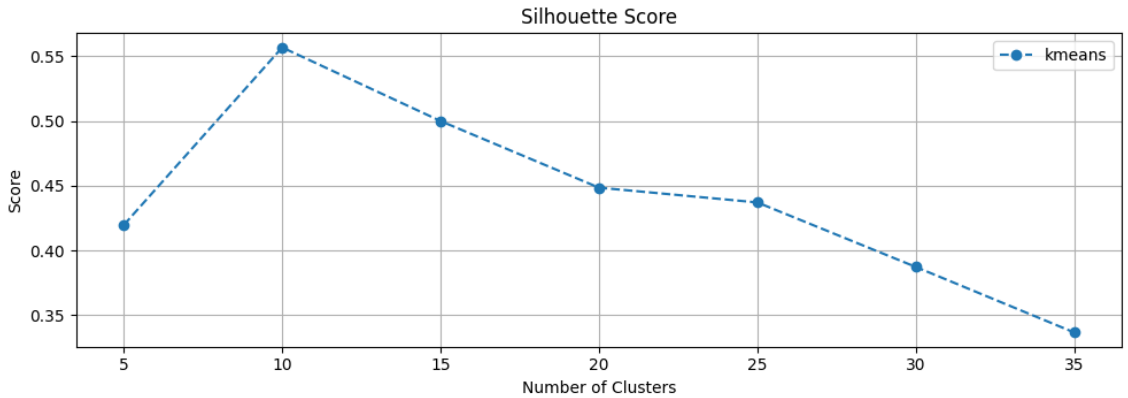


Figure 5.4: Silhouette score for different k-values in a k-means algorithm for 10 malware families with 2000 samples per AV label.

The resulting k-means labels defined as the BCs, were used for all downstream analyses and for training the supervised classifiers described in Section (5.2 Metric learning Model).

5.2 Metric Learning Model

This section outlines the construction of the ML model used to generate embeddings from static malware features. It describes the preprocessing of input data, the design

and training of the model architecture, and the implementation of the triplet loss function used to guide learning.

5.2.1 EMBER Features and Behavioral Cluster Labeling

EMBER features were chosen as the input to our ML model. As the model consists of a fixed size of input parameters, the EMBER features provided the advantage of a standardized mapping from PE files to 2381 dimensions. Additionally, Anderson et al. [29] demonstrated that models trained on these engineered features outperformed deep learning approaches like MalConv, which attempt to learn directly from raw binary data.

The EMBER features were standardized using scikit-learn's StandardScaler, applied on a per-feature basis to account for differences in scale; some features had magnitudes several orders of magnitude larger than others.

All data samples were labeled with the BC generated from the Report embedding, defined by their malware behavior.

The dataset was split into 80% for training, 10% for validation (to ensure meaningful model learning), and 10% for testing.

5.2.2 Model Architecture and Training Configuration

The model chosen for embedding EMBER features was a MLP with a triples loss function. The network structure, consisting of several linear layers transforming the input data to the embedding space, is shown in Table 5.1. Stochastic Gradient Descent (SGD) was used as the optimizer, momentum of 0.9, and weight decay of 1e-5. To improve convergence, an exponential learning rate scheduler with a decay factor of 0.9 was applied during training.

We explored both Sigmoid and ReLU as activation functions. Due to instability in training when using ReLU, Sigmoid was selected for the final model. Several weight initialization strategies were tested, ranging from PyTorch's default uniform initialization within the [-1,1] range to Xavier and Kaiming distributions. However, no significant performance improvements were observed with any specific initialization, so we opted for the uniform distribution to keep the model configuration simple.

In order to find the best possible ML model for this task, we conducted a systematic search for optimal hyperparameters. The hyperparameters investigated included learning rate (ranging from 0.001 to 0.02), dropout rate (from 0 to 0.4) and embedding dimension. The embedding dimension **E** ranged from 16 to 256 in our experiments in search for the most appropriate size.

Layer	Operation	Input Size	Output Size
1	Linear (Fully Connected) Activation Function	`input_size`	4096
	Batch Normalization	4096	4096
	Dropout	4096	4096
2	Linear (Fully Connected) Activation Function	4096	2048
	Batch Normalization	2048	2048
	Dropout	2048	2048
3	Linear (Fully Connected) Activation Function	2048	1024
	Batch Normalization	1024	1024
	Dropout	1024	1024
4	Linear (Fully Connected) Activation Function	1024	512
	Batch Normalization	512	512
	Dropout	512	512
5	Linear (Fully Connected) Activation Function	512	512
	Dropout	512	512
6	Linear (Fully Connected)	512	**E**

Table 5.1: Network Architecture of the ML model. The network first up-scales the 2381-dimensional input, and then narrows it down to the embedding dimension.

5.2.3 Triplet Loss Setup

The triplet loss function was implemented according to the previously described theory. For each batch of samples, the corresponding embedding feature vectors $\mathcal{E}$ and labels $\mathbf{y}$ were used. The margin m was set to 1. The process was carried out in the following steps in algorithm 5.3:

Algorithm 5.3 Triplet Loss Computation

Require: $\mathcal{E}$: Embeddings of batch samples, $\mathbf{y}$: Corresponding labels, m: Margin
Ensure: Average triplet loss for the batch
Compute pairwise distance matrix $\mathcal{D}$ from embeddings $\mathcal{E}$
Set diagonal of $\mathcal{D}$ to zero (remove self-distances)
Create adjacency matrix A where $A[i][j] = 1$ if $\mathbf{y}[i] = \mathbf{y}[j]$, else 0
Initialize empty list $\mathcal{L}_{triplet}$ to store losses
for each anchor i **do**
  for each positive j where $A[i][j] = 1$ and $i \neq j$ **do**
    for each negative k where $A[i][k] = 0$ **do**
      if $D[i][k] - D[i][j] + m > 0$ **then**
        $\ell \leftarrow D[i][k] - D[i][j] + m$
        Append ℓ to $\mathcal{L}_{triplet}$
      end if
    end for
  end for
end for
return average of all values in $\mathcal{L}_{triplet}$ =0

In line with the recommendations from the FaceNet paper, a custom batching function was implemented to ensure the presence of sufficiently many relevant samples in each batch. While it was not necessary for all class labels to be represented in every batch, a subset of labels was designated as the focus for each batch in order to form anchor points. For these selected classes, it was important to include enough samples to construct valid anchor-positive pairs. Without this adjustment, the loss could become unreasonably fluctuating due to unfavorable sampling, where the lack of intra-class examples led to ineffective or invalid triplet formation. Early empirical results related to this motivated the choice of a batch size of 256.

5.3 Evaluation Metrics on Embeddings

Determining whether the Report embedding and the ML embedding effectively captured meaningful aspects of malware similarity was not a straight forward task. Unlike traditional classification problems with clear accuracy metrics, evaluating the quality of an embedding requires a more nuanced approach. Several strategies were employed, combining both visual inspection and quantitative analysis suitable for each embedding.

5.3.1 Report Embeddings Validation

Validation of both Report embeddings began by assessing their alignment with malware family labels. For the BM25-based embedding, a cosine similarity matrix ($N \times N$) was first computed between all sample pairs. Due to the high dimensionality of the original BM25 vectors ($N \times V$, where V is the vocabulary size), running t-SNE directly on them was computationally infeasible. Instead, t-SNE

was applied to the precomputed similarity matrix to obtain a 2D representation for visualization. Each point was labeled with its corresponding AV label to inspect clustering patterns. Additional validation metrics of the BM25-based Report embedding, particularly regarding the clustering on the embedding, involved investigating intra-cluster cosine similarity against that of AV labels and random sample groupings.

In contrast, the Transformer-based Report embedding had a more compact dimensionality, making it feasible to apply t-SNE directly on the embedding vectors ($N \times \mathbf{e}$) without requiring a similarity matrix. This allowed a more direct view of how the transformer grouped samples based on learned semantic representations.

5.3.2 ML Model and Embedding Evaluation

First, the implementation of the ML model was evaluated by training it to embed samples based on malware families. Prior research [27] had shown that meaningful separation between malware families was achievable. To assess this, a k-Nearest Neighbors (kNN) classifier was used by projecting test samples into the training embedding space and classifying each one based on the majority malware family among its k nearest neighbors. Additionally, goodware samples were introduced into the dataset to evaluate whether the embedding could capture more fundamental aspects of behavior; was the activity malicious or not.

Evaluation of the embedding produced by the ML model, when trained using BCs, was conducted through two main approaches. First, t-SNE visualizations were examined to assess the alignment between the embedding structure and the BCs. This analysis was repeated with increasing dataset complexity by including 10, 20, and 30 malware families. Second, to evaluate whether the embedding captured behavior-based similarity, results were compared to a cosine similarity matrix derived from dynamic report analysis. The similarity of samples positioned closely in the embedding was measured and benchmarked against intra-family similarity and random sampling baselines. Furthermore, we measured kNN distances for test samples belonging to clusters that were absent from the training set to investigate the model's ability to identify novel or unseen classes.

