Minimum Viable Malware-image Resolution for ESP32-S3 Edge Triage

Phil Steadman *, P. Jenkins, Rajkumar Singh Rathore *, and Chaminda Hewage

Cardiff School of Technologies, Cardiff Metropolitan University, United Kingdom
E-mail: st20280948@outlook.cardiffmet.ac.uk (P.S.); pjenkins2@cardiffmet.ac.uk (P.J.); rsrathore@cardiffmet.ac.uk (R.S.R.); chewage@cardiffmet.ac.uk (C.H.)
*Corresponding author

***Abstract*—Image-based malware classification can achieve high accuracy on workstation-class hardware; however, the suitability of such methods for Internet of Things (IoT) security depends on whether they can produce useful representations that can fit within the physical limitations (memory, latency and energy consumption) of microcontroller-class devices. Therefore, this work investigates a narrower problem than general malware detection: what is the lowest resolution that malware can have while still being useful for edge triage given the Espressif Systems ESP32-S3 Series System-on-Chip (ESP32-S3) microcontroller? A convolutional neural network was trained and converted using TensorFlow Lite to four different input sizes, and evaluated on-device using local test inputs to keep inference results clear of overhead from network transfer. Subsequent controlled re-runs of the same 9.334 usable maling images have shown that gives a family classification accuracy of 97.30% on a test split of 1.410 samples, which is only slightly lower than the 97.45% family classification accuracy of. When performing a 100-sample serial repeat of an ESP32-S3 for inference, mean live latency was measured at as being 24.23 s and as being 5.68 s, with lower model storage and lower maximum Pseudo Static Random Access Memory (PSRAM) use with the lower resolution model. Results suggest that for periodic edge triage is the minimum useful resolution based on this experimental setup; however, a separate benign/malware detection feasibility study was conducted as an extension to address the practical aspect of the detection scope for future research.**

***Keywords*—Tiny Machine Learning (TinyML), malware images, Systems ESP32-S3 Series System-on-Chip (ESP32-S3), edge triage, Convolutional Neural Networks (CNNs)**

I. Introduction

The increasing number of Internet of Things (IoT) devices connected to the Internet today is projected to surpass 18.8 billion globally by the end of 2024, thereby changing daily life and work [1]. However, despite the benefits of the increasing number of IoT devices connected to the Internet, the introduction of so many connected devices have also created numerous cyber security vulnerabilities, making many IoT networks attractive targets for malware attacks [2]. A number of empirical studies conducted to date have revealed that a large percentage of currently deployed IoT devices are vulnerable to attacks via botnets and other types of cyber-attacks [3]. The constant evolution of malware design has resulted in malware authors employing obfuscation and polymorphism to hide their malicious code, further complicating the ability of traditional signature-based detection mechanisms to keep up with these sophisticated malware attacks [4, 5].

Due to the difficulties faced by traditional signature-based detection systems, many researchers have now turned to Artificial Intelligence (AI) and Machine Learning (ML) as behavior-based, heuristic, and anomaly-oriented methods of conducting malware analysis [6]. A static-analysis method involves converting malware binaries to 2-dimensional binary image representations and developing Convolutional Neural Networks (CNNs) to classify those binary images by malware family or by harmful/benign status [6−8]. The deployment problem that remains unresolved is whether a very low-resource platform, such as the Systems ESP32-S3 Series System-on-Chip (ESP32-S3), Microcontroller Unit (MCU), can classify malware images at an acceptable level of granularity while retaining enough discriminating texture for useful triage.

The ESP32-S3 is an MCU-class platform, so it has much less available Random-Access Memory (RAM) and processing capability than a Raspberry Pi-type Single-Board Computer (SBC). Still, the ESP32-S3 is a viable option for low-cost IoT deployments and offers the ability to perform local pre-filtering of potential malware artefacts on the endpoint device before transferring them to a cloud service or other destination for further analysis. Therefore, an analysis methodology that can be executed practically on an SBC may fail when implemented within the much lower resource limits of a Microcontroller Unit (MCU).

This paper evaluates the effect of reducing malware-image resolution to identify the minimum resolution before malware-family classification performance begins to degrade on an ESP32-S3

Manuscript received March 3, 2026; revised May 23, 2026; accepted June 15, 2026; published September 18, 2026.

deployment. The paper also reports the migration of a Python/TensorFlow training pipeline into an equivalent C++/TensorFlow Lite Micro deployment workflow to measure the trade-offs between input resolution, accuracy, inference latency, energy consumption, model size, and Pseudo-Static Random-Access Memory (PSRAM) usage. The intended use case is periodic malware-image triage or pre-filtering, not continuous real-time endpoint defense (see Fig. 1).

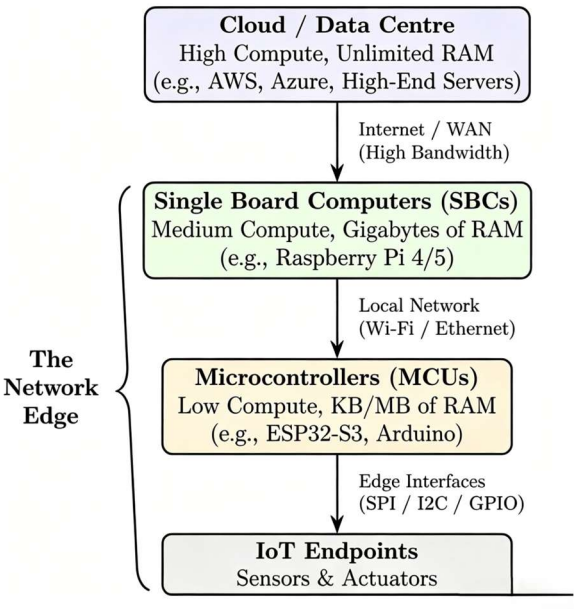

Fig. 1. The hierarchical architecture of IoT networks, highlighting the severe resource constraints encountered when pushing complex AI models down to the microcontroller edge.

Four primary novel contributions are made by this research effort:

- Minimum viable resolution: The identification of 32×32 as the lowest tested resolution for malware image representations that still supports useful family classification performance on the ESP32-S3 for the experimental conditions described.
- Formal resource trade-off: A quantification of the trade-offs in accuracy, latency, energy consumed, model size, and PSRAM consumed by reducing the malware image representation from 64×64 down to 8×8.
- MCU deployment evidence: The demonstrated ability to successfully convert a trained TensorFlow model to inference using TensorFlow Lite Micro via local SD-card based input.
- Operational boundary: Measured latency for the malware images provides the capability to support periodic edge triage and escalation workflows, but not sufficient for providing real-time continuous defence.

The remainder of this paper is organised as follows: Section II provides an overview of related image-based and edge malware classification research; Section III describes the hardware systems and underlying data sources used in this research, the training methodology, the conversion process, and the logging techniques; Section IV presents the results of the experimental comparison between various levels of the trade-offs defined above and the discussed workload analysis; Section V provides the summary of limitations and operational risks associated with the methodologies described; and finally, Section VI provides the conclusions and possible future research directions.

