S. Lopez-Flores and A. Munoz, "Resource-aware electromagnetic malware classification on edge platforms: a controlled cross-platform comparison using Raspberry Pi 5 EM traces", *Journal of Computer Virology and Hacking Techniques*, vol. 22, 2026.
http://doi.org/10.1007/s11416-026-00661-w
NICS Lab. Publications: https://www.nics.uma.es/publications

Resource-Aware Electromagnetic Malware Classification on Edge Platforms: A Controlled Cross-Platform Comparison Using Raspberry Pi 5 EM Traces

Sergio López-Flores[a], Antonio Muñoz[a,*]

[a]*Network, Information and Computer Security (NICS) Lab, University of Malaga, Spain*

Abstract

Electromagnetic side-channel analysis provides a non-intrusive approach to malware characterization in settings where host-based instrumentation is unavailable, undesirable, or too costly to deploy. In practice, however, the usefulness of this approach depends not only on predictive performance but also on whether the selected models remain practically deployable under constrained hardware resources. This paper presents a controlled cross-platform evaluation of electromagnetic malware classification under resource-limited deployment conditions.

A single corpus of electromagnetic traces acquired from a Raspberry Pi 5 using the EM-Sense acquisition workflow is reused unchanged throughout the study. This fixed dataset is used to train and evaluate four classical machine-learning pipelines (LDA+NB, LDA+SVM, LDA+RF, and LDA+XGBoost) and four neural architectures (MLP, CNN, RNN, and GRU) across four labeling granularities: *Packer*, *Virtualized*, *Type*, and *Family*. The same experimental protocol is executed on a virtual machine and three representative edge platforms (Raspberry Pi 5, Jetson Nano, and Jetson Orin Nano), allowing the influence of computational resources to be isolated from the acquisition process.

The results show a clear task-dependent trade-off. In the coarse-grained classification tasks, the classical pipelines provide the most favourable accuracy–cost balance while remaining computationally inexpensive across all evaluated platforms. In the finer-grained tasks, the neural architectures achieve higher predictive performance, although at a substantially greater computational cost. The experiments also identify a practical deployment boundary on Jetson Nano, where the more demanding neural configurations no longer retain a practically useful GPU-accelerated operating mode. Under the evaluation conditions considered in this study, these findings support a resource-aware approach to model selection for electromagnetic malware classification on edge platforms.

Keywords: Malware classification, Electromagnetic side-channel analysis, Edge computing, Resource-constrained devices, Deep learning, Classical machine learning

1. Introduction

Electromagnetic (EM) side-channel analysis offers a non-intrusive way to characterize software execution from externally observable physical signals. In malware analysis, this is especially relevant for settings in which host-based instrumentation is unavailable, undesirable, or too costly to maintain. By capturing processor-level EM emissions, it becomes possible to distinguish execution patterns associated with benign and malicious activity without modifying the monitored system.

Recent work has shown that EM traces can support malware detection and classification on embedded platforms. Most studies, however, have focused on signal acquisition, dataset construction, or classification performance under a single computational setting. Much less attention has been paid to a practical deployment question: how the choice of classifier changes once the same EM classification tasks are executed on hardware platforms with very different resource budgets.

That question matters because edge deployment imposes constraints that do not appear in workstation-class environments. A model that is attractive in terms of predictive performance may become impractical when memory, runtime, or software compatibility are limited. This is particularly relevant in EM-based malware analysis, where the operational goal is not only to obtain accurate classifications, but to do so under realistic hardware conditions.

This paper addresses that problem through a cross-platform evaluation of classical and neural models for EM-based malware classification under a controlled electromagnetic acquisition protocol. The study uses a single corpus of EM traces acquired from a Raspberry Pi 5 under a controlled workflow derived from the EM-Sense methodology. The resulting dataset is kept fixed and reused across all experiments so that the comparison isolates the effect of the computational platform rather than the acquisition stage. The acquisition protocol, sensing hardware, probe placement, and recording configuration were kept unchanged

*Corresponding author.
Email addresses: sergiolf@uma.es (Sergio López-Flores), anto@uma.es (Antonio Muñoz)

throughout the experimental campaign to ensure that all evaluated models were compared under common acquisition conditions. On this basis, we evaluate three classical pipelines based on linear discriminant analysis (LDA+NB, LDA+SVM, LDA+RF and LDA + XGBOOST) and four neural architectures (MLP, CNN, RNN, and GRU) across four labeling granularities: *Packer*, *Virtualized*, *Type*, and *Family*.

The objective is not to introduce a new acquisition device or a new learning architecture, but to examine how different modelling approaches behave when task granularity and hardware constraints are considered jointly. To this end, we compare predictive performance, computational cost, and execution feasibility on a virtual machine and three representative edge platforms: Raspberry Pi 5, Jetson Nano, and Jetson Orin Nano. The evaluated models span classical machine-learning pipelines, feed-forward neural networks, convolutional architectures, and recurrent sequence models, covering a broad range of computational characteristics under a common experimental protocol. Accordingly, the study is intended to characterise the trade-offs between predictive performance and deployment cost rather than to provide an exhaustive benchmark of one-dimensional time-series classifiers or a comprehensive assessment of cross-session generalisation.

The main contributions of this paper are as follows:

- We present a controlled cross-platform evaluation of classical and neural classifiers for EM-based malware classification using a fixed corpus of Raspberry Pi 5 EM traces.
- We compare model behavior across four labeling granularities, from binary scenarios (*Packer* and *Virtualized*) to finer-grained multiclass tasks (*Type* and *Family*).
- We show that the relative behavior of classical and neural models is strongly task-dependent under the controlled acquisition protocol adopted in this study: classical pipelines provide the strongest reported accuracy–cost balance in coarse-grained scenarios, whereas neural models reach the strongest observed accuracies in finer-grained classification at a substantially higher computational cost.
- We document platform-level feasibility constraints, particularly on Jetson Nano, and show that the practical boundary is determined not only by whether a configuration can be executed, but also by whether it can still be executed under a hardware mode that preserves a meaningful advantage.

The remainder of the paper is organized as follows. Section 2 reviews the relevant literature. Section 3 describes the experimental setup, hardware platforms, and classification scenarios. Section 4 presents the dataset construction and input representation used in the experiments. Section 5 introduces the evaluated models and the experimental protocol. Section 6 reports the experimental results. Section 7 discusses the main findings and their deployment implications. Section 8 addresses threats to validity, and Section 9 concludes the paper.

2. Related Work

Electromagnetic (EM) side-channel analysis has long been studied in embedded security, particularly in relation to hardware characterization, cryptographic implementations, and externally observable execution behavior. In this setting, EM emissions are useful because they provide access to processor-level activity without requiring software instrumentation inside the monitored device. This makes EM-based observation especially attractive in environments where intrusive monitoring is undesirable or where software-level telemetry can be restricted, manipulated, or simply unavailable Prvulovic et al. (2017); Sehatbakhsh et al. (2016); Khan et al. (2018).

More recently, several studies have extended this perspective from hardware-oriented analysis to software characterization and malware-related tasks. These works show that EM traces can encode reproducible behavioral patterns associated with program execution, enabling anomaly detection, malware detection, and, in some cases, finer-grained classification. Early approaches were largely framed as binary or anomaly-based problems, focusing on the distinction between benign and malicious behavior or on deviations from a known baseline. Subsequent work has moved toward richer classification settings, including malware type, family, and obfuscation-related distinctions, thereby showing that EM signals can support more than simple detection Khan et al. (2019b); Sehatbakhsh et al. (2019); Chawla et al. (2021).

Adjacent exploratory work has also examined fileless malware specifically, with encouraging binary results under controlled conditions but a marked degradation once concurrent user processes are introduced, which reinforces the importance of evaluating EM-based malware analysis beyond highly simplified execution states Welagedara (2025).

Within this line of research, an important practical issue concerns the accessibility of the acquisition and analysis workflow. A substantial part of the literature relies on specialized measurement setups and tightly controlled capture environments, which limits reproducibility and complicates transfer to realistic embedded deployments. More recent studies have moved toward workflows that are more applicable to operational settings while still preserving sufficient signal quality for downstream classification. This shift is important because it moves EM-based malware analysis from proof-of-concept laboratory conditions toward more practical and reproducible evaluation settings Khan et al. (2019b); Chawla et al. (2021).

A more recent direction has sought to simplify the signal-analysis stage itself. Rather than relying on broad-spectrum representations, Oh *et al.* propose a clock-centred

EM detector that monitors periodic modulation of the dominant clock emission, showing that narrowband monitoring can support external malware detection and coarse-grained behavioral identification while extending the effective sensing distance. This line of work is relevant because it emphasizes a different deployment trade-off from broader learning pipelines, namely lower signal-processing complexity and simpler external monitoring at the cost of a more specialized detection model Oh et al. (2025).