6
Results

The results presented in this section follow a structured order: first, the Report embedding generated using BM25; second, the Report embedding created with Transformer-based vector representations; and third, the ML model along with its generated embedding and corresponding evaluations of both the model and the embedding.

6.1 Embedding Analysis Overview

We evaluated two vectorizations of the dynamic reports: sparse BM25 term-frequency vectors (Section 5.1.2) and dense 1,536-D Transformer embeddings (Section 5.1.3). The first of the two we used cosine similarity to create vector embeddings while for the other we used the produced vector space.

6.1.1 BM25-based Report Embedding

The t-SNE plot of the cosine similarity matrix, constructed from the dynamic analysis reports using BM25, demonstrates strong cohesion with AV labels, as shown in Figure 6.1. This result suggests that the Report embedding captured meaningful behavioral based patterns which are also represented in the malware family labeling. Additionally, some intra-family variation is observable within the AV labels, indicating richer behavioral distinctions than malware families alone.

Using the clustering methodology described in Section 5.1.4, all reports were clustered into 60 Behavioral Clusters (BCs) using k-means on the cosine similarity matrix (k=3 $\times$ $n_{families}$). Figure 6.2 shows the result (note that it uses fewer colors than the number of clusters). To evaluate the behavioral coherence within the BCs, cosine similarity was measured within each BC and compared against benchmarks for intra-malware-family similarity and random groupings. Several samples from each BC were randomly selected, and their similarity scores relative to the different groupings are presented in Table 6.1. The results confirm that the clustering is more meaningful than the random benchmark. However, it is not evident that the BC and intra-malware-family similarity distributions differ.

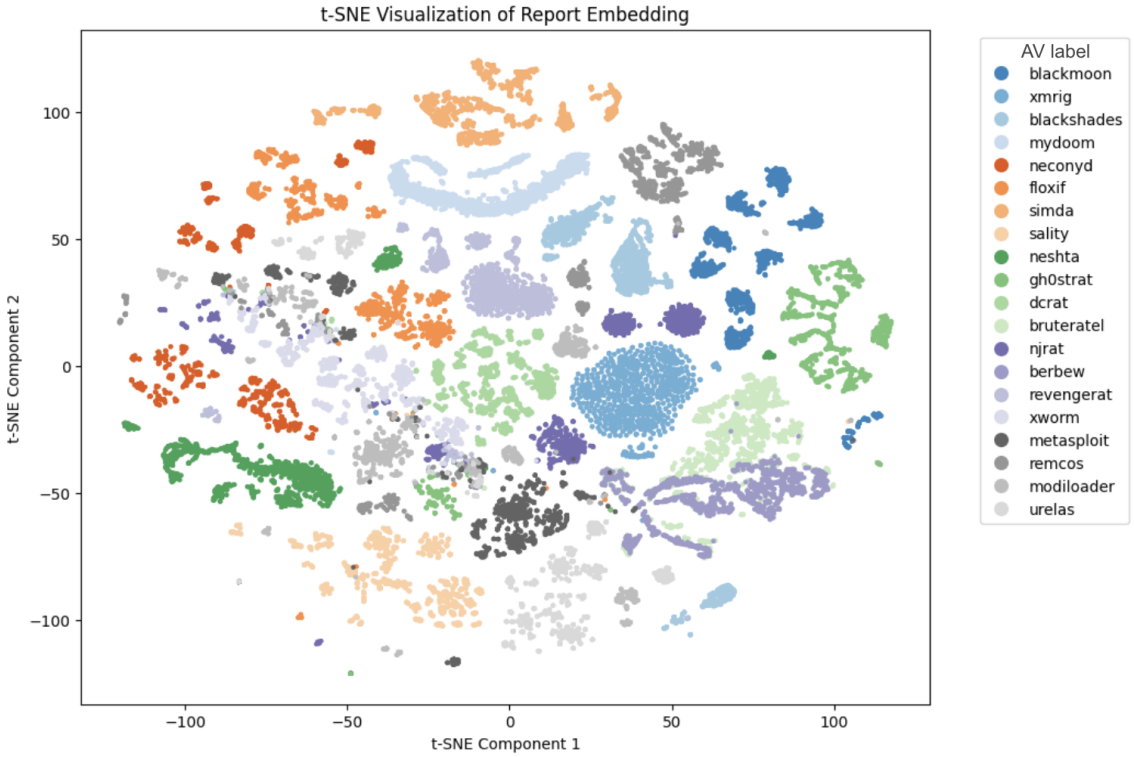


Figure 6.1: Report embedding visualized in two-dimensions using a t-SNE transformation on the cosine similarity matrix. Samples are labeled according to AV label, the resulting plot indicating cohesion between Report embedding and malware family. It also reveals intra-family variation among some malware families.

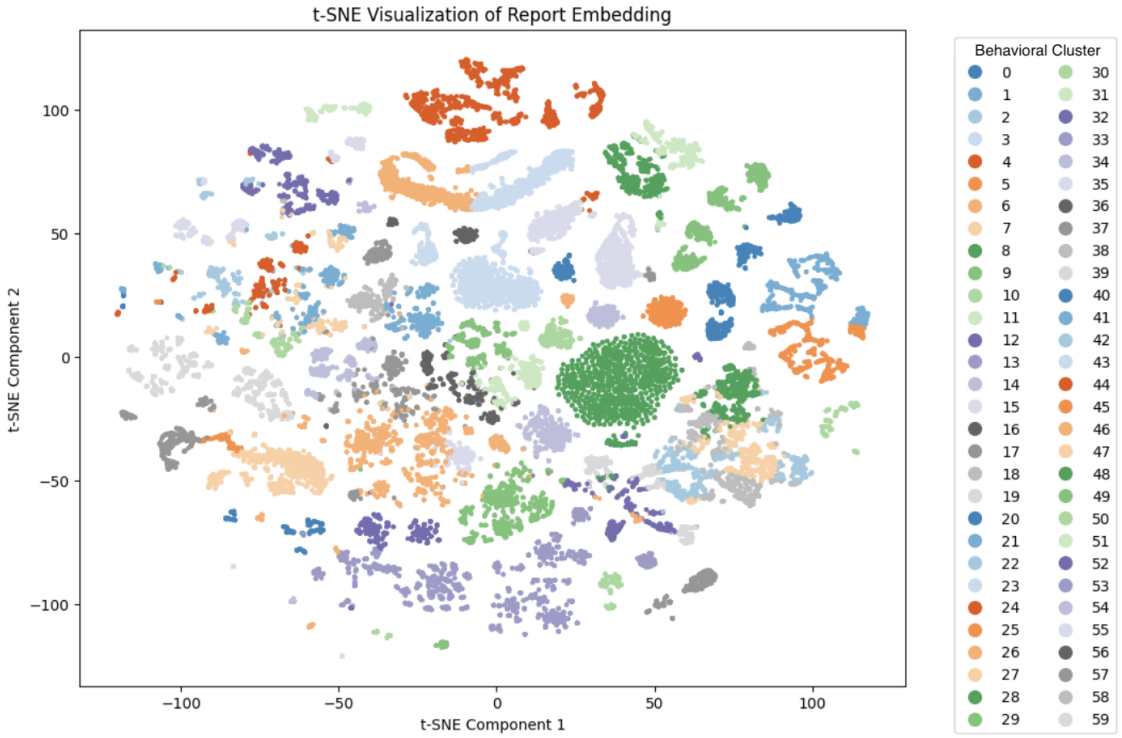


Figure 6.2: t-SNE visualization of Report embedding, with BCs produced with k-means on the cosine similarity matrix. Important to note that there are 60 clusters but only 40 unique colors.

Table 6.1: Cosine similarity statistics for several randomly selected samples from each BC and their relations to samples within the same BC along with malware family and random groupings. The results show that similarity within BCs is clearly higher than among random samples, but not statistically significantly higher than within malware families.