II. Related Work

The integration of cybersecurity and AI with edge computing is a rapidly growing area of research due to the ever-evolving cyber threat landscape and the associated proliferation of vulnerable IoT devices. As such, many evolutionary patterns have been established in terms of how malware detection systems have evolved from traditional signature-based methods to behaviour-based, heuristic and anomaly detection methods [4, 9]. The adoption of ML and Deep learning (DL) methodologies have been fundamental to advancing cybersecurity research for many years [6, 10].

A. Image-based Malware Classification

The methodology for classifying malware binaries by reverting them into two-dimensional images has resulted in an extremely successful subfield of research. Pioneered by Nataraj *et al.* [7], researchers have observed that the grey scale layout of the various malware families created an image with a very unique visual texture. Researchers were able to extract the typical features of an image and run simple classifiers (k-Nearest Neighbours (k-NN)) to classify the sample malware family within the Malimg dataset with 98% accuracy. It was also possible to see these textures with the naked eye.

With the evolution of new AI technologies and their increasing maturity, size and performance, several researchers have applied Convolutional Neural Networks (CNN) to models of malware detection. Kalash *et al.* [8] developed a deep learning CNN architecture to classify images of malware in grey scale, obtaining 98.52% accuracy in classifying images of malware from the Malimg dataset and 99.97% accuracy in classifying images of the Microsoft big 2015 dataset. Likewise, Makandar and Patrot [17] demonstrated success with Artificial Neural Networks (ANN) for classifying malware based on their texture. In addition, Vasan *et al.* [13] created an ensemble approach, which consisted of multiple CNN architectures for extracting higher-quality features from the images of unpacked and packed samples of malware. This method yielded over 99% accuracy from unpacked malware and 98% accuracy from packed malware.

B. AI-enabled Security in IoT and Edge Computing

As IoT threats continue to grow, researchers have been focusing on modifying the highly accurate, yet resource-demanding deep learning models for use in resource-constrained edge devices like ESP32's [18]. Traditional deep learning models impose large requirements on Central Processing Unit (CPU) and memory, which is not available on the edge of the network, Artificial Intelligence at the Edge (EdgeAI) [19].

Research has been conducted to create lightweight classifiers for specifying a model's architecture to reduce its resource overhead in terms of CPU and memory while maintaining an adequate performance level. Su *et al.* [11] designed a lightweight CNN classifier for classifying IoT malware, specifically Distributed Denial-of-Service (DDoS) bots such as Mirai, producing a 94.0% success rate. Roseline *et al.* [12] created a three-layer vision-based model that uses images of resolution with a low computational cost and a success rate for classifying malware of 97%, demonstrating the feasibility of running AI on a Single Board Computer (SBC) such as the Raspberry Pi. Yuan *et al.* [15] took this one step further than the earlier works by transforming malware binary files into multidimensional Markov images and classifying them with a lightweight CNN that uses Depthwise Convolutions to substantially reduce the size of the resulting model versus the standard architectures such as Visual Geometry Group 16 (VGG16) while achieving a 95% accuracy. It should also be noted that VGG16 models are very large compared to models like EfficientNet family.

In addition to developing lightweight models, recent research has explored using distributed and federated architectures to address resource limitations. Bhandari *et al.* [16] presented a distributed deep neural network Music Information Retrieval (MIR) middleware that allows the deployment of inference models on edge devices such as Raspberry Pi and NVIDIA Jetson to minimize CPU and power consumption overheads. Rey *et al.* [20] investigated federated learning approaches to train models to classify malware found in IoT devices while allowing them to protect user privacy through decentralized training of the data used to learn and train the model.

C. Challenges: Data Imbalance and Architectural Constraints

An ongoing challenge in developing Deep Learning Models for malware detection using CNNs is the data imbalance found in some malware datasets, where specific malware families have substantially more samples than others. Hemalatha *et al.* [14] sought to mitigate the negative effects of data imbalance by developing a DenseNet-based model adopting a reweighted class balanced loss function, producing a 98.23% accuracy when tested on the extremely imbalanced Maling malware dataset. Similarly, Yue and Wang [21] employed a similar strategy to develop a new class-weighting parameter for training neural-network models to counteract the effects of data imbalance found in the currently utilized datasets and ensure that lesser represented classes are appropriately weighted without consisting of high computational overheads during inference.

D. Summary of Findings and Gaps

Martin *et al*. [22] and Steadman *et al.* [23] support the notion that AI-based malware detection can be successfully performed using Single-Board Computers (SBCs) (e.g. the Raspberry Pi), but does not directly correlate to the (Microcontroller Unit (MCU) deployment question. Microcontrollers (e.g., the ESP32-S3) generally only provide Kilobyte (KB) to a few Megabyte (MB) of available RAM for operation; whereas, SBCs provide Gigabyte (GB) of memory and provide a richer Operating System experience. As shown in Table I, most of the high accuracy image-based malware literature has been produced for use on personal computers, cloud-based services, or non-extreme edge platforms. The gap being addressed through this revision is the lack of a systematic, empirical resolution threshold for malware triage by using images located under the MCU constraints. Controlled reruns of the Malimg set, a separate benign/malware trial, and a lightweight INT8 depthwise base are added and the thesis remains constrained to the resolution threshold study of the MCU.

TABLE I. COMPARISON OF PUBLISHED WORKS ON IMAGE-BASED AND IOT MALWARE DETECTION

Reference	Proposed Method/Model	Target Platform/Domain	Dataset(S) Used	Reported Accuracy
[7]	Grayscale Image Processing + k-NN	General PC/Server	Malimg	98.00%
[8]	Deep Convolutional Neural Network	General PC/Server	Malimg, BIG 2015	98.52%
[11]	Lightweight CNN (One-channel grayscale)	IoT Systems	DDoS Malware/Goodware	94%
[12]	3-Layer Lightweight CNN (32 × 32 images)	SBCs (e.g. aspberry Pi)	Malimg, BIG 2015	97%
[13]	Ensemble of CNN Architectures (IMCEC)	General PC/Server	Malimg	99%
[14]	DenseNet with Class-Balanced Loss	General PC/Server	Malimg, MaleVis	98.23%
[15]	LCNN with Markov Images	IoT Systems	Microsoft, IoT Datasets	99.35%
[16]	Distributed Deep Neural Network	SBCs (Raspberry Pi, Jetson)	Heterogeneous IoT Malware	93%
Proposed Work	Resolution-threshold CNN study (TensorFlow Lite Micro)	MCU (ESP32-S3, 2MB PSRAM)	Malimg family classification; separate benign/malware feasibility split	97.3% controlled 32 × 32 Malimg test accuracy 98.85% separate binary feasibility accuracy

III. Methodology

The purpose of this investigation is to assess the ability to deploy image-based malware classification models on extreme edge devices where there are significant constraints to CPU capabilities and memory state. This section reviews the hardware utilised in this project, the overall model development pipeline, the process of converting the malware image from the original binary form to model-execution-compatible format, as well as the data collection processes during the testing and evaluation phases.

