

AI-Driven Malware Detection in Linux: A Survey of Techniques, Challenges, and Future Directions

Aayush Srivastava^{1,*}, Yash Upadhyay¹, Krishna Garg¹, Aniket Kumar Singh¹, Sonu Kumar Thakur¹, Vikram Kapoor¹

Abstract

With the rapid growth of Linux-based systems across cloud computing, Internet of Things (IoT) devices, servers, and embedded platforms, Linux environments have increasingly become attractive targets for sophisticated malware attacks. This survey provides a comprehensive review of Artificial Intelligence (AI)-based approaches for detecting Linux malware, focusing on research published between 2022 and 2026. More than 40 different detection methodologies are examined, covering both traditional Machine Learning (ML) techniques, such as Random Forest, Support Vector Machines (SVM), and related classifiers, and advanced Deep Learning (DL) approaches, including Convolutional Neural Networks (CNNs), Graph Neural Networks (GNNs), and Transformer-based models such as BERT and large language models (LLMs). Particular attention is given to the growing shift from static malware analysis toward behavioral detection, especially through the examination of system call traces and runtime activities. The increasing use of eBPF is also discussed because of its potential to provide efficient, real-time visibility into Linux system behavior with relatively low performance overhead. The reviewed approaches are compared according to important factors such as detection accuracy, computational cost, scalability, adaptability, and resilience to adversarial attacks. Despite significant progress, several challenges remain, including limited availability of standardized Linux malware datasets, difficulties in reproducing experimental results, high computational requirements of some DL models, and insufficient use of Explainable AI (XAI). Based on these findings, the survey identifies key research gaps and outlines promising directions for developing more accurate, efficient, transparent, and robust AI-driven Linux malware detection systems.

Keywords: Linux malware detection, artificial intelligence, deep learning, system call analysis, eBPF, explainable AI

*Author for Correspondence

Aayush Srivastava
E-mail: Aayushsri75@gmail.com

¹ Student, Department of Computer Science and Engineering, Greater Noida Institute of Technology (GNIT), Greater Noida, Uttar Pradesh, India

Received Date: June 24, 2026

Accepted Date: August 05, 2026

Published Date: August 28, 2026

Citation: Aayush Srivastava, Yash Upadhyay, Krishna Garg, Aniket Kumar Singh, Sonu Kumar Thakur, Vikram Kapoor. AI-Driven Malware Detection in Linux: A Survey of Techniques, Challenges, and Future Directions. Journal of Operating Systems Development & Trends. 2026; 13(2): 7–14p.

INTRODUCTION

Linux powers over 90% of public cloud workloads and the majority of IoT devices worldwide. Owing to this dominance, there has been a surge in Linux-specific malware, such as ransomware, rootkits, and botnets (e.g., various variants of Mirai). Unlike Windows malware, which has been the subject of extensive research, Linux malware exploits specific system characteristics, such as the diversity of distributions and architectures (x86, ARM, and MIPS). Traditional signature-based detection methods are no longer as effective against polymorphic and stealthy threats. Consequently, researchers have begun employing AI-based approaches to automate feature extraction and enhance detection rates.