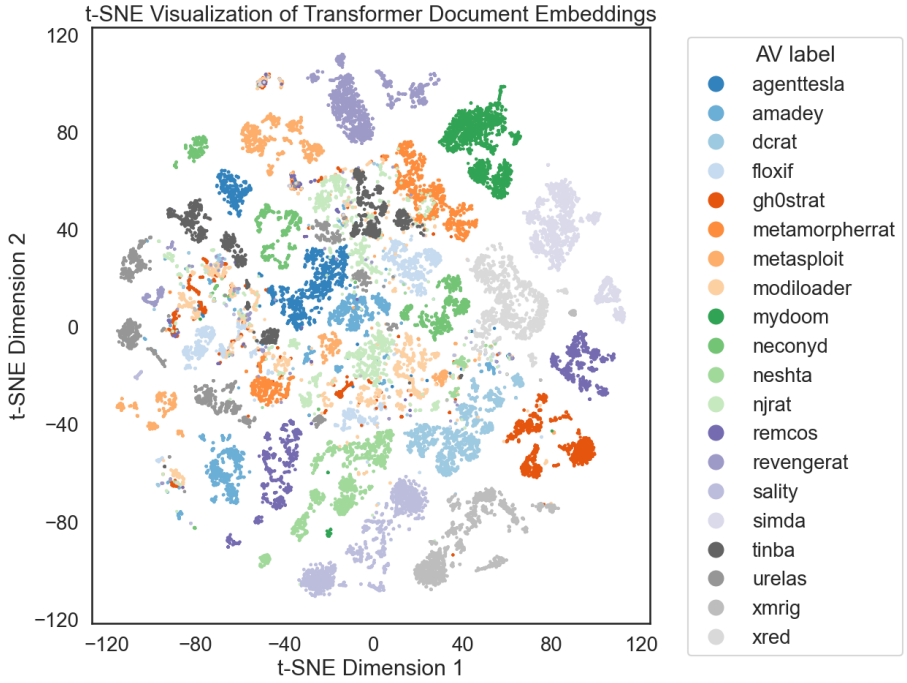
Type	Mean Cosine Similarity	Std Dev
Intra-cluster	0.122808	0.107453
Intra-malware-family	0.098925	0.097349
Random	0.016428	0.032922

6.1.2 Transformer-based Report Embeddings

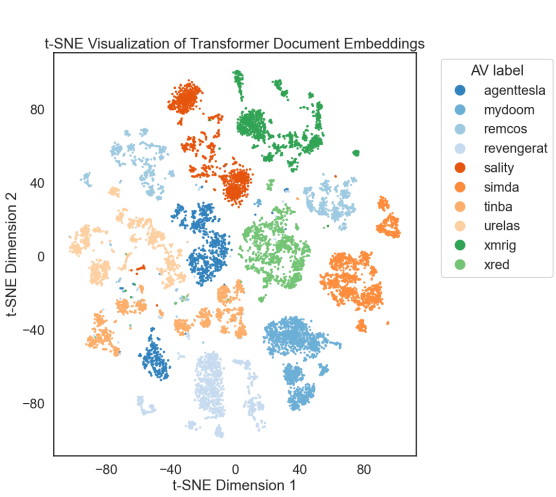
As described in Section 5.1.3, we used a Transformer-based embedding network to map dynamic malware behaviors into a 1536-dimensional latent space. Direct inspection of such a high-dimensional space is impractical; we therefore once again applied t-SNE to obtain a two-dimensional visualization.

Figure 6.3 shows the t-SNE visualization for datasets containing 10, 20, and 30 malware families, respectively, with 2000 samples per label. Despite the severe dimensionality reduction, the plots reveal clear inter-family separation. Within many

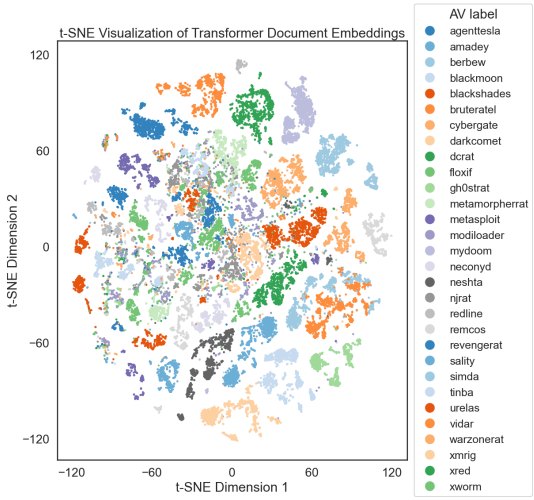
families, additional sub-clusters are apparent, indicating behavioral diversity that is nonetheless captured by the embedding model.



(a) 20 malware families.



(b) 10 malware families.



(c) 30 malware families.

Figure 6.3: Two-dimensional t-SNE visualization of the 1536-dimensional embedding space for (a) 10, (b) 20, and (c) 30 malware families (2000 samples per family). Overlap increases with the number of families due to the compression inherent in the projection; nevertheless, quantitative metrics in the original space indicate that clusters remain distinct.

As expected, introducing more malware families compresses inter-cluster distances in the two-dimensional plane, leading to increased visual overlap. Importantly, this

Table 6.2: Mean and standard deviation for pair-wise distance within the same malware family. Increasing amount of malware families and with that an increase of 2000 samples per family. The mean distance stays almost the same even with increasing amount of data showing within family separation remains the same.

nr. of malware families	mean distance	Std. Dev.
10	0.565	0.005
20	0.556	0.008
30	0.542	0.022

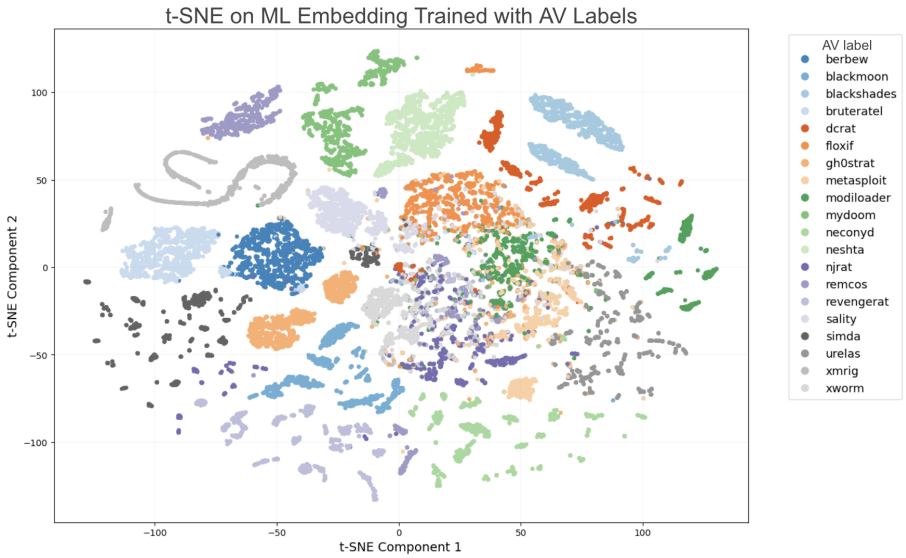
phenomenon is a limitation of the projection rather than the embedding itself: pair-wise distances computed in the original 1536-dimensional space, see Table 6.2 confirm that families remain well separated.

6.2 Metric Learning Model

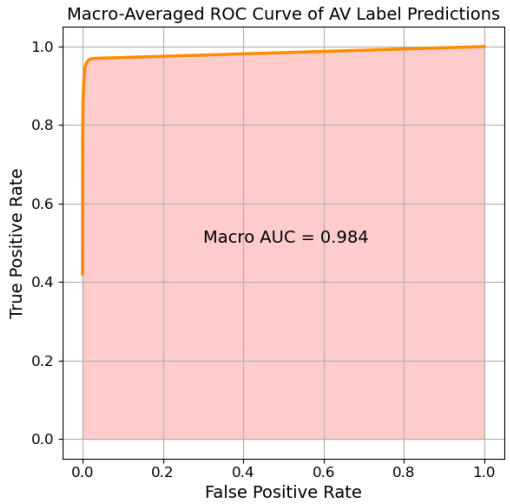
The results from the Metric learning (ML) model are presented in this section. First, the ML model was validated by training with AV labels as targets in order to separate malware families. The inclusion of goodware and its impact on the embedding was analyzed as validation of the ML model. Second, the model was trained using BCs as targets to generate a similarity-based embedding that captured malware behavior. This ML embedding was then analyzed to assess whether it reflected meaningful behavioral similarities.

6.2.1 Evaluation of ML Model

To evaluate both the ML model and the implementation of the Triplet loss, the model was initially trained using the AV labels as targets. The resulting 32-dimensional embedding is shown in Figure 6.4a, visualized in two dimensions using t-SNE. The clear separation between families suggests a successful implementation. Figures 6.4b and 6.4c display the averaged ROC curve across all families and the F1-scores per family, respectively, based on kNN classification of the ML embedding on a test set. An area under the curve (AUC) of 0.987 and a weighted average F1-score of 0.95 further indicates that the ML embedding meaningfully separated the families. Additionally, families with lower F1-scores correspond well to those that appear less distinctly separated in the t-SNE plot.



(a) t-SNE visualization of the embedding produced by a ML model trained using AV labels.



(b) Macro-Averaged ROC curve for AV family prediction along with Macro AUC.

