

Evaluating a Novel Voting System for Malware Dataset Curation with Siamese Networks

1st Sahil Singh

Department of Computer Science
Rhodes University
Grahamstown, South Africa
g21s6211@campus.ru.ac.za

2nd Stones Dalitso Chindipha

Department of Computer Science
Rhodes University
Grahamstown, South Africa
s.chindipha@ru.ac.za

Abstract—Replicating research findings is among the most substantial evidence that a scientific claim reflects a real phenomenon rather than chance, bias, or idiosyncrasies of a single laboratory. In malware security, however, replication is especially difficult and is widely regarded as one of the most challenging areas for reproducible research. Those difficulties arise from the domain itself: malware is inherently hostile, constantly evolving, and entangled with legal and ethical constraints that limit data sharing and experimental control. To address these barriers, this study developed a novel malware voting system that aggregates multiple analysis sources to produce robust labels and, in doing so, generated a new, carefully curated dataset. Using that dataset, we successfully replicated the malware classification experiment originally proposed by Hsiao et al. (2019), replicating key performance patterns and verifying central claims about classifier behaviour. To validate the voting system and dataset, we verified visual cluster consistency using perceptual hashing techniques and measured classifier generalisation under varying N-way constraints. The results demonstrate that our voting approach yields consistent labels that enable faithful replication while mitigating some domain-specific obstacles. This work therefore provides both a practical tool for future studies and empirical evidence that replication in malware research, though difficult, can be achieved with thoughtful methodology and rigorous safeguards.

Index Terms—Average Hash, Malware Image Classification, One-Shot Learning, Siamese Convolutional Neural Networks.

I. INTRODUCTION

Malware is a term for all software designed to deliberately exploit vulnerabilities in any programmable device. Malware is strategically deployed by cybercriminals to extort sensitive information, financial assets, deliberately crash critical systems, or gain unauthorised access to networks and data. These threats can be categorised into various families based on their unique methods of exploitation, such as Viruses, Worms, and Trojans [1, 2]. According to AV Test’s analysis¹, 100 million new malware variants were produced between 2023 and 2024. This 10% spike in new malware variants highlights the flaws of most anti-malware solutions [3]. The two primary malware analysis methods researchers employ to enhance malware classification are static and dynamic analysis. Static Analysis is done by breaking down the components of malware programs and analysing the underlying logic. Signature-based detection

is a commonly used static analysis technique that extracts the signature, a unique set of bytes of a file. The extracted signature is then compared to an existing database of known malware signatures to determine the maliciousness of a given file [4]. A significant issue of this technique is its dependence on frequent updates of malware signatures. This hinders its ability to keep up with the constant growth of malware [5]. To further sabotage the efficacy of static analysis, malware authors developed exploits that could circumvent these detection methods, such as obfuscation. Obfuscation hinders any attempt to extract, modify, or reverse-engineer the source code by inserting useless code and disrupting the program’s logical structure [6]. In contrast, however, dynamic analysis involves monitoring the live execution of malware in a controlled environment. This approach, therefore, helps identify behavioural traits that can be used to detect other unknown malware variants, thus bypassing the need to de-obfuscate malware for detection [7]. Despite its advantage in detecting obfuscated malware a range of challenges arises. Extensive monitoring is time-consuming as it involves continuous supervision of malware actions over a prolonged period. Additionally, the inability to replicate real-world conditions in a controlled environment produces inconsistent behaviour of certain types of malware, particularly those that are triggered by factors such as current time or specific user inputs [8, 9]. Due to this, this study aims to address the following Objectives:

- 1) To curate and produce an updated malware dataset that reflects current threat landscapes.
- 2) To conduct a comprehensive replication study of existing Siamese Networks used in one-shot malware classification.
- 3) To demonstrate a solution for malware family classification.

II. REPRODUCIBILITY CHALLENGES IN MALWARE RESEARCH

Replicating experiments in malware research faces unique and difficult obstacles that stem from the nature of the subject matter. Malware artefacts are hostile as they can damage systems, exfiltrate data, or spread uncontrollably if mishandled

¹<https://portal.av-atlas.org/>

[10, 11]. Because of that risk, many experiments cannot be rerun casually in shared environments, and results produced in one lab may depend on containment measures, sandbox configurations, or live networks that other researchers cannot or will not reproduce. Safety and containment, therefore, become first-order constraints on reproducibility: labs need controlled networks, virtualised or air-gapped environments, and strict operational procedures, all of which affect what can be shared or rerun.

Legal, ethical, and policy constraints further limit replication [12, 13, 14]. Possession and distribution of malware samples, sinkholed domains, or captured command-and-control traffic can violate laws and vendor terms, and handling real victim data risks privacy breaches [13, 14]. Consequently, many datasets are unavailable or heavily redacted, and researchers may be unable to share raw binaries or full telemetry [10]. Even when samples can be shared under NDAs or restricted repositories, the friction of access hampers independent verification and follow-up work [15].

Data quality and ground-truth labelling present another major challenge. Malware labelling is noisy: different scanning engines disagree, vendor names and signatures change over time, and a single sample may exhibit multiple behaviours depending on execution context [11]. Evasion techniques such as packing, polymorphism, and anti-analysis measures can hide behaviours in some runs but not others, producing inconsistent labels [6, 16]. These issues mean that two teams using the “same” dataset ostensibly may actually be training and evaluating on materially different labellings, undermining direct comparisons.

Experimental environments and non-determinism also reduce the replicability of results. Dynamic analysis traces depend on OS versions, installed libraries, timing, randomness, and network conditions; small changes in sandbox instrumentation or network responses can alter observed behaviour [10]. External dependencies — third-party APIs, live domains, or C2 servers — may disappear or behave differently, making exact replay impossible. Additionally, tool and library version drift in feature extractors and machine-learning frameworks can change results even when the dataset is unchanged.

Evaluation protocols and reporting gaps are an avoidable but common barrier. Papers often omit precise train/test splits, random seeds, preprocessing steps, or hyperparameter choices; they may report aggregate metrics without distributional detail. That lack of detail turns reproduction into guesswork and encourages unintentional discrepancies. Similarly, class imbalance, sampling bias, and subtle differences in metric computation can produce apparently conflicting outcomes.