A. Hardware Utilisation and Architecture

The board chosen for this study is the Seeed Studio XIAO ESP32-S3 Microcontroller. The ESP32-S3 has a processor with a 240 MHz dual core architecture, along with 2 MB of Pseudo-Static Random-Access Memory (PSRAM). In order to accommodate the TensorFlow Lite Micro version of the malware detection model and maintain its efficiency, PSRAM must be present in order to house the Tensor Arena, which manages the Tensor allocation dynamically as it allocates and deallocates portions of memory during model inference.

Initial testing of the prototypes utilised the ESP32-S3 Wi-Fi module to dynamically retrieve the malware images from an Amazon Web Services (AWS) S3 bucket via Hypertext Transfer Protocol (HTTP) over Transport Layer Security (TLS), while also storing the TensorFlow Lite model directly to the internal flash of the microcontroller. Ultimately, the Wi-Fi module in this configuration proved to be too resource demanding in terms of power draw and latency for this type of application since the added overhead made it impossible to definitively isolate the power draw and latency performance specifics related to the CNN itself. It was also difficult to isolate the latency from Wi-Fi and the Internet, compared with a wired connection directly to the media.

As a result of this limitation, the overall design architecture utilised in this study was modified to include a dedicated SD Card Module attached to the XIAO ESP32-S3 via a Serial Peripheral Interface (SPI). In this setup, the inference is run exclusively using locally stored data, thus, power and latency metrics associated with inference are not affected by network activity and Wi-Fi interference. The use of SPI as a method of transferring data from the SD Card Module to the ESP32-S3 created engineering challenges, as the default SPI channel pins were used for the PSRAM, this resulted in a configuration clash, and custom pinouts were required.

The data logger consisted of a Universal Serial Bus Type-C (USB-C) to USB-C power meter (Fig. 2). This meter enabled the continuous tracking of the wattage consumed by the CNN while running inference (Fig. 3). Overall, the power draw measured during the testing phases remained relatively constant while in a state of active testing. Fig. 3 provides the circuit diagram representing XIAO ESP32-S3 serially connected to the SD Card Interface. Later builds used the ESP32-S3 Sense, which includes a built-in Secure Digital (SD)-card interface, reducing the risk of wiring error and loose external connections. Fig. 4 shows the binary output of the images.

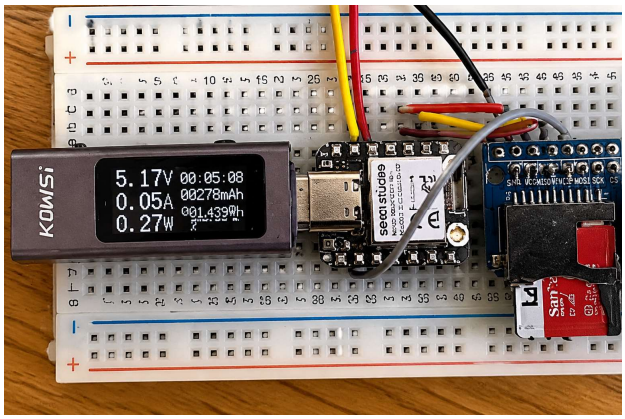

Fig. 2. USB-C power meter used for manual recording of power draw during inference.

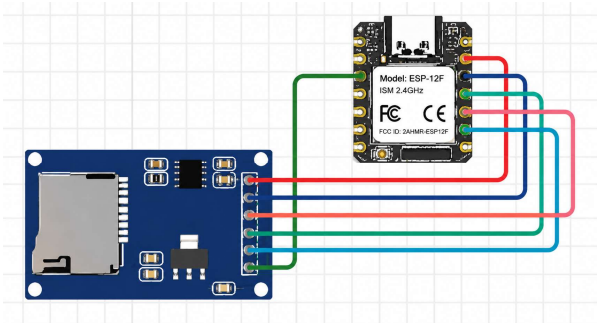

Fig. 3. Circuit diagram of the ESP32-S3 and SD breakout board connected via Serial Peripheral Interface (SPI).

0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
0x8c, 0x00, 0x00, 0x00, 0xe4, 0x00, 0x00, 0x00, 0xf0, 0xde, 0x45, 0x00,
0x00, 0xdf, 0x45, 0x00, 0x3c, 0xeb, 0x45, 0x00, 0x03, 0x00, 0x00, 0x00,
0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xfe, 0x1f, 0xba, 0xff,
0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x3c, 0x00, 0x00, 0x00,
0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f,
0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74, 0x00, 0x01, 0x00, 0x00, 0x00,
0x04, 0x00, 0x00, 0x00, 0x90, 0xff, 0xff, 0xff, 0x15, 0x00, 0x00, 0x00,
0x04, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x6f, 0x75, 0x74, 0x70

Fig. 4. An example of malware image as hexadecimal (Snippet).

B. Modelling Training and Data Pipeline

A custom Convolutional Neural Network (CNN) was created and trained using a high-performance Ubuntu server. A custom architecture was chosen over using pre-trained models like Mobile Network (MobileNet) or Residual Network (ResNet) to allow direct control of the layer structure for extreme compression and optimization. As pre-trained models are typically weighted towards standard ImageNet category weights (animals, and vehicles), these models would be unnecessarily large and bloated for a custom task of identifying grayscale malware textures.

The experiments for the controlled revision study used a 70/15/15 train/validation/test split that was reproducible, with a random seed value of 1337. The overall classification problem was set up to include 6.523 training samples, 1401 validation samples, and 1.410 test samples using a custom Convolutional Neural Network (CNN) model. For the CNN, a series of 3 convolutional blocks with Batch Normalisation, Max-Pooling (when resolution allowed), Global Average Pooling, Dropout and Softmax output layer was used to construct a model that was trained using Adam as a training algorithm, Sparse Categorical Cross-Entropy as the loss function, class weight balancing, batch size of 128, early stopping, and a maximum of 60 epochs to train the model. The same data split and preprocessing that was performed to create the training and validation datasets was consistent for the 8.8, 16.16, 32.32, and 64.64 resolution datasets so that the differences reported in the results are strictly representative of the resolution size of the images and the costs associated with deploying them to the edge.

The CNN model that was trained to classify the samples from the Malimg dataset [24], was also used to carry out the local audit of the training samples using the methodology defined in the flowchart of Fig. 5 which guided the research from training through TensorFlow Lite conversion and edge deployment.