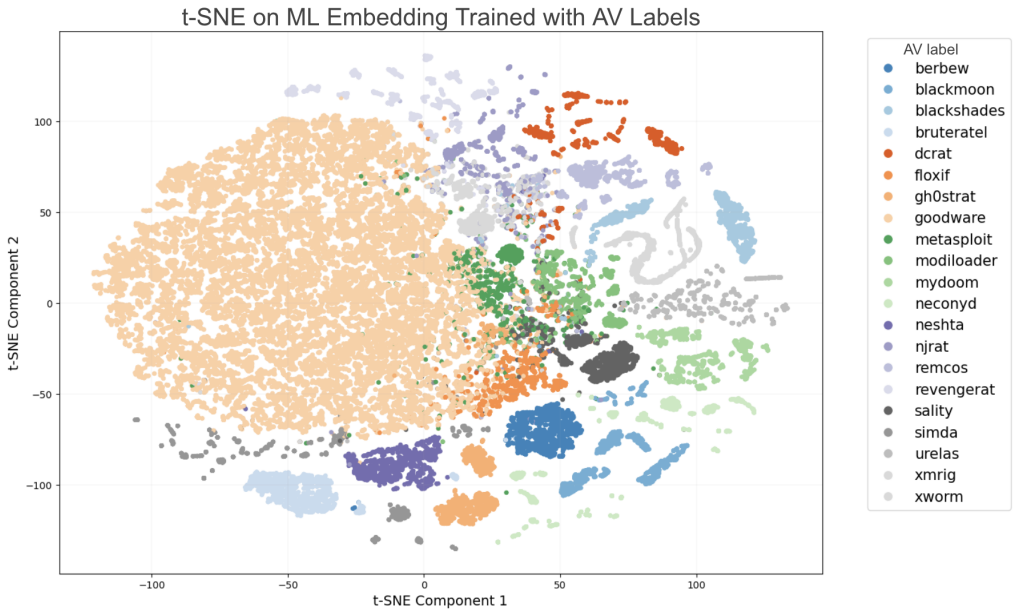
(c) F1-scores for AV-label prediction on the ML-embedding test set.

AV Family	**F1-score**
berbew	0.98
blackmoon	0.99
blackshades	0.99
bruteratel	0.98
dcrat	0.98
floxif	0.91
gh0strat	0.96
metasploit	0.83
modiloader	0.89
mydoom	1.00
neconyd	0.99
neshta	0.99
njrat	0.84
remcos	0.92
revengerat	0.98
sality	0.90
simda	0.97
urelas	0.99
xmrig	1.00
xworm	0.81
Mean (weighted)	**0.95**

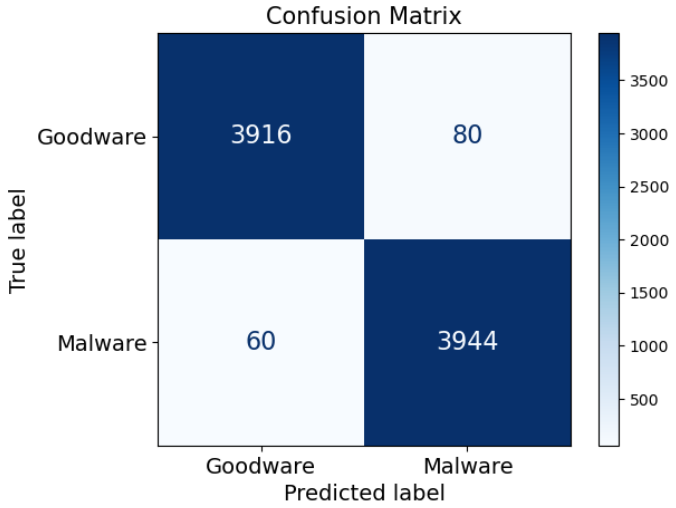
Figure 6.4: ML-embedding created by training with AV-labels (top), and family-prediction results using kNN on that embedding (bottom). When trained on EMBER features extracted from binary files, the model appears to learn meaningful distinctions between malware families.

Including goodware as a separate training category improved the evaluation of mean-

ingful separation in the ML embedding. Figure 6.5a shows the t-SNE visualization of the resulting embedding, which highlights a promising distinction between malware and goodware. The confusion matrix in Figure 6.5b further confirms this separation, as high classification scores were achieved when training a kNN-classifier on the embedding. The confusion matrix shows near-perfect separation between malware and goodware, with minimal false positives, suggesting strong discriminative ability. This indicates that the embedding captures discriminative features not only between malware families but also between malware and goodware.



(a) t-SNE visualization of ML embedding with training data containing 50% malware and 50% goodware.



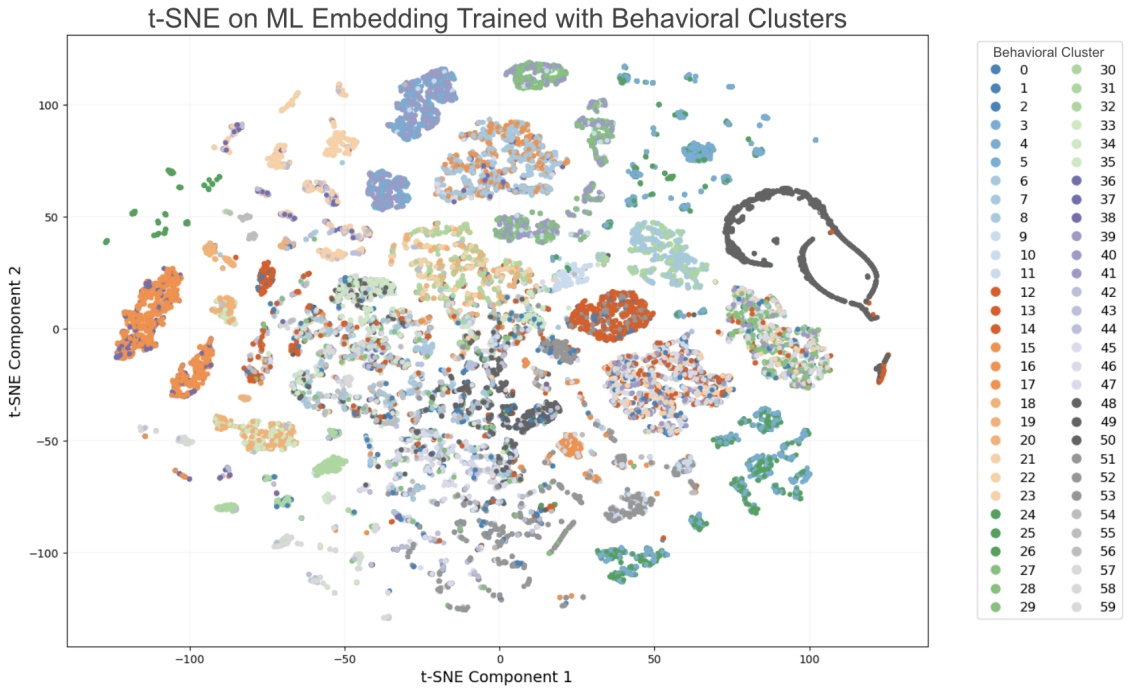
(b) Confusion matrix of kNN classifier trained on ML embedding and tested for malware vs goodware classification.

Figure 6.5: Results showing how goodware is represented in the ML embedding space. The t-SNE visualization reveals a clear separation from malware, and classification performance is high when distinguishing between the two. This suggests that the ML model can effectively capture fundamental differences between malicious and benign software using EMBER features.

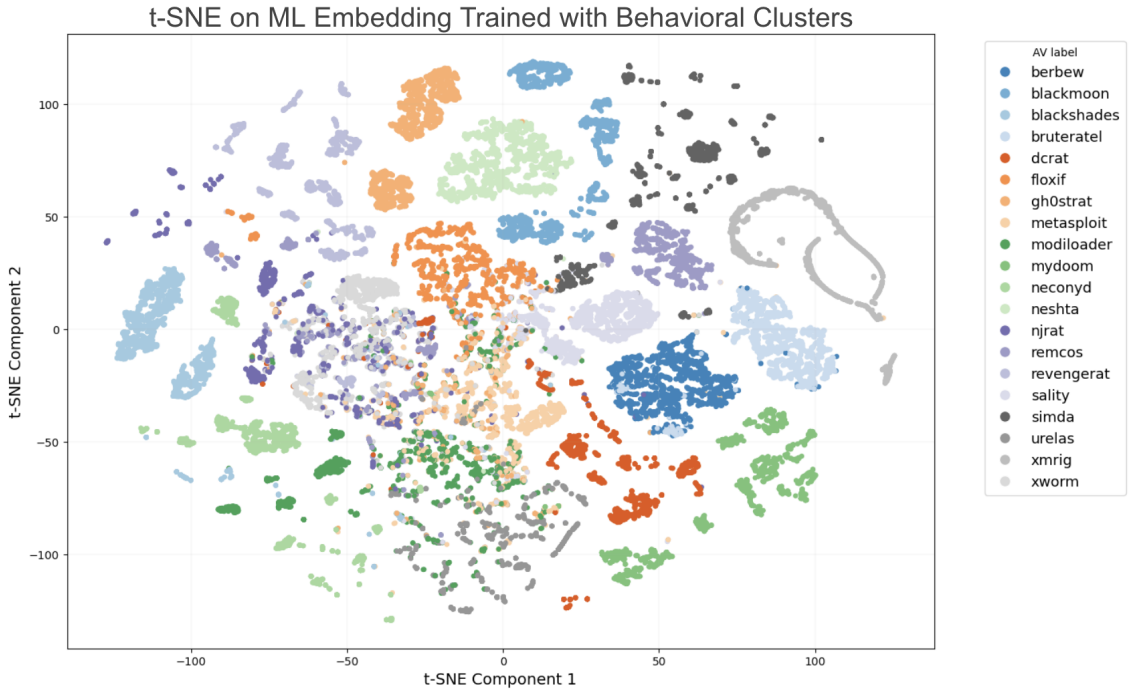
6.2.2 ML Embedding for Behavioral Clusters

After validating the ML model implementation, the objective was to train it using BCs as targets (instead of AV labels) in order to reflect the behavioral relationships

established in the Report embedding. The resulting ML embedding is shown in Figure 6.6a, after applying t-SNE to reduce it to two dimensions, with samples colored according to their BCs. The embedding does not appear as promising as when trained with AV labels. While some clustering is visible, labeling with BCs reveals that the clusters do not correspond well to the intended BCs. However, when the same embedding space is labeled with AV labels (Figure 6.6b), the clusters show a stronger alignment with malware families.