Practical mitigations reduce but do not eliminate these difficulties. Researchers can publish sanitised artefacts (feature vectors, behaviour summaries, or synthetic proxies) when raw samples cannot be shared, and provide reproducible environment descriptors such as container images, VM snapshots, or

precise configuration manifests. Consensus labelling (voting across multiple engines, human triage) and thorough provenance metadata (timestamps, feed sources, collection policies) help make datasets interpretable. Deterministic pipelines — fixed seeds, containerised workflows, and shared scripts for preprocessing and feature extraction — increase the likelihood that others can reproduce results. Finally, explicit reproducibility checklists in the methods (versions, splits, seeds, containment procedures) and robust sensitivity analyses (time-split tests, cross-validation, and multiple metrics) improve transparency and scientific confidence [14].

A. Malware Development

Due to the rapid growth of malware, manual labelling methods have been proven to be time-consuming and ineffective [17]. Furthermore, the sophistication of malware evasion techniques has reduced the reliability of existing malware classification techniques, thereby making the collection and labelling of new malware samples significantly more challenging [18]. While malware continues to evolve technically, its success is frequently enabled by non-technical failures. Recent cybersecurity-readiness studies highlight that organisational vulnerabilities and human-factor weaknesses remain critical entry points for attacks. For instance, fragmented policy enforcement and low digital literacy often exacerbate susceptibility to breaches, as inconsistent management practices create gaps that malware authors exploit [19]. Furthermore, with human error contributing to over 90% of successful cyberattacks [20].

B. Concept Drift

Concept drift is a term in Machine Learning (ML) that describes the gradual change in underlying patterns in data, which affects the accuracy of ML models. As malware improves through feature updates and bug fixes, the similarities between the original software and its future iterations begin to slowly degrade [21]. Consequently, the prediction performance of a model begins to decline as malware evolves [22]. To effectively detect and classify emerging malware, ML models must be updated and retrained once enough drift samples become available.

C. Open and Public benchmarks

The limited amount of publicly available malware datasets can be attributed to legal restrictions prohibiting the sharing of malicious files [23]. Although virus-sharing repositories such as Virus Share exist, the majority of these repositories provide raw binaries, often without labels. Therefore, third-party services such as VirusTotal are employed; however, these sites are rate-limited, making the generation of large-scale benchmark datasets infeasible [24]. This challenge in creating a reliable benchmark dataset is further exacerbated by labelling inconsistencies across multiple Antivirus engines.

III. NEURAL NETWORKS FOR LIMITED-DATA CLASSIFICATION

The large data dependence of Machine Learning and Deep Learning models, necessary to provide robust and accurate predictions and performance, has been addressed by the development of specialised neural networks and techniques that thrive in limited data environments.

A. Siamese Networks

The Siamese Network is a novel technique widely used in image detection. Unlike traditional machine learning and deep learning classification methods that focus on absolute classifications, Siamese Networks focus on the relative comparison between a pair of samples [25, 26]. The Siamese Network utilises two identical networks with shared weights. These sub-networks work in parallel to identify the underlying features and assess the similarity between the input samples [27]. The output is compared, and a similarity score is calculated using a loss function to determine how closely the two images are related.

B. Few-Shot Learning

Few-Shot Learning (FSL) is a Machine Learning technique that aims to train a model on N classes with K samples. The support and query sets contain a few example images per class during training. FSL malware classification can reduce the data dependence of deep learning algorithms and has significant implications for practical applications. Therefore, it is an important and urgent problem to study FSL malware classification [7].

C. One-Shot Learning

One-Shot Learning (OSL) is a type of FSL that enables a model to recognise new object categories from a single sample, allowing for rapid generalisation from minimal data. One-Shot Learning involves classifying an input into one of N novel classes (N -way), given only one support example for each of those N classes.

[28] explored using a Bayesian learning framework in OSL. The proposed model compared the probability that the object is present in the image with the probability that only the background noise is present in the image. The model would compare the probabilities produced for both images to decide whether the object is within the image. This novel technique was able to learn new categories with fewer training images, resulting in classification accuracy up to 95%.

[29] incorporated Hierarchical Bayesian Program Learning (HBPL) in OSL. HBPL is a novel technique that uses the principles of compositionality and causality to better capture the underlying nuances of the structure of an image. Unlike traditional methods that memorise the pixel formation,

HBPL represents the different image structures as generative programs by learning the steps needed to create the sample image, thus providing more context to the sample image. The proposed HBPL model could effectively utilise the prior knowledge gained from previous samples to classify the new images better. The model produced a remarkable error rate of 4.8%, not only nearing the general human error rate of 4.5% when detecting objects in a given image but also showing a significantly better accuracy than the alternatives, boasting a 73.5% improvement in accuracy over the traditional neural networks.

Building on [28]’s OSL model, [30] implements a Siamese Network. The network comprises an array of Convolutional layers, each with its respective filters that vary in size. An image is parsed through the Siamese Network. Each layer extracts specific features, with the final layer integrating these features into a single vector representation, commonly referred to as a feature embedding. A sigmoidal output unit calculates the distance between each sub-network. The sigmoidal unit σ ensures the calculation is between 0 and 1, whereby 1 describes the images as identical and zero describes having no similarities. The model demonstrated better performance compared to the reference models, achieving a precision of 92%. However, it did not outperform the HBPL model proposed by [29] but showed potential in its ability to perform close to its successor despite not utilising prior knowledge.

Texture-based analysis, as proposed by [31], leverages the visual texture patterns contained in malware. Although it has demonstrated superior accuracy and is resistant to obfuscation, it is heavily based on including known obfuscated samples within the test dataset, making this model less robust against unknown malware variants. The heights of the malware images were not fixed, resulting in varying image sizes, which led to increased computation times.

While [31] laid the groundwork in malware image conversion, [32] conducted one of the first key studies in demonstrating the efficacy of Siamese Networks in malware image classification through OSL. The authors rescaled the grey-scaled images collected of the malware samples, which were then used to train the Siamese Network to classify them into their respective malware families. The study showed that the OSL Siamese model outperformed the Nearest Neighbour and Random Guessing models when dealing with a limited dataset typical of new malware variants. This is crucial as the ability to classify malware in minimal examples can significantly reduce the time and resources required for detection. To address the lack of adaptability of unknown malware variants, [32]’s OSL model demonstrated remarkable accuracy and robustness compared to the other deep learning techniques when dealing with limited data. However, the accuracy of the model declines as the number of malware classes increases. Classifying malware samples is essential for detecting zero-day malware.

