

Malware Detection at the Edge with Lightweight LLMs: A Performance Evaluation

Christian Rondanini*, Barbara Carminati*, Elena Ferrari*, Antonio Gaudiano*

*DiSTA, University of Insubria, Italy

{christian.rondanini, barbara.carminati, elena.ferrari, agaudiano}@uninsubria.it

Ashish Kundu†

†Cisco Research, USA

ashkundu@cisco.com

Abstract—The rapid evolution of malware attacks calls for the development of innovative detection methods, especially in resource-constrained edge computing. Traditional detection techniques struggle to keep up with modern malware’s sophistication and adaptability, prompting a shift towards advanced methodologies like those leveraging Large Language Models (LLMs) for enhanced malware detection. However, deploying LLMs for malware detection directly at edge devices raises several challenges, including ensuring accuracy in constrained environments and addressing edge devices’ energy and computational limits. To tackle these challenges, this paper proposes an architecture leveraging lightweight LLMs’ strengths while addressing limitations like reduced accuracy and insufficient computational power. To evaluate the effectiveness of the proposed lightweight LLM-based approach for edge computing, we perform an extensive experimental evaluation using several state-of-the-art lightweight LLMs. We test them with several publicly available datasets specifically designed for edge and IoT scenarios and different edge nodes with varying computational power and characteristics.

Index Terms—Malware; Edge Computing; Large Language Model; Security

I. INTRODUCTION

In recent years, the rapid evolution of malware attacks has necessitated the development of innovative approaches for their detection, particularly in the resource-constrained edge computing domain. While foundational, traditional detection techniques have struggled to keep pace with modern malware’s increasing sophistication and adaptability. This has prompted a shift towards exploring advanced methodologies, including using lightweight Large Language Models (LLMs), to enhance malware detection capabilities in edge environments.

Among the various lightweight LLMs, models such as DistilBERT, TinyBERT, Llama-3.2-1B, DistilGPT-2, and TinyT5 have emerged as promising solutions. These models leverage techniques such as distillation and pruning to significantly reduce their size and computational requirements, making them more suitable for edge-devices deployment. Despite their smaller footprint, these models retain much of their larger counterparts’ pattern recognition and contextual understanding capabilities, allowing them to process and analyze complex, unstructured data streams effectively. In the context of malware detection, they offer the potential for improved accuracy, real-time adaptability, and continuous learning while addressing the strict energy, storage, and computational constraints of edge computing environments.

However, deploying LLMs for malware detection in edge computing is not without challenges. Key issues include:

- **Model performance:** maintaining high accuracy under the constraints of edge environments remains a significant hurdle. While LLMs excel in natural language understanding and pattern recognition, their generalizability across diverse edge scenarios is often limited, particularly when faced with malware’s dynamic and adaptive nature [1].
- **Susceptibility to bias:** overfitting during fine-tuning can make LLMs prone to biases, potentially limiting their ability to detect novel malware threats. To mitigate these risks, robust generalization requires diverse training datasets and advanced regularization techniques.
- **Feasibility of deployment:** LLMs’ computational and energy demands challenge their practicality on resource-constrained edge devices. Addressing this requires further advancements in lightweight model design through techniques such as quantization, knowledge distillation, and efficient architecture optimization.

To tackle these challenges, this paper proposes an architecture that leverages the strengths of lightweight LLMs while addressing their limitations in edge computing scenarios, such as reduced accuracy and insufficient computational power. The proposed architecture enhances malware detection capabilities, improves adaptability, and supports continuous learning in resource-constrained.

To evaluate the effectiveness of the proposed lightweight LLM-based approach for edge computing, we perform an extensive experimental evaluation using several state-of-the-art lightweight LLMs (such as distilBERT [2] and TinyBERT [3]).

We run our experiments on different publicly available datasets, specifically designed for edge and IoT scenarios, including Edge-IIoTset [4], X-IIoTID [5], TON-IoT [6], and the CIC IoT Dataset 2023 [7]. These datasets provide a rich variety of real-world network traffic and malware samples, enabling a comprehensive assessment of the models’ performance in detecting sophisticated threats. Finally, we test the feasibility of the proposed approach on edge nodes with different computational power and characteristics. To the best of our knowledge, we are not aware of other proposals performing such an extensive performance evaluation on the deployment of lightweight LLMs in the edge scenario.

The remainder of this paper is organized as follows. Section II provides background information on LLM and relevant datasets. Section III surveys related work. Section IV introduces our proposed architecture. The results of our extensive performance evaluation are presented in Section V, whereas

Section VI concludes the paper and outlines future research directions.