(a) ML embedding with BCs as labels.



(b) The same ML embedding with AV labels.

Figure 6.6: ML embedding generated with ML model trained with BCs as target labels (visualized with t-SNE). The same embedding is colored with a) BC and b) AV label. The figures highlight that the ML model struggles with separating the samples into BCs, and instead still separates mainly on AV label.

6.2.3 Report Similarity in ML Embedding

Further analysis of the ML embedding focused on the similarity in dynamic behavior between neighboring samples. This was measured using the original cosine similarity between each sample's dynamic report. For several randomly selected samples from each cluster, the average cosine similarity to their ten nearest neighbors in the ML embedding was computed. As a comparison, each sample was also matched with ten random samples from the same AV label group and ten completely random samples.

Figure 6.7 presents the results of this experiment (showing every fifth BC for clarity). The nearest neighbor and AV label groups generally exhibit higher cosine similarity than the completely random group. However, it is not clear that nearest neighbors consistently outperform AV label samples in similarity. This aligns with earlier results (Figure 6.6), which indicated a strong influence of AV label separation in the ML embedding.

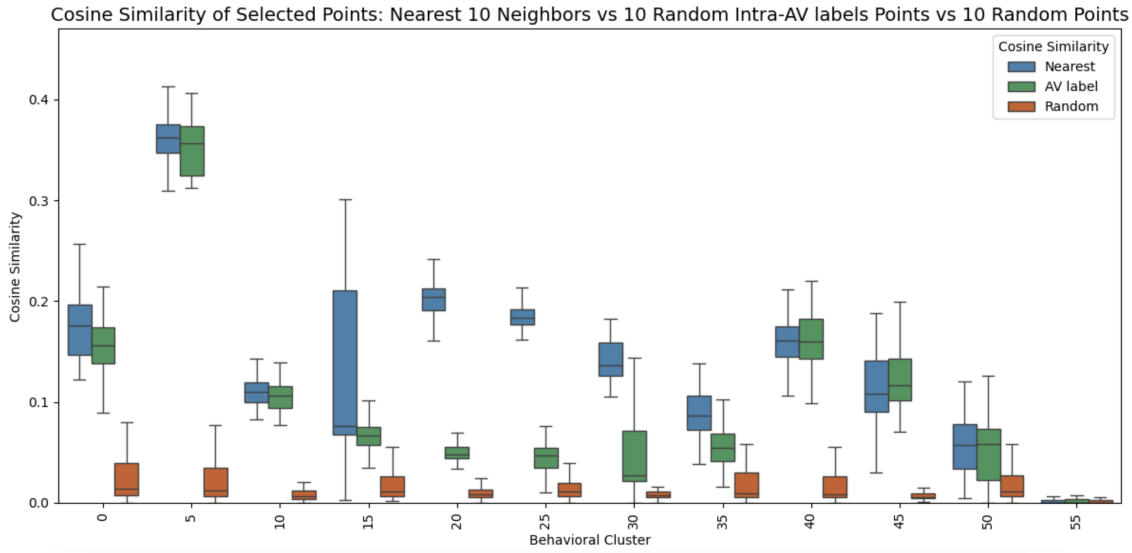


Figure 6.7: Cosine similarity among samples in the ML embedding for different groups: 1) similarity with the 10 nearest neighbors, 2) similarity for 10 random samples within the same malware family and 3) 10 random samples. Each iteration consists of sampling one random sample in each cluster, introducing variation in the results. The results show that similarity among nearest neighbors is only slightly higher than within the same malware family, and significantly higher than among random samples.

To investigate the impact of novel malware types and behavior on the ML embedding space, several BCs were deliberately excluded from the training dataset. Two test sets were then constructed: one containing samples from the same distribution as the training data, and another containing samples from the excluded clusters. Figure 6.8 shows the distribution of average distances from test samples to their nearest neighbors in the training embedding. Samples from unseen clusters exhibit a higher frequency of longer distances, although some overlap with seen clusters for smaller distances. These results suggest that novel samples are generally distinguishable in

the learned embedding space. However, it would still be hard to investigate and confirm this in specific single case scenarios.

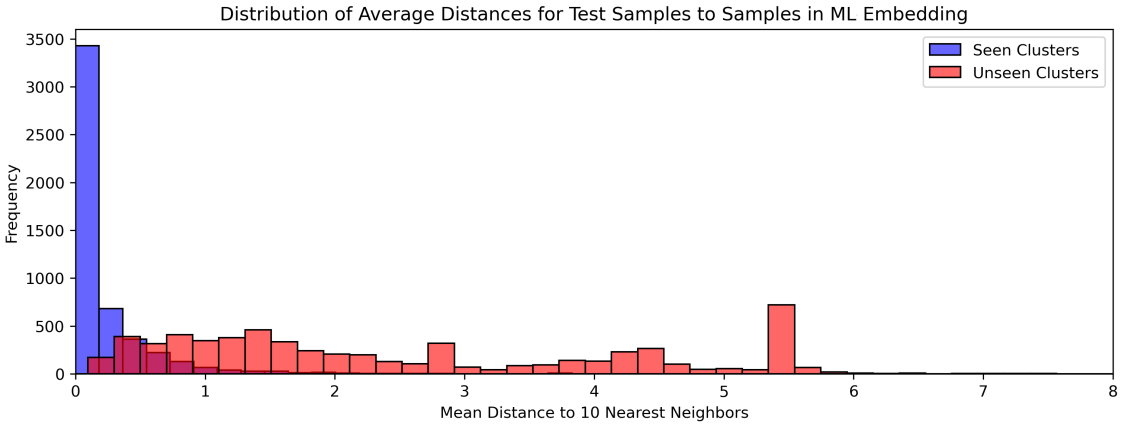
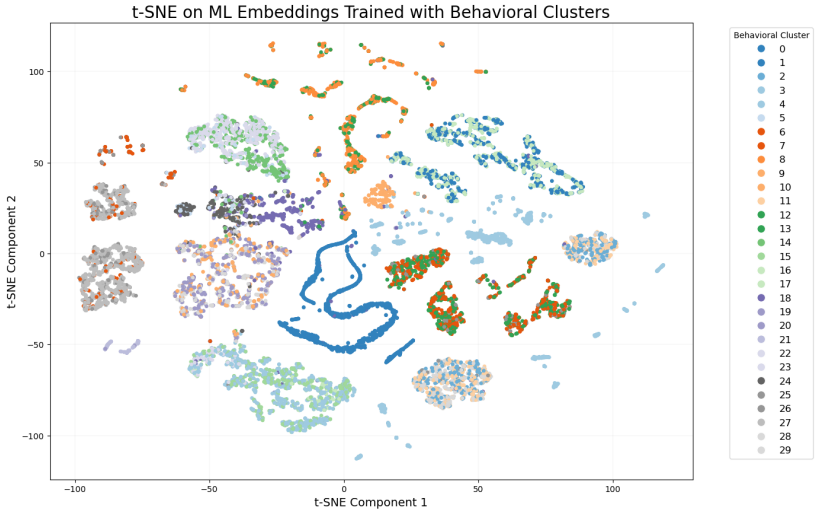


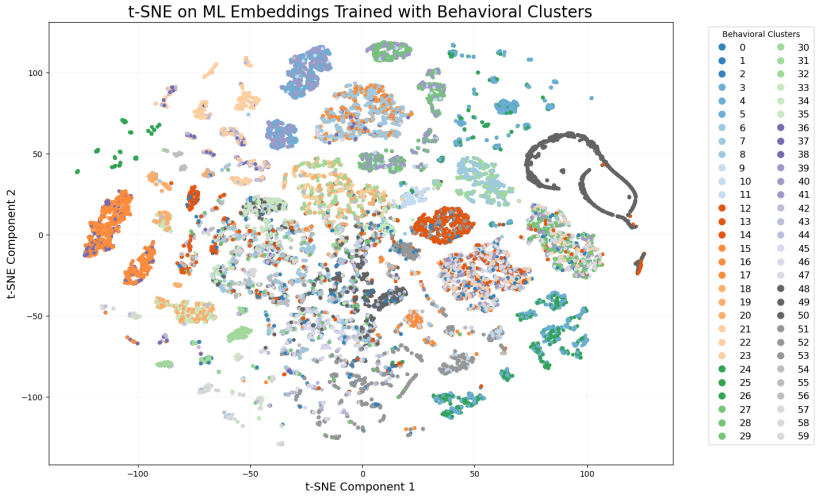
Figure 6.8: Distribution of average distances to the 10 nearest neighbors in the training embedding, for two types of test samples: (1) those from known classes (mirroring the training data), and (2) those from unseen classes whom were not present in the training data. The generally higher mean distance for the second group suggests that novel behaviors tend to stand out in the embedding space.