In response to this problem, the inclusion of byte code and opcode frequency-based images in the Dual Siamese model

produced a 3.3% increase in accuracy at a 2-way evaluation compared to [32]’s model. This could be due to the 128x128 pixel scaling applied to all images generated, which may allow for better classification by retaining more data in the image and potentially helping to extract additional features from each sample. However, the increased dimensionality of malware images demands greater computational resources and processing power, potentially leading to prolonged training times.

The literature indicates a significant shift towards utilising OSL Siamese networks for malware detection, as the capability of classifying malware-limited samples addresses not only the challenges that arise with the rapid growth of malware but also reinforce the importance of reliable malware-detection pipelines within real-world environments.

IV. METHODOLOGY AND IMPLEMENTATION

A practical approach to improving label quality is a malware voting (consensus) system: aggregate detections from multiple scanning engines or detectors, compute per-engine weights or trust scores from historical performance, and produce consensus labels with associated confidence scores. Thresholds (e.g., majority, weighted majority, or confidence cutoffs) determine final labels; ambiguous or low-confidence cases are flagged for human triage or further dynamic analysis. Record the full voting vector, engine versions, and timestamps so downstream users can re-evaluate labels as engines evolve. This pipeline reduces single-engine bias, makes labelling decisions auditable, and provides per-sample uncertainty, which is useful for robust evaluation [11, 33].

As illustrated in Figure 1, the system is divided into five main stages: Data Gathering, Data Conversion, Hash Organisation, Model Creation, and N-way Evaluation. The data conversion stage involves transforming each malware binary sample into an image. The hash organisation stage clusters visually similar images together to create variants. In the model creation stage, manual hyper-parameter tuning and various ensembling methods were implemented, followed by the N-way evaluation stage, whereby the accuracy between the enhanced and the original model was evaluated at varying N-way ranges.

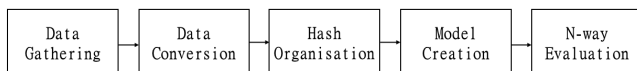


Fig. 1: System Overview

A. Data Gathering

A malware dataset was compiled from the VirusShare² academic collection, using a selection of malware collected from February 18, 2025, to March 28, 2025. The dataset used

²<https://virusshare.com/research>

provides more recent samples compared to Ember³ and other Kaggle datasets⁴. Due to the novelty of the malware samples used in this study, no official labels were available at the time of analysis. To mitigate the lack of labels and the possibility of mislabelling, VirusTotal⁵ was used to label each sample, and a voting system was employed. Each sample was scored based on the number of antivirus engines that identified it as malicious. Samples detected by more than 50% of the antivirus engines were deemed malicious, and the most popular label was used to classify these samples.

Algorithm 1 illustrates the operation of the antivirus voting system used to label malware samples. For each file, the number of antivirus (AV) engines detecting it as malicious (*MalwareVotes*) and the total number of engines that scanned it (*TotalVotes*) were recorded. The *Consensus* score was calculated by dividing *MalwareVotes* by *TotalVotes*. If the consensus exceeded the threshold of 0.5, the sample was classified as *malicious*. Otherwise, it was labelled as *benign* and discarded. For malicious samples, the highest weighted malware family name suggested by the Antivirus engines were assigned as the *Final Label*.

To address any label noise and vendor inconsistency, a weighted scoring system was implemented. We assigned higher confidence weights ($w=3$) to vendors historically recognised for low false-positive rates (e.g., Microsoft, Kaspersky, ESET) consistent with performance benchmarks from independent auditing bodies such as AV-Comparatives⁶, while assigning baseline weights ($w=1$) to generic engines. Furthermore, raw labels were normalised to strip generic prefixes (e.g., ‘Win32/’, ‘Heur:’) to ensure that ‘W32/Emotet’ and ‘Trojan:Emotet’ are correctly grouped as the same family. Although this normalisation prioritised high-precision grouping, it preserved specific identifiers in ambiguous cases. For example, consensus labels yielding high-confidence but generic terms (e.g., ‘trojan.js’) were retained only when the samples formed distinct, dense clusters. We validate this decision by ensuring these generic samples form structurally verified clusters during the hashing stage as detailed in Section IV-C. Rather than forcing these samples into broader families, which risks introducing label noise, we create a new family cluster based on this generic consensus.

Additionally to prevent voting inflation, we consolidated detections from engines sharing the same core technology (e.g., GData and Emsisoft utilising the Bitdefender engine) into a single weighted vote. Furthermore, multi-engine scanners that aggregate signatures from multiple distinct parents (e.g., Qihoo 360) were excluded entirely to prevent complex cross-cluster correlations. This ensures that the consensus score reflects independent verification rather than redundant

³<https://github.com/elastic/ember>

⁴<https://www.kaggle.com/datasets/ang3loliveira/malware-analysis-datasets-top1000-pe-imports>