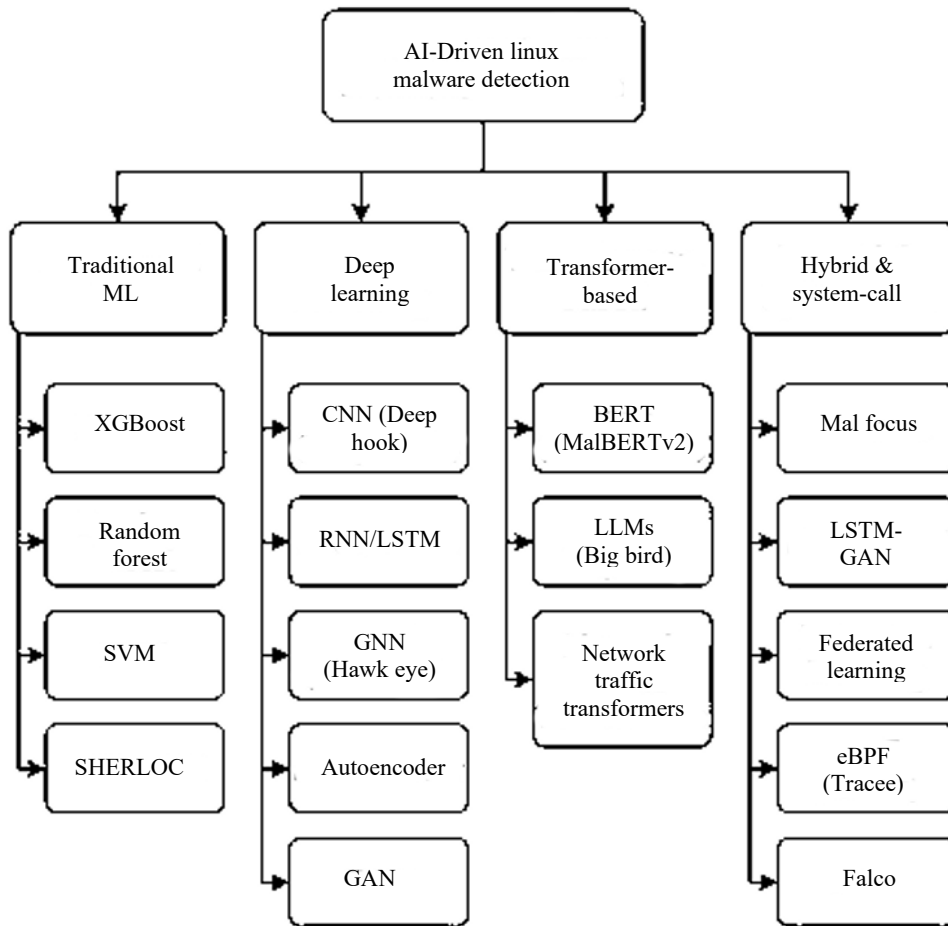


Figure 1. Proposed taxonomy of AI-driven Linux malware detection techniques covered in this survey.

This study reviews the current landscape of AI-based Linux malware detection, categorizing the techniques into traditional ML, deep learning, transformer-based, and hybrid system-call methods. Figure 1 illustrates the proposed taxonomy of the detection techniques covered in this survey.

BACKGROUND

Most Linux malware focuses on ELF files. Examples of common malware include:

Linux Malware Landscape

- *Botnet* (e.g., *Mirai*, *Gafgyt*): Uses infected IoT devices to conduct DDoS attacks.
- *Ransomware* (e.g., *ESXi server locker*): Encrypts critical files.
- *Rootkit*: Hides processes and persists (user-mode and kernel mode versions exist).

Detection Paradigms

- *Static analysis*: Examines the file without executing it (opcodes, strings, and ELF header). It is fast but vulnerable to obfuscation techniques such as UPX packing.
- *Dynamic analysis*: Malware is executed in a controlled environment, and its actions (system calls, network connections) are tracked. It is obfuscation-resistant but resource-intensive.
- *Memory forensics*: Inspects RAM to identify code injection and rootkits.

Extraction Mechanisms

- *Syscalls Trace*: The ordered chain of kernel requests (such as open, read, and fork) is the best way to characterize the behavior. Feature extraction methods can be static (based on ELF files) or dynamic (with strace, eBPF, etc.).

- *Opcode Sequence Analysis*: The statistical examination of machine code opcodes (such as n-grams of mov, push, and pop instructions) helps detect the use of the same opcode sequences by malware from a family.
- *Control Flow Graph (CFG)*: Representing the program as a graph of its basic blocks provides valuable information for GNNs.
- *Network Flow Analysis*: Packet sizes, timestamps, inter-arrival times, and DNS requests can indicate Botnet Command and Control (C2) communications as well as data exfiltration.
- *Cryptocurrency Miner*: Mines cryptocurrencies using processor cycles.