The local audit identified 9.334 usable Portable Network Graphics (PNG) files in the Malimg malware corpus. Earlier notebook records reported 9.336 and 9.339 files, so the revised manuscript uses the reproducible count of usable images rather than the earlier notebook totals. The scope of the primary evidence is therefore defined by the Malimg dataset: it supports malware-family classification, but it does not provide a direct basis for measuring benign-versus-malicious false positives in a production environment. The resolution threshold identified in this analysis should therefore be interpreted as the minimum viable image representation for separating Malimg malware-family memberships under MCU constraints. A separate experiment on the feasibility of detection of benign samples and malware samples was performed using the Malware Benign Image Classification Dataset [25]; the results of this are separate from the Malimg resolution threshold analysis. This dataset was also used by Shahnawaz *et al.* [26], who derived greyscale images from runtime Application Programming Interface (API)-call argument behaviour and reported 98.36% average classification accuracy including perfect reported metrics for the Downloader class but weaker recall for classes such as Adware, Backdoor, and Spyware A significant technical challenge arose during the deployment phase. The standard PNG decoding libraries used for the ESP32 could not interpret the images from the test set accurately into RAM, even though the images themselves were all valid Portable Network Graphics (PNGs) (confirmed using ImageMagick tool on Linux). Therefore, an offline pre-processing stage was developed on the Ubuntu server to create a better way of ensuring proper loading of image data into the Edge Device. All PNG images were converted to raw hexadecimal binary format (.bin) files and organized into directories according to resolution. This drastically reduced the amount of computational cost on the ESP32 by eliminating the overhead of converting the PNG files to tensor format during real-time processing. Fig. 6 shows examples of malware images from the Malimg dataset in the form of grayscale images generated through their binary structures.

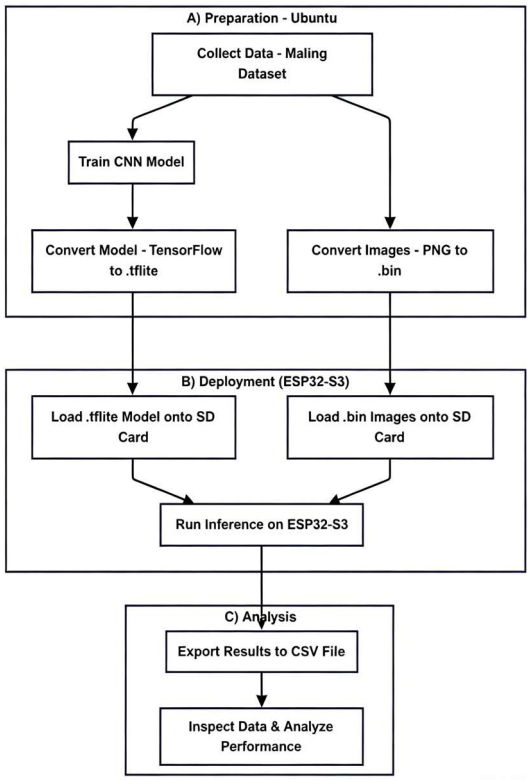

Fig. 5. Flowchart of the proposed methodology, illustrating the pipeline from model training on the Ubuntu server to TensorFlow Lite conversion and deployment on the ESP32-S3.

The CNN model was converted from TensorFlow format to TensorFlow Lite format (TFLite) following the completion of training. The conversion from TensorFlow to TFLite resulted in a decrease of file size from 12.6 MB to 4.2 MB, thus shortening the amount of memory required for the model's execution, without requiring significant quantization. The resulting *.tflite file containing the trained CNN model was stored onto the SD Card along with its associated .bin image files.

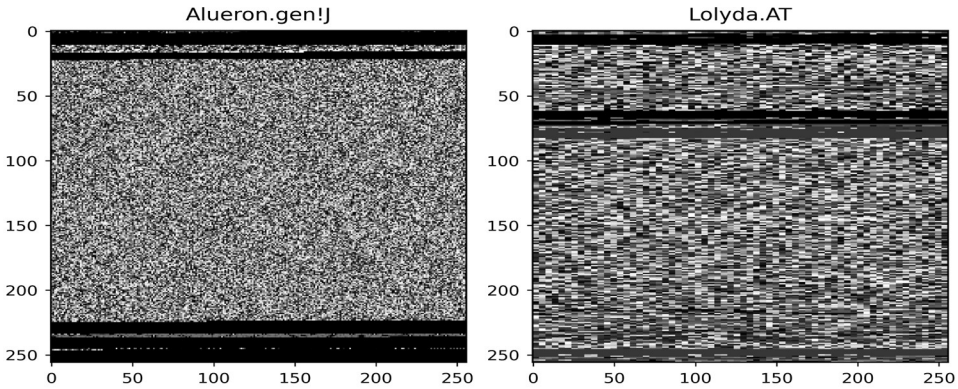

Fig. 6. Two examples of malware from the Malimg dataset, visualised as grayscale images based on their binary structure.

Notably, for each resolution, the CNN model needed to be re-trained and re-exported for input resolutions. For instance, a CNN model trained with a resolution of 64 × 6464 × 64 pixels will not accept a 32 × 3232 × 32-pixel input due to the difference in size between the input (4096 bytes) and the corresponding data buffer (1024 bytes). Therefore, four distinct models were trained for each required pixel resolution (64 × 64, 32 × 32, 16 × 1616 × 16, and 8 × 8), then each trained model was converted to TFLite and stored onto the SD Card.

The separate-model design avoids conflating resolution scaling with runtime resizing on the microcontroller. Each model therefore represents a complete training and deployment point on the resource trade-off curve.

The revised experiments also included MobileNetV2-small and a depthwise-separable CNN as baselines, with TFLite post-training quantisation used as an additional deployment check. These baselines are not intended to replace the ESP32-S3 measurements; rather, they evaluate whether established lightweight CNN families can perform similarly to the custom CNN under the same split and preprocessing.

Furthermore, the Malimg dataset is severely unbalanced, as shown in Fig. 7. Some previous studies have indicated it may be possible to somewhat reduce the negative effect of this severe class imbalance on Small Board Computers (SBCs), like those in Raspberry Pi, during classification [27]. However, due to the highly limited capabilities of the ESP32-S3, the device could not generalise appropriately under such conditions. Additionally, the class_weights parameter was used during training on the Ubuntu Server. As a result, the weighted model appropriately assigned more predictive power to underrepresented malware families, thereby improving its performance.

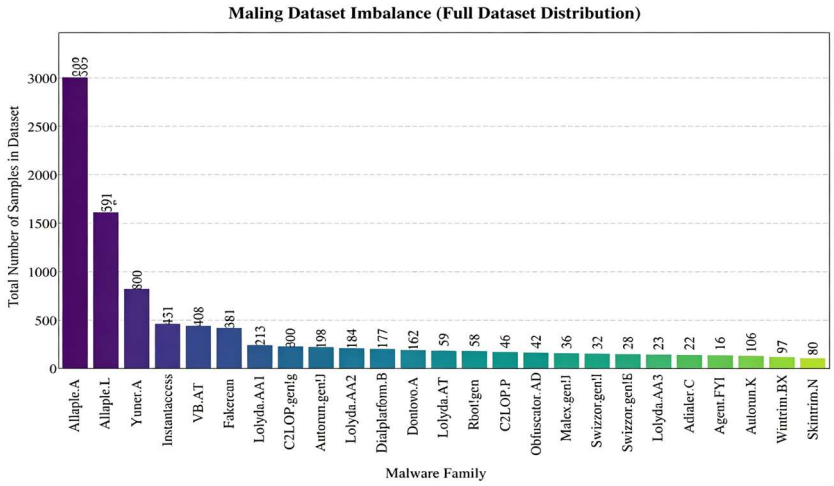

Fig. 7. Distribution graph of the Malimg dataset, showing the extreme class imbalance in the 25 malware families.