⁵<https://www.virustotal.com/>

⁶<https://www.av-comparatives.org/tests/summary-report-2024/>

Algorithm 1: Robust Malware Labelling with Weighted Voting and Normalisation**Input:** Set of malware samples S **Output:** Curated dataset D with robust labels**Data:** Trusted vendors T

```

1 foreach  $s_i \in S$  do
2    $h_i \leftarrow \text{ExtractHash}(s_i);$ 
3    $R_i \leftarrow \text{QueryVirusTotal}(h_i);$ 
   // Sort vendors so Trusted ones
   // come first (Prioritise W=3)
4   Sort  $R_i$  such that  $v \in T$  come first;
   // Step 1: Weighted Malicious
   // Consensus
5    $V_{\text{mal}} \leftarrow 0;$ 
6    $V_{\text{total}} \leftarrow 0;$ 
7    $\text{Seen}_{\text{engines}} \leftarrow \emptyset;$ 
8    $\text{Active}_{\text{voters}} \leftarrow \emptyset;$ 
9   foreach  $\text{vendor } v \in R_i$  do
   // Explicitly Exclude
   // Multi-Engine Scanners
10  if  $\text{IsMultiEngine}(v)$  then
11    continue;
12   $E_{\text{core}} \leftarrow \text{GetCoreEngine}(v);$ 
13  if  $E_{\text{core}} \notin \text{Seen}_{\text{engines}}$  then
14     $\text{Seen}_{\text{engines}} \leftarrow \text{Seen}_{\text{engines}} \cup \{E_{\text{core}}\};$ 
15     $\text{Active}_{\text{voters}} \leftarrow \text{Active}_{\text{voters}} \cup \{v\};$ 
16     $w_v \leftarrow (v \in T) ? 3 : 1;$ 
17    if  $v.\text{detection} = \text{Malicious}$  then
18       $V_{\text{mal}} \leftarrow V_{\text{mal}} + w_v;$ 
19       $V_{\text{total}} \leftarrow V_{\text{total}} + w_v;$ 
20   $\text{Score}_{\text{consensus}} \leftarrow V_{\text{mal}}/V_{\text{total}};$ 
21  if  $\text{Score}_{\text{consensus}} > 0.5$  then
   // Step 2: Label Normalisation
   // (Using ONLY Active Voters)
22   $\text{Map}_{\text{labels}} \leftarrow \emptyset;$ 
23  foreach  $\text{vendor } v \in \text{Active}_{\text{voters}}$  where
     $v.\text{detection} = \text{Malicious}$  do
24     $L_{\text{norm}} \leftarrow \text{Normalize}(v.\text{label});$ 
25     $w_v \leftarrow (v \in T) ? 3 : 1;$ 
26    if  $L_{\text{norm}} \neq \text{Generic}$  then
27       $\text{Map}_{\text{labels}}[L_{\text{norm}}] \leftarrow$ 
         $\text{Map}_{\text{labels}}[L_{\text{norm}}] + w_v;$ 
28   $L_{\text{final}} \leftarrow \arg \max(\text{Map}_{\text{labels}});$ 
   // Step 3: Family Confidence
   // Check
29   $W_{\text{final}} \leftarrow \text{Map}_{\text{labels}}[L_{\text{final}}];$ 
30   $\text{Score}_{\text{family}} \leftarrow W_{\text{final}}/V_{\text{mal}};$ 
31  if  $\text{Score}_{\text{family}} \geq \theta_{\text{fam}}$  then
32     $D \leftarrow D \cup \{(s_i, L_{\text{final}})\};$ 
33  else
34    Keep  $s_i$  as Generic label;
35  else
36    Mark  $s_i$  as Benign(Discard);
37 return  $D$ 

```

licensing agreements. The labelling decisions are based on majority agreement among multiple Antivirus engines, thereby reducing the likelihood of misclassification from any single detection source. This robust process ensures that all samples collected are indeed malware. Table I provides a detailed overview of the antivirus engines utilised.

TABLE I: Antivirus Engine Clusters and Excluded Multi-Engine Scanners

Family / Cluster	Engine Name	Role
BitDefender Cluster	BitDefender	Parent-Engine
Avast/AVG Cluster	Avast	Parent-Engine
Avira Cluster	Avira	Parent-Engine
Kaspersky Cluster	Kaspersky	Parent-Engine
McAfee Cluster	McAfee	Parent-Engine
ESET Cluster	ESET-NOD32	Parent-Engine
TrendMicro Cluster	TrendMicro	Parent-Engine
Sophos Cluster	Sophos	Parent-Engine
CommTouch Cluster	CommTouch	Parent-Engine
Microsoft Cluster	Microsoft	Parent-Engine
Agnitum Cluster	Agnitum	Parent-Engine
BitDefender & Avira	Qihoo 360	Multi-Engine
BitDefender & Sophos	ALYac	Multi-Engine

*All engines not mentioned here are taken as generic.

B. Data Conversion

Before attempting to improve the existing OSL Siamese Networks for malware detection, the practical use of images in malware detection was explained. The transformation of a binary into an image enables the visualisation of its different sections, providing an analyst with an initial orientation for targeted examination [34]. Malware authors can adapt parts of their malware files to change their inherent signature, thereby evading antivirus detection through polymorphism and obfuscation. However, these modifications to the original malware binary do not produce an extreme change in the overall texture of the binary image. Binary images can capture all the nuanced alterations, while keeping the global structure of the malware file.

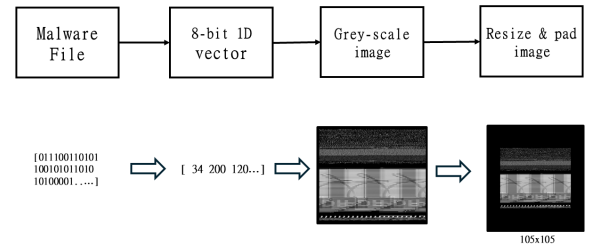


Fig. 2: Data Conversion Framework

As depicted in Figure 2, the malware binary code is read as a one-dimensional (1D) vector of 8-bit unsigned integers, where each byte is mapped to a single pixel. Each pixel's value corresponds to a value between 0 and 255, representing a shade of grey.

The samples are then converted into greyscale images, followed by appropriate rescaling and padding to ensure uniform input dimensions. Given the diverse file sizes of malware binaries, the corresponding binary would produce images of different dimensions, as illustrated in Table II. These inconsistencies in dimensions across different malware binaries would lead to errors in the fully connected layer [35].

TABLE II: Image width based on file sizes

File Size Range (KB)	Image Width (pixels)
< 10 KB	32
10 - 29 KB	64
30 - 59 KB	128
60 - 99 KB	256

To mitigate this issue, bicubic interpolation is used to resize the images, and the image dimensions of 105 x 105 are utilised to adhere to the original paper's preprocessing setup [32]. Algorithm 2 outlines the procedure for converting raw malware samples into binary image representations suitable for ML analysis. Each malware file is first read into memory as *raw_data* and converted into an array of 8-bit integers (*byte_array*), preserving the original byte structure of the file. The image width is determined using a pre-defined function (*GetWidth*), and the height is calculated by dividing the total number of bytes by this width. To ensure uniform image dimensions, the byte array is padded centrally (*padToCenter*), and then reshaped into a two-dimensional array representing the image grid.

This grid is converted into a greyscale image where each byte value corresponds to a pixel intensity. The image is then resized to fixed dimensions (*newHeight*, *newWidth*) to maintain consistency across the dataset. Finally, the processed binary image is saved in PNG format, producing a standardised image dataset that retains the structural patterns of the original malware binaries for further analysis.