A complementary recent direction shifts the emphasis from instantaneous spectral separation to longer-term behavioral monitoring. Guo *et al.* study controlled memory radiation and argue that anomalous and normal states may remain close in the spectrum unless temporal evolution is taken into account, which motivates combining denoising, behavior-oriented features, and explicit timing analysis. This is relevant to the present work because it underlines that, in EM-based security monitoring, discriminative structure may emerge either from richer feature representations or from longer observation over time, depending on the underlying hardware activity being targeted Guo et al. (2024).

In parallel, the malware analysis literature has increasingly incorporated both classical machine learning and deep learning models. Classical pipelines remain attractive because they are computationally inexpensive, easier to train, and often sufficiently strong in lower-complexity classification tasks. Deep architectures, by contrast, can exploit higher-capacity representations and may provide an advantage when the target problem requires finer discrimination. That advantage, however, comes at a substantially higher computational cost and with greater sensitivity to hardware and software constraints. In edge-oriented settings, this trade-off is not secondary: it can determine whether a model is merely accurate in principle or actually deployable in practice James (2013); Prvulovic et al. (2017).

Despite these advances, two limitations remain visible in the current literature. First, EM-based malware classification is usually evaluated in a single computational environment, even when the intended deployment context is resource-constrained. As a result, predictive performance is often reported without a corresponding account of runtime cost, memory pressure, or platform-level feasibility. Second, model comparisons are commonly tied to a single labeling task, even though deployment requirements may change substantially with label granularity. A model that performs well for binary discrimination may not remain suitable for malware type or family attribution. A related limitation is that prior EM- or power-based studies are often evaluated under restricted analysis conditions, with limited variation in malware samples or benign activity. This makes it difficult to assess how well results would transfer to broader malware-analysis settings beyond narrow detection or anomaly-based formulations Clark et al. (2013); Khan et al. (2019b); Sehatbakhsh et al. (2019); Wang et al. (2018); Chawla et al. (2021); Ding et al. (2020).

The present work is positioned at that intersection. It does not propose a new acquisition device or a new classification architecture. Instead, it takes a fixed corpus of EM traces acquired under a controlled workflow derived from the EM-Sense methodology and studies how representative classical and neural model families behave when the same classification problems are executed on heterogeneous compute platforms. This design makes it possible to separate the effect of the acquisition stage from the effect of the deployment platform and to examine reported model behavior as a joint function of task granularity and hardware budget. In that sense, the contribution of this paper is not another accuracy benchmark in isolation, but a controlled cross-platform comparison of classical and neural classifiers for EM-based malware analysis under realistic edge constraints.

3. Experimental Setup and Classification Scenarios

3.1. Objective and experimental rationale

The objective of the experimental campaign is to examine how model family and compute platform jointly affect the observed practical viability of EM-based malware classification under the executed pipeline. To isolate that effect, all models are trained and evaluated on the same fixed corpus of electromagnetic traces, while only the computational platform changes. This design allows reported predictive performance, computational cost, and execution feasibility to be compared without conflating them with differences in data acquisition.

3.2. Hardware platforms

Raspberry Pi 5 plays a dual role in this study. First, it is the target device from which the electromagnetic traces are acquired. Second, it is one of the platforms on which the classification models are trained and evaluated. Model execution is compared across four platforms with different resource budgets:

- *Virtual Machine on AI Server:* reference execution environment used as a baseline.
- *Raspberry Pi 5:* low-power ARM platform representing a constrained edge setting.
- *Jetson Nano:* embedded platform with limited memory and a restricted practically useful operating regime for the more demanding neural configurations.
- *Jetson Orin Nano:* higher-capacity embedded platform used to assess how additional resources affect model viability.

The reference execution environment was hosted on an AI server equipped with NVIDIA L40S accelerators.

Table 1: Unified summary of the hardware platforms and execution environments used in the cross-platform evaluation.

Platform	*CPU / RAM*	*GPU*	*Operating environment*	*Software and acceleration stack*	*Execution mode*
Virtual Machine on AI Server	Intel Xeon Silver 5418Y (2× CPUs), 2.0 GHz x86 (Sapphire Rapids); 512 GB DDR5-4800	4× NVIDIA L40S PCIe (48 GB each)	Debian 12.12; Linux 6.1 amd64; NVIDIA driver 580.95	Python 3.11.2; TensorFlow 2.20.0; Keras 3.12.0; scikit-learn 1.7.2; CUDA 12.4; cuDNN 9.1.1; TensorRT not installed	GPU
Raspberry Pi 5	2.4 GHz ARM Cortex-A76; 8 GB LPDDR4X	–	Raspberry Pi OS 64-bit (Debian 11 Bullseye); Linux kernel 5.10	Python 3.11.2; TensorFlow 2.19.0; Keras 3.9.2; scikit-learn 1.6.1; no CUDA/cuDNN/TensorRT	CPU
Jetson Nano	1.43 GHz ARM Cortex-A57 MPCore; 4 GB LPDDR4	NVIDIA Maxwell GPU (128 cores)	Linux (JetPack 4.6.1, L4T R32.6.1); kernel 4.9.253-tegra	Python 3.6.9; TensorFlow 2.4.1; Keras 2.4.0; scikit-learn 0.24.2; CUDA 10.2; cuDNN 8.2.1; TensorRT 8.0.1; NVCC 10.2.300	Mixed CPU/GPU[a]
Jetson Orin Nano	1.7 GHz ARM Cortex-A78AE; 8 GB LPDDR5	NVIDIA Ampere GPU (1024 cores, 32 Tensor Cores)	Linux (JetPack 5.1, L4T R35.5.0); kernel 5.10.192-tegra	Python 3.8.10; TensorFlow 2.11.0+nv23.3; Keras 2.11.0; scikit-learn 1.3.2; CUDA 11.4.19; cuDNN 8.6.0; TensorRT 8.5.2; NVCC 11.4.315	GPU

[a]Jetson Nano used GPU execution for some configurations, whereas GRU- and RNN-based runs were executed on CPU.

3.3. Acquisition workflow

The EM corpus used in this study was acquired once on a Raspberry Pi 5 and subsequently reused without modification across all computational platforms. No additional traces were collected on the virtual machine, Jetson Nano, or Jetson Orin Nano, which were used exclusively for model training and evaluation.

Signal acquisition followed the EM-Sense workflow. Malware and benign samples were executed on bare-metal Raspberry Pi 5 hardware running Raspberry Pi OS 64-bit, accessed remotely via SSH, while background services remained enabled to preserve a realistic execution environment. The target device was connected to an isolated network so that malware execution could be safely controlled without affecting other systems. This design choice is methodologically relevant because highly restricted or heavily instrumented environments may alter malware behaviour and expose analysis artifacts that facilitate evasion techniques. Executing the samples on bare-metal hardware while preserving a realistic operating context therefore provides a more representative execution environment than a simplified analysis setup Raffetseder et al. (2007); Yokoyama et al. (2016).

Electromagnetic emissions were captured using an Arduino UNO R4 WiFi connected to a commercial Tekbox EMC probe set equipped with a planar magnetic near-field (H-field) probe. The probe output passed through a passive analogue conditioning circuit incorporating a voltage divider before being sampled by the Arduino analogue input and stored directly on a microSD card for offline processing. The analogue front-end operated with a 30 dB RF amplifier (1–2000 MHz), while the acquisition pipeline sampled the signal every 1 ms (approximately 1 kHz). During the experimental campaign, the sensing loop was centred above the Raspberry Pi 5 SoC and held parallel to the processor package using a mechanical support. Probe position, orientation, analogue front-end, and acquisition parameters remained unchanged throughout all recording sessions to ensure consistent measurement conditions.

Table 2: Characteristics of the datasets used for the evaluated classification tasks. The same preprocessed datasets were used across all hardware platforms.

Task	Classes	Samples	Imbalance	Entropy
Family	9	9,920	2.09	0.976
Types	5	6,668	2.68	0.971
Packer	2	2,224	1.96	0.923
Virtualized	2	2,035	1.73	0.948

3.4. Dataset composition and task definitions

The corpus contains benign and malicious executions recorded from the Raspberry Pi 5 under the acquisition workflow described above. In total, the raw corpus comprises 18 capture sessions and approximately 70 MB of recorded EM data.

To enable a controlled comparison across hardware platforms, exactly the same preprocessed datasets, sample ordering, and one-hot encoded labels were used throughout the experimental campaign. Consequently, any differences observed between platforms are attributable to the execution environment rather than to variations in dataset composition. Table 2 summarizes the characteristics of the four classification tasks.

Table 3 summarizes the coverage of the experimental campaign across all evaluated hardware platforms. Every platform–model family–classification task combination is explicitly identified as either reported or not evaluated. This distinction clarifies the scope of the study and avoids interpreting unavailable results as failed executions.

The coverage shown in Table 3 reflects the scope of the experimental campaign reported in this paper. The three LDA-based pipelines were evaluated on all four hardware platforms and across all classification scenarios. In contrast, the complete set of neural-model experiments was not executed on Jetson Nano during the revision period because of the time required to complete the full training campaign. Consequently, the absence of these entries should not be interpreted as execution failures or as evidence of hardware

Table 3: Coverage of the experimental campaign across the evaluated hardware platforms. Reported indicates that the corresponding configuration was executed and contributes to the experimental evaluation. Not evaluated denotes configurations that were not executed as part of the reported experimental campaign.