Common Datasets

Research in the area employs various databases:

- *VirusShare*: An extensive collection of malware, including Linux binaries (over 20 K samples), in studies such as MalFocus.
- *IoT-23*: Labeled collection of network packets for 23 scenarios (20 malware + 3 benign) that include threats such as Mirai, Torii, and other botnets.
- *MalFocus Dataset*: Hybrid of public and proprietary collection of nearly 20k Linux samples used in hybrid deep learning studies.
- *Drebin/AMD*: Originally Android-oriented but still used by Linux malware researchers because of the common kernel architecture.

TRADITIONAL MACHINE LEARNING

Traditional ML algorithms serve as a baseline for accuracy and interpretability.

- **XGBoost**. achieved 99.66% accuracy in distinguishing malicious ELF files using a combination of static header features and dynamic syscall counts (2023) [1].
- **Random Forest**. It was effectively utilized (in 2023) to detect ransomware by analyzing file metadata and trusted strings, although it faced difficulties in cases involving highly obfuscated samples [2].
- **SVM**. In Virtualization-based Kernel Rootkit Detection (VKRD, 2023), it was utilized to classify malicious memory and CPU states from the hypervisor level [3].
- **Architecture-agnostic detection**: A 2024 study applied entropy and statistical static analysis to IR opcode sequences to detect IoT malware across x86, ARM, and MIPS architectures, achieving an F1-score of 97% [4].

Limitations. These methods rely heavily on manual feature engineering and are often vulnerable to adversarial modifications, as shown in Figure 2.

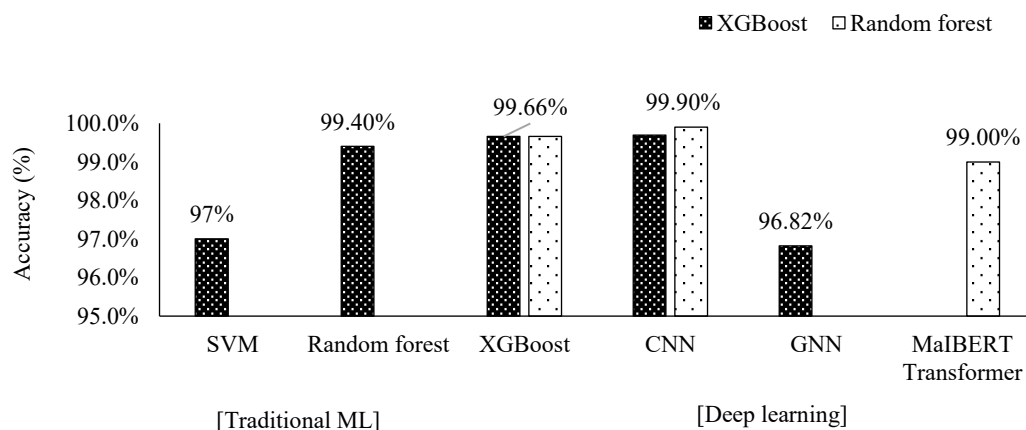


Figure 2. Comparative accuracy of traditional ML vs. deep learning approaches for ELF malware detection.