Algorithm 2: Convert Malware Samples to Binary Images

Input: Malware samples
Output: Binary image set

```

1 samples ← Malware files;
2 for i ← 0 to length(samples) - 1 do
3   raw_data ← read(samples[i]);
4   byte_array ← ConvertTo8bitInts(raw_data);
5   width ← GetWidth(samples[i]);
6   height ← length(byte_array) / width;
7   padded ← padToCenter(byte_array);
8   image_array ← reshape(padded, (height, width));
9   img ←
    ConvertToGrayscaleImage(image_array);
10  resized ← Resize(img, newHeight, newWidth);
11  binaryImage ← save_Image(resized, "PNG");
```

It can be observed that malware variants within a family exhibit similar textual patterns and are distinct from those of different families. This similarity in textural patterns results from many malware variants within a family being developed based on the original malware [35]. Conversely, the dissimilarity between malware families enhances the Siamese Network's inherent ability to distinguish between similar and dissimilar malware samples.

C. Hash Organisation

Despite malware samples being grouped into the same malware family, not all malware variants within a family share common textual patterns. Different file types, attack vectors, and obfuscation techniques can cause high intra-class variance [32]. To mitigate this variance and strictly adhere to the original paper's dataset configuration, the Average Hash (aHash) algorithm was implemented. The aHash algorithm clusters similar visual samples together, producing a unique fingerprint for each clustering. The proposed Hash system would compute pairwise Hamming distances and group samples whose aHash values differ by no more than a predefined threshold, thereby creating sub-types within a family.

TABLE III: False Match Rate (FPR) and False Reject Rate (FNR) at Different Hamming Distance Thresholds

Threshold	False Match (FPR) [%]	False Reject (FNR) [%]
0	10.59	10.27
3	11.68	0.40
6	19.33	0.00
9	26.26	0.00

This ensures that every entry in the dataset represents a structurally coherent malware variant, regardless of its naming granularity. Inconsistent or noisy samples, which typically fail to form clusters, are automatically discarded. Consequently, generic families in our dataset represent verified high-density visual patterns, challenging the model to generalise across the diverse structural subtypes often grouped under broad vendor tags. As detailed in Table III, lower thresholds enforce stricter similarity, admitting only nearly identical samples, while a higher threshold permits more variation and consequently broader clusters. Notably threshold $T = 3$ provides the best balance between false matches and false rejects, reducing the FNR to 0.40% while keeping the FPR relatively low compared to higher thresholds. Lower thresholds suffer from excessive false rejections, while higher thresholds significantly increase false matches without further reducing the FNR.

Figure 3 further demonstrates that a threshold of 3 exhibits a strong intra-class compactness with most same-variant samples exhibiting Hamming distances $T \leq 2$ while interclass distances are well separated. This threshold ensures that closely related samples are grouped together while still accommodating minor variations. Despite this, a limitation remains regarding distinct malware families that exhibit near-identical structures. These high levels of similarity make them difficult

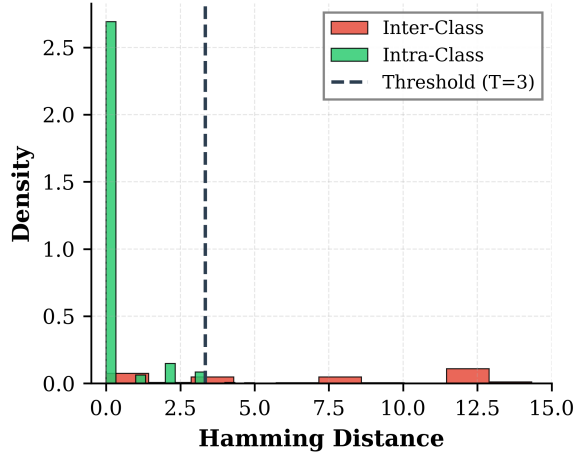


Fig. 3: Validation of aHash Clustering

to differentiate, even when applying the strictest hamming distance thresholds.

D. Baseline Siamese Network

A Siamese Network is a specialised neural network that consists of two sub-networks working in parallel to assess the similarity between input samples. Each sub-network produces an independent output value, and the distance between these outputs is used to determine the similarity between the given inputs. The design and development of the baseline Siamese Network in this study follow the architectural design choices presented in the original paper.

Table IV provides the exact details and values used in the design and development of the baseline Siamese Network. The model takes in a single-channel 105 x 105 image, which is then processed through multiple filters to extract local features. The stride determines the amount of local detail to be captured. The max pooling layer then reduces the spatial dimensionality while keeping the most important features to feed into the next layers. The increase in filters is used to capture more intricate details within the image, allowing for a feature vector to be created that computes the similarity distances. After this, the distance between the feature vectors is calculated and passed through the Sigmoid function to ensure the value is between 0 and 1. Values closer to 1 signify that the pairs of images are similar, while values closer to 0 indicate that there are no similarities between the input images.

V. N-WAY EVALUATION STRATEGY

The dataset was divided into two disjoint sections for evaluation: training and testing. The training set was used to train the model to detect and categorise malware samples based on the visual patterns that had been learned. The testing set was used to test the final trained model on malware samples not included in the training set, simulating the model's

TABLE IV: Baseline Siamese Network Architecture (One Branch)

Layer Type	Filters	Kernel / Pool Size	Output Size
Input	–	–	$1 \times 105 \times 105$
Conv + ReLU	64	10×10 (Stride 1)	$64 \times 96 \times 96$
Max Pooling	–	2×2 (Stride 2)	$64 \times 48 \times 48$
Conv + ReLU	128	7×7 (Stride 1)	$128 \times 42 \times 42$
Max Pooling	–	2×2 (Stride 2)	$128 \times 21 \times 21$
Conv + ReLU	128	4×4 (Stride 1)	$128 \times 18 \times 18$
Max Pooling	–	2×2 (Stride 2)	$128 \times 9 \times 9$
Conv + ReLU	256	4×4 (Stride 1)	$256 \times 6 \times 6$
Flatten	–	–	9216
FC (Sigmoid)	–	Input: 9216	4096
L1 Distance	–	–	4096
FC (Sigmoid)	–	Input: 4096	1

interaction and determining its classification and generalisation capabilities towards unseen malware variants.