Platform	Model family	*Packer*	*Virtualized*	*Type*	*Family*
Virtual Machine	Neural models	Reported	Reported	Reported	Reported
	LDA-based models	Reported	Reported	Reported	Reported
Raspberry Pi 5	Neural models	Reported	Reported	Reported	Reported
	LDA-based models	Reported	Reported	Reported	Reported
Jetson Orin Nano	Neural models	Reported	Reported	Reported	Reported
	LDA-based models	Reported	Reported	Reported	Reported
Jetson Nano	Neural models	Not evaluated	Not evaluated	Not evaluated	Not evaluated
	LDA-based models	Reported	Reported	Reported	Reported

limitations, but simply as configurations that fall outside the scope of the reported evaluation.

After acquisition, the recorded measurements were grouped into fixed-length blocks of 1000 consecutive values, each block was assigned the corresponding class label, and the resulting files were merged at the scenario level before one-hot label encoding for downstream training and evaluation. This procedure is described in the experimental materials released with the study. Given the 1 kHz acquisition rate, each constructed sample represents approximately one second of recorded signal.

Four classification scenarios were considered: *Packer*, *Virtualized*, *Type*, and *Family*. The first two are binary tasks representing coarse-grained discrimination, whereas the latter two are multiclass problems requiring progressively finer attribution. This progression makes it possible to analyse how model behaviour changes as label granularity increases and reflects different operational objectives, from coarse-grained triage to family-level attribution Pham et al. (2021).

Table 4 reports the per-class sample distribution for each classification scenario. Together with the summary statistics in Table 2, it provides a complete description of the datasets used throughout the experimental evaluation.

The composition of the benign class should also be considered when interpreting the results. Previous studies have shown that EM-based malware datasets built from highly restricted benign workloads may artificially simplify the discrimination problem. In operational environments, benign activity typically comprises a broader mixture of utilities, background services, and long-running processes, which should be taken into account when comparing results across studies Chawla et al. (2021); Khan et al. (2019a); Sehatbakhsh et al. (2019); Nazari et al. (2017).

To facilitate reproducibility, the labeled datasets and the scripts used in the experimental campaign are publicly available at `https://github.com/Sergiolopezflores/Gadget-Accessible-Electromagnetic-Fault-Injection/tree/main`.

3.5. Platform-specific constraints

Jetson Nano imposed the strongest practical constraints among the evaluated platforms. In particular, the software environment required adaptation due to the runtime error `cannot allocate memory in static TLS block`. In the less demanding configurations, some models could still be executed with GPU support. However, for the more demanding architectures and classification scenarios, GPU execution ceased to provide a stable or practically useful operating mode, so the experiments were carried out in practice under CPU-only execution. Under those conditions, runtime increased substantially and the platform no longer offered a clear advantage over Raspberry Pi 5. These limitations are treated as part of the observed platform behavior rather than as incidental implementation issues.

4. Dataset Construction and Input Representation

The classification pipeline used in this study did not rely on a time–frequency transformation or on a feature-selection stage. Instead, the recorded electromagnetic measurements were organized directly into fixed-length samples for supervised learning. According to the dataset-construction pipeline used in the experiments, the captured values were grouped into consecutive blocks of 1000 measurements, and each block was associated with the corresponding class label. Given the acquisition configuration described in Section 3, each block represents approximately one second of recorded signal. After the labeled files for a given scenario had been generated, they were merged into a single dataset and the labels were converted to one-hot form for downstream training and evaluation.

All EM recordings were acquired under a single controlled experimental campaign using the same Raspberry Pi 5 target, sensing hardware, probe placement, acquisition parameters, and measurement configuration. Fixed-length blocks were subsequently extracted from these recordings for model training and evaluation. Consequently, the observed experiments quantify classifier behaviour under consistent acquisition conditions and should not be interpreted as an assessment of robustness across independently acquired recording sessions or different sensing environments.

Given the acquisition rate used in the experiments, namely 1000 measurements per second, and the fixed block length of 1000 consecutive values, each constructed sample

Table 4: Class distribution for the four classification scenarios after dataset construction. Sample counts correspond to the complete datasets used throughout the experimental evaluation.

Task	Class distribution
Packer	Packed (1472), Unpacked (752)
Virtualized	Native (745), Virtualized (1290)
Type	Goodware (1535), DDoS (1595), Ransomware (1463), Rootkits (595), Spyware (1480)
Family	Bashlite (1503), Encoder (778), Gonnacry (1508), Goodware (1595), Kisni (783), Mak (795), Mirai (1406), Snoopy (788), Wirenet (764)

corresponds to approximately one second of recorded signal. If T denotes the capture duration in seconds, the number of constructed samples is

$$N_s = \left\lfloor \frac{f_s \cdot T}{L} \right\rfloor,$$

where $f_s = 1000$ is the sampling rate and $L = 1000$ is the block length. Under the configuration used in this study, this reduces to approximately one constructed sample per second of capture.

Let $\mathbf{x} \in \mathbb{R}^{1000}$ denote one constructed input sample, where each component corresponds to one value extracted from the recorded trace. The target label associated with $\mathbf{x}$ was encoded by means of a one-hot vector $\mathbf{z} \in \{0,1\}^C$, where C is the number of classes in the corresponding scenario. For a class label $y \in \{1,\ldots,C\}$, the encoding is given by

$$z_c = \begin{cases} 1, & \text{if } c = y, \\ 0, & \text{otherwise,} \end{cases} \qquad c = 1,\ldots,C.$$

Under this representation, each row of the final dataset consists of 1000 input values followed by a scenario-dependent one-hot target vector. This format was used uniformly across the binary and multiclass tasks considered in the study.

Before model training, the input vectors were standardized in order to reduce scale differences across dimensions and to provide a consistent numerical representation for the evaluated classifiers. After standardization, the representation was adapted only as required by each model family. In the classical branch, the input remained in vector form and was subsequently projected through linear discriminant analysis (LDA) before classification with NB, SVM, or RF. The MLP operated on the same vector representation. By contrast, the CNN-, RNN-, and GRU-based models received the standardized input reshaped as a one-dimensional sequence of length 1000 with a single feature channel per time step.

The resulting preprocessing chain is intentionally simple. It preserves the original captured values as the basis of the obtained classification task and introduces only the transformations required for label encoding, normalization, and model-specific input formatting. No spectrogram computation, spectral ranking, or frequency-band selection stage was applied in the experiments reported in this paper.

5. Classification Models and Evaluation Protocol

5.1. Evaluated models

The experimental comparison includes three classical pipelines and four neural architectures. The classical family comprises *LDA+NB*, *LDA+SVM*, *LDA+RF* and *LDA+XGBOOST*, where linear discriminant analysis (LDA) is used as a supervised projection stage before classification. The neural family comprises *MLP*, *CNN*, *RNN*, and *GRU*. This design allows the comparison of lightweight classifiers operating on standardized vector inputs with higher-capacity models that process the same data under architecture-specific formats.

The classical models were selected because they provide inexpensive baselines for fixed-length EM vectors once dimensionality has been reduced through LDA. In this setting, Naïve Bayes (NB) offers a simple probabilistic classifier with very low computational cost, Support Vector Machines (SVM) provide margin-based discrimination in the projected feature space, Random Forests (RF) add a non-linear ensemble alternative and Extreme Gradient Boosting (XGBoost) incorporates a gradient boosting strategy capable of capturing complex non-linear relationships while maintaining high predictive performance. The role of LDA in this branch is to produce a compact supervised representation before classification.

For the traditional machine-learning pipeline, the dataset was first partitioned into training and testing subsets before applying any preprocessing stage that required parameter estimation. The `StandardScaler` was fitted exclusively on the training subset and subsequently applied to the corresponding test data. Similarly, Linear Discriminant Analysis (LDA) was trained only on the scaled training samples together with their associated labels, and the resulting projection was then applied to the held-out subset. The transformed training data were subsequently used to train the NB, SVM, and RF classifiers, while all reported performance metrics were computed exclusively on the test partition.

The neural branch includes one fully connected baseline and three sequence-based architectures. The *MLP* operates directly on the standardized 1000-dimensional input vector and serves as the simplest neural baseline. The *CNN* is implemented as a one-dimensional convolutional model applied to the input sequence. The *RNN* is implemented through a bidirectional LSTM-based architecture

with dropout, and the *GRU* uses a stacked recurrent design followed by dense layers. Across the neural models, dropout was used as regularization, and the CNN additionally applied regularization in its dense layers. These design choices were adopted to balance predictive performance and computational feasibility on constrained platforms.

The purpose of the comparison is not to claim novelty in model design, but to assess how representative classical and neural approaches behave under the same EM-based classification tasks and across different hardware platforms. The architectural and training details that could be confirmed directly from the implementation used in the experiments are summarized in Tables A.8 and A.9.