C. Experimental Setup and Metrics Logging

The ESP32 device was programmed to run in a continuous loop evaluation mode. For each of the four models trained on resolutions, the ESP32 would load in a set of binary image tensors (that were pre-converted from an image format) from the SD storage card, run batch inference on each of those images, and store logging of every classification in an append-only Comma-Separated Values (CSV) format. The PSRAM allocated to the TensorFlow Lite Micro interpreter for the tensor arena also provided a location in memory to write the inference output for each image using the SD storage card preventing the CSV log of outputs from being counted against the tensor memory used by the models.

Results obtained from the live ESP32-S3 device provide deployment evidence for latency, model size, and peak PSRAM utilisation during live inference. The live-device results are separated from the controlled offline dataset

results; the live ESP32-S3 percentages shown in Table are therefore interpreted as deployment-sample accuracy, not as the primary statistical accuracy claim. A second serial repeat of the ESP32-S3 Node ID:01 device captured 100 valid inference rows, comprising 25 valid inferences for each of the resolutions at which the models were trained (8×8, 16×16, 32×32, 64×64). All rows displayed success with the SD-card load, TensorFlow Lite Micro inference success and CSV writing status.

TABLE II. DATA POINTS RECORDED FOR EACH IMAGE CLASSIFICATION

Data Point	Description
Image Resolution	The Resolution of the input image (e.g., 64 × 64).
Image Filename	The Original filename of the image.
True Malware Class	The Groundtruth Class from the File name.
Inference Time (ms)	The Time for Classification in ms.
Model Size (KB)	The. tflite File Size.
Peak RAM (KB)	The Peak Memory Used by the TFLite Interpreter.
Is Top-1 Correct	A Flag (1/0) for Standard Accuracy.
Top-1 Predicted Class	The Malware Class with the Highest Confidence.
Top-1 Confidence Score	The Probability of the Top-1 Class (Raw).
Is Top-3 Correct	A Flag (1/0) if True class within Top 3 predictions.

At the end of each batch, the program paused so the onboard Light-Emitting Diode (LED) could flash as an operator alert. The external Universal Serial Bus Type-C (USB-C) power meter described in sub-Section III was then read manually to derive the average power/wattage for each classification batch. The resolved power/wattage was recorded using the Serial Terminal on the ESP32 and appended to the CSV file. Observations revealed an approximate consistency of power/wattage across all iterations, with active inference resulting in approximately 0.65 watts and idle operation resulting in approximately 0.27 watts.

During the classification of each malware image, the relevant data points listed in Table II were recorded to provide a detailed view of the model's predictive performance-related constraints.

D. Power Consumption Measurement

In order to measure power consumption during the inference process for the device, a USB-C power meter was placed in line between the power supply and ESP32-S3 Microcontroller as seen in Fig. 2. This allowed for real-time Voltage, Current, and Power/Wattage readings to be derived from the Power Meter measurements of the device's Power/Wattage draw. The device's active wattage measurements were taken for each resolution Model during inference as well as the device's baseline wattage measurement (Idle State) to allow for an average wattage difference between the Inference and Baseline tests. The average wattage measured for each inference was derived by multiplying the active power draw by the inference latency.

The revised analysis distinguishes average active power from energy per inference and reports the measurement limitations of the manual USB-C meter. Because the meter was read manually at batch boundaries, the power data should be interpreted as batch-level energy evidence rather than high-frequency power profiling.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, the overall performance and accuracy results as recorded during both stages (offline evaluation during model training on the Ubuntu server and live edge-inference using ESP32-S3 Microcontroller), will be analysed. Comparison of both evaluation phases provides a comprehensive evaluation of the compromise(s) needed to generate extreme edge-deployment classification models.

A. Offline Training and Evaluation Metrics

The theoretical classification ability of the TensorFlow Lite models was tested before deploying the models for use on the ESP32-S3. Testing of the TensorFlow Lite models for their theoretical classification ability was performed during the training phase on a high-performance Ubuntu server using standard machine learning evaluation metrics. The Adam optimiser was selected for this evaluation due to previous studies indicating that it converged faster and produced less loss when compared to other optimisers when classifying malware images [28].

The Malimg dataset is heavily imbalanced; therefore, there is a need to use more than just global accuracy when evaluating model performance. Precision, Recall and F1-Score (both Macro and Weighted) were calculated to ensure that the models did not overfit to the majority classes.

TABLE III. CONTROLLED OFFLINE MALIMG TEST METRICS FOR THE CUSTOM CNN

Metric	64 × 64	32 × 32	16 × 16	8 × 8
Test samples (*n*)	1.410	1.410	1.410	1.410
Top-1 Accuracy	97.45%	97.30%	88.01%	78.79%
Top-3 Accuracy	99.79%	99.65%	97.80%	94.68%
Test Loss	0.0923	0.1048	0.4284	0.7922
Full INT8 TFLite Size	106.6 KB	106.6 KB	106.6 KB	106.4 KB

The data in Table III show that the controlled offline custom CNN maintains a high level of family classification accuracy at 32×32 while substantially reducing the representation size relative to 64×64. The 32×32 model produced 97.30% Top-1 and 99.65% Top-3 accuracy on the 1.410-sample test split (just behind the 64×64 model's 97.45% Top-1). In contrast, as the resolution decreased to 16×16 the Top-1 accuracy dropped significantly (to 88.01%) and again at 8×8 (to 78.79%). Since the results shown in this controlled server-based test indicate that aggressive down-sampling can result in the removal of useful family-level textural features from the image below the 32×32 resolution, this offers further evidence to support the resolution threshold described in the paper. Additionally, the measurements for the ESP32-S3 representative for these controlled results

are given in Table V and provide an associated device cost for the defined threshold. Fig. 8 plots Top-1 and Top-3 test accuracy under four input resolutions, showing obvious accuracy drop below 32 × 32.

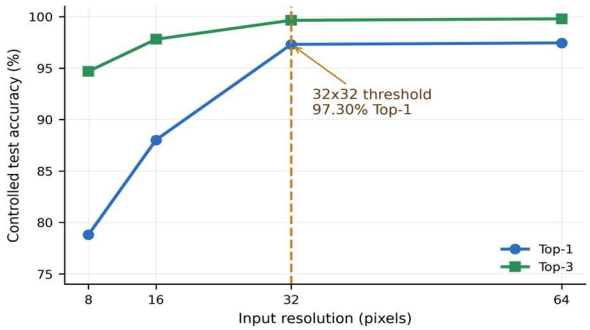

Fig. 8. Controlled malimg test accuracy by input resolution.

The revision audit also verified that old recovered Keras and TFLite artefacts existing in the project repository do not produce the reported accuracy results when run against the revised manifest. When audited the accuracy of all the recovered models provided approximately 1.35% Top-1 accuracy on the 1.410-sample split of the test. As the class listing is verified to be identical to the audited manifest, these recovered artefacts are therefore considered to be provenance-broken. This means that they will not be included as part of the evidence establishing the revised paper's accuracy. Consequently, the revised accuracy results rely entirely on controlled reruns where each of the following has been documented: manifests, seed randomness, preprocessing, model summary, confusion matrix, and export settings for TFLite.