To comprehensively assess the robustness and generalisability of the models, this study will utilise five distinct random seeds to shuffle the samples within the test set. For each seed, consequently, the N-way accuracy values are aggregated and averaged. This approach, therefore, ensures that the performance evaluation is not biased by a particular data ordering and thus provides a more reliable estimate of model behaviour across diverse scenarios.

VI. EXPERIMENTAL SETUP

This section describes the various components of the experimental environment and evaluation procedures. The system was equipped with an AMD Ryzen™ 5 9600X CPU, an NVIDIA GeForce RTX™ 4080 16GB GPU, and 32 GB of G.Skill Flare™ X5 RAM.

Experiments were conducted on a custom VirusTotal dataset of greyscale images, curated via the voting system detailed in Section IV-A. The aHash algorithm clustered binary image samples into sub-families of exactly three visually similar images, discarding any groups that failed to meet this threshold. The dataset is publicly available for research purposes.⁷

Our analysis of the dataset reveals significant class imbalance as depicted in both Table V and Table VI, a common challenge in real-world malware environments. Dominant families like *faceliker* provides ample support sets for One-Shot Learning, whereas rare families like *balrok* present extreme Few-Shot scenarios. This distribution validates the need for OSL Siamese Network strategies that are robust against data scarcity, as standard classifiers would likely overfit to the majority classes.

As listed in Table VII, the total training dataset comprises 578 subtypes distributed across 37 distinct malware families,

⁷ <https://github.com/sahil2803/Evaluating-a-Novel-Voting-System-for-Malware-Dataset-Curation-with-Siamese-Networks>

TABLE V: Training set per-family statistics

Family	Subtype Count	Image Count
agentjs	46	138
agenttesla	9	27
babar	1	3
barys	6	18
copak	4	12
dnst	2	6
dacic	28	84
ghol	1	3
loki	17	51
mydoom	1	3
trojan.startpage	3	9
trojan.downldr	11	33
trojan.iframe	59	177
trojan.script	32	96
zussy	1	3
agensla	2	6
clickjack	7	21
cryxos	21	63
kroxxu	3	9
mirai	7	21
redirba	6	18
salgorea	1	3
trojan.adinject	7	21
trojan.agentb	1	3
trojan.apost	2	6
trojan.decode	2	6
faceliker	273	819
trojan.fragtor	8	24
graftor	1	3
trojan.hllp	1	3
trojan.includer	1	3
jiagu	1	3
trojan.malscript	3	9
trojan.popunder	2	6
scriinject	6	18
tofsee	1	3
balrok	1	3

TABLE VI: Testing set per-family statistics

Family	Subtype Count	Image Count
andromeda	4	12
blackhole	1	3
bulz	1	3
cryptominer	19	57
expiro	45	135
ramnit	4	12
razy	1	3
fakejquary	1	3
trojan.js	2	6
trojan.ppatre	1	3
coinminerx	9	27
hidelink	10	30
refresh	2	6
trojan.aabe	5	15
trojan.fundf	1	3
trojan.krucky	1	3
trojan.stone	1	3
trojan.ulise	3	9
trojan.upatre	1	3

consisting of a total of 1,734 images. The testing dataset used to evaluate the model's generalisability comprised 112 subtypes across 19 malware families, totalling 336 images. The discrepancy in sample distribution between the training and testing sets ensures that the model is evaluated on a limited number of previously unseen malware classes. Therefore, simulating the real-world scenario of detecting zero-day malware variants. To guarantee a clear division of malware families and their corresponding subtypes, the malware samples are subsequently arranged into a hierarchical directory structure. The different malware families are represented by the parent

directories. The variants of each malware family are then created by further grouping each malware family sample into several subtype folders based on their aHash value, as covered in Section IV-C. The training and testing datasets are physically separated and arranged.

TABLE VII: Detailed breakdown of dataset distribution.

Dataset	Sub-type	Families	Images	Subtype
Training		37	1,734	578
Testing		19	336	112

A. Baseline Hyperparameters

The hyperparameters shown in Table VIII are taken directly from [32].

TABLE VIII: Training Hyperparameters

Hyperparameter	Value	Description
Mini-batch size	6	Number of image pairs propagated through the network per gradient update.
Learning rate	0.00006	Parameter in the Adam optimiser controlling step size to approach the optimal solution.
Iterations	2000	Total number of iterations in the training process.
N-way	1 to 15	Number of classes selected for testing in each one-shot task.
N-tasks	150	Number of one-shot tasks used for validation.

B. N-Way One-Shot Accuracy

In N-way OSL tasks, the goal is to correctly classify a query image among the N candidates, given only one example image per class that the model has not been trained on. Accuracy is measured across multiple tasks to ensure that the results reflect the actual generalisation ability, rather than random luck [36]. The model is tested on M independent N-way OSL tasks, where Q is the number of correct predictions. The overall accuracy is calculated by Equation 1 [32]:

$$\text{Accuracy} = \left(\frac{100 \times Q}{M} \right) \% \quad (1)$$

Accuracy in this context captures not only the correctness of the prediction but the reliability of the model when dealing with minimal training data. An *anchor image* is selected from a test class, and a *support set* is created consisting of one image from the same class as the query and one image from N different classes. The Siamese network computes similarity scores between the anchor and each support image. If the support image with the highest similarity score belongs to the same class as the anchor, the prediction is considered correct [32, 35].

C. N-way evaluation Setup

The testing process was conducted as described in Section V. Due to the limited number of test classes, the test evaluation only extended to 15 classes, in accordance with the maximum reported in the original study. The details of the parameters that were set for testing are shown in Table IX.

TABLE IX: Experimental setup parameters.

Parameter	Value	Description
N-way	1 to 15	Number of classes chosen for testing in each one-shot task.
N-tasks	150	Number of one-shot tasks used for evaluation.
Seed values	[0, 1, 42, 123, 999]	Random seeds used for reproducibility.

VII. RESULTS AND DISCUSSION

When evaluated in a 15-way setting, the replicated OSL model demonstrated performance comparable to that observed in the original study. While significant discrepancies are observed between the original and replicated models at the 9-way, 10-way, and 13-way settings, the non-monotonic nature of these deviations suggests they are attributable to the stochasticity of random N-way sampling and dataset splits, rather than a systematic model failure. Additionally, as outlined in Section II-B, this difficulty is further compounded by the natural evolution of malware, which introduces greater variance and complexity into the dataset's N-way evaluation tasks.