6.2.4 Scalability of the ML Embedding with Increasing Data Diversity

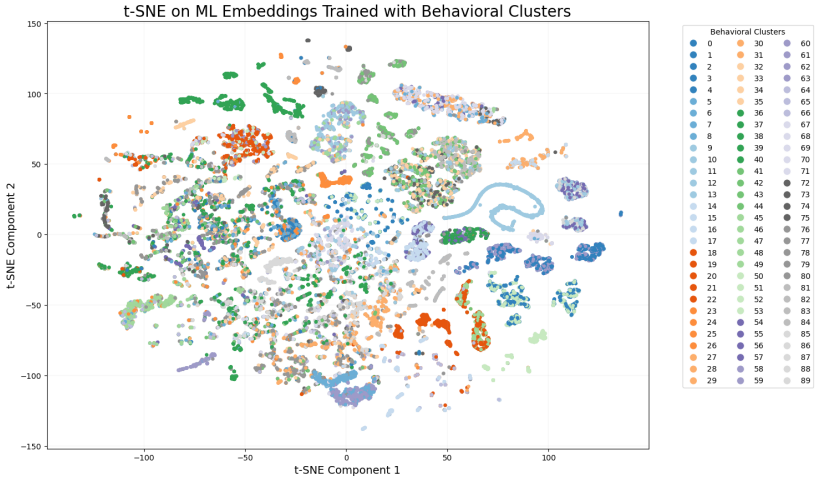
The scalability of the ML model and its embedding was analyzed by introducing more variation in the data by making it more diverse, which was done by including additional malware families. The ML embedding is visualized in two dimensions (using t-SNE) for datasets containing 10, 20, and 30 malware families, each with 2000 samples per family (Figure 6.9). As before, it becomes evident that the model struggles to preserve the behavioral structure from the Report embedding, which the BCs represent. This issue becomes more pronounced as the variation in the data increases.



(a) 10 malware families



(b) 20 malware families



(c) 30 malware families

Figure 6.9: ML embedding visualized with t-SNE for varying degrees of diversity in the training data. While the embedding generally struggles to separate BCs, increased data variety appears to exacerbate this issue.

6.3 Insights on Model Parameter Selection

Investigating the ML model across different architectures and hyperparameter combinations, the results suggest a high level of robustness. When training the model for malware family embedding, test F1-scores consistently ranged between 96% and 97.5% (Figure A.1), regardless of learning rates, embedding dimensions and dropout rates. Notably, performance tended to increase with higher dropout rates, which may be counterintuitive. This is addressed further in the Discussion section. Figure A.2 presents the F1-scores for various configurations when the model was instead trained using BCs as target labels, with scores ranging from 54% to 56%. Although some variation exists, no parameter combination resulted in a significantly poorer performance. This aligns with earlier findings indicating that the model struggles to form meaningful embeddings when trained on BCs.

In contrast, the model is sensitive to batch size and activation functions. ReLU was initially chosen for its ability to mitigate the vanishing gradient problem. However, it often led to exploding gradients and unstable training. Sigmoid, on the other hand, offered a more stable training process. Regarding batch size, we were able to validate FaceNet's recommendation to avoid small batch sizes, as a sufficient number of samples is needed to form meaningful triplets.

7
Discussion

This chapter discusses the implications of the previously presented results. It examines the strengths and limitations of the report embedding techniques and provides explanations for the performance of the ML embedding, particularly its challenges in separating Behavioral Clusters (BCs). The chapter concludes with reflections on potential future work and relevant ethical considerations.

7.1 Report Embeddings

Both report embedding techniques (BM25 and Transformer embedding) show a cohesion with malware families, while also revealing richer correlations, such as subclusters within the families (Figure 6.1 and 6.3). These observations highlight the potential of the dynamic report-based embedding to serve as a foundation for behavior driven analysis. Cosine similarity among BCs in the BM25 embedding further indicates a higher degree of internal similarity compared to malware families. While malware families can reach cosine similarity values comparable to those of the BCs (Table 6.1), they also frequently exhibit a wider range of similarities. This highlights that AV families can be broad, with sub genres in behavior.

The vectorized Report embedding obtained with the Transformer offers advantages over the BM25 approach, particularly for faster behavior-based analysis. Since the Transformer is pretrained, the resulting embedding is deterministic and not relative to other reports in corpus. Once the malware database is embedded, the similarity of new malware samples can be computed instantly, making this approach much more suitable for rapid and scalable analysis.

In contrast, BM25 depends on a fixed size document corpus, which poses challenges when analyzing new malware samples in relation to the existing malware dataset. To include new samples, one must recompute their similarity to the entire corpus, making the process inefficient and potentially infeasible for large datasets.

7.2 Metric Learning Embedding

The ML model showed strong performance when trained to embed malware families. Compared to previous work using EMBER features as input, the classification of malware families based on the ML embedding achieved F1-scores comparable to, and in some cases exceeding, previous studies [27]. It is however difficult to directly

compare results when the data sets are not identical nor standardized, which often is the case within malware research. The results also showed that including goodware as a labeled class in the training set enabled the ML model to clearly distinguish between malicious and benign behavior. This separation is practically important, as the value of the embedding would be limited if it could not effectively differentiate between malicious and benign software.

When training the ML model using BCs as target labels, the results were less encouraging. The t-SNE visualization of the ML embedding in two dimensions revealed that the samples were not clearly separated according to the BCs generated from the Report embedding. Instead, stronger clustering patterns remained within the malware family groupings.

A possible explanation for the ML model's difficulty in separating samples by BCs is the limited representation of dynamic behavior in the EMBER feature set (as they are generated through static analysis [28]). Without sufficient behavioral information in the input features, the model may struggle to identify meaningful patterns that distinguish BCs. It is essential that the information defining the sample labels is adequately represented in the input for the model to learn effectively. For example, FaceNet's implementation of the triplet loss was highly successful in separating photographs of faces and clustering them by individuals [13]. This success is due to the fact that facial photographs contain rich visual information that distinctly defines each person. In contrast, our study aimed at capturing malware behavior using EMBER features, can be likened to using FaceNet not to identify individuals, but to group them by personality traits. This is inherently more abstract and less directly observable in the data.

Why is the information that forms the BCs (the dynamic behavior in the reports) not represented in the EMBER features? One reason could be that some malware uses API calls to retrieve additional code to execute. This new code is not present in the original PE file's binary and therefore cannot be captured by the static EMBER features. For example, AV families such as Metasploit use automatically generated payloads and are encoded using the Shikata Ga Nai encoder, making every sample have a unique signature to avoid signature-based detection [55]. Future work could include filtering out malware samples that make such calls for additional executable code and those who encode the payload using Shikata Ga Nai encoding.

The ambiguous results presented in Section 6.3, where classification performance remained consistent across different hyperparameter combinations of the ML model, highlight a deeper issue related to the use of EMBER features and the evaluation of the resulting embedding. First, EMBER features inherently support strong classification performance [28], which explains why the model performs well even with unusually high dropout rates. This leads to a second, more fundamental challenge: how can we effectively validate a similarity-based embedding when classification accuracy alone may not reflect the true quality or structure of the learned space? This is why t-SNE visualizations played such a central role in our results section;

they offered an intuitive way to assess the structure of the embedding beyond what classification metrics alone could reveal.

7.3 Using Dynamic Behavior for Malware Analysis

Our methodology was to use the Report embedding to define BCs and then use these labels to train a model that captures malware similarity using only features extracted from the binary file. This approach required clustering the Report embedding, as seen in Figure 6.2. Striking a balance when selecting appropriate number of clusters presented a conceptual challenge: too few clusters risked grouping together behaviorally distinct samples, while too many could potentially introduce fine-grained distinctions that are harder to capture using static features alone.

Clustering samples based on their behavioral reports is both challenging and possibly unnecessary. If the Report embedding is used directly for malware similarity analysis, there is no strict need to define clusters for training an ML model. In downstream tasks, where the goal is often to retrieve samples similar to the one being analyzed, clustering adds little value. Simply identifying the 10–50 most similar samples may be sufficient, removing the need for labeling the data into BCs.

7.4 Future Work

We have already shown that the static EMBER feature vector is too sparse to capture a sample's true behavior, even when a deep neural network (DNN) is thrown at it. The obvious next step is to bridge that information gap between EMBER features and the dynamic reports.