Fig. 9 presents per-class F1-Scores of the 32 × 32 model, where minority malware families suffer lower performance due to data imbalance.

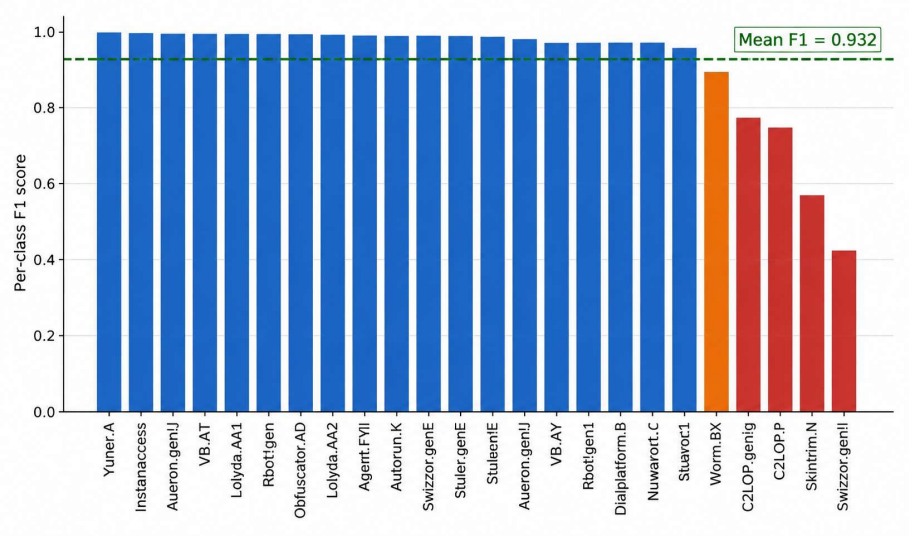

Fig. 9. Per-class F1-Scores for the controlled 32 × 32 Malimg custom CNN.

TABLE IV. BASELINE AND BENIGN/MALWARE FEASIBILITY RESULTS FROM CONTROLLED RERUNS

Dataset/Task	Model	Resolution	Test Accuracy (%)	INT8 Size (KB)
Malimg family	Custom CNN	32 × 32	97.30	106.6
Malimg family	MobileNetV2-0.35	32 × 32	31.42	640.4
Malimg family	Depthwise CNN	32 × 32	1.35	30.1
Benign/malware	Custom CNN	32 × 32	97.89	103.1
Benign/malware	Depthwise CNN	32 × 32	98.43	27.4
Benign/malware	Depthwise CNN	64 × 64	98.85	27.4

Table IV addresses benchmarking and dataset-scope concerns. The MobileNetV2-0.35 baseline provides a direct comparison with a named modern lightweight CNN family; in this controlled 32 × 32 Malimg run it reached 31.42% Top-1 accuracy with a 640.4 KB full-INT8 TFLite model, substantially below the custom CNN's 97.30% accuracy and larger than the custom INT8 artefact. The depthwise-separable baseline was much smaller after INT8 conversion, but it also did not learn the 25-class Malimg family task at 32 × 32. Conversely, the same depthwise approach performed strongly on the separate benign/malware image corpus, reaching 98.43% at 32 × 32 and 98.85% at 64 × 64 on a 3310-sample test split. These binary results are promising but are not merged with the Malimg family-classification claim: they are reported as a separate feasibility check showing that benign-versus-malicious evaluation is possible and must be developed as a distinct experimental track.

B. Edge Computational and Resource Costs Analysis

Resource constraints of embedded systems such as the ESP32-S3 are often deemed to be of greater importance than pure prediction accuracy. The mean computational footprint for each of the four model scales during inference is tabulated in Table V.

Perhaps the most relevant outcome is the peak RAM usage reduction in Table V. The baseline (largest) model using 64×64 required 643.38 KB of PSRAM for initialising its Tensor Arena, whereas the 32×32 model required only 163.38 KB (an overall improvement of 3.94). Fig. 10 visually compares accuracy and storage cost of all baseline and binary classification models.

TABLE V. EDGE INFERENCE COMPUTATIONAL AND RESOURCE FOOTPRINT

Metric	64 × 64	32 × 32	16 × 16	8 × 8
Inference Latency (L_t) [ms]	24.234.76+/−0.17	5.680.52+/−0.20	1.248.00+/−0.00	236.44+/−0.20
Model Size (M_s) [KB]	4,474.95	1,402.95	634.95	442.95
Peak PSRAM Usage (M_{ram}) [KB]	643.38	163.38	43.38	13.38
Energy consumed per inference (joules)	15.75	3.69	0.81	0.15

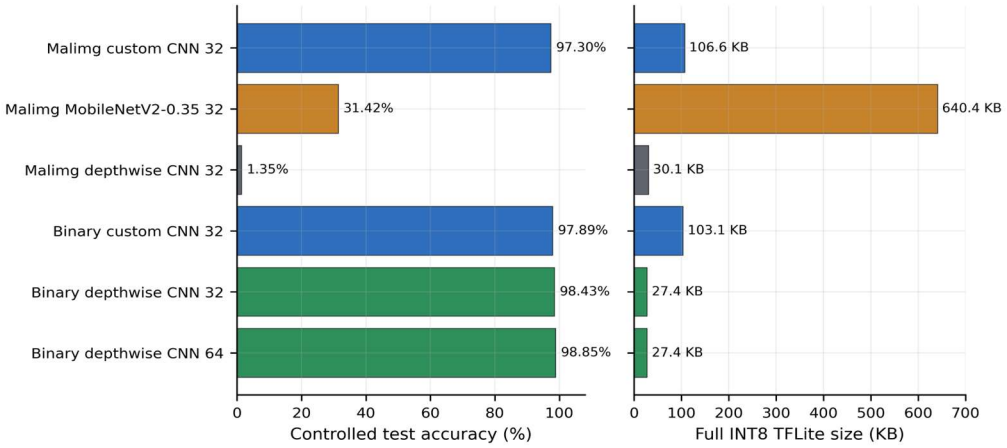

Fig. 10. Controlled baseline and benign/malware feasibility comparison.

This optimisation is extremely critical for the implementation of edge security as it will free up the majority of the microcontroller's RAM for performing other mission-critical tasks in parallel with properly performing the primary function of maintaining device operation while, for example, running an operation to perform security with respect to maintaining the device's network stack, performing O/T/A updates and managing and reporting cryptographic configuration, and therefore, eliminating the necessity of using higher-priced MCUs with larger amounts of RAM available.

Fig. 11 illustrates the downward trend of latency, PSRAM occupation and energy as resolution decreases.