To rigorously assess the quality of our replication and the robustness of the proposed voting pipeline, we performed a statistical comparison between our results and the original baseline [30] across all N-way settings as detailed in Table X. As an initial measure of methodological fidelity, a Pearson correlation test revealed an exceptionally strong positive relationship ($r=0.968$, $p < 0.001$) between the original and replicated accuracy curves. This near-perfect correlation acts as a statistical proxy for the validity of the data generation pipeline, suggesting that our weighted voting system and aHash Clustering successfully reconstructed class structures sufficiently distinct to reproduce the original performance trends. Despite the lack of original data, the system captured the exact performance degradation trend characteristic of the original OSL model.

While correlation assesses agreement in performance trends, it does not capture systematic differences in absolute accuracy values. To address this, a paired t-test was conducted, confirming a statistically significant difference in mean accuracy ($p=0.0189$). However, quantifying this deviation reveals a Mean Absolute Percentage Error (MAPE) of only 6.4%. This low error magnitude suggests that while exact numeric reproduction is hindered by the use of a novel, modern dataset,

TABLE X: Statistical Comparison: Replicated OSL Model vs. Original OSL Model [32] Performance

Test	Value	P-value	Interpretation
Pearson r	0.968	3.31e-09	Excellent correlation; trends match closely
Paired t-test	-2.681	0.0189	Bias detected
MAPE	6.41%	—	Mean absolute percentage error

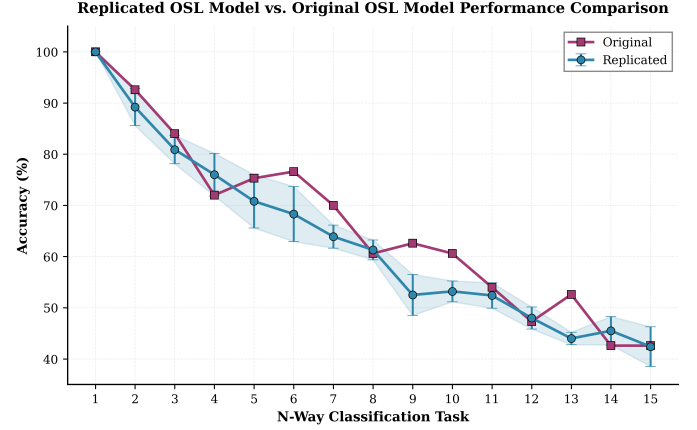


Fig. 4: Comparison of Classification Accuracy

the system successfully captures the fundamental degradation characteristics of the original OSL model. This degradation is attributable to the increased complexity of modern malware as discussed in Section II-B.

However, as detailed in Table XI and illustrated in Figure 4, the replicated model's accuracy begins to mirror the original model, with the smallest differences in results observed at the 11-way, 12-way, 14-way, and 15-way settings, thereby strongly demonstrating that the replicated model aligns with the original paper's findings. This convergence in performance, particularly in higher N-way settings, underscores the robustness of the underlying architectural design despite variations in the training data and preprocessing methodologies. Although the numeric results slightly deviate from those in the original paper, this setup represents the closest possible replication of the authors' methodology using the data from this study.

VIII. LIMITATIONS

A primary limitation was the voting system's inability to completely isolate all malware families, leading to the distributional overlaps described in Section IV. This reduced separability is likely a consequence of concept drift, where the evolving nature of modern malware presents an increasingly challenging landscape for the model to generalise effectively.

IX. CONCLUSION

This study successfully demonstrates that, despite the inherent and significant challenges to replication in malware security research, faithful replication is achievable. By developing a novel malware voting system to generate robust

TABLE XI: Comparison of Average N-way Accuracy between Replicated Baseline and Original study's Siamese Models (%)

Model	N-way														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Original [†]	100.0	92.6	84.0	72.0	75.3	76.6	70.0	60.6	62.6	60.6	54.0	47.3	52.6	42.6	42.6
Replicated	100.0±0.0	89.2±3.63	80.9±2.78	76.0±4.16	70.8±5.21	68.3±5.38	63.9±2.25	61.3±1.93	52.5±4.01	53.2±2.04	52.4±2.48	48.0±2.15	44.0±1.22	45.5±2.78	42.4±3.87

[†] Results taken directly from [32].

labels and a carefully curated dataset, the work replicated a key prior experiment, verifying its central claims. The research concludes that with a thoughtful methodology, aggregating multiple analysis sources to ensure label consistency and mitigate domain-specific obstacles, such as data hostility and evolution—replication in this field can be accomplished. Beyond technical verification, this work reinforces the necessity of automated One-Shot classification as a safety net for the organisational vulnerabilities and human-factor weaknesses identified in recent literature. Since manual defences are often compromised by fragmented policy enforcement or user error, robust automated labelling becomes indispensable. Ultimately, this research provides both a practical tool for consistent dataset curation and empirical evidence that with rigorous methodology, reproducible science can thrive even in the hostile domain of malware security.

ACKNOWLEDGMENT

Finally, I recognise that the opinions, findings, conclusions, or recommendations presented in this work are solely those of the author(s), and no responsibility or liability is assumed by any of the aforementioned individuals or sponsors. This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA. The authors acknowledge that opinions, findings, and conclusions or recommendations expressed here are those of the author(s) and that the above-mentioned sponsor accepts no liability whatsoever in this regard.