II. BACKGROUND

In this section, we first present the key elements of LLMs (Section II-A). Then, we illustrate the public datasets we used in our experimental evaluation (Section II-B).

A. Large language models

The earliest language models were developed in the early 1990s [8], using basic machine learning approaches and simpler probabilistic models, such as n-grams [9] that estimated word probabilities based on previous context. A significant advancement in the field was the introduction of neural networks, such as recurrent neural networks (RNNs) (e.g., the RNNLM [10]) and their variations, such as long short-term memory (LSTM) [11]. These models became popular for a variety of natural language applications, including text creation, machine translation, and text categorization, since they provide improved text sequence handling capabilities. Nevertheless, they still struggled to effectively convey long-term interdependence in text.

However, it was 2017 that marked a significant breakthrough in natural language processing (NLP), with the introduction of the transformer architecture by Vaswani et al. [12]. This innovation enabled parallel processing while effectively capturing long-range dependencies in data, revolutionizing the field. Its self-attention mechanisms and feed-forward neural networks allow models to assign varying weights to words in an input sequence based on their contextual relevance. Additionally, multi-head attention extends self-attention by enabling the model to simultaneously analyze multiple aspects of the data. This capability allows models to dynamically adjust the importance of different words within a context while simultaneously processing input sequences in parallel, leading to significant improvements in both efficiency and performance. The transformer architecture includes encoder and decoder modules enabled by self-attention mechanisms. Thanks to self-attention and feedforward, the encoder processes the input sentences and produces a representation of it. In contrast, the decoder generates a sequence with respect to the encoder representation (if any) and produces a prediction of the next token, again thanks to self-attention and cross-attention.

The advent of the transformer architecture and the availability of powerful yet affordable hardware led to the era of Large Language Models (LLMs). Most LLMs today are based on the transformer architecture, and they mainly differ by the number of parameters used, with those exceeding one billion parameters categorized as large pre-trained language models (PLMs). However, at the time of writing, there is no formal agreement within the academic community on the minimum parameter threshold for LLMs, as model capacity is closely tied to the size of the training dataset and the computational resources utilized.

To categorize existing LLMs, in what follows, we adopt the framework proposed by Pan et al. [13], which organizes them into three main categories: encoder-only, encoder-decoder, and decoder-only models.

The encoder-only models use only an encoder to encode sentences and understand the relationships between words. Encoder-only models are mainly designed for tasks that involve understanding, analyzing, or classifying text. A notable example in this category is BERT (Bidirectional Encoder Representations from Transformers) [14], introduced by Google in 2018. In contrast, decoder-only models only adopt the decoder module to generate target output text. These models are mainly designed for generative tasks, such as the generation of code or text. A notable example in this category is the Generative Pre-trained Transformer (GPT) model [15]. Lastly, encoder-decoder models combine the capabilities of previously described models to handle complex tasks, such as machine translation and text-to-text transformation. A notable example in the encoder-decoder category is T5 (Text-To-Text Transfer Transformer)[16] by Google.

A further step in LLM evolution is given by the growing emphasis on developing light LLM models, also called small language models (SLMs) [17]. These models are designed to require significantly reduced computational resources while still delivering high accuracy and performance levels. Techniques such as model pruning, quantization, knowledge distillation, as well as efficient architecture design are commonly employed to achieve a balance between computational requirements and achieved accuracy.

Examples of SLMs are DistilBERT [2] and TinyBERT [3], which are respectively a distilled and tiny version of the BERT model. DistilBERT retains 97% of BERT's original performance while being 40% smaller and 60% faster. This substantial reduction in size (i.e., a lesser number of parameters) and inference time is achieved through a process called knowledge distillation, where a smaller student model learns to replicate the outputs of a larger teacher model. DistilBERT's encoder-only architecture, focused on bidirectional contextual understanding, makes it especially effective for classification tasks, such as distinguishing between malicious and benign activities.

Unlike distilled models, tiny models are not derived from larger ones but are explicitly built to be lightweight and efficient. TinyBERT builds on the BERT architecture with a more aggressive optimization strategy, employing a two-stage distillation process to maximize both speed and memory efficiency. This results in a model significantly smaller and faster than its predecessor, yet capable of preserving critical linguistic features. TinyBERT is particularly advantageous for edge-based malware detection in scenarios where rapid, real-time analysis of contextual data is required, such as identifying anomalous patterns in network traffic or system logs.