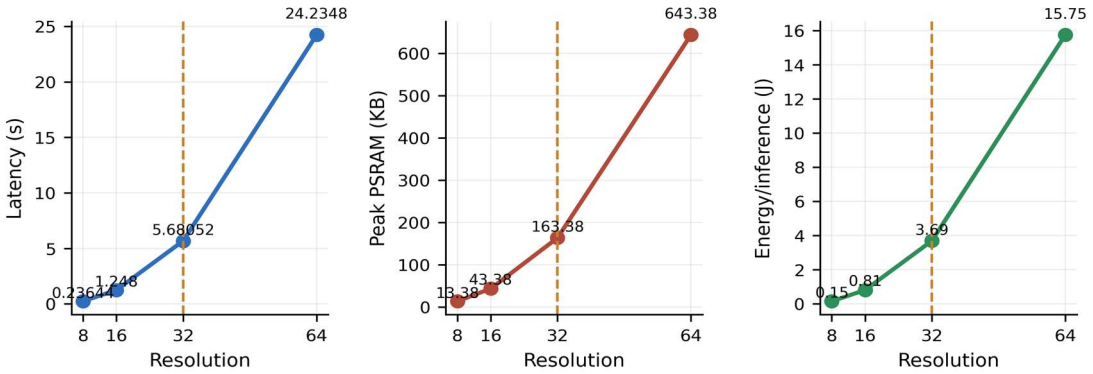

Fig. 11. ESP32-S3 resource cost by malware-image resolution.

C. Power and Energy Consumption

The power consumption of the ESP32-S3 during its operation was measured to be very stable overall through the inference process, regardless of the model resolution being processed. In its idle state, the power usage of the ESP32-S3 was measured as 0.27 Watts. During the active phase of inference, the average power consumption increased to a steady 0.65 Watts. However, since the peak amount of power consumed is common throughout testing with each model resolution, it is important to note that, for edge devices with battery power and a limited number of resources, the true impact on battery life is the total amount of energy consumed for each inference. The amount of energy consumed (E) can be calculated using the following equation: $E = P_{active} \times L_t$.

The 64×64 base model exhibits significant energy consumption (15.75 J) per inference run due to an average

processing latency of 24.23 s; however, by reducing to an optimal size of 32×32, the energy consumption is dropped by approximately 76.62% (3.69 J per scan) based on the same 0.65 W active-power assumption. Besides the implications for keeping energy consumption low enough for RAM, the ability to significantly reduce energy consumption through aggressive scaling of resolution will also help to extend the longevity of IoT-enabled devices that have been installed in the field.

The controlled offline test provides an estimate of statistical accuracy; however, the live ESP32-S3 execution also provides evidence of deployment as both converted models execute on the target board and produce logged outputs. For the repeat deployment sample, Top-1 Accuracy (Acc_1) is used to evaluate performance as shown in Table VI.

$$\mathrm{Acc}_1 = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\,(y_i = \hat{y}_i)$$

where N is the total number of test samples, y_i is the actual class label for test sample i, $\hat{y}_i$ is the predicted class label, and $\mathbb{I}(\cdot)$ is the indicator function.

The live accuracy sample from the ESP32-S3 shown in Table VI provides separate evidence that the converted models executed end-to-end on the target board, but it is intentionally separated from the controlled-accuracy evidence in Table III and Fig. 8. The live sample provides repeatability with latency statistics and increases the amount of on-device deployment evidence, yet it remains too small to replace the 1.410-record controlled test split as the statistical source for accuracy.

TABLE VI. LIVE ESP32-S3 TOP-1 DEPLOYMENT SAMPLE ACCURACY

Metrics	64 × 64	32 × 32	16 × 16	8 × 8
Deployment Samples (N)	25	25	25	25
Valid Serial Rows	25	25	25	25
Top-1 Accuracy (%) (Acc_1)	76	68	48	28

In the following subsections, the results of the live Top-1 deployment sample and the controlled offline Top-3 family-ranking evidence will be interpreted to meet different thresholds of resolution.

D. Resolution Thresholds for Top-1 Accuracy

The Acc_1 data provide evidence of a well-defined degradation threshold associated with a decrease in image resolution. The highest performing classification result for family-classification from the ESP32-S3 belongs to the 64×64 model; however, due to high latency and greater PSRAM costs, it cannot be considered an MCU deployment candidate. The lowest resolution model tested with respect to Top-1 acceptable accuracy combined with a significant reduction in energy consumption, PSRAM usage, and latency is the 32×32 model. Both the 16×16 and 8×8 models have lower computational costs but the texture that differentiates the family labels is beyond the level of usefulness to allow for reliable primary family classification.

E. Family-ranking Triage Using Top-3 Accuracy

Top-3 accuracy is not useful with regard to family ranking; its purpose is only to provide a general means to review accuracy results with respect to the family ranking of a given malware detection. Therefore, since all malware is derived from the same interface and is family specific in the Malimg dataset, the controlled results do not reflect a true benign versus malicious determination of the test-sample outputs.

For the 32×32 model, the controlled Top-3 accuracy result suggests that the family associated with the target file may still be contained among the top predicted candidates; even in the event that Top-1's first prediction is incorrect; therefore, Top-3 accuracy may provide a supporting mechanism for escalated workflow assessments of dubious files.

This conclusion must be carefully separated from the 16×16 model where the difference in methodology is clearly illustrated. The 16×16 model has a smaller latency; however, given lower confidence levels associated with the weak Top-1 results from both controlled and live accuracy assessments, it cannot be employed as a primary classifier. As such, Top-3 accuracy would be of limited value, as this model must be treated as producing low-confidence ranking in a triaged output rather than a singular automated determination of the primary malware family.

V. LIMITATIONS TO FINDINGS AND OPERATIONAL RISKS

A number of limitations have been identified that impact the outcome of the results presented in this study. First, because the experiment was conducted with the use of the Malimg dataset, the data is inherently imbalanced and designed to provide classification by a hierarchy of malwares through their corresponding familial classification system. There are no benign (non-malicious) applications represented in the present dataset, thus limiting the ability to define a realistic false positive outcome for a benign-vs-malicious deployment.

Second, an increase in operational impact exists in relation to a single error first on classification as it is present in the operationally more severe form of a false negative compared with family classification. In other words, a missed malicious artefact would remain within device/network containment and could potentially go undetected, whereas the detection of a false positive would incur only CPU and user time associated with gateway or analyst resources.

Third, the inference result from the ESP32-S3 should not be used as an autonomous malware verdict without confirmation from higher-reliability measures.

Future directions of the study should evaluate the impact of adversarial evasion on malware-image performance, including packing, obfuscation, padding, adversarial byte manipulation, and historical benchmark samples compared with current malware activity. In addition, operational use of malicious-versus-benign

confidence thresholds would require escalation processes and periodic retraining to maintain accuracy for both benign and malicious datasets once sufficient contemporary test cases are available. The MobileNetV2-small and INT8 depth wise baseline results also indicate that architecture selection depends on the specific classification task; MobileNetV2-0.35 and compact depth wise implementations should not be assumed to be universally optimal substitutes for custom CNN modelling. The present study claims only that the custom 32×32 model met the observed performance requirements of the Malimg ESP32-S3 resolution-threshold experiment for the stated technology and must not be viewed as a definitive solution for all classification tasks.

The custom 32×32 model represents the lowest level of resolution that achieved sufficient multifactor usage reductions in energy consumption, PSRAM usage, latency, and dimensions for the considered study group. Additionally, based on the present findings, the results should be viewed as providing justification for periodic triage on a Microcontroller-Class Device (MCU) rather than continuous real-time malware mitigation capability.