5.2. Evaluation protocol

All experiments were conducted using the same EM corpus acquired from the Raspberry Pi 5 setup described in Section 3. The identical preprocessed dataset, sample ordering, and one-hot encoded labels were reused across all hardware platforms so that any differences in predictive performance or execution behaviour could be attributed to the evaluated model–platform combination rather than to variations in data acquisition or dataset composition.

The final dataset consists of fixed-length samples of 1000 input values with a scenario-dependent one-hot encoded target vector, as described in Section 4. An identical 80/20 hold-out partition was applied in all experiments, with 80% of the available samples used for model development and the remaining 20% reserved for evaluation. All neural models were trained for 300 epochs using a batch size of 32 and the categorical cross-entropy loss function. The held-out partition was used for validation during training. Table A.9 summarizes the input representation, network architecture, and optimizer used for each neural model.

For the classical pipelines, feature standardization was performed by fitting the `StandardScaler` exclusively on the training partition and applying the resulting transformation to both the training and test subsets. LDA was subsequently fitted on the standardized training data and the learned projection was applied to both partitions before classifier training. The neural models were trained and evaluated using the same train/test partition. The MLP operated directly on the standardized input vectors, whereas the CNN-, RNN-, and GRU-based models received the corresponding one-dimensional sequence representation.

All deep learning models were trained for 300 epochs. The reported neural results therefore correspond to the final model obtained after that training schedule rather than to a separately selected checkpoint. Training progress was monitored through loss and accuracy curves in order to inspect learning behavior during development. The same scenario definitions were used throughout the evaluation: *Type* and *Family* as multiclass tasks, and *Virtualized* and *Packer* as binary tasks.

The results should therefore be interpreted as observed outcomes under the executed experimental pipeline. They do not constitute estimates obtained under a strictly isolated validation-and-test protocol, nor do they provide variance-based uncertainty estimates across repeated runs.

5.3. Performance measures

Classification accuracy is retained as the primary summary metric because it provides a direct and easily comparable measure across classification scenarios, computational platforms, and model families. To obtain a more complete assessment of predictive performance, macro-F1, weighted-F1, and balanced accuracy are also reported for every classification task. For the binary *Packer* and *Virtualized* scenarios, ROC-AUC and PR-AUC are additionally reported as threshold-independent measures of discrimination. Per-class precision, recall, and F1-score are obtained from the corresponding classification reports generated during evaluation.

In addition to predictive performance, deployment characteristics were assessed through preprocessing time, training time, inference latency, throughput, memory footprint, and serialized model size.

In addition to predictive performance, the study evaluates computational cost and execution feasibility. The reported timing values retain the semantics of the underlying implementations. For neural models, the reported training time corresponds to wall-clock time per epoch, whereas for the classical LDA-based pipelines it corresponds to the total fitting time of the complete pipeline. These measurements are presented separately because they quantify different stages of the learning process and should not be interpreted as a single normalized cost metric.

Execution feasibility is treated as a first-class experimental outcome. When a model no longer provided a practically useful operating point on a given platform because of memory limitations, software-environment constraints, or the loss of an effective hardware-accelerated execution mode, that behaviour was considered part of the experimental results rather than discarded.

Complete per-class classification reports, confusion matrices, ROC/PR analyses, and the deployment-oriented measurements collected for every evaluated model and hardware platform are available through the accompanying public repository together with the remaining experimental artefacts.

5.4. Statistical treatment

The values reported in this paper correspond to the measured runs available for each platform–model–scenario combination under a fixed experimental configuration. The data partition used in the scripts is deterministic through a fixed random seed, but the study does not include repeated runs across multiple seeds or alternative splits. Accordingly, the paper reports observed results rather than mean $\pm$ standard deviation estimates, and no formal statistical significance claims are made between model families. The confirmed architectural and training details used in the experiments are summarized in Tables A.8 and A.9.

6. Experimental Results

6.1. Accuracy across scenarios and platforms

Table 5 summarizes the predictive performance obtained for every evaluated model, classification scenario, and hardware platform. Each entry reports both classification accuracy and macro-F1, allowing overall predictive performance and class-balanced performance to be assessed simultaneously.

Under the executed experimental protocol, the relationship between predictive performance and model family depends strongly on the classification scenario. In the coarse-grained tasks (*Packer* and *Virtualized*), both classical and neural approaches achieved competitive results, although their behaviour differed across scenarios. For *Packer*, the highest reported accuracy was obtained by the CNN model on Raspberry Pi 5 (0.964/0.959), while the classical LDA-based pipelines consistently achieved accuracies close to 0.89 across all evaluated platforms, providing stable performance with substantially lower computational requirements. In contrast, the *Virtualized* scenario favoured the classical pipelines, with LDA+RF reaching the highest reported accuracy (0.786/0.758) on every evaluated platform, whereas the best neural models remained between 0.660 and 0.695.

The trend changed for the finer-grained classification tasks. In *Type*, the recurrent and convolutional neural models consistently outperformed the classical pipelines across all evaluated platforms, reaching accuracies of up to 0.968 and macro-F1 scores of up to 0.971, whereas the best classical models remained close to 0.86. The separation became more pronounced in the *Family* scenario, where the GRU architecture achieved the highest predictive performance on every evaluated platform with available results, reaching 0.902/0.858 on Jetson Orin Nano, while the classical pipelines remained close to 0.64 accuracy and 0.52 macro-F1.

The transition from competitive classical performance in the coarse-grained tasks to a clear neural advantage in the finer-grained tasks is summarized in Table 6. Neural models showed only marginal differences with respect to the best classical pipelines in *Packer* and were consistently outperformed in *Virtualized*. Conversely, they exceeded the best classical results by approximately 0.1 accuracy points in *Type* and by more than 0.2 points in *Family*. No results are reported for the neural models on Jetson Nano in the *Type* and *Family* scenarios because these configurations were not evaluated during the reported experimental campaign.

6.2. Per-model results and computational cost

Table A.10 reports the measured platform–scenario–model combinations used in the comparative analysis. For neural models, Time denotes wall-clock time per epoch. For the classical LDA-based pipelines, Time denotes total fitting time. These quantities are reported together for transparency, but they should not be interpreted as a single normalized cost variable.

Across all runs, MLP never achieved the best accuracy in any scenario–platform pair. The recurrent baseline labeled as RNN was also consistently dominated by CNN or GRU whenever results were available. Within the classical branch, the three LDA-based pipelines showed identical wall-clock times within each scenario–platform pair under the current implementation, although their predictive performance differed slightly.

The cost gap between model families was substantial. The classical pipelines remained inexpensive on all platforms with results, with total fitting times between 1 and 34 s depending on platform and scenario. By contrast, neural models showed a much stronger dependence on both architecture and hardware. CNN remained the least expensive neural model, ranging from 1–3 s per epoch on the virtual machine, 10–66 s on Raspberry Pi 5, 4–5 s on the Jetson Nano scenarios with CNN runs, and 1–7 s on Jetson Orin Nano. The recurrent models were much more demanding. GRU required 33, 26, 122, and 161 s per epoch on the virtual machine across *Packer*, *Virtualized*, *Type*, and *Family*, respectively; 127, 100, 490, and 638 s on Raspberry Pi 5; and 11, 8, 42, and 61 s on Jetson Orin Nano. No GRU results are reported on Jetson Nano.

Within the reported runs, these measurements indicate that Jetson Orin Nano provides the most favorable operating point among the evaluated edge devices for the neural branch. Raspberry Pi 5 remained usable, but recurrent training became slow in the more demanding scenarios. Jetson Nano should be interpreted differently: rather than as a uniformly failing platform, it marks the point at which the more demanding neural configurations cease, in the runs, to retain a practically useful accelerated operating mode and no longer provide a meaningful advantage over Raspberry Pi 5. In contrast, the classical pipelines remained inexpensive across all platforms with reported executions, which means that the practical benefit of the Orin-class hardware is concentrated mainly in the neural regime.

6.3. Scenario-wise behavior

The four classification scenarios represent progressively different operational objectives, ranging from coarse-grained binary discrimination to fine-grained malware attribution. Consequently, they differ not only in predictive difficulty but also in the level of information expected from the resulting classifier.

Packer. The *Packer* scenario was the least demanding binary task. Both classical and neural models achieved high predictive performance, although the best results depended on the evaluated platform. The highest reported accuracy was obtained by the CNN model on Raspberry Pi 5 (0.964/0.959), whereas the LDA-based pipelines consistently achieved accuracies close to 0.89 on every evaluated platform. These results indicate that packer detection can be addressed successfully using either model family,

Table 5: Predictive performance across the evaluated hardware platforms. Each entry reports Accuracy / Macro-F1. N/E indicates that the corresponding configuration was not evaluated as part of the reported experimental campaign.