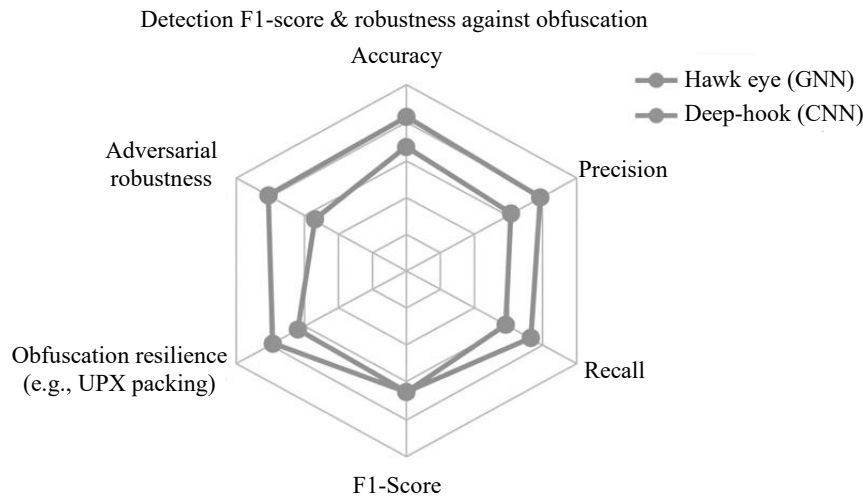


Figure 3. Detection F1-Score and robustness against obfuscation: HawkEye (GNN) vs. Deep-Hook (CNN).

DEEP LEARNING APPROACHES

Deep learning automates feature extraction, making it possible to detect complex nonlinear patterns (Figure 3).

- CNNs. Deep-Hook (2021) utilized CNNs on volatile memory images to detect rootkits with 99.9% accuracy [5]. Multichannel CNNs (2024) have also been applied to IoT malware by combining system calls and opcode sequences.
- RNNs/LSTMs. Sequential models are ideal for analyzing system-call traces. Static Analysis of System Calls (2024) achieved 93.29% accuracy on IoT datasets using LSTMs [6]. Crypto-ransomware detection (2022) combines CNNs with Bi-LSTMs to analyze I/O logs [7].
- GNNs. HawkEye (2021) represents binaries as Control Flow Graphs (CFGs) to detect structural anomalies (96.82% accuracy) [8]. PSI-Graph (2018) uses Deep Graph CNNs (DGCNN) on printable string graphs for cross-architecture detection [9].
- Autoencoders. Sparse Autoencoders (2024) have been successfully used for unsupervised anomaly detection in Linux audit logs, identifying unseen threats with 95% accuracy [10].
- GANs. WGAN-GP (2023) generates synthetic malware opcodes to augment the training datasets, addressing data scarcity.

TRANSFORMER-BASED APPROACHES

The success of Transformers in Natural Language Processing (NLP) has been adapted to malware detection, treating code or system calls as language.

- BERT. MalBERTv2 (2023) treats source code tokens as text and utilizes a pre-trained BERT model to identify malicious intent, achieving high F1 scores ranging from 82% to 99% [11].
- LLMs. In a recent study (2025), Large Language Models (LLMs), such as BigBird, were applied to system call traces using transfer learning, achieving an F1 score of 86% [12]. These models handle long-range dependencies more effectively than RNNs but incur higher inference latency.
- Network Traffic Transformers. These have been applied to high-volume network logs (2025) to detect common threats with an accuracy of 97.8%. They effectively handle the massive scale of modern data centers [13].

HYBRID AND SYSTEM-CALL-BASED APPROACHES

Hybrid Models

Hybrid approaches combine multiple architectures to balance accuracy and comprehensiveness.

- MalFocus (2025). It combines static analysis of library functions with dynamic monitoring of

system calls using a hybrid deep learning (DL) architecture, achieving an accuracy exceeding 96% [14].

- **Federated Learning.** A 2022 study conducted on the IoT-23 datasets demonstrated that federated learning could achieve an accuracy exceeding 90% while simultaneously preserving data privacy on distributed edge devices [15].

System-Call and eBPF

System calls are the definitive interface between user space and the kernel, making them the “gold standard” for Linux behavioral analysis, as shown in Figure 4.

- **Extended BPF (Extended Berkeley Packet Filter).** Tracee (2022–2024) and Real-time Linux Ransomware Detection (2024) leverage eBPF for low-overhead kernel-level visibility. eBPF programs run safely in the kernel, enabling real-time blocking of `rw_encryptions` [16,17].
- **Gantikow.** A rule-based security (2019) is widely used in Kubernetes containers to detect abnormal system calls (e.g., a shell spawning in a database container) [18].
- **Memory Forensics.** Volatility 3 plugins and PUMAKIT (2024) focus on detecting rootkit hooks and hidden processes in kernel memory [19].