Further work must include validated journal-quality per-class visualisations, along with future experiments comparing externally curated malware and benign samples under realistic deployment conditions. Future live-device studies should include larger sample sizes and precise confidence intervals to provide stronger baseline references for confidence-based triage decisions.

VI. Conclusions

The study described here examined whether a small enough resolution could be used in a manageable way to allow deployment of a CNN-based malware image classifier through an embedded System on a Chip (SoC) such as the ESP32-S3, yet still yield a significant level of classification performance. In this use case, the 32×32-pixel resolution model provided the best balance of classification performance, speed, power consumption and size when compared to the $64 \times 6464 \times 64$-pixel resolution model, yielding an average classification accuracy of 97.30% for the Malimg family of malware.

From these results, it can be inferred that the classification of malware using CNNs could be applied periodically, as a form of triage to assist in filtering through data collected on resource limited edge devices such as the ESP32-S3 between many or few benign and/or malicious attachments. However, this method should not be viewed as a fully autonomous malware detection system as a stand-alone product; while Malimg supports family classifications of malware, any determination of benign vs malicious will necessitate another research avenue for future investigation. Overall, this study supports previous research showing that with an appropriate reduction of the size of image through research and testing, it is feasible to utilise CNNs for triaging malware for deployment in a microcontroller class IoT security environment.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Phil Steadman conceptualised the study, conducted the investigation, developed the methodology, and wrote the original draft; P. Jenkins supervised the work, provided resources, and reviewed the manuscript; Rajkumar Singh Rathore supervised the work and reviewed the manuscript; Chaminda Hewage supervised the work and reviewed the manuscript. All authors have approved the final version.

References

[1] S. Sinha. (2024). State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/

[2] T. Sasi, A. H. Lashkari, R. Lu, P. Xiong, and S. Iqbal, "A comprehensive survey on IoT attacks: Taxonomy, detection mechanisms and challenges," *Journal of Information and Intelligence*, vol. 2, no. 6, pp. 455–513, 2024.

[3] B. Zhao, S. Ji, W. H. Lee, C. Lin, H. Weng, J. Wu, P. Zhou, L. Fang, and R. Beyah, "A large-scale empirical study on the vulnerability of deployed IoT devices," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 1826–1840, 2022.

[4] B. B. Rad, M. Masrom, and S. Ibrahim, "Evolution of computer virus concealment and anti-virus techniques: A short survey," arXiv preprint, arXiv:1104.1070, 2011.

[5] S. Hosseinzadeh, S. Rauti, S. Laurén, J. M. Mäkelä, J. Holvitie, S. Hyrynsalmi, and V. Leppänen, "Diversification and obfuscation techniques for software security: A systematic literature review," *Information and Software Technology*, vol. 104, pp. 72–93, 2018.

[6] M. Gopinath and S. C. Sethuraman, "A comprehensive survey on deep learning-based malware detection techniques," *Computer Science Review*, vol. 47, 100529, 2023.

[7] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proc. 8th International Symposium on Visualization for Cyber Security. Pittsburgh Pennsylvania*, 2011, pp. 1−9.

[8] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *Proc. 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, pp. 1–5.

[9] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computing Surveys*, vol. 44, no. 2, pp. 1–42, 2012.

[10] A. Sharma, S. Rani, and M. Shabaz, "An optimized stacking-based TinyML model for attack detection in IoT networks," *Plos One*, vol. 20, no. 8, e0329227, 2025.

[11] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng, and K. Sakurai, "Lightweight classification of IoT malware based on image recognition," in *Proc. 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, 2018, vol. 2, pp. 664–669.

[12] S. A. Roseline, G. Hari, S. Geetha, and R. Krishnamurthy, "Vision-based malware detection and classification using lightweight deep learning paradigm," in *Proc. Communications in Computer and Information Science*, 2020, pp. 62–73.

[13] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of CNN architectures (IMCEC)," *Computers and Security*, vol. 92, 101748, 2020.

[14] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient densenet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, 344, 2021.

[15] B. Yuan, J. Wang, P. Wu, and X. Qing, "IoT malware classification based on lightweight convolutional neural networks," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3770–3783, 2022.

[16] G. Bhandari, A. Lyth, A. Shalaginov, and T. M. Grønli, "Distributed deep neural-network-based middleware for cyber-attacks detection in smart IoT ecosystem: A novel framework

and performance evaluation approach," *Electronics*, vol. 12, no. 2, 298, 2023.
[17] A. Makandar and A. Patrot, "Malware analysis and classification using artificial neural network," in *Proc. 2015 International Conference on Trends in Automation, Communications and Computing Technology (I-TACT-15)*, 2015, pp. 1–6.
[18] S. Smmarwar, G. Gupta, and S. Jumar, "AI-empowered malware detection system for industrial internet of things," *Computers and Electrical Engineering,* vol. 108, 108731, May 2023.
[19] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys and Tutorial*s, vol. 19, no. 4, pp. 2322–2358, 2017.
[20] V. Rey, P. M. S. Sánchez, A. H. Celdrán, and G. Bovet, "Federated learning for malware detection in IoT devices," *Computer Networks*, vol. 204, 108693, 2022.
[21] S. Yue and T. Wang, "Imbalanced malware images classification: A CNN based Approach," arXiv preprint, arXiv:1708.08042, 2017.
[22] E. D. Martin, J. Kargaard, and I. Sutherland, "Raspberry Pi malware: An analysis of cyberattacks towards IoT devices," in *Proc. 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, 2019, pp. 161–166.
[23] P. Steadman, P. Jenkins, R. S. Rathore, and C. Hewage, "Challenges in implementing artificial intelligence on the raspberry Pi 4, 5 and 5 with AI HAT," in *Proc. Contributions Presented at The International Conference on Computing, Communication, Cybersecurity and AI*, 2024, pp. 147–157.
[24] D. T. Mane, P. B. Kumbharkar, S. B. Javheri, and R. Moorthy, "An adaptable ensemble architecture for malware detection," in *Proc. Advances in Intelligent Systems and Computing*, 2021, pp. 647–659.
[25] Malware benign image classification dataset. [Online]. Available: https://www.kaggle.com/datasets/bishwajitprasadgond/malware-benign-image-sample
[26] M. Shahnawaz, B. P. Gond, and D. P. Mohapatra, "Dynamic malware classification of windows PE files using CNNs and greyscale images derived from runtime API call argument conversion," arXiv preprint, arXiv:2505.24231, 2025.
[27] P. Steadman, P. Jenkins, R. S. Rathore, and C. Hewage, "Low-cost malware detection with artificial intelligence on single board computers," *Future Internet*, vol. 18, no. 1, 46, 2026.
[28] P. Steadman, R. S. Rathore, C. Hewage, and P. Jenkins, "AI optimizers and malware detection using imagary on the rasberry Pi," in *Proc. 57th International Conference on Information Systems and Computer Networks (ISCON*), 2025, pp. 1–8.