Scenario	Platform	MLP	CNN	RNN	GRU	LDA+NB	LDA+SVM	LDA+RF	LDA+XGBoost
Packer	VM	0.520 / 0.519	0.857 / 0.855	0.453 / 0.312	0.860 / 0.859	0.892 / 0.881	0.888 / 0.871	0.892 / 0.881	0.890 / 0.879
	Raspberry Pi 5	0.915 / 0.905	0.964 / 0.959	0.676 / 0.403	0.780 / 0.733	0.892 / 0.881	0.888 / 0.871	0.892 / 0.881	0.890 / 0.879
	Jetson Nano	N/E	N/E	N/E	N/E	0.892 / 0.881	0.888 / 0.871	0.892 / 0.881	N/E
	Jetson Orin Nano	0.571 / 0.571	0.809 / 0.805	0.453 / 0.312	0.897 / 0.896	0.892 / 0.881	0.888 / 0.871	0.892 / 0.881	0.890 / 0.879
Virtualized	VM	0.548 / 0.544	0.695 / 0.694	0.575 / 0.575	0.541 / 0.477	0.776 / 0.754	0.781 / 0.760	0.786 / 0.758	0.764 / 0.739
	Raspberry Pi 5	0.548 / 0.544	0.687 / 0.683	0.537 / 0.535	0.533 / 0.348	0.776 / 0.754	0.781 / 0.760	0.786 / 0.758	0.764 / 0.739
	Jetson Nano	N/E	N/E	N/E	N/E	0.776 / 0.754	0.781 / 0.760	0.786 / 0.758	N/E
	Jetson Orin Nano	0.560 / 0.555	0.660 / 0.659	0.529 / 0.360	0.533 / 0.348	0.776 / 0.754	0.781 / 0.760	0.786 / 0.758	0.764 / 0.739
Type	VM	0.775 / 0.763	0.954 / 0.957	0.963 / 0.967	0.966 / 0.971	0.861 / 0.872	0.860 / 0.871	0.848 / 0.862	0.844 / 0.853
	Raspberry Pi 5	0.761 / 0.738	0.954 / 0.959	0.952 / 0.958	0.968 / 0.971	0.861 / 0.872	0.860 / 0.871	0.848 / 0.862	0.844 / 0.853
	Jetson Nano	N/E	N/E	N/E	N/E	0.861 / 0.872	0.860 / 0.871	0.848 / 0.862	N/E
	Jetson Orin Nano	0.758 / 0.752	0.951 / 0.956	0.905 / 0.915	0.951 / 0.955	0.861 / 0.872	0.860 / 0.871	0.848 / 0.862	0.842 / 0.850
Family	VM	0.582 / 0.488	0.870 / 0.807	0.832 / 0.757	0.859 / 0.797	0.637 / 0.520	0.636 / 0.514	0.635 / 0.512	0.640 / 0.523
	Raspberry Pi 5	0.577 / 0.482	0.866 / 0.800	0.812 / 0.735	0.887 / 0.833	0.637 / 0.520	0.636 / 0.514	0.635 / 0.512	0.640 / 0.523
	Jetson Nano	N/E	N/E	N/E	N/E	0.635 / 0.518	0.636 / 0.514	0.639 / 0.517	N/E
	Jetson Orin Nano	0.579 / 0.483	0.864 / 0.801	0.828 / 0.752	0.902 / 0.858	0.637 / 0.520	0.636 / 0.514	0.638 / 0.517	0.640 / 0.522

Table 6: Accuracy difference between the best neural and the best classical model for each scenario-platform pair. Positive values indicate an advantage for neural models; negative values indicate an advantage for classical pipelines.

Scenario	*VM*	*RPi 5*	*Jetson Nano*	*Jetson Orin Nano*
Packer	-0.03	-0.01	-0.02	+0.01
Virtualized	-0.10	-0.11	-0.10	-0.13
Type	+0.11	+0.11	N/D	+0.09
Family	+0.23	+0.25	N/D	+0.26

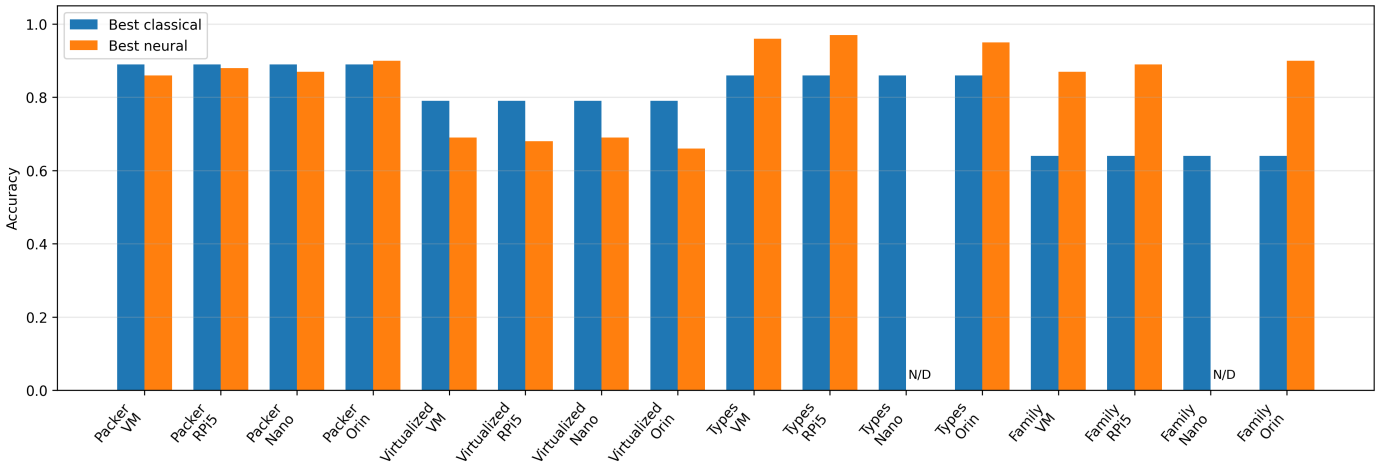

Figure 1: Best predictive performance obtained by the highest-performing classical and neural models for each scenario-platform combination. The figure illustrates the transition from comparable performance in the coarse-grained tasks to a clear neural advantage in the finer-grained classification scenarios.

Table 7: Classification results across hardware platforms. For neural models, Time denotes per-epoch time. For LDA-based pipelines, Time denotes total fitting time.

		MLP		*CNN*		*RNN*		*GRU*		*LDA+NB*		*LDA+SVM*		*LDA+RF*		*LDA+XGBoost*	
Platform	*Scenario*	*Time*	*Acc*	*Time*	*Acc*	*Time*	*Acc*	*Time*	*Acc*	*Time*	*Acc*	*Time*	*Acc*	*Time*	*Acc*	*Time*	*Acc*
Virtual Machine	Packer	< 1 s	0.52	1 s	0.86	15 s	0.43	33 s	0.86	2s	0.89	2s	0.88	2s	0.89	6s	0.89
	Virtualized	< 1 s	0.55	1 s	0.69	12 s	0.58	26 s	0.54	2s	0.78	2s	0.79	2s	0.79	6s	0.76
	Type	< 1 s	0.76	2 s	0.96	57 s	0.93	122 s	0.96	3s	0.86	3s	0.86	3s	0.85	14s	0.84
	Family	< 1 s	0.58	3 s	0.87	71 s	0.83	161 s	0.86	3s	0.64	3s	0.63	3s	0.64	15s	0.64
Raspberry Pi 5	Packer	1 s	0.52	∼16 s	0.88	90 s	0.45	127 s	0.87	5 s	0.89	5 s	0.88	5 s	0.89	5 s	0.89
	Virtualized	< 1 s	0.56	10 s	0.68	57 s	0.50	100 s	0.53	5 s	0.78	5 s	0.78	5 s	0.79	5 s	0.76
	Type	1 s	0.76	47 s	0.95	278 s	0.95	490 s	0.97	8 s	0.86	8 s	0.86	8 s	0.85	8 s	0.84
	Family	2 s	0.58	66 s	0.86	450 s	0.85	638 s	0.89	12 s	0.64	12 s	0.64	12 s	0.63	12 s	0.64
Jetson Nano	Packer	1 s	0.55	5 s	0.87	183 s	–	–	–	28 s	0.89	28 s s	0.88	28 s	0.89	28 s	0.88
	Virtualized	< 1 s	0.56	4 s	0.69	–	–	–	–	18 s	0.78	18 s	0.78	18 s	0.79	18 s	0.79
	Type	–	–	–	–	–	–	–	–	29 s	0.86	29 s	0.86	29 s	0.85	29s	0.86
	Family	–	–	–	–	–	–	–	–	34 s	0.63	34 s	0.64	34 s	0.64	34 s	0.64
Jetson Orin Nano	Packer	1 s	0.57	1 s	0.81	9 s	0.45	11 s	0.90	6 s	0.89	6s	0.88	6 s	0.89	6 s	0.89
	Virtualized	< 1 s	0.56	1 s	0.66	7 s	0.53	8 s	0.53	5 s	0.78	5 s	0.78	5 s	0.79	5 s	0.76
	Type	2 s	0.76	5 s	0.95	36 s	0.90	42 s	0.95	7 s	0.86	7 s	0.85	7 s	0.85	7s	0.84
	Family	3 s	0.59	7 s	0.86	49 s	0.83	61 s	0.90	12 s	0.64	12 s	0.64	12 s	0.64	12 s	0.64

with the classical pipelines providing a considerably lower computational cost.

Virtualized. The *Virtualized* scenario proved more challenging for the neural architectures. Across all evaluated platforms, the LDA-based pipelines consistently outperformed the neural models, with LDA+RF achieving the highest reported accuracy (0.786/0.758). By contrast, the best neural configurations remained between 0.660 and 0.695 accuracy depending on the platform. Under the evaluation conditions considered here, virtualization detection therefore appears to favour lightweight statistical models over the more complex neural architectures.

Type. The multiclass *Type* scenario marked the transition towards a clear neural advantage. CNN, RNN, and particularly GRU consistently exceeded the predictive performance of the classical pipelines across all evaluated platforms. The

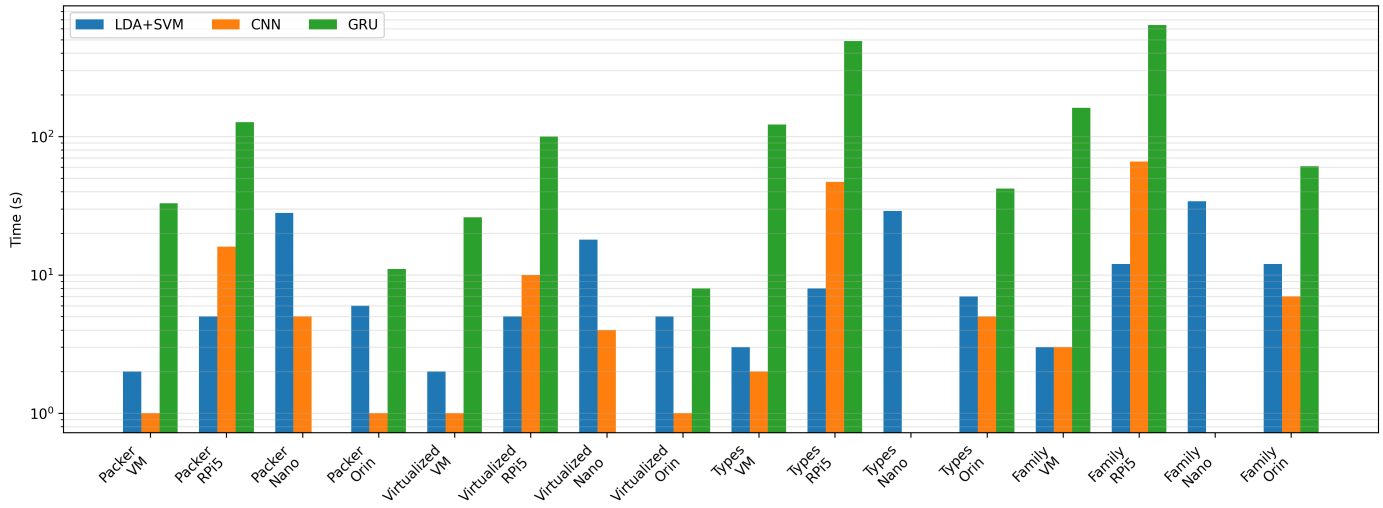

Figure 2: Representative computational cost on a logarithmic scale. Classical timings are shown with LDA+SVM because all three LDA-based pipelines exhibited the same measured wall-clock time within each scenario-platform pair. Neural timings are reported per epoch.

highest reported result was obtained by the GRU model on Raspberry Pi 5 (0.968/0.971), whereas the best classical pipeline remained close to 0.86 accuracy. These results suggest that the additional modelling capacity of the neural architectures becomes beneficial once the classifier must discriminate among multiple malware categories.

Family. The *Family* scenario represented the most demanding classification task because it required discrimination among individual malware families rather than broader behavioural categories. This increase in label granularity produced the largest separation between model families. The GRU architecture achieved the highest reported predictive performance on every platform with available results, reaching 0.902/0.858 on Jetson Orin Nano, while the classical pipelines remained close to 0.64 accuracy. The results therefore indicate that fine-grained malware attribution benefits substantially from higher-capacity sequence models, although at a considerably greater computational cost than the classical alternatives.

6.4. Execution limits and incomplete coverage

The partial Jetson Nano coverage should be interpreted as a substantive platform-level result. In the completed scenarios, the platform remained usable for some of the lighter configurations, but this behavior did not extend to the more demanding cases under a practically useful GPU-backed regime. In particular, *Packer* and *Virtualized* could be evaluated for MLP, CNN, and the classical LDA-based pipelines, whereas the recurrent models did not provide a practically useful operating point on that platform. For *Type* and *Family*, the experimental campaign was not extended initially to all model–scenario combinations once it became clear that the more demanding workloads no longer yielded a meaningful advantage over Raspberry Pi 5 and were effectively limited to substantially slower CPU-only

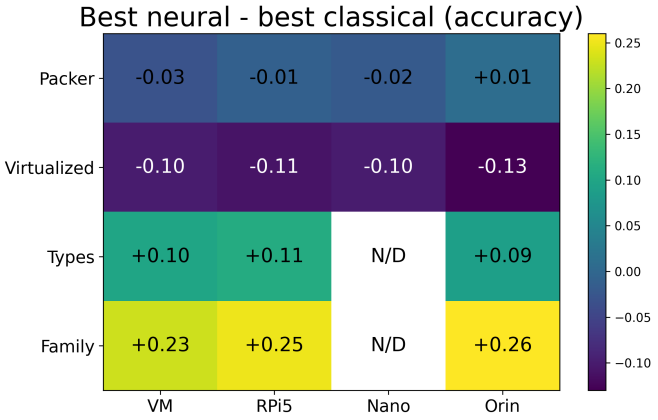

Figure 3: Accuracy advantage of the best neural model over the best classical model. Positive values indicate a neural advantage.

execution. Together with the software-environment issue described earlier, namely the `cannot allocate memory in static TLS block` runtime error, these outcomes indicate that Jetson Nano lies at or beyond the practical feasibility boundary for the more demanding configurations considered in this study.

7. Discussion

The main result of this study is that, under the executed experimental pipeline, EM-based malware classification on edge platforms is better understood as a resource-aware model selection problem than as a purely predictive benchmark. The reported evidence does not support a uniform preference for either classical or neural methods. Instead, it shows that observed model behavior depends jointly on task granularity and available hardware resources.

For the two coarse-grained scenarios, *Packer* and *Virtualized*, the classical pipelines were consistently preferable in

the results. In these tasks, the LDA-based models matched or exceeded the best neural alternatives on every platform with available results while retaining a much lower computational burden. The contrast was particularly clear in *Virtualized*, where the best classical models remained near-perfect across platforms and the neural branch stayed well below that level. Under these conditions, the additional cost of neural training does not produce a practical return.

The picture changes once the classification task becomes more fine-grained. In *Type* and especially in *Family*, neural models became competitive or superior wherever results were reported. The improvement was modest in some settings, but substantial in others, most clearly in *Family*. Within the present evaluation setup, this indicates that the additional modeling capacity of the neural architectures becomes relevant when the label space requires finer discrimination between malware behaviors and families. The interpretation of these results should also remain tied to the composition of the benign class. In EM-based malware analysis, overly restricted non-malicious workloads can make discrimination artificially easier, whereas more varied benign activity yields a more demanding and operationally more meaningful setting. From that perspective, model behavior should be read not only against label granularity, but also against the realism of the underlying activity mix Khan et al. (2019b); Chawla et al. (2021); Sehatbakhsh et al. (2019); Nazari et al. (2017).

The comparison within the neural branch is also informative. MLP never provided the best result in any platform–scenario combination, and the recurrent baseline labeled as RNN did not surpass CNN or GRU in any setting. In practical terms, the neural trade-off observed here is driven almost entirely by CNN and GRU. CNN provided the most balanced neural alternative, with strong predictive performance at a moderate computational cost. GRU achieved the strongest results in several of the fine-grained tasks, but at a much higher runtime cost, particularly on Raspberry Pi 5.

The platform comparison further sharpens these conclusions. Raspberry Pi 5 remained usable for lightweight models and for some neural workloads, but recurrent training became slow in the more demanding scenarios. Jetson Orin Nano offered a substantially better operating point for neural models, reducing training time sharply while preserving their advantage in *Type* and *Family*. By contrast, the classical pipelines remained inexpensive on all platforms with executions, which means that the practical value of the Orin-class hardware is concentrated mainly in the neural regime. Jetson Nano should be interpreted differently: not simply as a slower platform, but as a practical viability boundary for the more demanding configurations within the executed setup. In the lighter settings, some models could still make effective use of the available hardware. However, as model and task complexity increased, the platform ceased to provide a useful GPU-backed operating point and the remaining workload was effectively pushed toward much slower CPU-only execution. Under those conditions, Jetson Nano no longer offered a meaningful advantage over Raspberry Pi 5. For this reason, the execution status of every evaluated configuration is reported explicitly, distinguishing successful execution from CPU-only execution, out-of-memory failures, software incompatibilities, and configurations that were not attempted. These distinctions provide a more accurate interpretation of the platform's practical deployment limits than treating unavailable results as equivalent missing values.

Taken together, the results support a tiered deployment perspective within the evaluation conditions considered here. Under those conditions, classical models were the more appropriate option for front-line discrimination, coarse-grained triage, and low-budget edge settings. Neural models, particularly CNN and GRU, became justified when the objective shifted toward finer attribution and the hardware platform could sustain their computational cost. At the same time, the Jetson Nano results show that the relevant boundary is not only whether a configuration can be executed at all, but whether it can still be executed under a hardware mode that preserves a practical advantage. This is a more realistic deployment view than assuming that one model family should dominate the entire EM-based malware classification pipeline.

Recent narrowband EM monitoring results suggest that this tiering can also occur at the sensing level rather than only at the classifier level. In highly constrained settings, a lightweight detector centered on a dominant carrier such as the device clock may be sufficient for first-line anomaly screening, whereas richer learned models remain more appropriate when the objective is finer attribution across malware categories and deployment platforms Oh et al. (2025).

8. Threats to Validity

Several limitations of the present study should be stated explicitly.

Dataset validity. The conclusions depend on the representativeness of the EM corpus used in the experiments. Although the dataset includes benign activity, multiple malware types, family-level labels, and obfuscation-related conditions, it was acquired from a single target device and under a fixed experimental workflow. If the corpus does not cover sufficient diversity in execution contexts, malware variants, packer conditions, or benign behaviors, the results may not generalize to broader operational settings.

Acquisition validity. All traces were acquired from a Raspberry Pi 5 under a controlled setup derived from the EM-Sense workflow. This improves comparability across experiments, but it also means that the study does not evaluate sensitivity to changes in probe placement, target hardware, acquisition circuitry, or environmental interference. The results therefore reflect classifier behavior under one acquisition configuration rather than robustness across heterogeneous sensing conditions.

Dataset-construction and partitioning validity.. The evaluation depends on how the recorded EM traces were segmented and partitioned. In the implemented pipeline, the captured measurements were organized into fixed-length samples of 1000 consecutive values before dataset partitioning. Although the training and test subsets were subsequently preprocessing stages involving parameter estimation were performed independently for each partition, adjacent samples originating from the same execution may still appear in different partitions. As a result, the performance may be optimistic relative to a stricter protocol based on execution-level or acquisition-session separation. Evaluating such protocols remains an important direction for future work.

Validation protocol validity. For the neural-network models, the same held-out subset was used for validation during training and for the final performance evaluation. This protocol was applied consistently across all evaluated architectures to ensure a fair comparison under identical experimental conditions. However, because the final evaluation was not performed on an independent test set, the reported predictive metrics should be interpreted within the context of this experimental design rather than as unbiased estimates of generalization performance.

Measurement validity. The execution times are sensitive to software dependencies, runtime configuration, thermal conditions, and background load, especially on embedded platforms. In addition, the timing values are not homogeneous across model families: for neural models, the time corresponds to per-epoch training cost, whereas for the classical LDA-based pipelines it corresponds to total fitting time. These measurements are therefore informative about practical burden, but they should not be read as variance-estimated or fully normalized cost comparisons.

Internal validity. Classical and neural models differ in training dynamics, hyperparameter sensitivity, and implementation complexity. Although all models were evaluated on the same corpus and under the same scenario definitions, some imbalance in tuning effort or implementation maturity may still affect the comparison. Accordingly, the results should be interpreted as a controlled empirical comparison of representative model families under the executed pipeline rather than as evidence that every architecture has been optimized to its absolute limit or as an exhaustive benchmark of contemporary time-series architectures.

External validity. The findings apply to the evaluated platforms, namely the reference execution environment, Raspberry Pi 5, Jetson Nano, and Jetson Orin Nano, and to the specific acquisition workflow used to build the corpus. Different processors, operating environments, sensing devices, malware populations, or dataset-construction procedures may shift the relative behavior of classical and neural models. In particular, the platform conclusions should not be extrapolated automatically to other embedded systems without additional validation.

Reproducibility. Exact replication may be affected by platform-specific software issues, especially on Jetson Nano, where software-environment constraints and the loss of a practically useful GPU-backed operating mode limited the evaluation of the more demanding configurations. Reproducibility therefore depends not only on releasing the dataset and code, but also on documenting software versions, dependency constraints, and platform-specific execution workarounds. In addition, the present study reports observed runs under a fixed split and does not include repeated measurements across multiple random seeds, so uncertainty estimates remain outside the current scope.

The current evaluation is limited to recordings collected under a common acquisition protocol. Although this enables a controlled comparison between candidate machine learning models and execution platforms, additional acquisition campaigns involving independent recording sessions, probe repositioning, and environmental variability would be required to quantify robustness against acquisition-domain changes. .

9. Conclusion

This paper has presented a controlled cross-platform comparison of classical and neural models for electromagnetic malware classification using a fixed corpus of traces acquired from a Raspberry Pi 5 under a controlled workflow. The evaluation covered four classification scenarios with different label granularities and four compute platforms with clearly different resource budgets.

Within the executed experimental pipeline, the results indicate that model selection in EM-based malware analysis cannot be reduced to predictive accuracy alone. In the coarse-grained tasks, classical LDA-based pipelines consistently provided the strongest reported accuracy–cost balance and remained inexpensive across all platforms with available results. In the finer-grained tasks, neural models, particularly CNN and GRU, became competitive or superior, but at a substantially higher computational cost. Within the present setup, task granularity therefore becomes a central factor in deciding whether the added complexity of the neural branch is operationally justified.

The platform comparison is equally important. Raspberry Pi 5 remained viable for lightweight models and for some neural workloads, but recurrent training became slow in the more demanding scenarios. Jetson Orin Nano provided a substantially better operating point for the neural branch, whereas Jetson Nano marked a practical feasibility boundary for the more demanding configurations. In the lighter settings, some models could still make effective use of the available hardware. However, as model and task complexity increased, the platform ceased to provide a practically useful GPU-backed operating mode and no longer

offered a meaningful advantage over Raspberry Pi 5. These findings show that deployment constraints are part of the substantive result, not merely incidental implementation details.

Taken together, the experiments support a resource-aware view of EM-based malware classification under the evaluation conditions considered here. Within this setup, classical models were the more attractive option for coarse-grained discrimination on low-budget edge hardware, whereas neural models became more compelling when the objective shifted toward finer attribution and sufficient compute resources were available.

The conclusions drawn in this work therefore apply to the controlled acquisition conditions considered throughout the experimental campaign. Extending the evaluation to independent acquisition sessions constitutes an important direction for future work.

Future work should strengthen this comparison along four directions. First, inference latency, memory footprint, and energy consumption should be incorporated as primary deployment criteria alongside the training-time measurements reported here. Second, broader datasets, stricter partitioning protocols, repeated runs across multiple random seeds, and heterogeneous target devices would improve the generality of the conclusions. Third, a practical extension of this work is the study of hybrid pipelines in which classical models perform low-cost triage and neural models are reserved for fine-grained attribution. Fourth, controlled multi-trace aggregation at inference time deserves further study. Prior EM-based malware classification work has shown that averaging repeated measurements of the same binary can markedly improve simpler classifiers such as NB and SVM, especially in the more difficult scenarios, whereas the effect is less straightforward for neural models under variable user environments Pham et al. (2021). It would therefore be useful to determine whether controlled aggregation across repeated executions can strengthen low-cost classification on constrained hardware without removing temporal variation that some models may exploit.

Another practical direction is explicit temporal localization within each sample. Bergstedt reports that EM-based anomaly detection can benefit from dividing traces into sub-intervals, scanning them with sliding windows, and using probability scores to emulate a streaming operational setting, which suggests that only part of a captured trace may carry the strongest discriminative content Bergstedt (2022). Applying a similar analysis here would clarify whether the current 1000-value segments contain shorter subregions that dominate the classification result.

Acknowledgements

This study was funded by INCIBE (National Cybersecurity Institute) through the strategic research project C15.I07.P06.S21 and by the Spanish Ministry of Science and Innovation under project PID2022-139268OB-I00. The authors also acknowledge the support of the University of Malaga and the NICS Lab research group.

Appendix A. Hyperparameter configuration

This appendix summarizes the model configurations that could be confirmed directly from the implementation used in the experimental campaign. Only settings supported by the executed scripts are reported explicitly. Parameters not fixed in those scripts are intentionally left unspecified rather than inferred.

The evaluated models share a common dataset construction and partitioning scheme. Each sample consists of a standardized 1000-dimensional input vector derived from the captured EM measurements, together with a scenario-dependent target label encoded in one-hot form. The train/test split was fixed to 80/20 with `random_state=42`. In the classical branch, the one-hot labels were converted to class indices before fitting, and the number of LDA components was set to $\min(d, C-1)$, where d is the input dimensionality and C is the number of classes. In the neural branch, the MLP used the standardized input vector directly, whereas CNN, RNN, and GRU received the same input reshaped as a one-dimensional sequence of length 1000 with one feature channel per time step.

For readability, the configurations are reported separately for the classical and neural branches. This division reflects the role played by LDA in the classical pipelines and the architecture-specific input formatting used by the neural models.

The neural models were all trained for 300 epochs with batch size 32 and `loss='categorical_crossentropy'`. In the current implementation, the held-out subset was passed as validation data during training. The optimizer was `adam` in all cases except for the GRU model, which used `Adam(learning_rate=0.0005)`.

References

Bergstedt, M.A., 2022. Malware detection using electromagnetic side-channel analysis .

Chawla, N., Kumar, H., Mukhopadhyay, S., 2021. Machine learning in wavelet domain for electromagnetic emission based malware analysis. IEEE Transactions on Information Forensics and Security 16, 3426–3441.

Clark, S.S., Ransford, B., Rahmati, A., Guineau, S., Sorber, J., Xu, W., Fu, K., 2013. {WattsUpDoc}: Power side channels to nonintrusively discover untargeted malware on embedded medical devices, in: 2013 USENIX Workshop on Health Information Technologies (HealthTech 13).

Ding, F., Li, H., Luo, F., Hu, H., Cheng, L., Xiao, H., Ge, R., 2020. Deeppower: Non-intrusive and deep learning-based detection of iot malware using power side channels,

Table A.8: Configuration of the classical machine-learning pipelines. All pipelines use standardized input vectors followed by LDA with dimensionality $\min(d, C-1)$ before classifier training.

Model	Classifier	Implementation
LDA+NB	Gaussian Naïve Bayes	`GaussianNB()` with the default scikit-learn parameters.
LDA+SVM	Support Vector Machine	`SVC(kernel='rbf', C=1, gamma='scale', probability=True)`.
LDA+RF	Random Forest	`RandomForestClassifier(n_estimators=100, random_state=42)`.
LDA+XGBoost	Extreme Gradient Boosting	Binary tasks: `objective='binary:logistic'`, `eval_metric='logloss'`. Multiclass tasks: `objective='multi:softmax'`, `num_class=C`, `eval_metric='mlogloss'`. Common parameters: `n_estimators=300`, `max_depth=6`, `learning_rate=0.05`, `subsample=0.8`, `colsample_bytree=0.8`, `tree_method='hist'`, `random_state=42`.

Table A.9: Neural-model configurations used in the experimental evaluation. N denotes the number of output classes in the corresponding classification scenario.

Model	Input representation	Architecture	Optimizer
MLP	Standardized vector	`Dense(128, ReLU)` → `Dropout(0.3)` → `Dense(64, ReLU)` → `Dropout(0.3)` → `Dense(`N`, softmax)`	`Adam`
CNN	1D sequence	`Conv1D(128, 3, ReLU)` → `MaxPooling1D(2)` → `Dropout(0.4)` → `Conv1D(64, 3, ReLU)` → `MaxPooling1D(2)` → `Dropout(0.3)` → `Conv1D(32, 3, ReLU)` → `MaxPooling1D(2)` → `Dropout(0.3)` → `Flatten` → `Dense(128, ReLU, L2)` → `Dropout(0.4)` → `Dense(64, ReLU, L2)` → `Dropout(0.3)` → `Dense(`N`, softmax)`	`Adam`
RNN	1D sequence	`Bidirectional(LSTM(128))` → `Dropout(0.3)` → `Dense(64, ReLU)` → `Dropout(0.3)` → `Dense(`N`, softmax)`	`Adam`
GRU	1D sequence	`GRU(128, return_sequences=True)` → `Dropout(0.4)` → `GRU(64, return_sequences=True)` → `Dropout(0.3)` → `GRU(32)` → `Dropout(0.3)` → `Dense(128, ReLU)` → `Dropout(0.4)` → `Dense(64, ReLU)` → `Dropout(0.3)` → `Dense(`N`, softmax)`	`Adam` (5×10^{-4})

Table A.10: ROC-AUC and PR-AUC obtained by the classical machine learning models across all evaluated platforms for the binary classification scenarios.

		LDA+NB		*LDA+SVM*		*LDA+RF*		*LDA+XGBoost*	
Platform	*Scenario*	*ROC-AUC*	*PR-AUC*	*ROC-AUC*	*PR-AUC*	*ROC-AUC*	*PR-AUC*	*ROC-AUC*	*PR-AUC*
Virtual Machine	Packer	0.86	0.75	0.92	0.83	0.89	0.76	0.89	0.76
	Virtualized	0.74	0.74	0.83	0.89	0.74	0.77	0.74	0.77
Raspberry Pi 5	Packer	0.86	0.75	0.92	0.83	0.89	0.76	0.89	0.76
	Virtualized	0.74	0.74	0.83	0.89	0.74	0.77	0.74	0.77
Jetson Nano	Packer	0.86	0.75	0.92	0.83	0.89	0.76	0.89	0.76
	Virtualized	0.74	0.74	0.83	0.89	0.74	0.77	0.74	0.77
Jetson Orin Nano	Packer	0.86	0.75	0.92	0.83	0.89	0.76	0.89	0.76
	Virtualized	0.74	0.74	0.83	0.89	0.74	0.77	0.74	0.77

in: Proceedings of the 15th ACM Asia conference on computer and communications security, pp. 33–46.

Guo, J., Xu, Y., Zhang, M., Huang, W., Guo, H., 2024. Behavior recognition and anomaly detection utilizing memory electromagnetic emanation, in: MILCOM 2024-2024 IEEE Military Communications Conference (MILCOM), IEEE. pp. 530–535.

James, G., 2013. An introduction to statistical learning with applications in r.

Khan, H.A., Alam, M., Zajic, A., Prvulovic, M., 2018. Detailed tracking of program control flow using analog side-channel signals: a promise for iot malware detection and a threat for many cryptographic implementations, in: Cyber Sensing 2018, SPIE. pp. 23–36.

Khan, H.A., Sehatbakhsh, N., Nguyen, L.N., Callan, R.L., Yeredor, A., Prvulovic, M., Zajić, A., 2019a. Idea: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems. IEEE Transactions on Dependable and Secure Computing 18, 1150–1163.

Khan, H.A., Sehatbakhsh, N., Nguyen, L.N., Prvulovic, M., Zajić, A., 2019b. Malware detection in embedded systems using neural network model for electromagnetic side-channel signals. Journal of Hardware and Systems Security 3, 305–318.

Nazari, A., Sehatbakhsh, N., Alam, M., Zajic, A., Prvulovic, M., 2017. Eddie: Em-based detection of deviations in program execution, in: Proceedings of the 44th Annual International Symposium on Computer Architecture, pp. 333–346.

Oh, M.J., Danuor, P., Ju, G.D., Jung, Y., Kwon, K., Jung, Y.B., 2025. Enhanced em-based malware detection for iot devices using clock frequency analysis. IEEE Transactions on Computers .

Pham, D.P., Marion, D., Mastio, M., Heuser, A., 2021. Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification, in: Proceedings of the 37th Annual Computer Security Applications Conference, pp. 706–719.

Prvulovic, M., Zajić, A., Callan, R.L., Wang, C.J., 2017. A method for finding frequency-modulated and amplitude-modulated electromagnetic emanations in computer systems. IEEE Transactions on Electromagnetic Compatibility 59, 34–42. doi:10.1109/TEMC.2016.2603847.

Raffetseder, T., Kruegel, C., Kirda, E., 2007. Detecting system emulators, in: International Conference on Information Security, Springer. pp. 1–18.

Sehatbakhsh, N., Nazari, A., Alam, M., Werner, F., Zhu, Y., Zajic, A., Prvulovic, M., 2019. Remote: Robust external malware detection framework by using electromagnetic signals. IEEE Transactions on Computers 69, 312–326.

Sehatbakhsh, N., Nazari, A., Zajic, A., Prvulovic, M., 2016. Spectral profiling: Observer-effect-free profiling by monitoring em emanations, in: 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE. pp. 1–11.

Wang, X., Zhou, Q., Harer, J., Brown, G., Qiu, S., Dou, Z., Wang, J., Hinton, A., Gonzalez, C.A., Chin, P., 2018. Deep learning-based classification and anomaly detection of side-channel signals, in: Cyber Sensing 2018, SPIE. pp. 37–44.

Welagedara, A., 2025. Fileless malware detection using electromagnetic side-channel analysis.

Yokoyama, A., Ishii, K., Tanabe, R., Papa, Y., Yoshioka, K., Matsumoto, T., Kasama, T., Inoue, D., Brengel, M., Backes, M., et al., 2016. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion, in: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer. pp. 165–187.