Rather than taking the two-step detour of clustering on dynamic data and then metric learning from the static EMBER vector, we should stay entirely in the transformer-based embedding space. A transformer encoder gives us a fixed-length, high-dimensional vector for every report, delivers near-instant nearest-neighbor lookups, and scales linearly with corpus size, unlike the BM25 + cosine combo, which blows up quadratically. The embedding is deterministic, too, so similarity scores are stable and reproducible. Finally, the transformer model we used here was straight off the shelf, which means there's still plenty of head-room for domain-specific fine-tuning.

Future work could also explore a more structured and domain-informed interpretation of the reports in their original JSON format. We acknowledge that the hierarchical structure of the JSON data likely encodes meaningful relationships between fields. However, due to limited prior knowledge of malware analysis, our approach

was to simplify the reports by extracting and concatenating all values into a flat list of strings (see Listing 5.1). Developing methods that better preserve and leverage the structural relationships within the JSON format may lead to higher-quality report embeddings and improved downstream performance.

7.5 Ethical Considerations

The field of malware analysis inherently exists within a dynamic adversarial landscape, where attackers and defenders are in a constant arms race. This raises ethical considerations that are important to acknowledge.

First, while the goal of this work is to improve malware analysis through behavioral embeddings, the same techniques could be misused. For example, a malicious actor might reverse-engineer such an embedding space to evade detection. By incrementally modifying malware to shift its position in the embedding space, nudging it toward regions associated with benign or unrelated behavior, they could potentially bypass classifiers or mislead analysts.

Second, the broader methodology of creating behavioral embeddings based on report data has implications beyond cybersecurity. Similar approaches could be applied to analyze and cluster user behavior, which might be exploited for surveillance or targeted advertising. If applied without ethical oversight, this could intrude on individual privacy or be used to manipulate people.

8
Conclusion

The project set out with two main objectives. First, to produce an embedding that captures malware behavior, thereby enabling similarity-based analysis of malware. Two alternative approaches were explored to create this embedding: 1) BM25 combined with cosine similarity for document comparison, and 2) Transformer-based document embedding. Both methods yielded promising results, successfully producing embeddings that showed cohesion with malware families, along with additional nuances, particularly within families. However, the latter method offers greater potential for fast and computationally efficient analysis, as it allows new malware samples to be directly compared to the existing dataset.

The second objective was to investigate whether it is possible to create a behavioral-based embedding using only input from the binary file. While achieving separation by malware family was very feasible, behavior-based separation proved too difficult when using EMBER features as the sole input to the Metric learning model. This is likely because EMBER features, derived solely from static analysis of PE files, do not contain sufficient information about a sample's dynamic behavior for the model to effectively distinguish between behavior-based categories.

You can't judge a book by its cover nor a malware by its binary. This holds true, at least when the binary is represented using EMBER features and used to infer the malware's behavior. The main motivation for binary analysis was to enable faster behavior-based assessments and reduce computational demands. However, Recorded Future Triage already successfully processes up to 1.5 million malware samples daily. Given that sandbox environments are already in place and producing dynamic behavioral reports, analyzing these reports for malware similarity using a transformer-based embedding is, in our view, the most effective path forward.

Bibliography

[1] SentinelOne. *Key Cyber Security Statistics for 2025*. Updated March 28, 2025. Mar. 2025. URL: https://www.sentinelone.com/cybersecurity-101/cybersecurity/cyber-security-statistics/ (visited on 05/13/2025).

[2] Control D. *Malware Statistics Trends in 2023*. Accessed: 2025-05-20. 2023. URL: https://controld.com/blog/malware-statistics-trends.

[3] Ömer Aslan Aslan and Refik Samet. "A Comprehensive Review on Malware Detection Approaches." In: *IEEE Access* 8 (2020), pp. 6249–6271. DOI: 10.1109/ACCESS.2019.2963724.

[4] Roberto Perdisci Babak Rahbarinia Marco Balduzzi. *Exploring the Long Tail: The Life of Rare Malware*. Trend Micro White Paper. Accessed: 2025-02-10. 2018. URL: https://documents.trendmicro.com/assets/wp/wp-exploring-the-long-tail.pdf.

[5] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. "Tracking concept drift in malware families." In: *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. AISec '12. Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 81–92. ISBN: 9781450316644. DOI: 10.1145/2381896.2381910. URL: https://doi.org/10.1145/2381896.2381910.

[6] Edward Raff et al. *Malware Detection by Eating a Whole EXE*. 2017. arXiv: 1710.09435 [stat.ML].

[7] Marek Krčál et al. "Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only." In: *International Conference on Learning Representations*. 2018. URL: https://api.semanticscholar.org/CorpusID:69911013.

[8] Haojun Wang et al. "Deep Learning and Regularization Algorithms for Malicious Code Classification." In: *IEEE Access* 9 (2021), pp. 91512–91523. DOI: 10.1109/ACCESS.2021.3090464.

[9] Martin Apel, Christian Bockermann, and Michael Meier. "Measuring similarity of malware behavior." In: *2009 IEEE 34th Conference on Local Computer Networks*. 2009, pp. 891–898. DOI: 10.1109/LCN.2009.5355037.

[10] Songsong Liu et al. "Enhancing malware analysis sandboxes with emulated user behavior." In: *Computers Security* 115 (2022), p. 102613. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2022.102613. URL: https://www.sciencedirect.com/science/article/pii/S0167404822000128.

[11] Ulrich Bayer et al. “Scalable, Behavior-Based Malware Clustering.” In: Jan. 2009.

[12] Infosecurity Magazine. *Ransomware Groups Linked via Shared Code and Ransom Notes*. Accessed: 2025-05-20. Infosecurity Magazine. 2022. URL: https://www.infosecurity-magazine.com/news/ransomware-shared-code-ransom-notes/.

[13] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “FaceNet: A unified embedding for face recognition and clustering.” In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[14] Lan Luo et al. “On Teaching Malware Analysis on Latest Windows.” In: *Journal of The Colloquium for Information Systems Security Education* 9 (Mar. 2022), p. 7. DOI: 10.53735/cisse.v9i1.152.

[15] Joshua Saxe and Konstantin Berlin. “Deep neural network based malware detection using two dimensional binary program features.” In: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. 2015, pp. 11–20. DOI: 10.1109/MALWARE.2015.7413680.

[16] Mansour Ahmadi et al. *Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification*. 2016. arXiv: 1511.04317 [cs.CR]. URL: https://arxiv.org/abs/1511.04317.

[17] Mihai Christodorescu and Somesh Jha. “Static Analysis of Executables to Detect Malicious Patterns.” In: *12th USENIX Security Symposium (USENIX Security 03)*. Washington, D.C.: USENIX Association, Aug. 2003. URL: https://www.usenix.org/conference/12th-usenix-security-symposium/static-analysis-executables-detect-malicious-patterns.

[18] Edward Raff and Charles Nicholas. *A Survey of Machine Learning Methods and Challenges for Windows Malware Classification*. 2020. arXiv: 2006.09271 [cs.CR]. URL: https://arxiv.org/abs/2006.09271.

[19] AV-TEST Institute. *Malware Statistics & Trends*. Accessed: February 5, 2025. 2024. URL: https://www.av-test.org/en/statistics/malware/.

[20] J. Zico Kolter and Marcus A. Maloof. “Learning to Detect and Classify Malicious Executables in the Wild.” In: *Journal of Machine Learning Research* 7.99 (2006), pp. 2721–2744. URL: http://jmlr.org/papers/v7/kolter06a.html.

[21] David Mohaisen and Omar Alrawi. “AV-Meter: An Evaluation of Antivirus Scans and Labels.” In: July 2014, pp. 112–131. ISBN: 978-3-319-08508-1. DOI: 10.1007/978-3-319-08509-8_7.

[22] Jasmin Bharadiya. “Machine Learning in Cybersecurity: Techniques and Challenges.” In: *European Journal of Technology* 7 (June 2023). DOI: 10.47672/ejt.1486.

[23] Maksim E. Eren et al. *Catch’em all: Classification of Rare, Prominent, and Novel Malware Families*. 2024. arXiv: 2403.02546 [cs.CR].

[24] Mesut Guven. "Dynamic Malware Analysis Using a Sandbox Environment, Network Traffic Logs, and Artificial Intelligence." In: *International Journal of Computational and Experimental Science and Engineering* 10 (Sept. 2024). DOI: 10.22399/ijcesen.460.

[25] Vandana Srivastava and Vaibhav Sharma. "Sandbox Technology in a Web Security Environment: A Hybrid Exploration of Proposal and Enactment." In: (June 2022).

[26] Recorded Future. *Recorded Future: Threat Intelligence Cloud*. https://www.recordedfuture.com/. Accessed: 2025-02-07.

[27] Ethan M. Rudd et al. *Efficient Malware Analysis Using Metric Embeddings*. 2022. arXiv: 2212.02663 [cs.LG]. URL: https://arxiv.org/abs/2212.02663.

[28] Hyrum S. Anderson and Phil Roth. *EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models*. 2018. arXiv: 1804.04637 [cs.CR]. URL: https://arxiv.org/abs/1804.04637.

[29] Hyrum S Anderson and Phil Roth. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models." In: *arXiv preprint arXiv:1804.04637*. 2018.