Comparative Analysis

Table 1 presents a comparative analysis of the AI-driven Linux malware detection approaches.

CHALLENGES

Limitations of Existing Machine Learning Methods

Despite the interpretability of conventional ML methods, such as Random Forest and SVMs, these methods have major disadvantages.

- *Shallow Feature Dependence:* Traditional ML approaches depend largely on manual feature engineering (e.g., statistical properties of n-grams in system calls). It is vulnerable to countermeasures, including polymorphisms and packing.
- *Scalability:* Models such as SVMs cannot efficiently process very high-dimensional input vectors of system call traces.

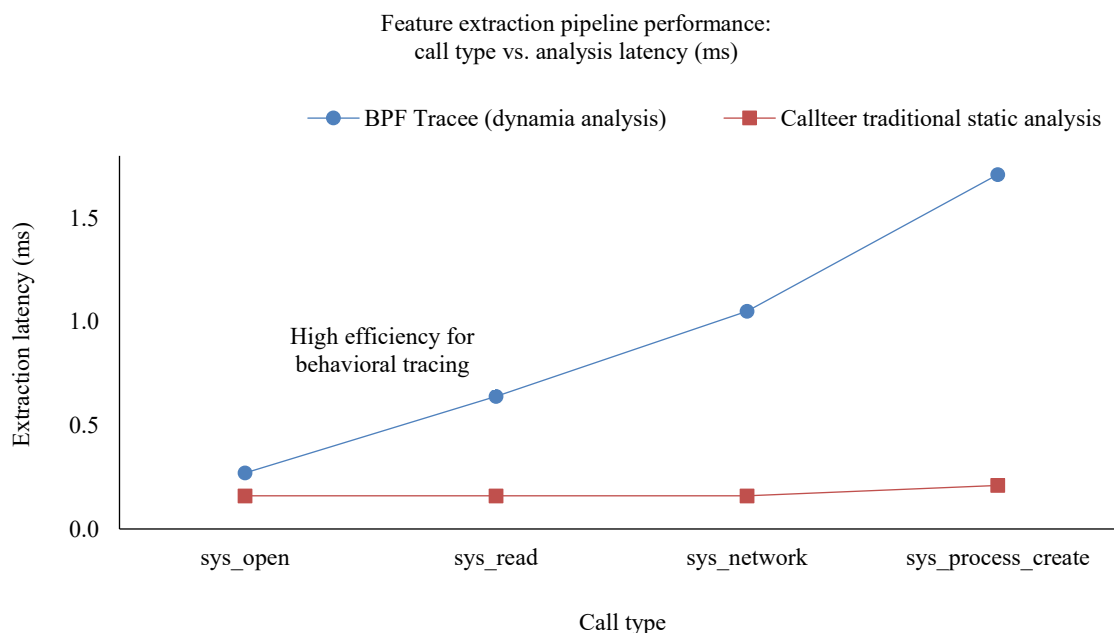


Figure 4. Feature extraction pipeline performance: system call type vs. analysis latency comparing eBPF Tracee (dynamic) and traditional static analysis.

Table 1. Comparative analysis of AI-driven Linux malware detection approaches.

Paper/approach	Year technique	Dataset	Accuracy / F1	Limitations
Deep-Hook [5]	2021 CNN	Memory Dumps (22k)	99.90%	Computationally intensive
XGBoost [1]	2023 XGBoost	Linux ELF	99.66%	Feature engineering overhead
MalFocus [14]	2025 Hybrid DL	Public/Private (~20k)	>98%	Hybrid model complexity
Transformer [13]	2025 Transformer	Network Traffic	97.80%	Resource intensive
Crypto-Ransom [7]	2022 CNN, Bi-LSTM	Custom I/O Logs	97.78%	Requires I/O window
Arch-Agnostic [4]	2024 Static (Entropy)	IoT (Multi-Arch)	97% F1	Binary lifting correctness
HawkEye [8]	2021 GNN	Linux Malware	96.82%	Graph construction overhead
Sparse AE [10]	2024 Sparse AE	Linux Audit Logs	95%	False positives
Static IoT [6]	2024 RNN, LSTM	IoT Dataset	93.29%	Vulnerable to obfuscation
Federated [15]	2022 Federated Learning	IoT-23	>90%	Poisoning attacks
MalBERTv2 [11]	2023 BERT	Drebin/AMD	82–99% F1	Needs source code
LLM SysCall [12]	2025 LLM (BigBird)	System Traces	86% F1	Inference latency
PSI-Graph [9]	2018 DGCNN	Multi-Arch	High	Relies on string presence

Limitations of Deep Learning Methods

DL methods have been demonstrated to offer more reliable performance; however, they suffer from certain issues.

- *Adversarial Robustness*: DL classifiers have shown poor robustness to adversarial attacks. One possible attack vector involves adding non-functional “no-op” system calls or attaching random garbage data [20] as an addition to evade classification.
- *Lack of Interpretability (“Black Box”)*: Deep Neural Networks lack intrinsic interpretability. In SOC, an analyst must be able to reason why a suspicious binary has been detected, which makes XAI for binary classification a serious challenge.
- *Computational Complexity*: Modern DL classifiers, such as MalBERTv2, which are built using the BERT transformer, demand huge amounts of computing resources.

Dataset Limitations

The limitations of AI detection capabilities are related to the limitations of datasets, as they have the following issues:

- *Lack of Large Linux Datasets*: Contrary to Windows and Android malware, Linux does not contain large publicly available databases. The Drebin dataset for Android applications and VirusShare can be utilized as possible sources.
- *Label Inconsistencies*: Labeling malware based on the AV scanning output may result in noise, thus negatively affecting the ability of algorithms to generalize.
- *Outdatedness*: Malware tends to develop very fast (Concept Drift), making the dataset outdated within a short period of time.

Real-World Deployment Challenges

Connecting theory and practice when it comes to balancing academic and practical needs is achieved by:

- *Kernel dependency*: Tools such as dynamic analysis tools (eBPF) have a close relationship with the kernel version. Ensuring compatibility across diverse Linux distributions is a challenging task.
- *False positives cannot be tolerated*: On servers, the interruption of a valid operation can lead to downtime. Many existing studies tend to maximize Accuracy or F1 Score without restricting the false positive rate (<0.01%).
- *Latency restrictions*: Prevention in real time requires instant decision-making. The latency introduced by the inference pipeline may become a deal-breaker.

FUTURE DIRECTIONS

- Explainable Artificial Intelligence (XAI). Building models that do not merely classify malicious software but provide explanations (i.e., the particular assembly code sequences or syscall traces causing the classification).
- eBPF-Native AI. Placing the inference model physically close to the data source. Lightweight models, such as Decision Trees, can be directly run in an eBPF virtual machine for zero-latency prevention.
- Cross-Platform Generalization. Building “architecture-agnostic” models that efficiently learn semantics regardless of the platform, such as x86, ARM, or MIPS, without any additional training.
- Federated Learning for Privacy Preservation Providing malware detection capabilities without sharing any private data with other parties through federated learning schemes.

CONCLUSION

The area of Linux malware detection using AI technology is developing quickly, from signature-based approaches to dynamic analysis with Deep Learning and Transformer models. Although Deep-Hook and HawkEye are excellent examples of models capable of achieving more than 96% accuracy, their application in a production environment faces challenges in the form of resource usage and lack of explainability. Future efforts should be directed toward the development of hybrid techniques that will benefit from the observability provided by eBPF together with lightweight AI capabilities.

Acknowledgments

The authors would like to thank the Greater Noida Institute of Technology (GNIT) for providing the resources and support necessary to conduct this research.

Disclosure of Interests

The authors declare no competing interests relevant to the content of this article.

REFERENCES

1. Sharma S, Gupta N, Bundela B. A GWO-XGBoost machine learning classifier for detecting malware executables. In: 2023 International Conference on Disruptive Technologies (ICDT); 2023 May 11. IEEE; 2023. p. 247-51. Available from: <https://ieeexplore.ieee.org/document/10150993>
2. Singh A, Mushtaq Z, Abosaq HA, Mursal SNF, Irfan M, Nowakowski G. Enhancing ransomware attack detection using transfer learning and deep learning ensemble models on cloud-encrypted data. *Electronics (Basel)*. 2023;12(18):3899. Available from: <https://www.mdpi.com/2079-9292/12/18/3899>
3. Tian D, Ma R, Jia X, Hu C. A kernel rootkit detection approach based on virtualization and machine learning. *IEEE Access*. 2019;7:91657-66.
4. Shim HJ, Jung JW, Kinnunen T, Evans N, Bonastre JF, Lapidot I. a-DCF: an architecture agnostic metric with application to spoofing-robust speaker verification [Preprint]. 2024. arXiv:2403.01355.
5. Landman T, Nissim N. Deep-Hook: a trusted deep learning-based framework for unknown malware detection and classification in Linux cloud environments. *Neural Netw*. 2021;144:648-85.
6. Ramamoorthy J, Gupta K, Kafle RC, Shashidhar NK, Varol C. A novel static analysis approach using system calls for Linux IoT malware detection. *Electronics (Basel)*. 2024;13(15):2906.
7. Alqahtani A, Sheldon FT. A survey of crypto ransomware attack detection methodologies: an evolving outlook. *Sensors (Basel)*. 2022;22(5):1837.
8. Xu P, Zhang Y, Eckert C, Zarras A. HawkEye: cross-platform malware detection with representation learning on graphs. In: *International Conference on Artificial Neural Networks*; 2021 Sep 7. Cham: Springer International Publishing; 2021. p. 127-38.
9. Nguyen HT, Ngo QD, Le VH. IoT botnet detection approach based on PSI graph and DGCNN classifier. In: *2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP)*; 2018 Sep 28. IEEE; 2018. p. 118-22.
10. Bussmann B, Leask P, Nanda N. BatchTopK sparse autoencoders [Preprint]. 2024. arXiv:2412.06410.

11. Rahali A, Akhloufi MA. MalBERTv2: code aware BERT-based model for malware identification. *Big Data Cogn Comput.* 2023;7(2):60.
12. Akhtar S, Khan S, Parkinson S. LLM-based event log analysis techniques: a survey [Preprint]. 2025. arXiv:2502.00677.
13. Akuthota UC, Bhargava L. Transformer-based intrusion detection for IoT networks. *IEEE Internet Things J.* 2025;12(5):6062-7.
14. Huang W, Lin C, Xiang L, Zhang Z, Meng G, Xue L, et al. Malfocus: locating malicious modules in malware based on hybrid deep learning. *IEEE Trans Dependable Secure Comput.* 2025;22(5):5060-76.
15. Rey V, Sánchez PM, Celdrán AH, Bovet G. Federated learning for malware detection in IoT devices. *Comput Netw.* 2022;204:108693.
16. Brodzik A, Malec-Kruszyński T, Niewolski W, Tkaczyk M, Bocianiak K, Loui SY. Ransomware detection using machine learning in the Linux kernel [Preprint]. 2024. arXiv:2409.06452.
17. Kim R, Ryu J, Kim S, Lee S, Kim S. Detecting cryptojacking containers using eBPF-based security runtime and machine learning. *Electronics (Basel).* 2025;14(6):1208.
18. Gantikow H, Reich C, Knahl M, Clarke NL. Rule-based security monitoring of containerized workloads. In: *CLOSER 2019*; 2019 May 2. p. 543-50.
19. The Hacker News. New Linux rootkit PUMAKIT uses advanced stealth techniques to evade detection [Internet]. *daily.dev*; 2024. Available from: <https://daily.dev/posts/new-linux-rootkit-pumakit-uses-advanced-stealth-techniques-to-evade-detection-q4wb9qedf>
20. Lin HY, Biggio B. Adversarial machine learning: attacks from laboratories to the real world. *Computer.* 2021;54(5):56-60.