REFERENCES

- [1] J.-Y. Kim and S.-B. Cho, "Obfuscated Malware Detection Using Deep Generative Model based on Global/Local Features," *Computers & Security*, vol. 112, p. 102501, 2022.
- [2] E. Gandotra, D. Bansal, and S. Sofat, "Malware Analysis and Classification: A Survey," *Journal of Information Security*, vol. 2014, 2014.
- [3] G. Karat, J. M. Kannimoola, N. Nair, A. Vazhayil, V. Sujadevi, and P. Poornachandran, "CNN-LSTM Hybrid Model for Enhanced Malware Analysis and Detection," *Procedia Computer Science*, vol. 233, pp. 492–503, 2024.
- [4] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu, "Combining File Content and File Relations for Cloud Based Malware Detection," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 222–230.
- [5] S. K. Smmarwar, G. P. Gupta, and S. Kumar, "Android malware detection and identification frameworks by leveraging the machine and deep learning techniques: A comprehensive review," *Telematics and Informatics Reports*, p. 100130, 2024.
- [6] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. IEEE, 2010, pp. 297–300.
- [7] Y. Chai, J. Qiu, L. Yin, L. Zhang, B. B. Gupta, and Z. Tian, "From Data and Model Levels: Improve the Performance of Few-Shot Malware Classification," *IEEE Transactions on Network and service Management*, vol. 19, no. 4, pp. 4248–4261, 2022.
- [8] P. Shijo and A. Salim, "Integrated Static and Dynamic Analysis for Malware Detection," *Procedia Computer Science*, vol. 46, pp. 804–811, 2015.
- [9] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, "Detection of Malicious Code Variants Based on Deep Learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.
- [10] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM computing surveys (CSUR)*, vol. 44, no. 2, pp. 1–42, 2008.
- [11] M. Bailey, J. Oberheide, J. Andersen, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," 2007, proceedings / technical report — add full venue details.
- [12] U.S. Congress, "Computer Fraud and Abuse Act (CFAA), 18 U.S.C." <https://www.law.cornell.edu/uscode/text/18/1030>, 1986, u.S. federal statute; consult official text for current language.
- [13] "Regulation (EU) 2016/679 (General Data Protection Regulation)," <https://eur-lex.europa.eu/eli/reg/2016/679/oj>, 2016, eU regulation on data protection and privacy.
- [14] V. Stodden, F. Leisch, and R. D. Peng, *Implementing reproducible research*. Crc Press Boca Raton, FL, 2014, vol. 546.
- [15] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th international conference on mining software repositories*, 2016, pp. 468–471.
- [16] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Drebin, "Effective and explainable detection

- of android malware in your pocket,” in *Network and distributed system security symposium*, pp. 1–15.
- [17] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, “A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges,” *Future Generation Computer Systems*, vol. 130, pp. 1–18, 2022.
- [18] D. Gibert, C. Mateu, and J. Planes, “The rise of machine learning for detection and classification of malware: Research developments, trends and challenges,” *Journal of Network and Computer Applications*, vol. 153, p. 102526, 2020.
- [19] M. R. Ngandu, G. Mwansa, and Z. Mkabe, “Strengthening cybersecurity in a government department by addressing password management challenges and human factor vulnerabilities,” *Discover Computing*, vol. 28, no. 1, p. 148, 2025.
- [20] G. Mwansa, “Cybersecurity awareness in government institutions: A systematic review of behavioral strategies and policy readiness,” *Journal of Applied Informatics and Computing*, vol. 9, no. 5, pp. 2076–2088, 2025.
- [21] A. Singh, A. Walenstein, and A. Lakhota, “Tracking Concept Drift in Malware Families,” in *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, 2012, pp. 81–92.
- [22] Y. Jiang, G. Li, S. Li, and Y. Guo, “BenchMFC: A benchmark dataset for trustworthy malware family classification under concept drift,” *Computers & Security*, vol. 139, p. 103706, 2024.
- [23] M. G. Gaber, M. Ahmed, and H. Janicke, “Malware Detection with Artificial Intelligence: A Systematic Literature Review,” *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–33, 2024.
- [24] Sudhakar and S. Kumar, “An emerging threat Fileless malware: a survey and research challenges,” *Cybersecurity*, vol. 3, no. 1, p. 1, 2020.
- [25] A. Castellani, S. Schmitt, and S. Squartini, “Real-World Anomaly Detection by Using Digital Twin Systems and Weakly Supervised Learning,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4733–4742, 2020.
- [26] J. Contreras, S. Mostafapour, J. Popp, and T. Bocklitz, “Siamese Networks for Clinically Relevant Bacteria Classification Based on Raman Spectroscopy,” *Molecules*, vol. 29, no. 5, p. 1061, 2024.
- [27] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, “Signature Verification using a Siamese Time Delay Neural Network,” *Advances in neural information processing systems*, vol. 6, 1993.
- [28] L. Fei-Fei, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.
- [29] B. M. Lake, R. R. Salakhutdinov, and J. Tenenbaum, “One-shot learning by inverting a compositional causal process,” *Advances in neural information processing systems*, vol. 26, 2013.
- [30] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, “Siamese Neural Networks for One-shot Image Recognition,” in *ICML deep learning workshop*, vol. 2, no. 1. Lille, 2015, pp. 1–30.
- [31] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: Visualization and automatic classification,” in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.
- [32] S.-C. Hsiao, D.-Y. Kao, Z.-Y. Liu, and R. Tso, “Malware Image Classification Using One-Shot Learning with Siamese Networks,” *Procedia Computer Science*, vol. 159, pp. 1863–1871, 2019.
- [33] “VirusTotal,” <https://www.virustotal.com/>, malware-scanning aggregation service; useful example for multi-engine consensus workflows.
- [34] B. Marais, T. Quertier, and C. Chesneau, “Malware Analysis with Artificial Intelligence and a Particular Attention on Results Interpretability,” in *International Symposium on Distributed Computing and Artificial Intelligence*. Springer, 2021, pp. 43–55.
- [35] J. Zhu, J. Jang-Jaccard, and P. A. Watters, “Multi-Loss Siamese Neural Network With Batch Normalization Layer for Malware Detection,” *IEEE Access*, vol. 8, pp. 171 542–171 550, 2020.
- [36] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, “Matching Networks for One Shot Learning,” *Advances in Neural Information Processing Systems*, vol. 29, 2016.



Sahil Singh received the B.Sc. (Hons.) degree in computer science from Rhodes University, Grahamstown, South Africa, in 2025.

He has been recognized for academic merit and technical innovation. His work includes research in machine learning, with a particular interest in siamese networks and one-shot learning.

Mr. Singh was a recipient of the TeenTech Gold Award in 2017 and was selected to exhibit his work at The Royal Society in London.



Stones Dalitso Chindipha received his BSC honours in Computer Science and Applied Statistics from University of Malawi, MSc and PhD in Computer Science from Rhodes University in 2012, 2018 and 2022 respectively. From 2013 - 2015 he worked as a database administrator for a revenue authority organisation before pursuing his MSc which was immediately followed by PhD. He was appointed lecturer in 2020 before being promoted to Senior lecturer in 2024. His research interests include Malware analysis, cryptography and cybersecurity analytics.