[30] Edward Raff et al. "MalConv: Malware Detection by Eating a Whole EXE." In: *AAAI*. 2018.

[31] Manuel Egele et al. "A survey on automated dynamic malware-analysis techniques and tools." In: *ACM Computing Surveys* 44.2 (2012), pp. 1–42.

[32] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. "Scalable, behavior-based malware clustering." In: *NDSS*. 2009.

[33] Konrad Rieck et al. "Automatic analysis of malware behavior using machine learning." In: *Journal of Computer Security* 19 (June 2011), pp. 639–668. DOI: 10.3233/JCS-2010-0410.

[34] Hyrum S Anderson and et al. "Improving malware classification using dynamic analysis and multi-task learning." In: *ACM AISec*. 2012.

[35] Xinyuan Hu and et al. "DUET: Dynamic and Static Analysis for Malware Clustering." In: *ACSAC*. 2013.

[36] Wenliang Xia and et al. "Hybrid malware detection model based on RNN and CNN using static and dynamic features." In: *Computers & Security*. 2022.

[37] I.A Basheer and M Hajmeer. "Artificial neural networks: fundamentals, computing, design, and application." In: *Journal of Microbiological Methods* 43.1 (2000). Neural Computting in Micrbiology, pp. 3–31. ISSN: 0167-7012. DOI: https://doi.org/10.1016/S0167-7012(00)00201-3. URL: https://www.sciencedirect.com/science/article/pii/S0167701200002013.

[38] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks." In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009).

[39] Bernhard Mehlig. "Supervised Learning." In: *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021, pp. 73–74.

[40] Elad Hoffer and Nir Ailon. *Deep metric learning using Triplet network*. 2018. arXiv: 1412.6622 [cs.LG]. URL: https://arxiv.org/abs/1412.6622.

[41] Brian Kulis. "Metric Learning: A Survey." In: *Foundations and Trends® in Machine Learning* 5.4 (2013), pp. 287–364. ISSN: 1935-8237. DOI: 10.1561/2200000019. URL: http://dx.doi.org/10.1561/2200000019.

[42] Deen Dayal Mohan et al. *Deep Metric Learning for Computer Vision: A Brief Overview*. 2023. arXiv: 2312.10046 [cs.CV]. URL: https://arxiv.org/abs/2312.10046.

[43] Mahmut KAYA and Hasan Şakir BİLGE. "Deep Metric Learning: A Survey." In: *Symmetry* 11.9 (2019). ISSN: 2073-8994. DOI: 10.3390/sym11091066. URL: https://www.mdpi.com/2073-8994/11/9/1066.

[44] Yikai Li, C. L. Philip Chen, and Tong Zhang. "A Survey on Siamese Network: Methodologies, Applications, and Opportunities." In: *IEEE Transactions on Artificial Intelligence* 3.6 (2022), pp. 994–1014. DOI: 10.1109/TAI.2022.3207112.

[45] Karen Sparck Jones. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." In: *Journal of Documentation* 28.1 (Jan. 1972), pp. 11–21. DOI: 10.1108/eb026526. URL: https://doi.org/10.1108/eb026526.

[46] Stephen Robertson and Hugo Zaragoza. "The Probabilistic Relevance Framework: BM25 and Beyond." In: *Found. Trends Inf. Retr.* 3.4 (Apr. 2009), pp. 333–389. ISSN: 1554-0669. DOI: 10.1561/1500000019. URL: https://doi.org/10.1561/1500000019.

[47] Jiawei Han, Micheline Kamber, and Jian Pei. "2 - Getting to Know Your Data." In: *Data Mining (Third Edition)*. Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 39–82. ISBN: 978-0-12-381479-1. DOI: https://doi.org/10.1016/B978-0-12-381479-1.00002-2. URL: https://www.sciencedirect.com/science/article/pii/B9780123814791000022.

[48] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[49] OpenAI. *tiktoken: A fast BPE tokenizer for use with OpenAI's models*. https://github.com/openai/tiktoken. Accessed: 2025-05-20. 2022.

[50] OpenAI. *Embeddings Guide: Embedding Models*. Accessed: 2025-04-29. URL: %5Curl%7Bhttps://platform.openai.com/docs/guides/embeddings#embedding-models%7D.

[51] Hongjin Su et al. “One Embedder, Any Task: Instruction-Finetuned Text Embeddings.” In: *Findings of the Association for Computational Linguistics: ACL 2023*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 1102–1121. DOI: 10.18653/v1/2023.findings-acl.71. URL: https://aclanthology.org/2023.findings-acl.71/.

[52] James MacQueen. “Some methods for classification and analysis of multivariate observations.” In: *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*. Vol. 5. University of California press. 1967, pp. 281–298.

[53] Laurens van der Maaten and Geoffrey Hinton. “Visualizing Data using t-SNE.” In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.

[54] Peter Rousseeuw. “Rousseeuw, P.J.: Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Comput. Appl. Math. 20, 53-65.” In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1987), pp. 53–65. DOI: 10.1016/0377-0427(87)90125-7.

[55] Google Cloud Threat Intelligence. *Shikata Ga Nai Encoder Still Going Strong*. Accessed: 2025-05-23. 2019. URL: https://cloud.google.com/blog/topics/threat-intelligence/shikata-ga-nai-encoder-still-going-strong/.

A

Appendix 1

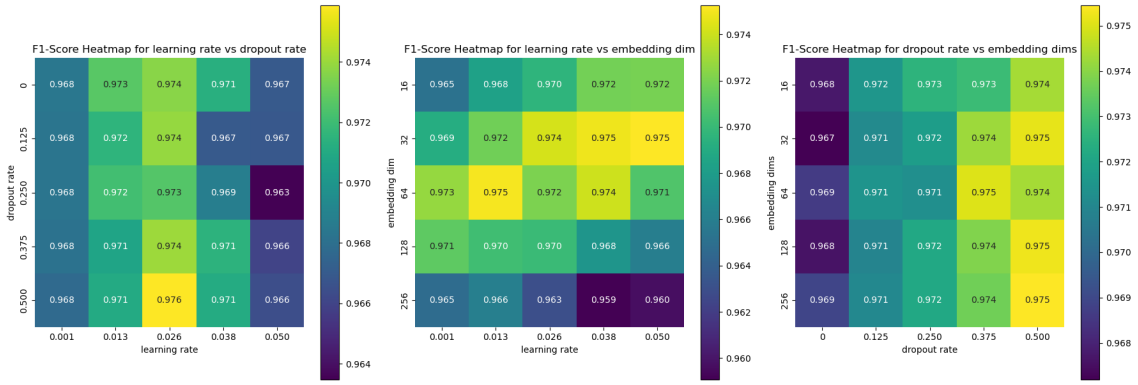


Figure A.1: F1-scores for AV-label classification on test data using kNN applied to the learned embedding, across different ML-model parameter combinations. In each subplot, two parameters are varied while one of the following is fixed at a time: learning rate (0.01), dropout rate (0.3), or embedding dimension (32), respectively. Each score represents an average over four runs with different random seeds for the train-test split. Although classification scores show minimal variation, the model's strong performance at high dropout rates is noteworthy.

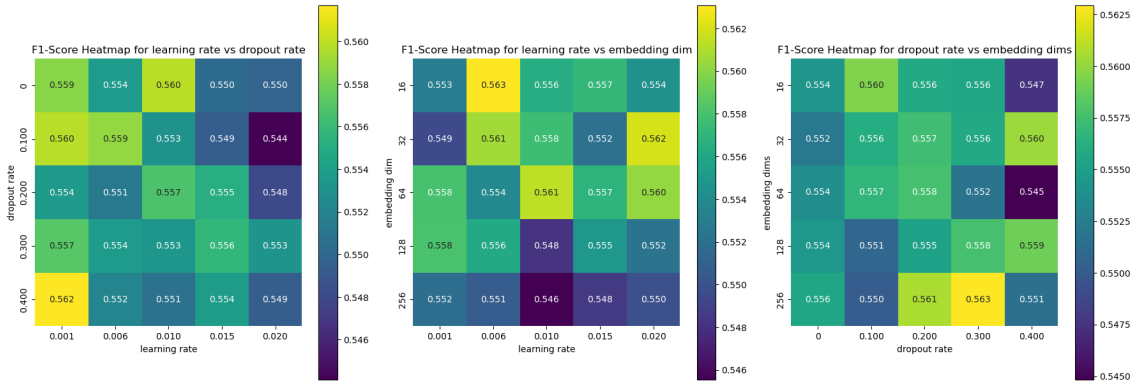


Figure A.2: F1-scores for Behavioral Cluster (BC) classification on test data using kNN applied to the learned embedding, across different ML-model parameter combinations. In each subplot, two parameters are varied while one of the following is fixed at a time: learning rate (0.01), dropout rate (0.3), or embedding dimension (32), respectively. Each score represents an average over four runs with different random seeds for the train-test split. The consistently low variance in classification scores suggests that the model struggles to produce a meaningful embedding for BCs.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY