Computer Modeling in Engineering & Sciences

Tech Science Press

https://doi.org/10.32604/cmes.2026.078451

ARTICLE

Towards Robust Malware Detection with a Multiclass Dataset for Intelligent Learning

Amjad Hussain[1,*], Ayesha Saadia[2,*], Chihhsiong Shih[3], Nazish Nawaz[2], Amir H. Gandomi[4,*] and Khursheed Aurangzeb[5]

[1]Department of Cyber Security, Main Campus, Air University, Islamabad, Pakistan
[2]Department of Computer Science, Main Campus, Air University, Islamabad, Pakistan
[3]Department of Computer Science, Tunghai University, Taichung City, Taiwan
[4]Faculty of Engineering & Information Technology, University of Technology, Sydney, NSW, Australia
[5]Department of Computer Engineering, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

*Corresponding Authors: Amjad Hussain. Email: 220324@students@au.edu.pk; Ayesha Saadia. Email: ayesha.saadia@students.au.edu.pk; Amir H. Gandomi. Email: gandomi@uts.edu.au

Received: 31 December 2025; Accepted: 19 March 2026; Published: 27 May 2026

ABSTRACT: Malware has evolved from the early Creeper virus into highly sophisticated and organized cyber threats. Over time, it grew in sophistication, adopting advanced techniques, stealth tactics, and autonomous propagation. Modern malware leverages encryption, obfuscation, zero-day exploits, and AI-assisted techniques to conduct stealthy and persistent attacks. Classification of its exact family is the end goal to defend and mitigate the latest attacks. Researchers have contributed significantly and introduced many techniques to tackle malware threats. Binary detection is performed at a large scale, but very little in multi-class classification. In this research, a hybrid technique is proposed by combining a sandbox with AI models to extract hidden patterns and classify its category and family with high accuracy. A dataset (AU-PEMAL-2025) is prepared, which includes 10,839 records of 26 malware families. Five ML and three DL models are trained on the newly created dataset to validate its effectiveness. The ML classifiers achieved the highest accuracies of 0.9945, 0.9788, and 0.9485, while the DL models achieved 0.9932, 0.9591, and 0.9286 accuracies with minimal losses in detection and multi-class classification of category and family, respectively. Our findings reveal that the proposed approach can efficiently detect the obfuscated malware variants and safeguard organizations from unseen malware threats.

KEYWORDS: Malware dataset; ransomware dataset; malware classification; malware family identification

1 Introduction

The Internet invention helped the world to communicate, share, and manage resources remotely [1]. At the same time, cyber criminals use it to extend criminal activities in a digital way, and the distribution of malicious software has become easy [2]. A new term for malicious software came into existence as malware, which is a combination of two words: "Mal", taken from the first three characters of malicious, and "ware", taken from the last three characters of software. Any malicious software, used by cybercriminals to harm a system or user, is called malware [3].

In 1989, the first digital crime came to notice when Joseph Popp, an AIDS researcher, mailed 20000 infected Floppy disks to WHO AIDS conference attendees. This digital crime was called AIDS Trojan [4]. Time passed, malware grew, and changed its shape into many categories, like Ransomware, Viruses, Trojans,

Bots, Spyware, RAT, Worms, Rootkits, Information Stealer, etc., and the categories further changed into families. Each malware category and family has unique characteristics and functionalities to harm a system [5]. Some malware categories encrypt important data or lock the victim's access to the system, such as ransomware [6], while others can permanently delete important data or physically damage a system [7]. Some categories can be used to launch an attack on a target indirectly by utilizing the online available vulnerable sources, which are called zombies or bots. It is essential to know the functionality of different categories and families to mitigate these attacks in a better way [8].

We must identify malware as soon as it attacks the computer systems and gain full control or cause damage to important data. The process of analyzing a suspicious file to determine if it contains malware or not is known as malware detection [9]. Classifying malware goes one step further, and once a file has been determined to be malicious, the process of categorizing malware involves identifying the specific malware category and family [10]. Various disciplines and techniques, like data science, machine learning, and heuristics, together with technology like cloud computing, big data, and blockchain, are employed in these procedures to boost the detection rate. Various methods exist for detecting malware that employs the techniques above and technology [11].

The first method to detect malware is by analyzing the malicious software statically without running it on a live system [12]. The header of the file, the strings it contains, the Dynamic Link Libraries (DLLs) it imports or exports, and some other characteristics are analyzed. The critical point in static analysis is that it is analyzed without executing the software, and information is captured safely [13]. However, without executing malware, we cannot get the proper functionality of malicious software and cannot get complete information about malware if it is obfuscated and performs some action when it gets internet or some other type of command [14]. To investigate and explore the functionalities of malware, dynamic analysis is performed [15]. In dynamic analysis, the system is set up in such a way that, before executing the malicious software, snapshots of the registry, processes, threads, and network activities are taken as the first part, then the monitoring tools are run to capture important information. After having the first snapshot and monitoring tools in running mode, the malicious software is run and watched [16]. Dynamic analysis is performed for a specific time, and after that time, a second snapshot is taken in which the changes made are compared with the previous snapshot. New registry entries, processes, and threads it started or stopped, files it created, deleted, or modified, and network activities it performed are analyzed. Other important tools, such as Suricata, are also utilized to capture more features and to understand the functionality of malware [17]. In advanced dynamic analysis, it is reverse-engineered to understand its working in a better way at the code level, and the malware code is analyzed line by line to see its workflow [18]. Thousands of the latest malware types are introduced every day, and it is impossible to analyze every malicious software separately one by one; moreover, after executing malicious software on a machine, it needs to be reset to its original state to run the next malicious software [19,20]. It also has limitations that require it to run in a sandbox-based controlled environment, and network functionality is disabled to avoid any dangerous activity it performs on a live system [21].

To find and stop malware, security experts have suggested several signature-based and behavior-based ML practices [22]. First approach uses distinct patterns or traits of known malware to locate and stop it [23], and the second approach checks how software behaves in order to identify any uncommon activity that may flag the presence of malware [24]. These approaches rely on matching every signature that has been retrieved from the current file to every signature that has been gathered and verified previously in the malware database. Whether the present file is malicious relies on how similar the signatures are [25]. The main problem with this approach is that it can only identify known malware variants that have already been identified because it depends on previously gathered and verified malware signatures. However, these

strategies are no longer thought to be successful due to the constant and ever-increasing varieties of malware [26]. As such, the current study has shifted its attention to malware identification systems that depend on machine learning [27].

As new malware variants are being introduced every day and the complexity of manual detection increases, it is difficult to analyze each malicious software one by one and is also time-consuming [28]. In addition, these traditional detection techniques failed to detect and classify real-world malware samples [29]. ML and DL have helped a lot in this field and have automated the detection and classification. Anti-virus software organizations have adopted it to detect malware at large [30]. Deep learning is an artificial intelligence area that derives from artificial neural networks (ANNs) and learns from examples [31]. While DL has been applied extensively in domains like voice control, image processing, and autonomous vehicles, it has not been sufficiently applied to malware detection and categorization [32]. The deep learning-based detection method significantly shrinks the feature dimension while operating at a high-performance level [33]. The rapid trend of advanced cyber threats has compelled a change from old traditional methods like signature-based detection to more powerful visualization-based analysis. Early researchers demonstrated the effectiveness of malware samples conversion into grayscale images, which enables the use of computer vision methods to recognize malicious structural patterns [34]. Deep learning structures like Convolutional Neural Networks (CNNs) have been integrated recently, which automate the feature extraction to work on large datasets [35]. However, the researchers start focusing on texture-based approaches for investigation due to problems of imbalanced datasets and the high cost of deep learning computing. Some researchers used optimization algorithms by drawing inspiration from nature to improve the classification boundaries in imbalanced environments [36]. Granulometry Analysis (GRA) based GRASE framework was introduced to handle these structural and high computational problems. The GRACE measures the size and density of code patterns efficiently by performing a multi-scale texture-based analysis of malware images by implementing morphological operations. The framework implements an Image Statistical Similarity Parameter (ISSP) set to provide a detailed mathematical depiction of malicious variants. It includes important metrics like PSNR, Normalized Cross-Correlation, and MSE. The framework also implements a hybrid strategy, namely the Semi Eager (SemiE) classifier, to maximize the trade-off between training effectiveness and local pattern robustness, in contrast to purely eager or lazy learning models. The development of such lightweight and highly discriminative systems for the detection of real-time malware has advanced toward hybrid classification and statistical texture analysis [37].

A lot of malware classification techniques have been proposed and applied to existing old datasets or by targeting old malware variants. In most works, multi-class classification on the latest known-to-world malware variants is also not performed. This is the motivation to create a dataset of the latest malware variants, extract important hidden functionalities, create a dataset with robust features, and validate it by training not only ML-based algorithms but also DL-based algorithms to perform binary as well as multi-class classification. The following contributions are made in this research:

- Collect and analyze the latest real-world malware variants dynamically and gather important features.
- Prepare a new dataset (AU-PEMal-2025) with robust features by employing feature engineering techniques with a combination of self-made analysis techniques.
- Validate the newly created dataset by training ML-based and DL-based algorithms to check and see if the selected features helped in identifying different malware variants.
- Perform binary, category-wise, and family-wise classification with high accuracy.

This work is further structured so that a review of earlier studies on malware detection employing ML-based and DL-based techniques is given in Section 2. A methodology for malware detection, classification, and dataset preparation is provided in Section 3. The experimental investigation, practical application, and

the results are reviewed, debated, and contrasted with similar work in Section 4. In Section 5, the conclusion and possible future research directions are presented.

2 Background and Related Work

2.1 Background

As soon as the malware was made public, security professionals and researchers also started work to identify and neutralize it using various static, dynamic, signature-based, and pattern-based techniques [38]. As time passed, there was a surge in malware assaults and new variants, necessitating the use of an automated system to identify new malware variations. ML-based and DL-based techniques have shown remarkable effectiveness when the right features are chosen and input into the training process [39]. Understanding the most significant aspects of malware is required for machine learning and to recognize and categorize malicious files; the correct pattern and behavior must be analyzed [40]. As the main goal of this study is the identification of malware in Windows, it is crucial to comprehend how portable executable (PE) binary files operate. The PE file structure is shown in Fig. 1. The OS loader receives information from the data structure of the file format on how to unfold the enfolded executable. This offers links to TLS data, resource management data, import tables, Application Programming Interface (API) export, and dynamic libraries [41].

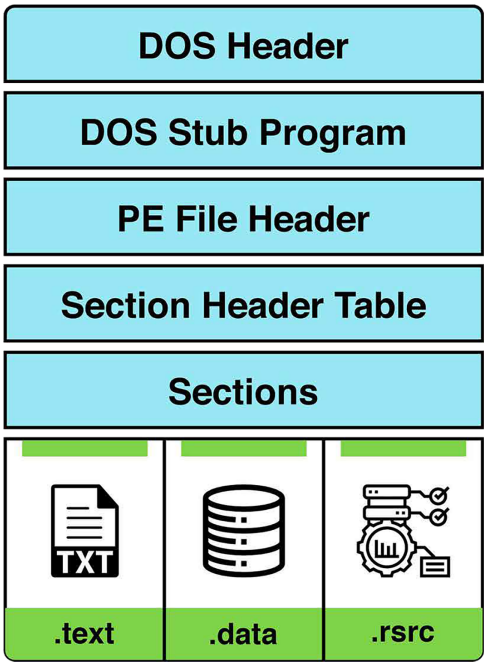

Figure 1: PE file structure.

DOS Header: For every PE file, the first thing is the DOS header that begins with 64 bytes of data. It includes details related to the machine, sections, time and date stamp, the number of symbols, the size of the optional header, the reference to the symbol table, the signature, and the characteristics [42].

DOS Stub Program: A stub is a little code or program that runs automatically when an application starts [43]. When an application is incompatible with Windows, this stub shows the warning message that the program cannot be run in DOS mode. If a Win32-based product is executed in a non-Win32 environment, this function generates a useful error message [44].

PE File Header: Magic in the PE file header determines whether an executable is 32-bit or 64-bit. AddressOfEntryPoint, the location at which the Windows loader will start operating, includes the Relative Virtual location (RVA) of the module's Entry Point (EP) and is normally contained in the.txt file [45]. The RVAs for the start of the code and data sections are kept by the BaseOfCode and BaseOfData elements. ImageBase is the address from which an executable file will be memory-mapped to a specific memory area. SectionAlignment and FileAlignment refer to the alignment of PE sections in memory and files, respectively. The SizeOfImage property specifies the amount of memory utilized by the PE file during execution. The Subsystem field of an executable file indicates the target subsystem [46,47].

Section Header: The Section Header table carries details about the various components of an executable file image. The first is SizeOfRawData, which gives the actual size of the file section [48]. The VirtualSize property indicates the size of the segment in memory. PointerToRawData is the file offset at which the Raw Data section begins. The RVA of the region in memory is VirtualAddress [49]. The RVA for the next section may be determined using the VirtualAddress and VirtualSize values. The characteristics specify the memory access rights for the region of memory that was supplied as flags [50].

Sections: The section names in PE file section headers are specified using a simple character array field called Name. It has a.txt file, which is usually the first part and includes the application's executable code [51]. This part also contains the application's entry point, which is the location of the first application instruction that will be executed. Following that is.data, which includes an application's initialized data, such as strings. The third and most crucial aspect is that.rdata or.idata are used for the sections containing the import table, and this table lists the Windows APIs that the program makes use of. Other sections include.reloc, which includes information about the address of the relocation table,.rsrc, which is a popular term for the resource-container section, which holds pictures used for the application's UI; and .debug, which provides debug information [52,53].

One of the most fundamental principles in comprehending any malware binary is the PE format. It provides a wealth of information on any executable, whether developed for a 32-bit or 64-bit platform, a DLL file, or any other file.

2.2 Literature Review

In this research, we conducted a literature review by analyzing the problem addressed, PE binaries collected, malware variants used, techniques applied, features selected, proposed solutions, ML-based and DL-based models implemented, and accuracies achieved.

In [54], the author proposed the BODMAS dataset, which had 57,293 malicious and 77,142 benign records. A combination of Sequential Feature Selection and a derivative of Forward Feature Selection was applied, and 25 features were selected to train the models. Gaussian Naïve Bayes (GNB), Local Discriminant Analysis (LDA), Logistic Regression (LR), Multilayer Perception (MLP), K-Nearest Neighbor (KNN), Gradient Boosting (GB), and Random Forest (RF) algorithms were trained using selected features. Models achieved 99.38% accuracy using the full set of features and 99.56% accuracy using the selected 25 features in detecting malware. In multi-class classification, the models achieved 97.59% accuracy on the full set of features and 97.69% accuracy on the selected set of features to categorize malware. In [55], the author utilized the BIG15 dataset to train the LR, ANN, and Convolutional Neural Network (CNN) models. Pertained models InceptionV3 and Long Short-Term Memory (LSTM) were also trained on the dataset to create the malware signature as a 2D representation. The InceptionV3 produced 98.76% accuracy on the test and 99.6% accuracy on the training set. In [56], the author proposed a DL-based technique to classify malware by utilizing Word2Vec for feature extraction, GB classifier for classification with k-fold cross-validation. The model achieved 96% accuracy on the provided set of samples from Microsoft dataset files.

In [57], the author utilized an ML and DL-based hybrid approach to classify malware on the BIG15 dataset by Microsoft. The Principal Component Analysis (PCA) technique was employed to extract features from malware samples. Several ANN algorithms in conjunction with SVM and KNN with k-fold cross-validation were applied to assess the effectiveness. The models achieved 96.6% accuracy with KNN, 94.6% with SVM, 95.6% with static, and 95.5% with dynamic feed-forward networks. In [58], the author proposed a Deep Residual Network (DRN) based malware identification approach with 18 layers as an alternative to feature extraction. The publicly available Malimg dataset was utilized for the experiment. The model obtained an 86% classification accuracy on average by using k5-fold cross-validation. In [59], the author proposed a memory forensics-based malware identification method in a controlled virtualized environment to collect the dump from volatile memory and turn it into a specific picture using PE. ML classifiers (SVM, RF, DT, and XGB) were trained on the modified pictures with dimensions of 112 × 112 and 56 × 56. The models achieved 97.01% accuracy on the provided data to detect malware.

In [60], the author applied a memory forensics-based technique for obtaining memory-based characteristics to help with malware identification and categorization. According to the authors, the characteristics that were extracted might be utilized to expose the malware's actions, such as process and DLL injection, activities with command and control, and obtaining the ability to carry out certain activities with elevated privileges. Features were converted into binary vectors using feature engineering, and an SVM classifier was trained. The model achieved 98.5% accuracy with a 1.24% false positive rate. Additionally, they created a memory-based dataset that includes 966 benign samples and 2502 malware files. They have made this dataset publicly available for use in future studies. In [61], the author proposed a file-less malware detection technique by first executing malware samples in a VM, then taking a snapshot of the VM and extracting a memory dump. The features were extracted from the collected memory dumps, a dataset was created, and fed to ML classifiers (DT, LR, KNN, SVM, GB, and XGB) for training and testing. RF classifier secured the highest accuracy of 93.33% in malware identification. In [62], the author extracted static features from the PE file and trained RF, SVM, DT, AdaBoost, GNB, and GB classifiers. The highest accuracy of 99.44% was achieved by the RF classifier among all applied classifiers.

In [63], the author extracted API call information in three formats. They collected the API call usage, frequency of these calls, and sequences of these calls to create feature sets. The TF-IDF technique was applied to enrich the feature sets. Different ML classifiers were trained on the selected feature set, and they achieved 99.6% of accuracy on the crafted API dataset in detecting malware. In [64], a ransomware detection and classification technique, namely BigRC-EML, was proposed using ensemble machine learning. The PCA feature engineering technique was utilized to reduce the feature dimensionality. The author utilized two datasets, in which the first dataset contains 842 benign and 582 ransomware samples, while the second dataset contains 500 applications. The ML classifiers (SVM, RF, KNN, XGB), along with the Neural Network (NN), were trained and tested. The NN performed well and achieved 98% accuracy on the selected features. In [65], the author proposed a hybrid ML-based and image analysis-based method to identify malware. Using the clustering technique and the AVClass program, the gathered binary samples were labeled. Grayscale pictures of maliciously labeled samples were produced in order to retrieve the global and local textural aspects. Global features were retrieved using HOG, GIST, and Hu Moments, while textural features were extracted by utilizing ORB, KAZE, and SIFT descriptors. In order to identify low-dimensional elements and create a local feature map of grayscale malware pictures, the bag of visual words (BoVW) method was implemented. The ML classifiers (SVM, NB, RF, ExtraTree (ET), and KNN) were trained on the stacked feature maps derived from various picture descriptors. For the experiment, 690 real-world malware samples from 22 families and the Mallmg dataset were utilized on honeypots. The models achieved 98.23% accuracy.

In [21], malware classification and identification were performed by analyzing malware samples dynamically and then feeding extracted data to ML-based and DL-based models. A total of 2576 malicious and 1080 goodware samples were collected from online sources and analyzed in the Cuckoo sandbox to generate reports, and the generated reports were parsed into a comma-separated-value (CSV) file. The CNN, RNN, SVM, KNN, XGB, RF, and GBC classifiers were trained, and the highest accuracy of 99% was achieved by RNN. In [66], the author dynamically analyzed the malware samples collected from open sources and created a dataset of API calls. Multiple ML classifiers along with DL models were trained on the dataset. The XGB classifier achieved 88.26% accuracy, and the MLP achieved 90.70% in detecting malware. In [67], the author proposed a statistical classifier to identify malware by first investigating and analyzing the usage rate of API sequence calls in both malware and benign samples. Two publicly available datasets were utilized for training the models. The first dataset contains 252 malicious and 200 goodware API calls, while the second dataset contains 278 different API calls shared by different malware families. The models secured 97% of accuracy in detecting malware.

In [68], the author collected ransomware API calls reports from Cuckoo Sandbox and trained two text-based BERT and LSTM models. A total of 1990 normal and 669 ransomware samples were gathered from online sources. To make it balanced, 535 goodware and 535 malicious samples were utilized for model training. The models achieved 97% accuracy with 0.16007 loss on 5 epochs, while 99.62% accuracy with 0.0442 loss on 10 epochs. The author declared an average of 95% accuracy in detecting ransomware. In [69], the author collected a total of 7398 malicious and 7462 benign samples from an anti-virus company. Cuckoo Sandbox was utilized for dynamic analysis, extracting API calls, names, categories, arguments, return values, and call frequency. A vector building module was implemented to produce a labeled dataset. The LightGBM, RF, XGB, CatBoost, ReNet1D, KNN, EfficientNetB0, and LR models were trained. The LightGBM achieved 97.42% accuracy in identifying malware. In [70], an obfuscated malware dataset, namely CIC-MalMem-2022, was created using VolMemLyzer, a publicly available framework for feature extraction. The dataset comprised three malware categories and fifteen families. Ensemble learning models, RF, SVM, DT, Linear Perception (LP), KNN, and NB were trained and validated on the proposed dataset and they achieved an accuracy of 99%.

2.3 Public Malware Datasets for Machine Learning-Based Detection

EMBER [71] a large-scale dataset specialized for Windows Operating System-based malware detection by analyzing static features of PE files. The static elements include byte-level histograms, header metadata, information about sections, import and export tables. EMBER is mostly used for traditional machine learning models; opcodes however, due to its dependencies on static features, are vulnerable to packing and techniques like obfuscation, which reduce their resistance to modern evasion techniques by malware creators. SOREL-20M [72] is another large-scale publicly available dataset with more than 20 million malicious and benign PE files. This dataset also relies on static features like PE structure and metadata; however, this dataset is very useful for malware detection due to its scalability and class imbalance. It is limited to binary detection and does not provide behavioral insights, which are very important for the identification of malware variants that utilize obfuscated and polymorphic techniques. The BODMAS dataset [73] made it possible for fine-grained classification tasks by providing labeled malware samples at the family level. The dataset consists of static PE features, including structural data and opcode that are taken from PE files. Although the dataset is very helpful for researchers to classify malware families on the basis of static features, however, due to non-availability of dynamic and behavioral features, it is limited. CIC-MalMem-2022 dataset [70] is the most robust and advanced dataset, which targets evasive and obfuscated malware families. It introduced memory-based dynamic features, including loaded modules, memory access behavior,

and memory allocated patterns. Memory-based artifacts rather than disk-based features make it more robust against packing and encryption techniques.

The summary of the characteristics and limitations of publicly available datasets and to position the proposed dataset within the current research landscape, Table 1 presents a comparative overview of widely used malware datasets.

Table 1: Comparative overview of public malware datasets used in recent research.

Dataset	Analysis Type	Feature Source	Samples	Label Granularity	Obfuscation/ Evasion Focus	Notes
EMBER 2024 [71]	Static	PE metadata, byte histograms	3.23M	Binary	–	Large-scale static benchmark
SOREL-20M 2020 [72]	Static	PE feature	20M	Binary	–	Industry-scale dataset
BODMAS 2021 [73]	Static	Byte-level, metadata	134,434	Family	–	Family-level static dataset
CIC-MalMem-2022 [70]	Dynamic	Memory features	2916	Binary/Family	Yes	Obfuscation-resilient
AU-PEMAL-2025	Hybrid	Static + Dynamic	21,703	Binary/Category/ Family	Yes	Multi-level, hybrid analysis

Although there are many large-scale datasets available, most of these only support binary or generic classification and use static, API sequence-based, or dynamic features. The need for a hybrid dataset that can support not only binary classification but also category-wise and family-wise identification of obfuscated and modern malware variants is largely observed. In this work, a benchmark dataset, AU-PEMAL-2025, variants, is introduced by selecting highly important static as well as dynamic features of malware binaries. Static features provide structural and code-level characteristics, while the dynamic features focus on runtime behaviors, API interactions, network logs, and execution patterns. The proposed dataset allows for malware detection, its category identification, and family attribution by enabling a systematic assessment across a variety of malware analysis tasks. This approach increases generalization across all malware variants and strengthens resistance to obfuscation techniques.

3 The Proposed Methodology

The proposed methodology is presented in this section with detail discussion on sample collections to dataset creations and from preprocessing to binary, category-wise, and family-wise malware classification. Fig. 2 presents the proposed methodology, which comprises five steps: sample collection, dynamic analysis, dataset creation, preprocessing, ML and DL models implementation, and result evaluation.

3.1 Samples Collection

In this work, a total of 15,500 benign and 15,500 malware samples were gathered from publicly available platforms such as VirusShare, Any.run, VirusTotal, GitHub, and MalwareBazar. The samples include 4 malware categories and 26 families. All samples were 32-bit PE files containing verified benign and malicious variants. The main focus while collecting samples was to gather the latest verified malware variants to address real-world problems and create a latest malware dataset. Therefore, samples of mainly 4 malware categories (Remote Access Trojan (RAT), Ransomware, Information Stealer, and Trojan Horse) were collected and categorized separately. Previously, in [74,75], the experiments were performed on different features of the

collected malware samples. To further explore and enhance the performance, this research is conducted with more robust techniques and features.

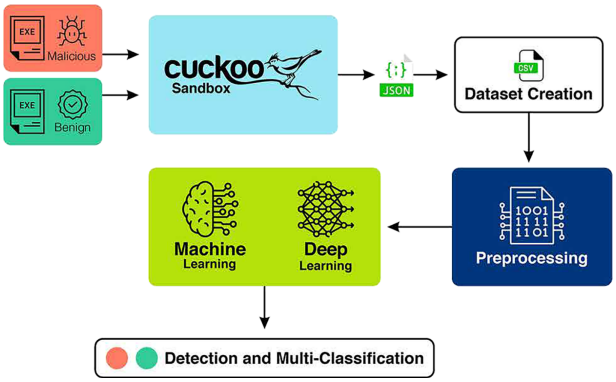

Figure 2: Proposed methodology.

3.2 Malware Samples, Categories, and Families

Collection: Malware samples were gathered from multiple publicly available repositories and threat intelligence sources. Each sample was stored in a dedicated folder corresponding to its malware family. **Family Assignment:** Samples in each folder represent a single malware family. Families were verified using public threat reports and documented behavioral patterns.

Category Mapping: Based on behaviors seen during sandbox execution and threat intelligence documentation, families of malware were categorized into more general categories (such as Trojan, Worm, Adware, and Ransomware).

Quality Control: Each sample was examined during dynamic analysis to make sure that the behavior observed correctly aligned with the exact category and family. To minimize and avoid label noise, samples that had unclear presentation or had inconsistent behaviors were removed or marked as low-confidence.

Statistics: The AU-PEMal-2025 has a total of four categories and 26 malware families with 10,839 malicious records. To facilitate the researchers' performance of classification at the family level, each family is assigned many variations.

3.3 Benign Sample Collection and Diversity

We tried our best to make sure that the benign samples in the AU-PEMal-2025 dataset are from real-world application software and not malicious. Diversity and verifications were in focus when collecting executable files from online sources. Samples were also verified on VirusTotal to check if any antivirus software detected them as malicious. We collected benign samples from official software repositories like Microsoft Windows System files, open-source applications, installer packages, system utilities, productivity software, media players, development tools, and software distributions that researchers widely use. We also made sure to add both executable and DLL files to check and see the different behaviors during execution in the sandbox. Only those files were included that were found to be safe by the antivirus engines. Benign samples checked on VirusTotal were executed in a sandbox, analyzed, and were confirmed that these files were using the registry, making network requests, accessing, and modifying files, but these were legitimate files, not malware. These files did not show any malware-specific behaviors; however, they produced many important behavioral traces. This methodology of benign sample collection and diversity checking makes

sure that the goodware files are from real-world applications and show realistic behavior patterns, and stops models from easily differentiating malicious samples from benign based on dataset artifacts only.

3.4 Malware Category and Family Labeling

We manually sorted the files into their respective category-wise and family-wise folders by checking publicly available threat reports and vendor-specific descriptions. Each folder had samples from a unique and specific family. Samples in each family had different hashes, different file types, and different propagation techniques. Each family had multiple versions, variants, and was grouped in a specific category, like Trojan, Ransomware, Spyware, based on the functionality and threat intelligence sources. Each sample was also tested on VirusTotal and Any.run to verify its family and category. After manually categorizing, the samples were run dynamically to check and make sure that their behavior matches the category and family. Unclear and inconsistent nature of samples was marked as low-confidence and were removed from the folders to reduce label noise. The technique helped us to make sure that categories and families of malware are accurate, consistent, and exactly match the behavioral features that were recorded during dynamic analysis.

3.5 Dynamic Analysis

After verifying and putting all samples in separate folders, the next task was to analyze each sample dynamically by allowing it to perform its functionality. HP Workstation Z230 with Core i7 processor (4790) with 16 Giga Byte Random Access Memory (RAM), and NVidia Quadro 4GB Graphic Processing Unit (GPU) was set up separately for all experimental tasks. Ubuntu 18.04.3 LTS with Cuckoo Sandbox 2.0.7 and Windows 7 (32-bit) as a guest OS was configured. Suricata, an important and powerful tool to analyze network behavior, was configured with a backend virtual machine to act as a DNS Server. Each malware sample was run one by one for 500 s in a controlled space to analyze its true functionality and behavior. Modern malware variants are aware of the environment and assess the guest OS and to reduce the chances of evasion, the following settings were configured:

Evasion-Aware Configuration: The sandbox environment was set up in such a way that it looks like a real guest OS. Hard Disk partitions were set up to look like normal systems, common software like Microsoft Office, Skype, and Notepad++ were installed to perform operations like system file modification, registry entries, and network activities. This type of setting makes it difficult for the malware to find the virtualized environment easily. All system artifacts, such as OS version, CPU, and RAM, were set up to resemble the normal machine used by a normal user.

Behavioral Coverage: In dynamic analysis, malware behavior was analyzed to see what type of processes it generates, what files it manipulates, what registry modifications it makes, what network communications it tries to perform, what logs it tries to generate and modify, and which DLL files it imports or exports. Some malware variants can take a long time to execute, like logic bombs; however, in recent research, it was shown that 500 s is sufficient time to start the functionalities.

Quality Control: Many samples did not run or showed no activity during execution. These samples were either 64-bit or corrupted files, so these suspicious samples were removed to reduce the label noise. Static features of these files were also removed to avoid the missing value problem in the dataset.

This methodology helped to make sure that the dataset contains all typical malware behaviors and features for strong model training and evaluation. A summary of each malware sample was generated from Cuckoo Sandbox and saved as a JavaScript Object Notation (JSON) file. Each JSON file contains static artifacts and behavioral features like registry operations, file operations, network activities, DLL imports, exports, and API call sequences.

Limitations: It is obvious that some modern and most sophisticated malware variants that are aware of such a controlled environment or some malware variants that rely on payloads which takes long time to arrive, might not be found using this technique. However, most of the typical and obfuscated malware variants are focused on in this research.

Malware samples collected and analyzed in this work were primarily collected in 2023; however, Windows 7 was selected as the sandbox operating system due to its compatibility with Cuckoo Sandbox and other automated malware analysis frameworks, as well as its stability. Windows 7 provides an environment with fewer built-in security restrictions that allow malware to execute, and it enables greater observation of behavioral indicators such as API call sequences, registry modifications, file operations, and network communications. New operating systems like Windows 10 and 11 provide additional security layers that may alter or restrict certain malware behaviors. The main objective of the study is to capture generalized behavioral patterns rather than OS-specific exploitation tactics. However, in future work, we will extend the analysis to a multi-operating system sandbox environment to further examine behavioral variations across modern platforms.

3.6 Feature Leakage Mitigation

We carefully processed each malware sample and its corresponding JSON file to derive dynamic features to avoid indirectly encoding family-specific artifacts that could leak information between training and validation sets. To mitigate the information leakage, the following configurations were set up:

Feature Abstraction: Raw dynamic logs, such as registry keys, process names, and file paths, were turned into important behavioral features like total files created, registry changes, network requests, processes started, or ended. Family-specific features, such as file names, IP addresses, system-specific paths, or family-related identifiers to make sure that behavioral patterns are included instead of hard-coded artifacts.

Valuation Splits: We implemented the family-disjoint technique for category-wise and family-wise malware classification tasks. By applying this, make sure that all samples from a specific malware family only show up in either training or validation. It helped to make sure that models cannot cheat by using specific family identifiers built into the features, so performance shows real behavioral generalization.

Preprocessing on Training Data Only: Maximum processing steps, including feature selection, imputation, and scaling, were fit mostly on the training dataset and then implemented on the test dataset. It helped to prevent leakage of distributional information from the test set into the model.

By combining all these techniques, AU-PEMal-2025 ensures that not only binary but category-wise and family-wise classification performance replicates true malware behavioral showing rather than reliance on specific family artifacts.

3.7 Dataset Creation

Both benign and malicious sample records, including static and dynamic artifacts, were put together into the AU-PEMal-2025 dataset. In Table 2, important characteristics like total number of samples, labeling protocols, duplication removal procedure, and evaluation splits are listed so that it can be reproduced easily. We added two versions of the dataset, one with all static and dynamic features with file hashes to identify each record, and a clean version with selected features.

The AU-PEMal-2025 Dataset Card is provided to ensure reproducibility and transparency. Important details from sample collection to data construction are provided in tabular form (Table 2). Following detail is keep in focus for better understanding:

Table 2: AU-PEMal-2025 dataset description (Dataset card).

Attribute	Description
Dataset Name	AU-PEMAL-2025
Purpose	Dynamic behavioral-based malware detection and hierarchical classification benchmarking
Collection Timeframe	[Jan–Aug 2023]
Sample Sources	Malware: Public repositories such as Any.run, VirusShare, VX-Heaven, Benign: Signed Windows software from official distributions and trusted and verified software applications
Number of Samples	Malware: 15,500 samples, Benign: 15,500 samples
Platforms	Windows PE 32-bit
Feature Type	Hybrid (static and dynamic behavioral-based features), extracted from PE files and from sandbox execution reports such as API calls, files, registry logs, network activities, or system calls
Labeling Protocol	Each malware sample was tested across multiple antivirus engines, verified, and labeled in its corresponding category and family. Samples with low confidence and conflict were removed
Duplication Removal	Exact duplicates were removed firstly via SHA-256 hashing, secondly via the same features, and near-duplication was filtered using behavioral similarity
Data Splits Provided	Stratified random 80/20 split; additional family-disjoint split for robust evaluation
Class Imbalance Handling	Balanced in binary detection, in category-wise classification, reduction of benign samples, and in family-wise classification, reduction of benign as well as majority malware families. Also tested class weighting and stratified sampling
Execution Settings	Each sample was run in the Cuckoo sandbox for 500 s. Suricata was configured to check network activities. Dynamic features aggregated to high-level statistics to minimize family-specific leakage
Accessibility	Most of the real-world malicious samples are not publicly available due to cyber threats and licensing. Feature selection and extraction scripts, preprocessing pipeline, train and test splits, and hyperparameters provided for reproducibility
Potential Limitations	Delayed-execution or environment-aware malware may not be fully captured; family-specific variants may still be underrepresented
Intended Use	Benchmarking ML and DL models for malware detection, category, and family classification.

Collection Timeframe and Sampling Strategy: Malicious and benign samples were collected from public repositories and trusted sources between January and August 2023. The main goal was to gather verified, latest real-world samples to construct a dataset with the latest features. Multiple online sources, threat reports, malware attacks, previous studies, and malware databases were explored.

Duplication Removal: In the first round, the basic hashing technique, SHA-256, was applied to remove duplication. In the second round, exact features were checked, and hashes were matched to check for near

duplication. High overlap API call sequence clustering was applied to filter out duplicate behavioral traces to keep only a single representative for each cluster.

Labeling Protocol: Samples were collected, tested, verified, and kept in separate folders, category-wise and family-wise. VirusTotal and Any.run antivirus engines were used to verify malware families. Samples with low confidence and conflict were removed to avoid label noise.

3.7.1 Feature Extraction

A hybrid feature space was created by gathering static and dynamic artifacts during the feature extraction process. In static artifacts, PE header, section counts, import/export table size, entropy values, opcode and API call frequencies, and the structural attributes of executable files were included. Dynamic features were extracted from dynamic analysis after execution of each sample for 500 s. Important patterns such as network activity logs, memory allocation, registry modifications, and runtime API call sequences are extracted in dynamic artifacts. This hybrid feature space helps to find obfuscated and behavior-based malware variants.

After gathering, combining, and initial preprocessing, we implemented the feature selection methods to remove redundant and unimportant attributes and improve generalization. Multiple available feature selection techniques were tested and validated along with manual feature analysis. A statistical and model-based criterion was implemented to keep features that contribute most to class discrimination.

Five different JSON files from each malware family were analyzed manually to understand the behavior of malware samples and select related features. After thorough analysis, 103 features were selected as labels, and a CSV file was prepared as a dataset. A Python-based script was written to parse each JSON file from a given folder path, check required parameters, extract data against labels, load a CSV file, and save data against each label.

3.7.2 Dataset Cleaning

A total of 23,930 samples out of 31,000 were executed successfully, and completed dynamic analysis. The next step was to analyze, check, and clean the newly created dataset. The raw dynamic-analysis feature vectors extracted from sandbox execution reports contained missing values due to optional or environment-specific behaviors. By analyzing, it was noticed that there were some entries that executed but stopped due to platform compatibility, and the values of those records were null or missing in the CSV file. A threshold was set in which 100 features were passed to check if a record had null values against those 100 features; keep it otherwise, drop it. The Heatmap, Mutual Information, Wrapper methods, RF classifier, and Feature Relevance techniques were implemented to check the importance and relevance of features against target columns (Class, Category, Family). After analyzing the results of the techniques, 39 features were selected for the final dataset. After cleaning the dataset, the final version consisted of 21,703 records, which included four malware categories (Info Stealer, Trojan Horse, Ransomware, and RAT), 26 malware families, and 39 features, with 10,864 benign and 10,839 malicious records. Figs. 3–5 provide binary, category-wise, and family-wise distribution, while Tables 3 and 4 show malware categories, families, and total records against each family.

ML models require preprocessed data before it is fed for training and validation. The first step in preprocessing is to check missing values after loading the dataset. These values are the values in the dataset that indicate data corruption or are wrongly recorded in the dataset creation process. These values can undoubtedly affect the performance of a model.

Missing values arise due to execution-path variability in dynamic analysis. Missing feature values were initially handled using the default imputation strategy applied consistently across all experiments. To assess

the sensitivity of model performance to the choice of imputation method, we additionally evaluated standard statistical imputers, including median imputation and mode imputation, where applicable. We also applied an Interquartile Range (IQR) based imputation strategy within semantically grouped feature blocks to reduce the impact of extreme values while maintaining internal feature consistency, along with a custom-built script to check five columns backward and five columns forward before filling the value. Features are organized into semantically related groups (e.g., file, registry, network, API statistics). The backward/forward rule operates only within each group to preserve statistical consistency and does not assume any temporal or ordinal dependency among features.

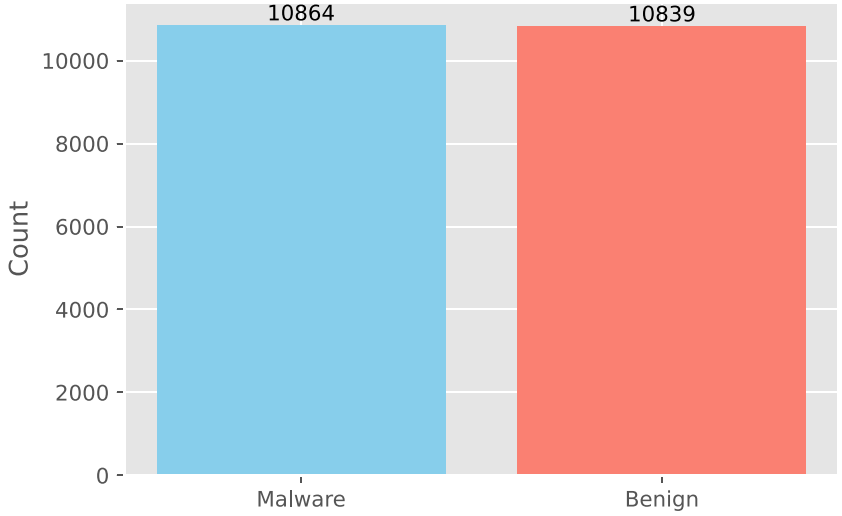

Figure 3: Class distribution of the AU-PEMal-2025 dataset.

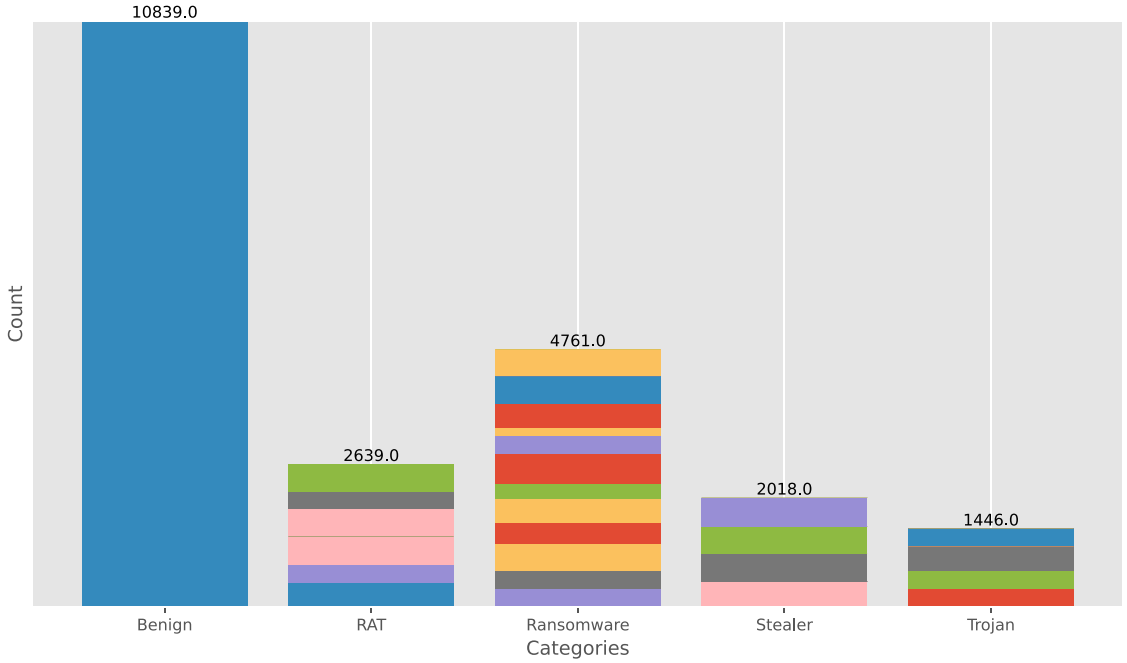

Figure 4: Category-wise distribution of the AU-PEMal-2025 dataset.

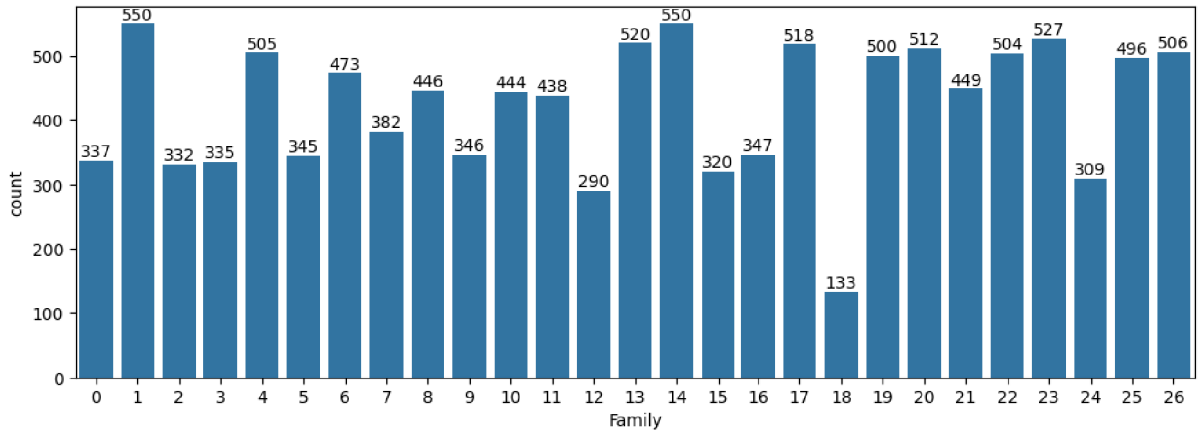

Figure 5: Family-wise distribution of the AU-PEMal-2025 dataset.

Table 3: Malware categories in the AU-PEMal-2025.

Category	Total Samples
Trojan Horse	1446
Info Stealer	2018
Ransomware	4761
RAT	2639
Benign	10,839

Table 4: Family-wise distribution in the AU-PEMal-2025.

Family	Samples	Family	Samples
AgentTesla	337	njRat	506
Emotet	345	Cerber	332
Guloader	444	DarkSide	335
Qbot	320	Dharma	505
Formbook	473	GandCrab	382
Raccoon	518	LockBit	438
RedLine	500	Maze	290
Snake	527	Phobos	550
Gh0st	446	REvil	347
Glupteba	346	Ragnar	133
NanoCore	520	Ryuk	449
Remcos	512	Shade	504
Ursnif	309	WannaCry	496

3.7.3 Sensitivity Analysis of Imputation

To evaluate the sturdiness of the proposed models to missing-value handling, we conducted a sensitivity analysis using standard imputers. Specifically, model performance was compared under median and mode-based imputation strategies. Across binary, category-wise, and family-wise classification tasks, performance

variations remained within a narrow range, demonstrating that the models are not sensitive to the choice of imputation method.

The next phase in the preprocessing is to check categorical values in the dataset. ML models require numeric data for training, and text or strings in the dataset are considered absolute values. The conversion of string data into numerical form is called categorical encoding. In the first phase of data creation, the categorical values are treated separately. Different features in the dataset represent different states, and converting them to the same pattern can lead to a wrong interpretation of captured data. At the time of model training, in our dataset, the target columns (Class, Category, and Family) contained categorical values, which were converted into numeric values. Tables 5–7 provide details of categorical encoding against each categorical value.

Table 5: Categorical encoding in the "Class" column.

Class	Encoding
Benign	0
Malicious	1

Table 6: Categorical encoding in the "Category" column.

Class	Encoding	Class	Encoding
Benign	0	Stealer	3
Ransomware	1	Trojan Horse	4
RAT	2		

Table 7: Categorical encoding in the "Family" column.

Class	Encoding	Class	Encoding
Benign	0	Guloader	13
Phobos	1	LockBit	14
Snake	2	Gandcrab	15
NanoCore	3	REvil	16
Raccoon	4	Emotet	17
NjRat	5	Glupteba	18
Dharma	6	AgentTesla	19
Shade	7	DarkSide	20
RedLine	8	Cerber	21
WannaCry	9	Qbot	22
Formbook	10	Ursnif	23
Gh0st	11	Maze	24
Ryuk	12	Ragnar	25

3.7.4 Handling Class Imbalance

The imbalanced data problem occurs in classification tasks most of the time. Imbalanced data refers to the distribution of classes as heavily skewed or disproportionate. When the total number of records in one class is high compared to other classes, then that situation is called imbalanced, which is also a common problem in multi-class tasks. If an imbalanced dataset is fed for training, the model can be biased toward the classes that have the majority of records and can mislead accuracy or face difficulty in learning patterns. In the malware detection phase, the dataset was mostly balanced. However, in category-wise classification, the dataset was slightly imbalanced, while in family-wise classification, the dataset became highly imbalanced. To address this, we considered multiple strategies:

Initial Approach: We reduced the number of benign samples in the training set to match the approximate scale of malware samples for a balanced baseline evaluation. This made sure that early tests didn't make models more likely to guess the majority class.

Evaluation of Standard Techniques: Along with downsampling, we carefully looked at other options:

- Class weighting: assigning higher loss weights to underrepresented malware classes.
- Stratified sampling: making sure that the proportions of classes in the train and test sets stay the same.
- SMOTE (Synthetic Minority Oversampling Technique): making fake samples for families that aren't well represented.
- Performance metrics for these methods were compared to controlled downsampling to ensure robustness.

Findings and Rationale: We found that class weighting and stratified splits worked just as well or even a little better than manual downsampling, and they didn't throw away any good data. Some baseline experiments kept downsampling to be consistent with previous work, while results using class weighting/oversampling are included in the Availability of Data and Materials Section to show that they are strong.

To reduce class imbalance, the SMOTE was only used on the training set. SMOTE makes fake feature vectors for minority classes by interpolating between the nearest neighbors in feature space. This keeps the model from being biased toward the most common malware families. Fig. 6 provides the overview of the feature extraction, selection, and balancing pipeline.

This approach ensures that valuable information from benign samples is preserved and that reported performance metrics reflect realistic classification scenarios.

3.7.5 Handling Outliers

An outlier in a dataset is a situation in which some data points have a large distribution as compared to others in the same column, which can mislead the models in the training process. To handle outliers, the IQR method is utilized. After thoroughly analyzing the outliers, the capping method is applied to avoid removing the records that contain outliers. After handling outliers, normalizing or standardizing a dataset to scale numerical columns within a specific range or distribution is an important factor in preprocessing. The primary goal of normalizing a dataset is to bring features to a similar scale without garbling differences in the range of values. In this work, the StandardScaler approach is applied to the dataset, which is a standardization technique.

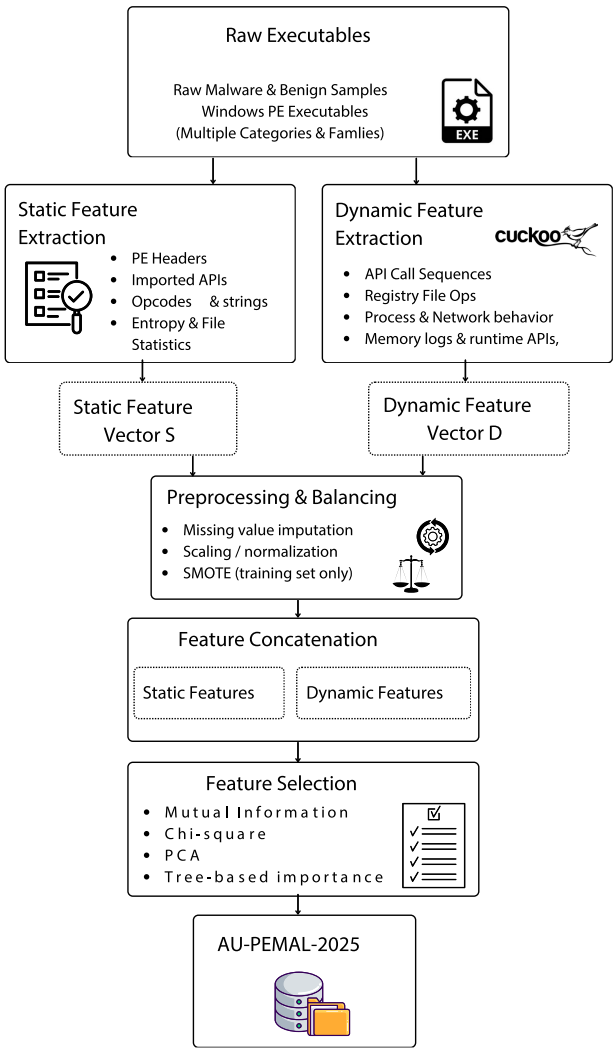

Figure 6: Overview of the feature extraction, selection, and balancing pipeline.

All preprocessing steps, including outlier threshold estimation, imputing missing values, scaling features, and feature selection, are performed on the training set and afterwards applied to the test set using the learned parameters. Imputation thresholds, scaling parameters, and feature-selection criteria are computed using the training data only and then applied unchanged to the held-out test set to prevent unintended data leakage. To assess the robustness of our imputation strategy, we performed a sensitivity analysis using standard alternatives (median and mode imputation). Results indicate that model performance varied minimally (±X% for accuracy/F1), demonstrating that our approach does not significantly bias the outcomes. Additionally, features were scaled using [Min–Max/StandardScaler] based on the training data distribution. Feature selection was performed using variance threshold/mutual information only on training data to maintain strict separation from the test set.

3.8 Detection Models

After preprocessing, the next phase is to train the models on that well-aligned data. There exist many algorithms and classifiers for detection and classification tasks. In this work, we choose five ML classifiers and three DL models to train on the newly created AU-PEMal-2025 dataset. The details are given below:

Logistic Regression (LR): LR is a supervised ML algorithm, widely used for tasks that involve classifying objects into two or more classes. It works by figuring out the chance that a certain input belongs to a certain class [76]. The model uses a sigmoid function to map predicted values into the range of 0 to 1, which stands for probabilities, and it uses a decision threshold (usually 0.5) to sort outcomes. Maximum likelihood estimation is used by the model to find a constant that maximizes the chances of the observed data [77].

Random Forest (RF): The RF ensemble learning approach is well-known for classification and regression applications. In the 2000 s, Leo Breiman proposed constructing a predictor ensemble using a group of decision trees that develop in randomly selected data subspaces [78]. During training, it creates many decision trees and generates the class, which is the mode or mean forecast of the individual trees [79].

XGBoost (XGB): XGB was primarily built for performance and speed using a gradient-boosted decision tree as it uses distributed gradient-boosting decision trees (GBDTs) [80]. It supports tree boosting in parallel, which is the best ML approach for classification, regression, and grouping tasks. Overall, the XGB concept is based on gradient-boosted trees with supervised learning as the core technique. Supervised learning predicts target values based on input data with several characteristics [81].

Support Vector Machine (SVM): Designed to find the best hyperplane to excellently distinguish data elements of different classes in a complex space. The SVM is also a supervised ML approach that is primarily utilized for classification and regression tasks. The main elements of SVM are the hyperplane itself (a decision boundary that divides classes), the distance between the hyperplane and the closest support vectors (which SVM seeks to maximize for better generalization), and support vectors (the vital data points neighboring the hyperplane that affect its position). The kernel approach allows SVM to handle linearly inseparable data by allowing complex decision boundaries through the mapping of input features into hyper dimensional spaces. Because of its structure, SVM can handle classification problems with distinct margin separation and is robust to high-dimensional data [82].

K-Nearest Neighbor (KNN): One well-liked distance-based learning method for classification applications is the KNN model. It functions by measuring the separation between two data points [83]. KNN classification originated from the need for discriminant analysis in scenarios where trustworthy parametric estimations of likelihood densities are either unfamiliar or hard to determine. In an unreported research conducted in 1950 at the US Air Force School of Aviation Medicine, Fix and Hodges offered a non-parametric system for pattern recognition. Since then, this method has been called the "k-nearest neighbor rule" [84].

Convolutional Neural Network (CNN): A CNN is a type of DNN that is particularly useful for processing grid-like, sequential, time series-like data. Lee Yang and Leon O. Chua initially proposed the CNN in 1988 [85]. Formally, the first CNN was a two-dimensional network with input, output, and dynamic state layers [86]. CNN is a sort of neural network that uses a feed-forward. The CNN structure is made of a convolutional layer, a pooling layer, and fully connected layers. The convolutional layers are made up of kernels or filters that traverse the input data convolutionally, extracting features using convolutions. The spatial dimensions of the convolved features are reduced by pooling layers while crucial information is retained. The terms max pooling and average pooling are often used. For classification or regression, fully connected layers are used. CNN is ideal for malware detection and classification jobs because it learns hierarchical patterns and features that allow it to classify malware based on typical patterns in the input data. For binary malware detection, the CNN architecture is defined as follows:

- Input layer: shape (F, 1)
- Conv1D with filters = 128, kernel = 3, activation = ReLU
- MaxPooling1D size = 2
- Dropout rate = 0.3
- Conv1D layer filters = 64, kernel = 3, activation = ReLU
- MaxPooling1D with pool size = 2
- Dropout with rate = 0.3
- Flatten layer
- Fully Connected layer units = 128, activation = ReLU
- Dropout with rate = 0.3
- Output layer activation = sigmoid
- Total trainable parameters: 82,882

For malware category classification, the same convolutional backbone was retained to ensure architectural consistency. The output layer was adapted to support multi-class prediction. The architecture differs from the binary model only in the final layer:

- Output layer with softmax activation and C neurons, where C denotes the number of malware categories.
- Trainable parameters for this configuration were 83,269.

Similarly, for malware family classification, the CNN architecture remains unchanged except for the dimensionality of the final softmax layer, corresponding to the number of malware families. This configuration contains 86,107 trainable parameters.

The CNN model architecture was designed to be shallow, minimizing overfitting and facilitating justifiable comparison with other traditional ML models. Interactions between different features that are close to each other are captured by Convolutional layers, and max-pooling reduces dimensionality and enhances generalization over time. It maintains a consistent backbone across all tasks to confirm that any differences in the performance of the model are due to the complexity of the tasks, rather than the CNN structure, which is provided in Table 8.

Table 8: CNN architectures and parameter counts.

Task	Output Activation	Classes	Total Parameters
Binary Detection	Sigmoid	2	82,882
Category Classification	Softmax	C	83,269
Family Classification	Softmax	F	86,107

Long Short-Term Memory (LSTM): LSTM is a type of RNN network that was designed to solve the dispersed and exploding gradients problems that were unable to be solved using regular RNNs to solve long-range dependencies in sequential data. A single LSTM unit has a unique memory cell that stores information over time and three main gates (input, forget, and output) that control information flow into and out of the cell [87]. Functionality of the output gate is to decide which part of the memory cell is visible to the next hidden layer, the forget gate functions to decide what information to forget or throw away, and the last input gate decides how much new unseen data is saved [88]. LSTS are well-suited for tasks such as NLP, time-series forecasting, and speech recognition due to their gating structure. Its powerful mechanism enables

it to efficiently identify both long and short-term dependencies [89]. The LSTM architecture designed for malware classification is as follows:

- Input layer: shape (F, 1)
- LSTM layer-units = 128, $return_{sequences}$ = True
- Normalization = Batch
- Dropout rate = 0.3
- LSTM layer-units = 64, $return_{sequences}$ = False
- Normalization = Batch
- Dropout rate = 0.3
- Fully connected layer-units = 64 units, activation = ReLU
- Dropout rate = 0.2
- Output layer-activation = sigmoid

The Adam optimizer was used for training with a learning rate of 0.001. A total of 138,050 trainable parameters out of 138,434 were set. For malware category classification, the same stacked LSTM backbone was used, with the final output layer adapted for multi-class prediction. The output layer employs a softmax activation with C neurons, where C represents the number of malware categories. The model contains a total of 138,629 parameters, of which 138,245 are trainable. For malware family classification, the architecture remains unchanged except for the dimensionality of the final softmax layer, corresponding to the number of malware families. This configuration contains 140,059 total parameters, including 139,675 trainable and 384 non-trainable parameters. Table 9 provides the architectures and parameter counts for LSTM.

Table 9: LSTM architectures and parameter counts.

Task	Output Activation	Total Params	Trainable	Non-Trainable
Binary Detection	Sigmoid	138,434	138,050	384
Category Classification	Softmax	138,629	138,245	384
Family Classification	Softmax	140,059	139,675	384

Bidirectional Long Short-Term Memory (BiLSTM): It is an extension of the LSTM that processes a series of data in a bidirectional manner in two independent LSTM layers. The identical data is processed in reverse order by the forward layer [90]. A normalization layer that splits channels into sets and standardizes the characteristics within each set is known as group normalization. It does not use the batch dimension; thus, its computation is not affected by batch size [91]. For binary malware detection, the BiLSTM architecture is defined as follows:

- Input layer: shape (1, F)
- Bidirectional LSTM layer-units per direction = 128, $return_{sequences}$ = True
- Dropout rate = 0.3
- Bidirectional LSTM layer-units per direction = 64
- Dropout rate = 0.3
- Fully connected layer-units = 64, activation = ReLU
- Dropout rate = 0.2
- Output layer-activation = sigmoid

The Adam optimizer and categorical cross-entropy as a loss function were used for training the model. The total trainable parameters were 340,674. For malware category classification, the same BiLSTM backbone was retained, with the final output layer modified to support multi-class prediction.

The output layer employs a softmax activation with C neurons, where C denotes the number of malware categories. This configuration contains 340,869 trainable parameters.

For malware family classification, the architecture remains unchanged except for the dimensionality of the final softmax layer, corresponding to the number of malware families. This configuration contains 342,299 trainable parameters. Table 10 provides the architectures and parameter counts for BiLSTM.

Table 10: BiLSTM architectures and parameter counts.

Task	Output Activation	Total Params	Trainable	Non-Trainable
Binary Detection	Sigmoid	340,674	340,674	0
Category Classification	Softmax	340,869	340,869	0
Family Classification	Softmax	342,299	342,299	0

Although CNN, LSTM, and BiLSTM models are traditionally used for images or sequences, we apply them to tabular features by organizing the extracted behavioral features into semantically related groups (e.g., process, file, registry, network). For CNNs, each group is treated as a 1D feature map, allowing convolutional filters to capture local correlations among related features. For LSTM/BiLSTM models, feature groups are fed sequentially to capture dependencies across behavioral dimensions.

3.9 Performance-Evaluation

ML and DL have different evaluation factors to measure performance, and achieving high accuracy does not guarantee that a model's performance is efficient. It can mislead, and there are chances that the model is overfit. In this work, we evaluated recall, precision, F1-score, and accuracy for performance evaluation of ML classifiers, while training and testing accuracy with loss for deep learning models, and confusion matrices to validate model predictions.

ACCURACY: The accuracy of an ML or DL model is the primary factor used to evaluate its performance. In DL models, it is checked and confirmed by evaluating it in each epoch and by calculating the mean accuracy at the end. Accuracy can be represented as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

PRECISION: Precision is an important metric to evaluate the performance of a model and is assessed by performing division on the accurately predicted positives by the total number of positives. It is presented as follows:

$$Pricision = \frac{TP}{TP + FP} \tag{2}$$

RECALL: The third important factor to evaluate an ML model is sensitivity analysis which is also called recall. It is calculated in such a way that it is the proportion of related specimens retrieved to the total number of specimens and is presented as follows:

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

F1-SCORE: Another essential aspect to consider when evaluating an ML model is the F1-score, which is generated by combining accuracy and recall, and is considered to represent the average weight of all values, and can be presented as below:

$$F1 - Score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4}$$

Evaluation Robustness and Generalization: To ensure robust and leakage-resistant evaluation, we extend our experimental protocol beyond a single random train–test split. Multiple random seeds are used in all experiments (binary, category-wise, and family-wise classification) to evaluate and report results with mean and standard deviation.

In addition to the standard 80/20 random split, we employ leakage-resistant evaluation strategies appropriate for malware analysis. These include family-disjoint splits, which means that the test set has malware families that aren't in the training set, so the model can't learn patterns that are specific to a family. This type of setting works the same like real-world when it comes to generalizing to new malware variants.

These types of evaluation protocols help models to lower the chances of data leaks that can occur when malware samples are very similar across all splits and give a more accurate representation of how well a model can generalize.

Evaluation Protocol: Multiple important evaluation strategies were implemented to reduce overfitting and to add robust generalization. To start, all experiments were done again with different random seeds, and the results are given as mean ± standard deviation. Secondly, the important settings that were resistant to leakage were also added. Family-disjoint splits, category-wise, and family-wise separation reduce the chances that the malware families that were in the test set do not exist or are not seen during the training dataset.

4 Results

In this section, the performance of the implemented models on the newly created dataset is provided in three phases (binary, category-wise, and family-wise). To validate the efficiency of the AU-PEMal-2025 dataset, malware is categorized into different classes, such as categories and families, to predict real-world and unseen variants. Similar work in the same field is discussed and compared with this research at the end of this section to validate the proposed technique. The experiments were performed on a Workstation that had a Core i7 4790 processor with 16 gigabytes of RAM and 4 gigabyte NVidea Quadro graphics card. The experiments are also performed on Google Colab to check the resources available for the proposed dataset. Anaconda 2.4.2 with required libraries and Jupyter Notebook was configured on the workstation. A standard 80/20 train-test split was utilized to train the models with random seeds, and family-disjoint splits to reduce the chances of overfitting and to make sure that models were not biased toward specific families. Samples were divided into two groups randomly, with 80% for training and 20% for testing. To get the mean and standard deviation, different random seeds were used. All samples from the same family were either fitted in training or the testing dataset to avoid leakage.

All preprocessing steps (imputation, scaling, feature selection) were fit on the training set only for each split to avoid test-set leakage. Metrics reported include both random and family-disjoint evaluations, with mean ± standard deviation across multiple seeds. A total of 39 features were fed to all models for training in detection, 38 features in category-wise classification, and 36 features in family-wise classification tasks. DL models were trained with categorical cross-entropy with a learning rate of 0.01 and the Adam optimizer.

We trained three DL models (CNN, LSTM, and BiLSTM) on the engineered dynamic-analysis tabular features. Although the extracted behavioral features are tabular, they exhibit implicit temporal and sequential structure derived from dynamic execution logs. To leverage this structure, the feature vectors were serialized into fixed-length sequences and provided as input to CNN, LSTM, and BiLSTM-based architectures. All deep learning models were implemented using the same input representation to ensure fair comparison. Early stopping and dropout were employed to mitigate overfitting. The detailed layer-wise configurations for DL models are shown in Table 11. For all architectures, we applied dropout (0.3) and early stopping on the basis of validation loss (patience = 10). Adam optimizer with 0.001 learning rate and batch size of 32 was set up. Hyperparameters were selected using validation-based tuning on the training set. All reported results represent the mean and standard deviation over multiple runs with different random seeds under leakage-resistant evaluation settings.

Table 11: Deep learning model architectures.

Model	Layer	Units/Filters	Kernel/Seq	Activation	Dropout	Output
CNN	Conv1D	128 Filters	Kernel = 3	ReLU	–	–
	Conv1D	64 Filters	Kernel = 3	RelU	–	–
	MaxPool1D	Pool Size = 2	–	–	–	
	Dense	128 Units	–	ReLU	0.30	–
	Output Dense	No. of Classes	–	Softmax	–	Classification
LSTM	LSTM	128 Units	Sequence Input	tanh	0.30	–
	Output Dense	No. of Classes	–	Softmax	–	Classification
BiLSTM	Bidirectional LSTM	128 Units	Sequence Input	tanh	0.30	–
	Output Dense	No. of Classes	–	Softmax	–	Classification

Hyperparameter tuning was conducted in a validation-driven manner to ensure fair comparison and to avoid information leakage from the test set. For all classical ML algorithms (LR, XGB, RF, KNN, and SVM), grid search was utilized to optimize hyperparameters with stratified cross-validation (5-fold) on the training data. The macro-averaged F1-score was used as the primary optimization metric due to class imbalance.

Each malware sample is depicted by a feature vector of length F extracted from dynamic analysis reports. The vector was reshaped into a one-dimensional sequence of length F with a single channel that enables convolutional and recurrent processing. The models learnt important patterns from feature co-occurrence and magnitude instead of semantic ordering between the features. A 10% from the training data was utilized as a validation split to fine-tune DL models. Manual and random seeds were implemented to look into important architectural and optimization parameters. To stop overfitting, patience is set to 10 epochs on the basis of validation loss. L2 regularization was implemented, and the final evaluation was based on the model checkpoint that worked best. The complete hyperparameter details are summarized in Table 12.

Table 12: Hyperparameter details.

Model	Hyperparameters Explored	Selected Configuration
LR	$C \in \{0.1, 1, 10\}$, $penalty \in \{12\}$	C = 1.0
RF	$n_estimators \in \{100, 300, 500\}$, $max_depth \in \{None, 10, 20\}$, $min_samples_split \in \{2, 5\}$	20
XGB	$Learning_rate \in \{0.01, 0.1\}$, $max_depth \in \{6, 10\}$, $n_estimators \in \{200, 500\}$	lr = 0.1, max_depth = 10
SVM	$C \in \{0.1, 1, 10\}$, $kernel \in \{linea, rbf\}$, $y \in \{scale, auto\}$	$C \in \{10\}$, $kernel = rbf$
CNN	$Filters \in \{32, 64\}$, $kernel_size \in \{3, 5\}$, $dropout \in \{0.3, 0.5\}$	2 conv layers, 64 filters, kernel = 3
BiLSTM	$Units \in \{64, 128\}$, $recurrent_dropout \in \{0.2, 0.3\}$	units = 128

4.1 Binary Classification

As the dataset was made such that it had three columns (Class, Category, and Family) as labels. The first label column had two (benign and malware) categorical values, which were converted into 0 and 1. In the first experiment, five ML classifiers (LR, RF, XGB, SVM, and KNN) were trained on the AU-PEMal-2025 dataset. The models achieved 0.9908, 0.9939, 0.9945, 0.9919, and 0.9631 accuracy, respectively, in binary classification. Table 13 provides evaluated results in tabular form, and Fig. 7 presents a graphical representation.

Across repeated runs, all classical machine learning models demonstrated highly stable performance with very small standard deviations (≤0.0015), indicating that the learned decision boundaries are not sensitive to random initialization or sampling variation. Among the evaluated models, XGBoost secured the best performance with an average accuracy of 0.9945 ± 0.0014, followed closely by Random Forest (0.9939 ± 0.0009). Logistic Regression and SVM also produced competitive results with an accuracy of over 0.99, but KNN's accuracy was lower (0.9631 ± 0.0017), which shows that it is sensitive to high-dimensional feature interactions. The consistently low variance across all metrics shows that the proposed feature representation can be used again and again and is strong.

Table 13: Performance of ML models in binary classification.

Model	Accuracy	Precision	Recall	F1-Score
LR	0.9908 ± 0.0009	0.9909 ± 0.0008	0.9908 ± 0.0009	0.9908 ± 0.0009
RF	0.9939 ± 0.0009	0.9940 ± 0.0009	0.9939 ± 0.0009	0.9939 ± 0.0009
XGB	0.9945 ± 0.0014	0.9946 ± 0.0014	0.9945 ± 0.0014	0.9945 ± 0.0014
SVM	0.9919 ± 0.0015	0.9919 ± 0.0015	0.9919 ± 0.0015	0.9919 ± 0.0015
KNN	0.9631 ± 0.0017	0.9631 ± 0.0017	0.9631 ± 0.0017	0.9631 ± 0.0017

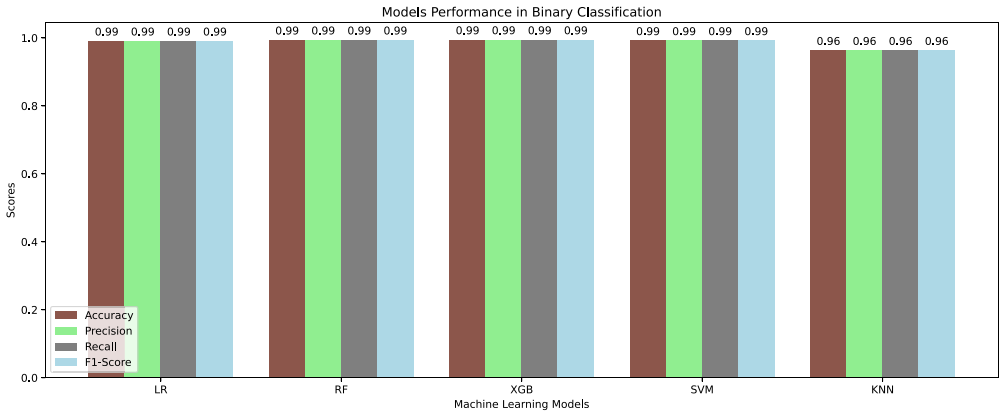

Figure 7: ML models' accuracies in binary classification.

The second experiment was conducted using DL models, and the CNN, LSTM, and BiLSTM algorithms were trained to detect malware. The performance of deep learning models was outstanding in detecting malware. The CNN achieved 0.9932 accuracy with an average 0.045 loss, the LSTM model achieved 0.9917 accuracy with an average 0.045 loss, and the BiLSTM model achieved 0.9922 accuracy with an average 0.034 loss. Table 14 provides the evaluated results along with standard deviation by DL models in tabular form, while the Figs. 8–10 provide the graphical representation of accuracies and losses of models.

Table 14: Performance of DL models in binary classification.

Model	Accuracy	Precision	Recall	F1 Score	Average Loss
CNN	0.9932 ± 0.0014	0.9933 ± 0.0013	0.9932 ± 0.0014	0.9932 ± 0.0014	0.045
LSTM	0.9917 ± 0.0009	0.9918 ± 0.0009	0.9917 ± 0.0009	0.9917 ± 0.0009	0.254
BiLSTM	0.9922 ± 0.0011	0.9923 ± 0.0011	0.9922 ± 0.0011	0.9922 ± 0.0011	0.034

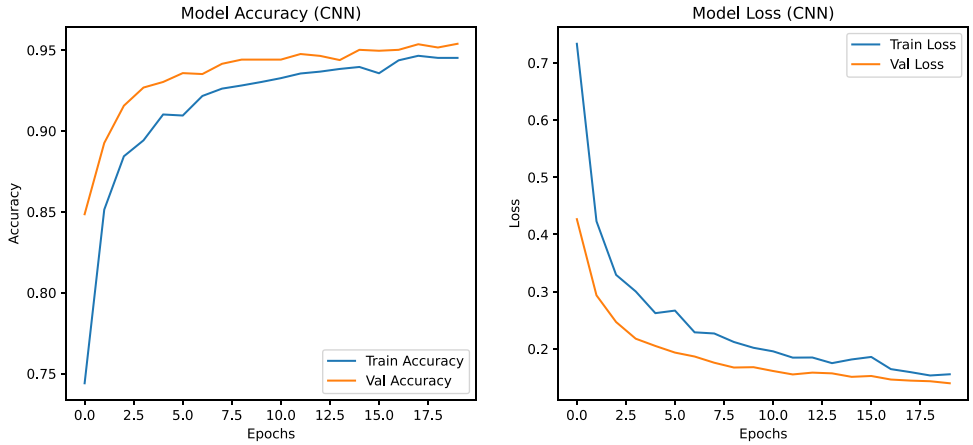

Figure 8: The CNN model accuracy and loss in malware detection.

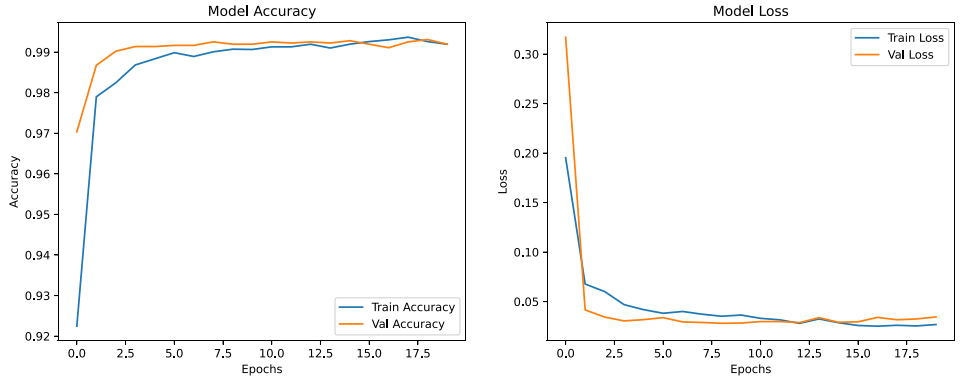

Figure 9: The LSTM model accuracy and loss in malware detection.

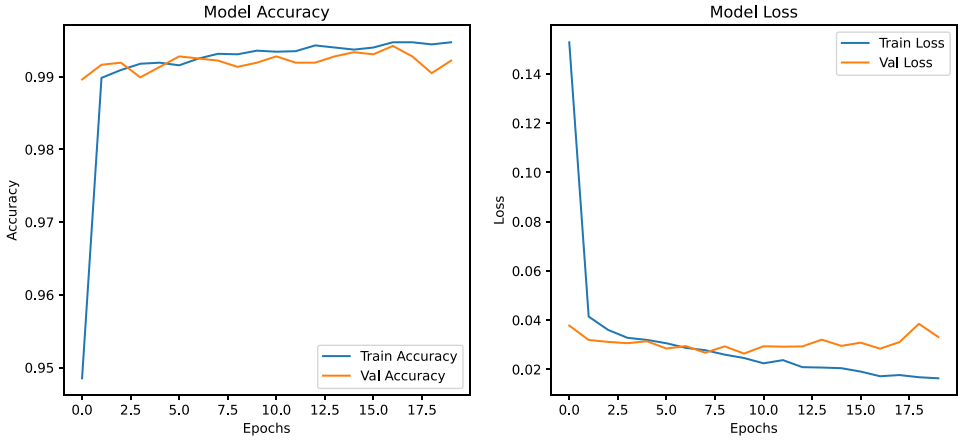

Figure 10: The BiLSTM model accuracy and loss in malware detection.

The deep learning models also demonstrated strong and consistent performance in the binary detection task across multiple random seeds. The CNN model had the best overall performance, with an accuracy of 0.9932 ± 0.0014, as well as high precision, recall, and F1-score values. The average of training and validation loss was 0.045, which showed that the model was stable in all seeds. The BiLSTM model achieved the second-highest accuracy of 0.9922 ± 0.0011 with the lowest average loss of 0.034. It showed that the bidirectional nature of the model successfully captured the feature space from both directions. The LSTM achieved 0.9917 ± 0.0009 accuracy with a higher loss of 0.254. The small standard deviations observed across all metrics confirm that the deep learning models produce reliable and repeatable results under different random initializations.

By analyzing the confusion matrix provided in Fig. 11, it is clear that the LR model misclassified 27 malicious samples and 12 benign records. The RF model misclassified 21 benign records as malicious and 8 malicious samples as benign. The XGB misclassified only 16 goodware samples and 11 malware samples. The SVM model misclassified 30 benign and 13 malware samples, while KNN predicted 80 benign and 81 malware samples incorrectly. Among the DL models, the CNN misclassified a total of 35, LSTM 37, and BiLSTM 39 samples. The XGB classifier, among the ML models, performed better than the other ML models in binary classification, while the CNN model secured the highest accuracy with minimal loss among the DL models.

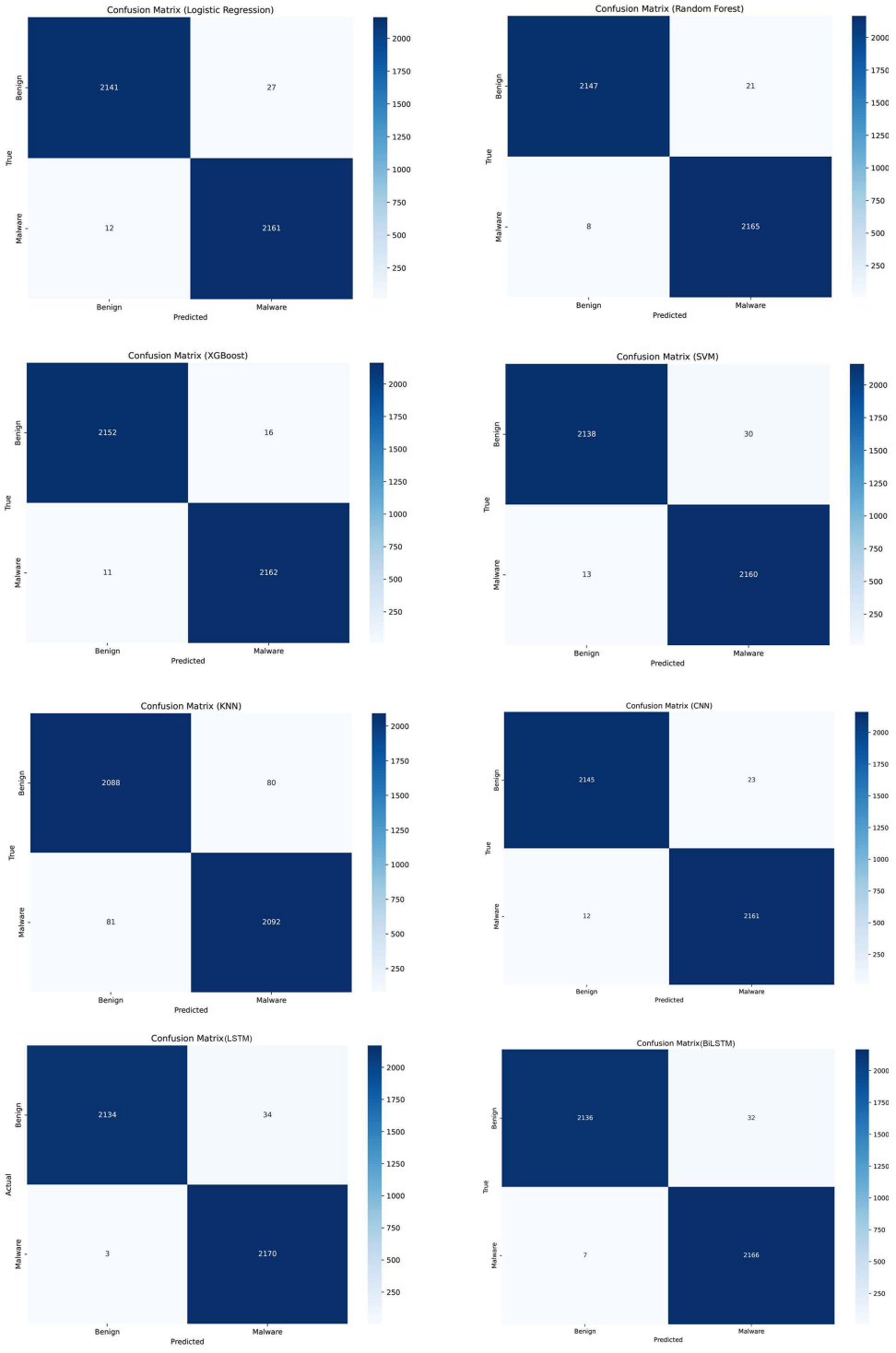

Figure 11: Models' confusion matrix in binary detection phase.

On the AU-PEMal-2025 dataset, all models performed admirably for malware detection, except the KNN, which performed poorly and misclassified 81 malware and 80 benign samples. The DL models achieved great accuracy while minimizing training and testing losses. The little training and testing loss suggested that the characteristics chosen and supplied to the models are effective and relevant to malware. Despite

high classification accuracy values often exceeding 99% performance remained consistent across different random seeds and leakage-resistant splits. Importantly, no significant discrepancy between training and testing performance was observed, which indicates limited overfitting.

4.2 Malware Category-Wise Classification

After successful detection of malware, the next task and defense is to find the exact category of malware. Identifying malware as Trojan, Ransomware, Info Stealer, or RAT will help to focus on specific malware categories and utilize resources in a better way to defend against malware attacks. To this purpose, the proposed AU-PEMal-2025 dataset was built in such a way that it had four malware categories. The ML and DL models were trained by specifying the "Category" column as the target. In category-wise malware classification, the LR, RF, XGB, SVM, and KNN achieved 0.8714, 0.9734, 0.9788, 0.9488, and 0.9186 accuracy, respectively. Table 15 provides evaluation results along with standard deviation performed by models in tabular form, while Fig. 12 provides a comparison in a graphical view.

Table 15: ML models' results in classifying malware categories.

Model	Accuracy	Precision	Recall	F1-Score
LR	0.8714 ± 0.0017	0.8716 ± 0.0012	0.8714 ± 0.0017	0.8713 ± 0.0016
RF	0.9734 ± 0.0007	0.9737 ± 0.0007	0.9734 ± 0.0007	0.9733 ± 0.0006
XGB	0.9788 ± 0.0015	0.9789 ± 0.0015	0.9788 ± 0.0015	0.9788 ± 0.0015
SVM	0.9488 ± 0.0015	0.9487 ± 0.0013	0.9488 ± 0.0015	0.9486 ± 0.0015
KNN	0.9186 ± 0.0024	0.9174 ± 0.0027	0.9186 ± 0.0024	0.9177 ± 0.0026

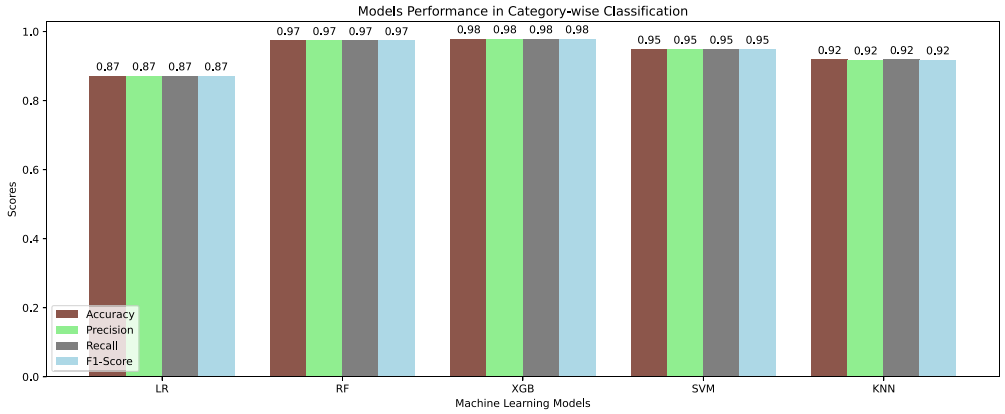

Figure 12: ML models' result comparison in category-wise classification.

In the category-wise malware classification task, ensemble-based models achieved the strongest performance. XGB delivered the best results with an accuracy of 0.9788 ± 0.0015, along with consistently high precision, recall, and F1-score, indicating its effectiveness in capturing complex, non-linear relationships among hybrid features. RF closely followed, getting an accuracy of 0.9734 ± 0.0007. This shows that it is very reliable and has the lowest variability across runs, which shows that it is very strong.

SVM classifier achieved an accuracy of 0.9488 ± 0.0015 among the ML models, which is lower than the LR, RF, and XGB, but shows that the model is competitive, although not as good at handling multiclass problems as compared to other models' accuracies. KNN achieved 0.9186 ± 0.0024 accuracy, which is slightly

higher than the standard deviation, which shows that the model is more sensitive to data spread. LR gave a linear baseline of 0.8714 ± 0.0017, which shows that non-linear and ensemble methods work better for category-level discrimination. Overall, the small standard deviations (±0.0007 – ±0.0024) across multiple random seeds verify the stability and reproducibility of the obtained results for category-wise classification.

DL models were also trained to detect the malware categories. The CNN achieved 0.9591 accuracy with 0.224 average loss. The LSTM model achieved 0.9502 accuracy with 0.242 average loss. The BiLSTM model achieved 0.9504 accuracy with 0.185 average loss. Table 16 provides the summary of achieved results by DL models in tabular form, while the Figs. 13–15 provide graphical representations.

Table 16: Performance of DL models in category-wise classification.

Model	Accuracy	Precision	Recall	F1 Score	Average Loss
CNN	0.9591 ± 0.0034	0.9595 ± 0.0030	0.9591 ± 0.0034	0.9592 ± 0.0032	0.224
LSTM	0.9502 ± 0.0009	0.9508 ± 0.0005	0.9502 ± 0.0009	0.9503 ± 0.0007	0.242
BiLSTM	0.9504 ± 0.0009	0.9509 ± 0.0008	0.9504 ± 0.0009	0.9505 ± 0.0009	0.185

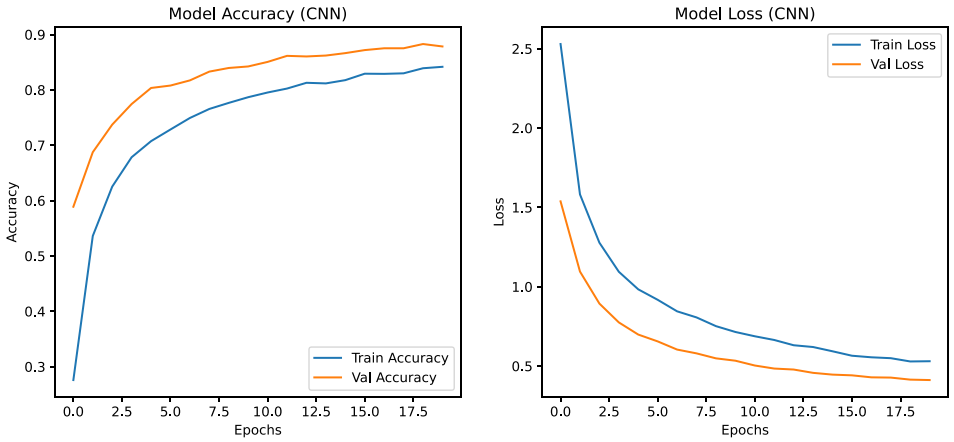

Figure 13: The CNN accuracy and loss in category-wise malware classification.

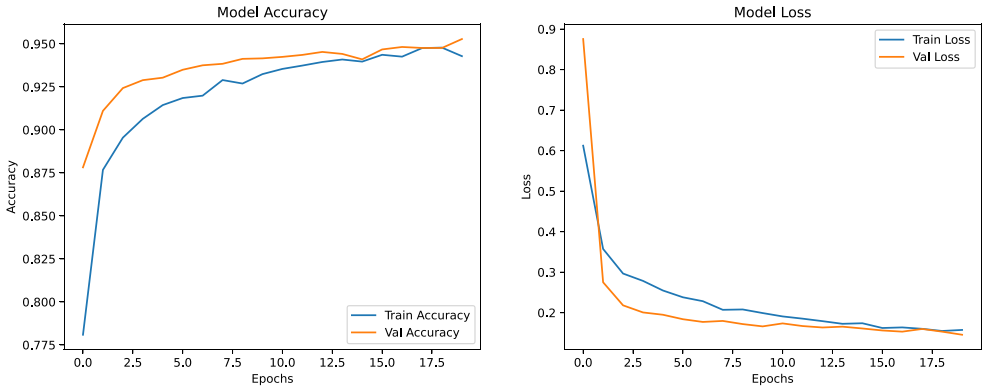

Figure 14: The LSTM accuracy and loss in category-wise malware classification.

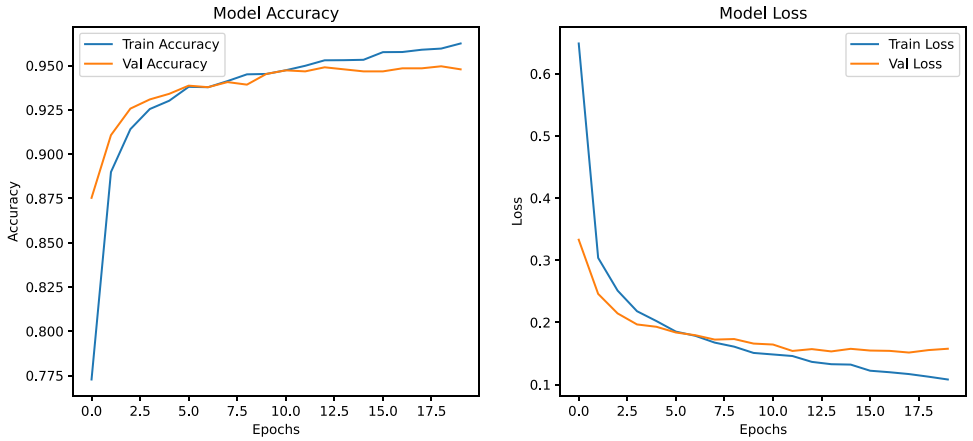

Figure 15: The BiLSTM accuracy and loss in category-wise malware classification.

The CNN achieved the highest results among the evaluated architectures, with an accuracy of 0.9591 ± 0.0034, along with comparable precision, recall, and F1-score values in the category-wise classification of malware. This shows that CNN can learn discriminative spatial patterns very well from the hybrid feature representation. The standard deviation is a little higher than for other DL models, which means that they are moderately sensitive to changes in initialization and sampling. This is normal for convolutional architectures trained on tabular feature embeddings.

The sequential models, LSTM and BiLSTM, gave similar results, with accuracies of 0.9502 ± 0.0009 and 0.9504 ± 0.0009, respectively. Their almost identical performance suggests that modeling temporal dependency doesn't add much value to this feature structure because the input isn't inherently sequential. But their very low standard deviations show that their learning behavior is very stable across many random seeds.

BiLSTM had the lowest average loss (0.185) in terms of training dynamics, followed by CNN (0.224) and LSTM (0.242). This shows that all architectures converged well. Overall, the results confirm that while DL models perform competitively for category-level classification, their gains over classical ensemble methods are modest, highlighting the strong suitability of the engineered hybrid features for traditional machine learning approaches.

By analyzing the confusion matrix in Fig. 16, the XGB secured the highest accuracy among the ML classifiers and misclassified only 85 samples. The RF algorithm misclassified 108, SVM 227, KNN 342, and LR 562 samples. The CNN misclassified 205 samples, LSTM 213, and BiLSTM 208 samples. The CNN model performed better than other models; however, by comparing training and validation losses, standard deviations, and analyzing graphs, the BiLSTM model performed better than other DL models in categorizing malware.

4.3 Malware Family-Wise Classification

After malware detection and category-wise classification, the next task is to identify its exact family. Identifying the malware family will help not only to mitigate the attack immediately but also save resources that are utilized in analyzing the whole network. The proposed AU-PEMal-2025 dataset contained 25 malware families and one benign family. Models were trained and validated by specifying the "Family" column as a target. In classifying malware families, LR, RF, XGB, SVM, and KNN achieved 0.7946, 0.9384, 0.9485, 0.8671, and 0.8143 accuracy, respectively. Table 17 provides models' results in tabular form, while Fig. 17 presents a model comparison in a graphical view.

Figure 16: Models' confusion matrix graphs in category-wise classification.

Table 17: ML Models' performance in classifying malware families.

Model	Accuracy	Precision	Recall	F1-Score
LR	0.7946 ± 0.0050	0.7949 ± 0.0061	0.7946 ± 0.0050	0.7932 ± 0.0058
RF	0.9384 ± 0.0040	0.9393 ± 0.0040	0.9384 ± 0.0040	0.9382 ± 0.0039
XGB	0.9485 ± 0.0015	0.9487 ± 0.0015	0.9485 ± 0.0015	0.9483 ± 0.0015
SVM	0.8671 ± 0.0011	0.8691 ± 0.0012	0.8671 ± 0.0011	0.8670 ± 0.0014
KNN	0.8143 ± 0.0035	0.8136 ± 0.0042	0.8143 ± 0.0035	0.8112 ± 0.0041

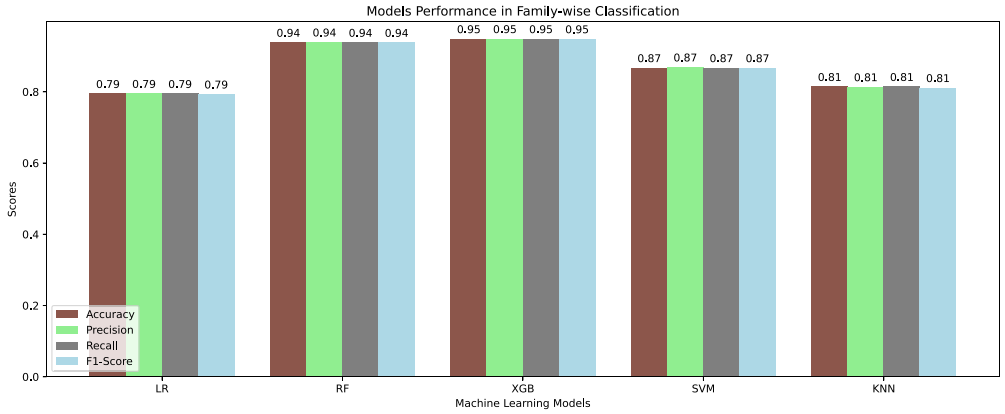

Figure 17: ML models' comparison on classifying malware families.

The family-wise malware classification task, which is the most detailed and difficult evaluation setting, shows bigger differences in performance between machine learning models because malware families are more similar to each other within the same class. The XGB model achieved the highest accuracy of 0.9485 ± 0.0015 and similar high values of 0.9487 ± 0.0015 precision, 0.9485 ± 0.0015 recall, and 0.9483 ± 0.0015 F1-score. The very small standard deviation of 0.0015 shows that the model performed amazingly with different random seeds. It shows that the model is good at recognizing malware categories and differentiating them from similar features of malware variants. The gradient-boosted decision tree showed good performance in finding small interactions between features in the hybrid feature space.

The RF model also achieved a good accuracy of 0.9384 ± 0.0040 and had stable performance in all random seeds. The model benefited from ensemble-based variance reduction, which made it suitable for handling the mixed nature of features (static and dynamic). The SVM model achieved 0.8671 ± 0.0011 accuracy, which shows that the model understands its decision boundaries very well, but is not suitable for capturing the complex and non-linear patterns among the malware categories. The KNN and LR models achieved the lowest accuracies of 0.8143 ± 0.0035 and 0.7946 ± 0.0050, respectively. These models heavily depend on simpler decision-making processes and understand only those samples that have similar behavioral and structural traits. Overall, the ML models performed perfectly in category-wise classification of malware variants. The XGB and RF models are good for large-scale family-wise classification tasks. The consistent low standard deviation across models also demonstrates that the evaluation protocols are stable when using multiple random seeds.

We also trained deep learning models (CNN, LSTM, and BiLSTM) to classify malware by family. The CNN model had an accuracy of 0.8906 and an average loss of 0.765, the LSTM model had an accuracy of 0.8825 and an average loss of 0.935, and the BiLSTM model had an accuracy of 0.9286 and an average loss of 0.426. Table 18 shows the accuracies and losses in a table, and Figs. 18–20 show them in a graph.

Table 18: Performance of DL models in family-wise classification.

Model	Accuracy	Precision	Recall	F1 Score	Average Loss
CNN	0.8906 ± 0.0092	0.8915 ± 0.0096	0.8906 ± 0.0092	0.8897 ± 0.0092	0.765
LSTM	0.8825 ± 0.0108	0.8825 ± 0.0112	0.8825 ± 0.0108	0.8814 ± 0.0111	0.935
BiLSTM	0.9286 ± 0.0035	0.9302 ± 0.0031	0.9286 ± 0.0035	0.9283 ± 0.0032	0.426

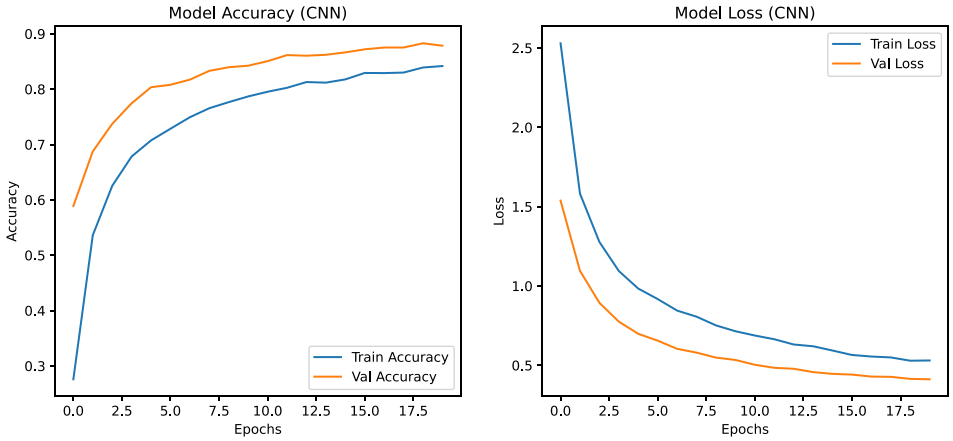

Figure 18: The CNN accuracy and loss graph in classifying malware families.

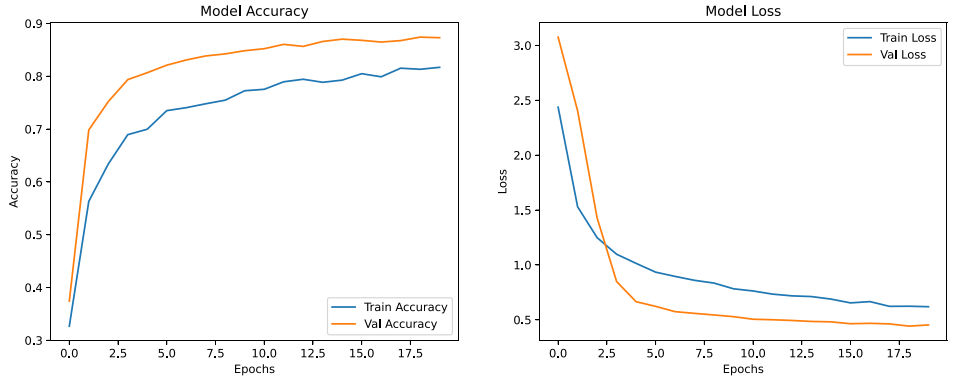

Figure 19: The LSTM accuracy and loss graph in classifying malware families.

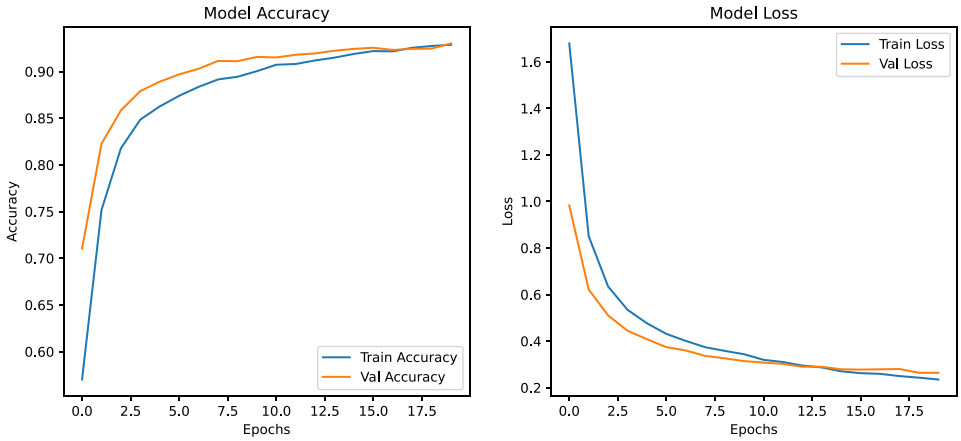

Figure 20: The BiLSTM accuracy and loss graph in classifying malware families.

When using deep learning architectures for family-wise classification, the job gets a lot harder because you have to tell the difference between very similar malware types. So, neural models don't work as well as binary detection, but they are still strong and consistent.

The BiLSTM model achieved the highest accuracy of 0.9286 ± 0.0035. A low standard deviation value indicates the good generalization of the model across the different random seeds. The bidirectional architecture of the model helped to understand the feature dependency in the hybrid feature representation. The CNN model secured 0.8906 ± 0.0092 accuracy, which shows that the model can learn the feature interaction in a specific area. However, the higher standard deviation shows that the model is very sensitive to the training samples and validation variants. Although it is normal for a fine-grained multi-class problem with overlapping behaviors. The LSTM performed poorly with an accuracy of 0.8825 ± 0.0108, which indicates that the sequential relationships alone do not work well for this dataset. By comparing average training and validation loss, the BiLSTM model had a lower average loss of 0.426 as compared to the CNN and LSTM, which had 0.765 and 0.935, respectively. Performance of the BiLSTM shows that it is best suited for the complex family-wise classification tasks and to identify the real-world obfuscated malware variants.

The model confusion matrix can be seen in Fig. 21. RF, DT, and XGB misclassified a few records; however, the KNN classifier misclassified a large portion of the samples. In family-wise classification, among the ML classifiers, XGB performed well and achieved the highest accuracy, and from the DL models, the BiLSTM model achieved the highest accuracy.

Under family-disjoint and category-wise evaluation settings, a moderate but expected performance drop was observed compared to random splits, further confirming that the models were not relying on memorization but instead learned discriminative behavioral patterns.

In all three phases, the LR achieved accuracies of 0.9908 in binary, 0.8714 in category-wise, and 0.7946 in family-wise classification. The RF classifier achieved 0.9939, 0.9734, and 0.9384, and the XGB achieved 0.9945, 0.9788, and 0.9485 accuracies, respectively. The SVM achieved 0.9919, 0.9488, and 0.8671 accuracies while the KNN achieved 0.9631, 0.9186, and 0.8143 accuracies. Performance comparison of ML models for binary and multi-class classification is provided in Table 19. Overall, in all three phases, the XGB classifier stands first with 0.9945 accuracy in binary classification, 0.9788 in category-wise classification, and 0.9485 accuracy in family-wise classification. Other ML classifiers also performed well on the selected feature.

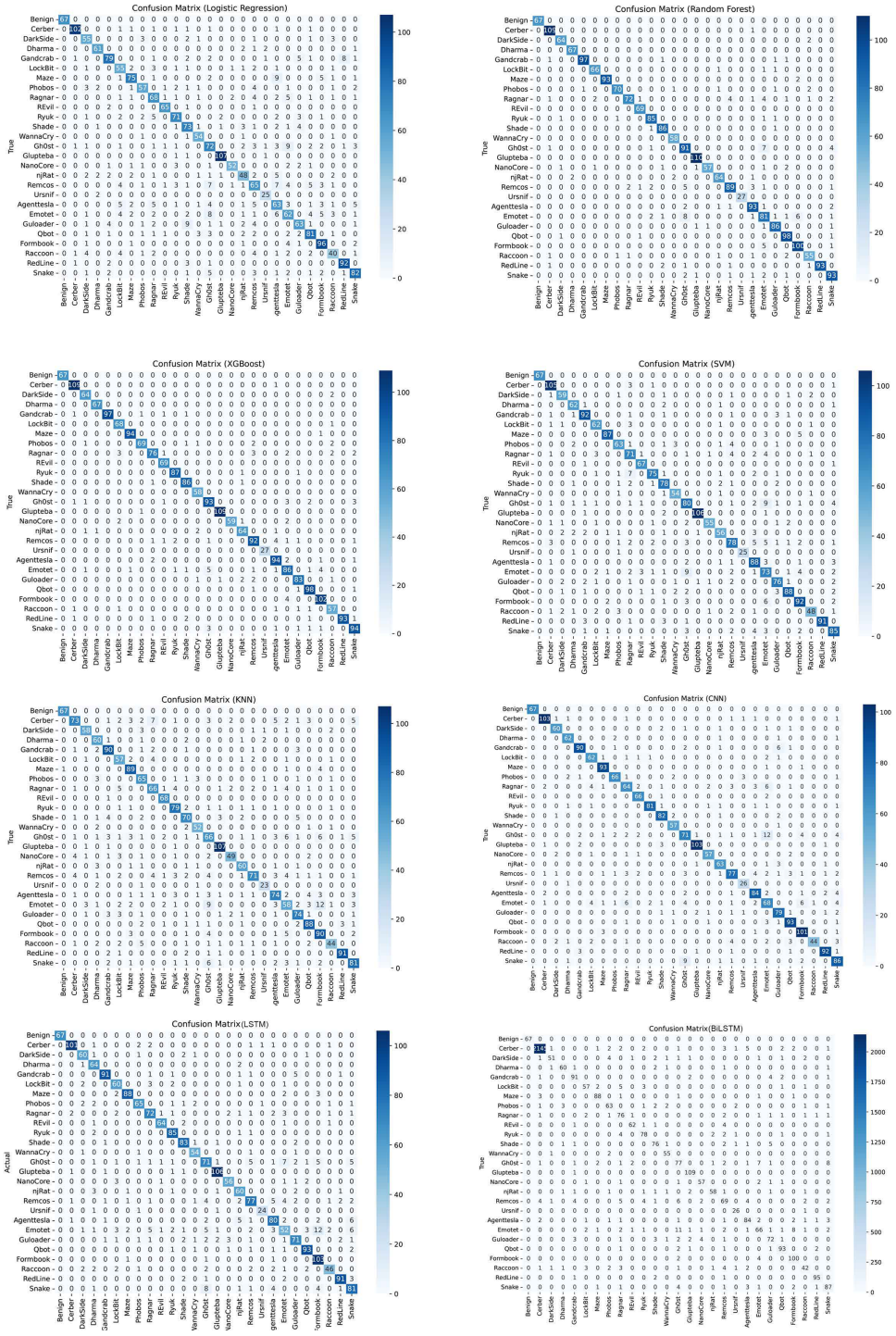

Figure 21: Model's confusion matrix in classifying malware families.

Table 19: Performance comparison of ML models across all phases.

Model	Binary (Acc ± Std)	Category-Wise (Acc ± Std)	Family-Wise (Acc ± Std)	Performance Trend
LR	0.9908 ± 0.0009	0.8714 ± 0.0017	0.7946 ± 0.0050	Large drop with granularity
RF	0.9939 ± 0.0009	0.9734 ± 0.0007	0.9384 ± 0.0040	Strong, stable
XGB	0.9945 ± 0.0014	0.9788 ± 0.0015	0.9485 ± 0.0015	Best overall
SVM	0.9919 ± 0.0015	0.9488 ± 0.0015	0.8671 ± 0.0011	Moderate scalability
KNN	0.9631 ± 0.0017	0.9186 ± 0.0024	0.8143 ± 0.0035	Distance-sensitive

Performance by DL models was also impressive. By validating the dataset, the features selected for Ransomware in the category-wise classification stand at the top, Trojan second, RAT third, and Info Stealer fifth. The CNN achieved 0.9932, 0.9591, and 0.8906 accuracies while the LSTM achieved 0.9917, 0.9502, and 0.8825 accuracies. The BiLSTM achieved 0.9922 in binary, 0.9504 in category-wise, and 0.9286 in family-wise classification. Performance comparison of DL models in all three tasks is provided in Table 20.

Table 20: Performance comparison of DL Models across all phases.

Model	Binary (Acc ± Std)	Category-Wise (Acc ± Std)	Family-Wise (Acc ± Std)	Avg. Loss (Binary/Cat/Fam)
CNN	0.9932 ± 0.0014	0.9591 ± 0.0034	0.8906 ± 0.0092	0.045/0.224/0.765
LSTM	0.9917 ± 0.0009	0.9502 ± 0.0009	0.8825 ± 0.0108	0.254/0.242/0.935
BiLSTM	0.9922 ± 0.0011	0.9504 ± 0.0009	0.9286 ± 0.0035	0.034/0.185/0.426

While stricter evaluation protocols typically lead to slightly reduced absolute performance compared to random splits, they provide a more realistic measure of generalization. The consistent performance across multiple seeds and leakage-resistant splits demonstrates the robustness of the proposed models.

The high accuracy achieved can be attributed to the use of well-engineered behavioral and statistical features that capture stable characteristics of malware activity. Similar trends have been documented in previous malware classification studies conducted in controlled and leakage-aware environments. It is also proven that the models performed well with a variety of random seeds and strict evaluation protocols, and generalize well instead of overfitting or being biased toward specific variants or families.

4.4 Comparison with Prior Work

It would be biased to perform a direct and fully controlled comparison with previous studies on malware classification tasks because the sample collection to dataset creation, and from feature extraction methods to class distributions, and from task formulations to evaluation, are all different. Most of the studies rely on private or non-public datasets, which makes replication unattainable. Therefore, the comparative analysis in this study is intended to be indicative rather than a strict head-to-head evaluation. To ensure transparency, we report standard evaluation metrics and clearly describe our experimental setup. Performance comparisons across studies should be interpreted with caution, and claims are limited to explaining the efficacy of the proposed approach within the scope of the evaluated datasets. Results from ML and DL models indicate that our proposed approach of feature engineering for new dataset creation was impressive and not biased towards machine learning models. To authenticate the performance of the proposed approach, a comparison with the related work in the same malware field is performed. Table 21 provides the comparison with previous works. We performed a comparison on the basis of applied methods, utilized datasets, malware quantity and quality, multi-class classification, and feature engineering techniques. We also compared the accuracies in all classification tasks, and if any contribution to the publishing dataset for future research is also considered.

Table 21: Comparison with related work.

Study/Dataset	Feature Type	Task	Model	Metric	Result (%)	Evaluation Protocol
[72] SOREL-20M	Static PE	Binary/Malware Tagging	LightGBM/FFNN	ROC-AUC	99.8%	Time-based
[73] BODMAS	Static PE	Binary/Family Attribution	Gradient Boosted Decision Tree (GBDT)	F-1 Score, FPR, Accuracy	98.62%, 92% to 97%	Time-based
[70] CIC-MalMem-2022	Memory Features	Binary	Two-layer Stacked Ensemble Learning Model	Accuracy, F1-Score, Precision, Recall	99.00% Accuracy and 99.02% F1-Score	Ensemble Validation
[21] 3655 samples	Dynamic API Call Sequences	Binary	CNN, RNN, SVM, RF, KNN, XGB, and GBC	Accuracy, Precision, Recall, F1-Score, ROC-AUC Score	99%	Random Train-Test Split
[69] 14,860 Samples	Dynamic API-based Hash Features	Binary	MLP, CNN, LSTM, RF, SVM, XGBoost	Accuracy, Precision, Recall, F1-Score	98.40%	K-Fold Cross-Validation with extensive Hyperparameter Tuning
[71] EMBER2024	Static PE	Binary	LightGBM	ROC AUC, PR AUC	99.69%	Temporal Split
AU-PEMal-2025	Hybrid (Static and Dynamic Features)	Binary/Category /Family	LR, RF, XGB, SVM, KNN, CNN, LSTM, BiLSTM	Accuracy, Precision, Recall, F1-Score, Loss	99.37%	Random Split and K-Fold Cross-Validation

The first work [72] provided a large-scale dataset comprising 20 million static features. The dataset has a huge sample collection for research purposes, but the static features at a commercial level can have high detection results. The dataset does not have dynamic behavioral features and is only useful for initial defense. The LightGBM and Feed-forward Neural Network (FFNN) are trained on the dataset with the ROC-AUC evaluation metric. A time-based protocol is implemented, and models achieved an impressive accuracy of 99.8% on static features. Models trained on this dataset would be very helpful as a first defense, but would be unable to identify the evasive and obfuscated malwares which have behavioral characteristics. The second work [73] introduces modular genome-style representations combined with feature hashing and vector embedding techniques to capture more semantic structure. Feature representations beyond basic static counts improve classifier discrimination, particularly under cross-validation setups, though the work still focuses on static feature extraction and conventional model families without dynamic behavioral integration. In the third work [70], a publicly available framework, VoMemlyzer, is used to extract features from memory. A total of 2916 malware samples of three categories and 15 families are collected for analysis. Each sample is run 10 times with a gap of 15 s, and generated 29,298 malicious memory dumps from all samples. To handle an imbalanced dataset, the SMOTE technique is implemented to generate synthetic samples. Six machine learning classifiers are trained and tested on the newly created dataset. The models achieved 99% accuracy in detecting malware. However, no results or details of the multi-class classification area were shared. Moreover, no deep learning model is trained to validate biases, and 29298 memory dumps are generated from a total of 2916 samples.

In the fourth work [21], 2576 malicious and 1080 goodware samples are collected from online sources, analyzed dynamically using Cuckoo sandbox, and extracted features into JSON format. Extracted features are parsed into a CSV file format and processed for machine learning. The proposed technique achieved 99% accuracy; however, no detection or multi-class classification details are provided. Moreover, no features,

other than API calls, are extracted from dynamic analysis to prepare the final dataset. Malware categories and family details are also not shared. MalMem-2022 is a memory-centric dataset that moves beyond static analysis by incorporating memory and runtime features extracted during execution. The methods tested on MalMem-2022 show that they can handle code obfuscation and packing, and they get high classification scores even on malware families that have been obfuscated. Still, MalMem-2022 tests often only look at binary or coarse-grained labeling, and the lack of detailed family-level annotations makes it hard to compare directly for multi-class tasks. In the fifth work [69], SecurAge2, a cybersecurity company based in Singapore, collects 7396 malicious samples. The Cuckoo sandbox dynamically analyzes all samples and uses the hashing trick method to extract features. API-based malware detection is done, and only API calls are used for malware analysis. LightGBM got an accuracy of 97.42%. The last work [71] shows (EMBER-2024), a big static analysis benchmark dataset made to test malware detectors in every way possible. Most of the time, methods tested on EMBER use engineered PE metadata and byte histograms with standard machine learning classifiers. The results reported on this dataset show good binary classification performance, but the evaluation mostly uses standard random train/test splits and doesn't report much on variance across seeds or protocols that are resistant to leakage.

In contrast, the proposed dataset combines both static and dynamic artifacts into a hybrid feature representation, which makes it possible to do strong evaluations of binary, category-wise, and family-wise classification. Our method consistently reports mean ± standard deviation metrics across various random seeds and utilizes leakage-resistant evaluation splits when suitable, thereby offering a more stringent evaluation of generalizability. This study's hybrid feature space outperforms or matches the performance of related work across tasks while fixing problems with feature diversity (static only), evaluation breadth (single split), and label granularity (family coverage). This positions the proposed work as a more comprehensive benchmark for modern malware classification research. In our proposed technique, no oversampling or undersampling technique is applied to make the original dataset balanced or create synthetic samples. More features from static and dynamic analysis are utilized to create the dataset. Machine learning bias is also validated by implementing deep learning-based models. The results from our proposed approach show that the selected features are perfect for identifying obfuscated malware, its categories, and families with higher accuracy as compared to other works. Moreover, results from family-wise malware classification indicate that the proposed approach achieved high accuracy as compared to all work done so far in this field.

4.5 Conclusion and Future Work

Malware has been a great challenge since the advent of the Internet. It infected all fields and changed into different categories and families. It damages systems by stealing, infecting, and encrypting important data or by controlling systems for some illegal activities without the user's permission. Security professionals contributed significantly to mitigating and defending from the day it came to be known. However, its large-scale spread in different fields and new variants made it challenging to detect and classify exact categories and families. Moreover, traditional signature-based or pattern-based techniques also failed to classify new variants. To address this challenge, a hybrid multi-class malware classification framework is proposed using ML and DL-based approaches. A total of 21,703 latest real-world samples of different malware categories and families are collected from online sources and dynamically analyzed in a sandbox environment. A new obfuscated dataset (AU-PEMal-2025) comprised 4 malware categories and 26 families is prepared with unique and important hidden features. To validate the effectiveness of selected features, five ML models and three DL models are trained and tested on the proposed dataset. The ML classifier XGB secured the highest accuracy of 0.9945, 0.9788, and 0.9485 in binary, category-wise, and family-wise classification of malware, respectively. The DL model BiLSTM achieved the highest accuracy of 0.9932, 0.9591, and 0.8906 in binary,

category, and family-wise classification, respectively. Results from models on the newly created dataset show that our proposed technique in preparing datasets by selecting efficient features has resulted in outstanding performance and can be utilized to mitigate unseen malware threats. In the future, we intend to combine memory-based log features in the dataset to add more information related to obfuscated malware families.

Acknowledgement: None.

Funding Statement: This Research is funded by Ongoing Research Funding Program (ORF-2026-947), King Saud University, Riyadh, Saudi Arabia, National Science and Technology Council under grant number NSTC 113-2622-8-029-004-IE.

Author Contributions: Conceptualization, Amjad Hussain and Ayesha Saadia; methodology, Amjad Hussain and Khursheed Aurangzeb; software, Amjad Hussain and Nazish Nawaz; validation, Amir H. Gandomi, Ayesha Saadia and Khursheed Aurangzeb; formal analysis, Chihhsiong Shih; investigation, Amjad Hussain and Ayesha Saadia; resources, Khursheed Aurangzeb and Chihhsiong Shih; writing original draft preparation, Amjad Hussain; writing review and editing, Ayesha Saadia and Khursheed Aurangzeb; data collection, Amjad Hussain and Nazish Nawaz; supervision, Ayesha Saadia; project administration, Ayesha Saadia and Khursheed Aurangzeb; funding acquisition, Chihhsiong Shih. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data are available at https://doi.org/10.5281/zenodo.16924564. Malware dataset with complete details and reference labels that comprise the AU-PEMal-2025 is released under the Creative Commons CC-BY 4.0 license. Additionally, all preprocessing scripts, model wrappers, and evaluation notebooks are open-sourced under the MIT License and are hosted at https://github.com/khbdevelopers/A-Multiclass-Dataset-for-Training-and-Evaluation-of-AI-Models, which also includes a direct link to the dataset.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Packard N. The ARPANET into the Internet: a tale of two networks. Stud Medium Commun. 2020;8(1):37. doi:10.11114/smc.v8i1.4783.
2. Gaurav A, Gupta BB, Panigrahi PK. A comprehensive survey on machine learning approaches for malware detection in IoT-based enterprise information system. Enterp Inf Syst. 2023;17(3):2023764. doi:10.1080/17517575.2021.2023764.
3. Ali shah I, Mehmood A, Khan AN, Elhadef M, Khan AUR. HeuCrip: a malware detection approach for Internet of battlefield things. Clust Comput. 2023;26(2):977–92. doi:10.1007/s10586-022-03618-y.
4. Warikoo A. Perspective chapter: ransomware. In: Malware-detection and defense. London, UK: IntechOpen; 2023. doi:10.5772/intechopen.108433.
5. Talukder S, Talukder Z. A survey on malware detection and analysis tools. Int J Netw Secur Appl. 2020;12(2):37–57. doi:10.5121/ijnsa.2020.12203.
6. O'Kane P, Sezer S, Carlin D. Evolution of ransomware. IET Netw. 2018;7(5):321–7. doi:10.1049/iet-net.2017.0207.
7. Hama Saeed MA. Malware in computer systems: problems and solutions. Int J Inform Dev. 2020;9(1):1–8. doi:10.14421/ijid.2020.09101.
8. Wainwright P, Kettani H. An analysis of botnet models. In: Proceedings of the 2019 3rd International Conference on Compute and Data Analysis; 2019 Mar 14–17; Kahului, HI, USA. p. 116–21. doi:10.1145/3314545.3314562.
9. Tahir R. A study on malware and malware detection techniques. Int J Educ Manag Eng. 2018;8(2):20–30. doi:10.5815/ijeme.2018.02.03.
10. Abusitta A, Li MQ, Fung BCM. Malware classification and composition analysis: a survey of recent developments. J Inf Secur Appl. 2021;59(2):102828. doi:10.1016/j.jisa.2021.102828.

11. Aslan O, Samet R. A comprehensive review on malware detection approaches. IEEE Access. 2020;8:6249–71. doi:10.1109/access.2019.2963724.
12. Hassen M, Carvalho MM, Chan PK. Malware classification using static analysis based features. In: Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI); 2017 Nov 27–Dec 1; Honolulu, HI, USA. p. 1–7. doi:10.1109/ssci.2017.8285426.
13. Nath HV, Mehtre BM. Static malware analysis using machine learning methods. In: Recent trends in computer networks and distributed systems security. Berlin/Heidelberg, Germany: Springer; 2014. p. 440–50. doi:10.1007/978-3-642-54525-2_39.
14. Aghakhani H, Gritti F, Mecca F, Lindorfer M, Ortolani S, Balzarotti D, et al. When malware is packin' heat; limits of machine learning classifiers based on static analysis features. In: Proceedings of the 2020 Network and Distributed System Security Symposium; 2020 Feb 23–26; San Diego, CA, USA. doi:10.14722/ndss.2020.24310.
15. Afianian A, Niksefat S, Sadeghiyan B, Baptiste D. Malware dynamic analysis evasion techniques: a survey. ACM Comput Surv. 2020;52(6):1–28. doi:10.1145/3365001.
16. Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic malware analysis in the modern era—a state of the art survey. ACM Comput Surv. 2020;52(5):1–48. doi:10.1145/3329786.
17. Ijaz M, Durad MH, Ismail M. Static and dynamic malware analysis using machine learning. In: Proceedings of the 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST); 2019 Jan 8–12; Islamabad, Pakistan. p. 687–91. doi:10.1109/ibcast.2019.8667136.
18. Gajrani J, Laxmi V, Tripathi M, Gaur MS, Zemmari A, Mosbah M, et al. Effectiveness of state-of-the-art dynamic analysis techniques in identifying diverse Android malware and future enhancements. In: Advances in computers. Amsterdam, The Netherlands: Elsevier; 2020. p. 73–120. doi:10.1016/bs.adcom.2020.03.002.
19. Murali R, Thangavel P, Velayutham CS. Evolving malware variants as antigens for antivirus systems. Expert Syst Appl. 2023;226(1):120092. doi:10.1016/j.eswa.2023.120092.
20. Chen L, Ye Y, Bourlai T. Adversarial machine learning in malware detection: arms race between evasion attack and defense. In: Proceedings of the 2017 European Intelligence and Security Informatics Conference (EISIC); 2017 Sep 11–13; Athens, Greece. p. 99–106. doi:10.1109/eisic.2017.21.
21. Alshmarni AF, Alliheedi MA. Enhancing malware detection by integrating machine learning with cuckoo sandbox. J Inf Secur Cybercrimes Res. 2024;7(1):85–92. doi:10.26735/wzng1384.
22. Ahmed F, Hasan MK, Alvi ST. A comprehensive review of machine learning-based approaches for malware detection. In: Proceedings of the 2025 5th International Conference on Emerging Smart Technologies and Applications (eSmarTA); 2025 Aug 5–6; Ibb, Yemen. p. 1–8. doi:10.1109/esmarta66764.2025.11132133.
23. Punyasiri D. Signature & behavior based malware detection [dissertation]. Malabe, Sri Lanka: Sri Lanka Institute of Information Technology; 2023.
24. Han R, Kim K, Choi B, Jeong Y. A study on detection of malicious behavior based on host process data using machine learning. Appl Sci. 2023;13(7):4097. doi:10.3390/app13074097.
25. Chandini SB, Rajendra AB. Recent trends and challenges in malware analysis and detection-a comparative study. In: Recent advances in computing sciences. London, UK: CRC Press; 2023. p. 166–72. doi:10.1201/9781003405573-29.
26. Reddy MM, Raghava S, Tarun NN, Reddy SC, Rao GR, Chandra JV. Analysing the malware by using checksum and signature-based detection techniques. Grenze Int J Eng Technol. 2023;9(2):376.
27. Akhtar MS. Analyzing and comparing the effectiveness of various machine learning algorithms for Android malware detection. Adv Mobile Learn Educ Res. 2023;3(1):570–8. doi:10.25082/amler.2023.01.005.
28. Haidros Rahima Manzil H, Naik SM. Detection approaches for Android malware: taxonomy and review analysis. Expert Syst Appl. 2024;238(6):122255. doi:10.1016/j.eswa.2023.122255.
29. Raghunath J, Kiran S, Rao GSN, Kumar JA, Anasuya R, Kumar CS. A machine learning technique to detect behavior based malware. Semicond Optoelectron. 2023;42(1):1268–78. doi:10.1109/confluence47617.2020.9058173.
30. Deldar F, Abadi M. Deep learning for zero-day malware detection and classification: a survey. ACM Comput Surv. 2024;56(2):1–37. doi:10.1145/3605775.

31. Manthena H, Kimmel JC, Abdelsalam M, Gupta M. Analyzing and explaining black-box models for online malware detection. IEEE Access. 2023;11:25237–52. doi:10.1109/access.2023.3255176.
32. Sharifani K, Amini M. Machine learning and deep learning: a review of methods and applications. World Inform Technol Eng J. 2023;10:3897–904.
33. Gopinath M, Sethuraman SC. A comprehensive survey on deep learning based malware detection techniques. Comput Sci Rev. 2023;47(3):100529. doi:10.1016/j.cosrev.2022.100529.
34. Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security; 2011 Jul 20; Pittsburgh, PA, USA. p. 1–7. doi:10.1145/2016904.2016908.
35. Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, Zheng Q. IMCFN: image-based malware classification using fine-tuned convolutional neural network architecture. Comput Netw. 2020;171(1):107138. doi:10.1016/j.comnet.2020.107138.
36. Cui Z, Xue F, Cai X, Cao Y, Wang GG, Chen J. Detection of malicious code variants based on deep learning. IEEE Trans Ind Inf. 2018;14(7):3187–96. doi:10.1109/tii.2018.2822680.
37. Deore M, Tarambale M, Raja Kumar JR, Sakhare S. GRASE: granulometry analysis with semi eager classifier to detect malware. Int J Interact Multimed Artif Intell. 2024;8(6):120–34. doi:10.9781/ijimai.2023.12.002.
38. Souri A, Hosseini R. A state-of-the-art survey of malware detection approaches using data mining techniques. Hum Centric Comput Inf Sci. 2018;8(1):3. doi:10.1186/s13673-018-0125-x.
39. Santos RS, Festijo ED. Generating features of windows portable executable files for static analysis using portable executable reader module (PEFile). In: Proceedings of the 2021 4th International Conference of Computer and Informatics Engineering (IC2IE); 2021 Sep 14–15; Depok, Indonesia. p. 283–8. doi:10.1109/ic2ie53219.2021.9649225.
40. Singh A. Identifying malicious code through reverse engineering. Boston, MA, USA: Springer Science & Business Media; 2009. doi:10.1007/978-0-387-89468-3.
41. Yousaf MS, Durad MH, Ismail M. Implementation of portable executable file analysis framework (PEFAF). In: Proceedings of the 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST); 2019 Jan 8–12; Islamabad, Pakistan. p. 671–5. doi:10.1109/ibcast.2019.8667202.
42. Visual C, Unit B. Microsoft portable executable and common object file format specification. Microsoft Corporation, Revision. 1999 [cited 2026 Jan 1]. Available from: https://www.academia.edu/download/60758328/coff20191001-89712-1v16uez.pdf.
43. Mohanta A, Saldanha A. Malware analysis and detection engineering: a comprehensive approach to detect and analyze modern malware. Berlin/Heidelberg, Germany: Springer; 2020.
44. Al-Khshali HH, Ilyas M. Impact of portable executable header features on malware detection accuracy. Comput Mater Continua. 2023;74(1):153–78. doi:10.32604/cmc.2023.032182.
45. Awan S, Saqib NA. Detection of malicious executables using static and dynamic features of portable executable (pe) file. In: Security, privacy and anonymity in computation, communication and storage. Berlin/Heidelberg, Germany: Springer; 2016. p. 48–58.
46. Kamble MT, Sridevi. Feature extraction and analysis of portable executable malicious file. In: Proceedings of the 2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA); 2022 Sep 8; Gunupur, India. p. 1–6. doi:10.1109/iccsea54677.2022.9936121.
47. Gritzalis S, Aggelis G, Spinellis D. Architectures for secure portable executable content. Internet Res. 1999;9(1):16–24. doi:10.1108/10662249910251273.
48. Kumar A, Aghila G. Portable executable scoring: what is your malicious score? In: Proceedings of the 2014 International Conference on Science Engineering and Management Research (ICSEMR); 2014 Nov 27–29; Chennai, India. p. 1–5. doi:10.1109/icsemr.2014.7043649.
49. Shin D, Kim Y, Byun K, Lee S. Data hiding in windows executable files. Perth, Australia: School of Computer and Information Science, Edith Cowan University; 2008.
50. Jiang J, Zhang F. Detecting portable executable malware by binary code using an artificial evolutionary fuzzy LSTM immune system. Secur Commun Netw. 2021;2021:3578695. doi:10.1155/2021/3578695.

51. Moreira CC, Moreira DC, de S de Sales C Jr. Improving ransomware detection based on portable executable header using xception convolutional neural network. Comput Secur. 2023;130(18):103265. doi:10.1016/j.cose.2023.103265.
52. Belaoued M, Guelib B, Bounaas Y, Derhab A, Boufaida M. Malware detection system based on an in-depth analysis of the portable executable headers. In: Machine learning for networking. Cham, Switzerland: Springer International Publishing; 2019. p. 166–80. doi:10.1007/978-3-030-19945-6_11.
53. Namanya AP, Ali Mirza QK, Al-Mohannadi H, Awan IU, Disso JFP. Detection of malicious portable executables using evidence combinational theory with fuzzy hashing. In: Proceedings of the 2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud); 2016 Aug 22–24; Vienna, Austria. p. 91–8. doi:10.1109/ficloud.2016.21.
54. Buriro A, Buriro AB, Ahmad T, Buriro S, Ullah S. MalwD&C: a quick and accurate machine learning-based approach for malware detection and categorization. Appl Sci. 2023;13(4):2508. doi:10.3390/app13042508.
55. Ahmed M, Afreen N, Ahmed M, Sameer M, Ahamed J. An inception V3 approach for malware classification using machine learning and transfer learning. Int J Intell Netw. 2023;4:11–8. doi:10.1016/j.ijin.2022.11.005.
56. Cakir B, Dogdu E. Malware classification using deep learning methods. In: Proceedings of the ACMSE, 2018 Conference; 2018 Mar 29–31; Richmond, KY, USA. p. 1–5. doi:10.1145/3190645.3190692.
57. Narayanan BN, Djaneye-Boundjou O, Kebede TM. Performance analysis of machine learning and pattern recognition algorithms for Malware classification. In: Proceedings of the 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS); 2016 Jul 25–29; Dayton, OH, USA. p. 338–42. doi:10.1109/naecon.2016.7856826.
58. Lu Y, Graham J, Li J. Deep learning based malware classification using deep residual network. In: Proceedings of the 2019 Modeling, Simulation and Visualization Student Capstone Conference; 2019 Apr 18; Suffolk, VA, USA. p. 1–6.
59. Shah SSH, Ahmad AR, Jamil N, Khan AUR. Memory forensics-based malware detection using computer vision and machine learning. Electronics. 2022;11(16):2579. doi:10.3390/electronics11162579.
60. Sihwail R, Omar K, Akram Zainol Ariffin K. An effective memory analysis for malware detection and classification. Comput Mater Continua. 2021;67(2):2301–20. doi:10.32604/cmc.2021.014510.
61. Khalid O, Ullah S, Ahmad T, Saeed S, Alabbad DA, Aslam M, et al. An insight into the machine-learning-based fileless malware detection. Sensors. 2023;23(2):612. doi:10.3390/s23020612.
62. Hussain A, Asif M, Bin Ahmad M, Mahmood T, Raza MA. Malware detection using machine learning algorithms for windows platform. In: Proceedings of International Conference on Information Technology and Applications. Singapore: Springer Nature; 2022. p. 619–32. doi:10.1007/978-981-16-7618-5_53.
63. Dabas N, Ahlawat P, Sharma P. An effective malware detection method using hybrid feature selection and machine learning algorithms. Arab J Sci Eng. 2023;48(8):9749–67. doi:10.1007/s13369-022-07309-z.
64. Aurangzeb S, Anwar H, Naeem MA, Aleem M. BigRC-EML: big-data based ransomware classification using ensemble machine learning. Clust Comput. 2022;25(5):3405–22. doi:10.1007/s10586-022-03569-4.
65. Kumar S, Janet B, Neelakantan S. Identification of malware families using stacking of textural features and machine learning. Expert Syst Appl. 2022;208(11):118073. doi:10.1016/j.eswa.2022.118073.
66. Banda BPR, Govan B, Roy K, Bryant K. Malware detection using explainable ML models based on feature extraction using API calls. In: Proceedings of the 2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD); 2023 Aug 3–4; Durban, South Africa. p. 1–7. doi:10.1109/icabcd59051.2023.10220515.
67. Daeef AY, Al-Naji A, Chahl J. Lightweight and robust malware detection using dictionaries of API calls. Telecom. 2023;4(4):746–57. doi:10.3390/telecom4040034.
68. Lin TL, Chang HY, Chiang YY, Lin SC, Yang TY, Zhuang CJ, et al. Ransomware detection by distinguishing API call sequences through LSTM and BERT models. Comput J. 2024;67(2):632–41. doi:10.1093/comjnl/bxad005.
69. Yau LQ, Lam YT, Lokesh A, Gupta P, Lim J, Singh IS, et al. A novel feature vector for AI-assisted windows malware detection. In: Proceedings of the 21st IEEE International Conference on Dependable, Autonomic & Secure Computing (DASC 2023); 2023 Nov 14–17; Abu Dhabi, United Arab Emirates. p. 355–61. doi:10.1109/dasc/picom/cbdcom/cy59711.2023.10361451.

70. Carrier T, Victor P, Tekeoglu A, Lashkari AH. Detecting obfuscated malware using memory feature engineering. In: Proceedings of the 8th International Conference on Information Systems Security and Privacy; 2022 Feb 9–11; Online. p. 177–88.
71. Joyce RJ, Miller G, Roth P, Zak R, Zaresky-Williams E, Anderson H, et al. EMBER2024—a benchmark dataset for holistic evaluation of malware classifiers. In: Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2; 2025 Aug 3–7; Toronto, ON, Canada. doi:10.1145/3711896.3737431.
72. Harang R, Rudd EM. SOREL-20M: a large scale benchmark dataset for malicious PE detection. arXiv:2012.07634v1. 2020. doi:10.48550/arXiv.2012.07634.
73. Yang L, Ciptadi A, Laziuk I, Ahmadzadeh A, Wang G. BODMAS: an open dataset for learning based temporal analysis of PE malware. In: Proceedings of the 2021 IEEE Security and Privacy Workshops (SPW); 2021 May 27; San Francisco, CA, USA. doi:10.1109/spw53761.2021.00020.
74. Hussain A, Saadia A, Alhussein M, Gul A, Aurangzeb K. Enhancing ransomware defense: deep learning-based detection and family-wise classification of evolving threats. PeerJ Comput Sci. 2024;10(1):e2546. doi:10.7717/peerj-cs.2546.
75. Hussain A, Saadia A, Alserhani FM. Ransomware detection and family classification using fine-tuned BERT and RoBERTa models. Egypt Inform J. 2025;30(1):100645. doi:10.1016/j.eij.2025.100645.
76. Hosmer DW Jr, Lemeshow S, Sturdivant RX. Applied logistic regression. Hoboken, NJ, USA: John Wiley & Sons; 2013. doi:10.1002/9781118548387.
77. Menard S. Applied logistic regression analysis. Thousand Oaks, CA, USA: SAGE Publications; 2001.
78. Breiman L. Random forests. Mach Learn. 2001;45(1):5–32. doi:10.1023/A:1010933404324.
79. Cutler A, Cutler DR, Stevens JR. Random forests. In: Ensemble machine learning. New York, NY, USA: Springer; 2012. p. 157–75. doi:10.1007/978-1-4419-9326-7_5.
80. Asselman A, Khaldi M, Aammou S. Enhancing the prediction of student performance based on the machine learning XGBoost algorithm. Interact Learn Environ. 2023;31(6):3360–79. doi:10.1080/10494820.2021.1928235.
81. Dhaliwal SS, Nahid AA, Abbas R. Effective intrusion detection system using XGBoost. Information. 2018;9(7):149. doi:10.3390/info9070149.
82. Cortes C, Vapnik V. Support-vector networks. Mach Learn. 1995;20(3):273–97. doi:10.1007/BF00994018.
83. Assegie TA. An optimized KNN model for signature-based malware detection. Tsehay admassu assegie "An optimized KNN model for signature-based malware detection"? Int J Comput Eng Res Trends (IJCERT). 2021;8(2):46–9.
84. Imandoust SB, Bolandraftar M. Application of k-nearest neighbor (KNN) approach for predicting economic events: theoretical background. Int J Eng Res Appl. 2013;3(5):605–10.
85. Chua LO, Yang L. Cellular neural networks: applications. IEEE Trans Circuits Syst. 1988;35(10):1273–90. doi:10.1109/31.7601.
86. Zarandy A, Rekeczky C, Szolgay P, Chua LO. Overview of CNN research: 25 years history and the current trends. In: Proceedings of the 2015 IEEE International Symposium on Circuits and Systems (ISCAS); 2015 May 24–27; Lisbon, Portugal. p. 401–4. doi:10.1109/iscas.2015.7168655.
87. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80. doi:10.1162/neco.1997.9.8.1735.
88. Greff K, Srivastava RK, Koutnik J, Steunebrink BR, Schmidhuber J. LSTM: a search space odyssey. IEEE Trans Neural Netw Learning Syst. 2017;28(10):2222–32. doi:10.1109/tnnls.2016.2582924.
89. Sherstinsky A. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Phys D Nonlinear Phenom. 2020;404(8):132306. doi:10.1016/j.physd.2019.132306.
90. Dang D, Di Troia F, Stamp M. Malware classification using long short-term memory models. arXiv:210302746. 2021.
91. Wu Y, He K. Group normalization. In: Proceedings of the European Conference on Computer Vision (ECCV); 2018 Sep 8–14; Munich, Germany. p. 3–19. doi:10.1007/978-3-030-01261-8_1.