For decoder-only models, distilGPT2 [2] and Llama 3.2 1B¹ are popular distilled versions of the GPT model [15], and

¹<https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct>

LLama (Large Language Model Meta AI) [18], respectively. DistilGPT2 is a streamlined version of the GPT-2 architecture, designed primarily for generative tasks but adaptable for classification when appropriately fine-tuned. Unlike encoder-only models, DistilGPT2 decoder-only structure is suitable for tasks requiring a focus on sequential dependencies, such as the detection of obfuscated or polymorphic malware encoded in text or script files. By reducing the computational overhead of the original GPT-2 model, DistilGPT2 can be an efficient solution for edge devices that require nuanced and context-sensitive anomaly detection.

Llama 3.2 1B [19] represents a distilled version of the LLaMA architecture optimized for inference on edge devices. Through techniques such as pruning, quantization, and architectural simplifications, Llama 3.2 1B achieves significant efficiency gains while maintaining much of the performance and versatility of its larger counterpart. Its design focuses on delivering high throughput and low latency, making it well-suited for resource-constrained environments that demand rapid and accurate threat analysis. Llama 3.2 1B excels in processing complex input patterns and can be particularly effective for identifying advanced persistent threats (APTs) or zero-day exploits.

Lastly, for the encoder-decoder category, TinyT5 [20] is a distilled version of Google T5 [16]. By leveraging knowledge distillation, TinyT5 achieves a smaller model size and faster inference times, making it highly suitable for deployment on edge devices with limited computational and memory resources. This lightweight model is optimized to handle a wide range of natural language processing tasks in a text-to-text format, maintaining strong performance across benchmarks.

Table I summarizes the main characteristics of the above discussed models. In the table, Vocabulary size is the number of unique tokens (words or chars) in the model's vocabulary, which determines the size of the input/output layers. Hidden layers express the number of intermediate layers between the input and output layers. Hidden size denotes the number of neurons in each hidden layer. The number of heads indicates the number of attention heads from the transformer architecture.

B. Datasets

The selection of datasets is a key factor in training and evaluating LLMs for malware detection in edge environments. In [21], we surveyed a comprehensive list of public datasets that can be exploited for testing LLM-based malware detection techniques in the edge computing scenario. From these datasets, we have selected those that focus on network traffic and are, to the best of our knowledge, the most appropriate for our testing activities, as they offer a good size, age and feature distribution combination. A detailed overview of the considered datasets is provided in the following, whereas Table II summarizes their main characteristics.

- **X-IIoTID.** This dataset was created at the University of New South Wales (UNSW), Canberra[5]. It constitutes a connectivity- and device-agnostic benchmark for the

evaluation and training of machine learning (ML) and deep learning (DL) intrusion detection systems (IDS) within the Internet of Things (IoT) and Industrial IoT (IIoT) environments. The dataset was created in a testbed mimicking the Industrial Internet Reference Architecture (IIRA). This architecture integrates various industrial and IoT devices, communication protocols, cloud services, and attack simulation tools. The dataset includes 820,834 records, comprising 421,417 normal and 399,417 malicious instances, with 65 features extracted from network traffic, device resources, and logs associated with devices and alerts. The captured traffic employs a variety of protocols, such as Modbus, MQTT, TCP, CoAP, and SMTP, which reflect the heterogeneous characteristics of IIoT ecosystems. Malicious records were systematically generated using three established frameworks: the Cyber Kill Chain (CKC), the Malware Lifecycle Chain (MALC), and the MITRE ATT&CK framework. The simulated attack processes were carefully designed to span multiple stages of a typical cyberattack lifecycle, facilitating a comprehensive assessment of IDS effectiveness. These stages encompass reconnaissance, weaponization, exploitation, lateral movement, command and control (C2), data exfiltration, tampering, cryptographic ransomware deployment, and ransomware-induced denial of service (Ransom DoS).

- **Edge-IIoTset.** The Edge-IIoTset [4] dataset, with a size of 1.2 GB, represents a well-calibrated balance between data richness and computational manageability. Tailored specifically for edge computing, it captures diverse IIoT traffic patterns and an extensive range of malicious activities, offering a broader threat landscape compared to smaller datasets, such as X-IIoTID [5]. Edge-IIoTset includes detailed metadata, facilitating a deeper understanding and detection of evolving malware tactics. Its medium-scale size supports advanced feature engineering while remaining suitable for lightweight models, ensuring compatibility with resource-constrained edge devices. The dataset comprises both benign and malicious traffic samples, encompassing a variety of attack types such as Distributed Denial-of-Service (DDoS), botnets, and ransomware. It also contains labelled data with network flow characteristics, system logs, and behavioural indicators, enabling comprehensive analysis of IIoT security threats.
- **TON-IoT.** The TON-IoT dataset, released in 2021 by Moustafa et al. [6], stands as one of the most comprehensive datasets for IoT-based threat detection, offering a detailed representation of network environments through diverse data sources and modalities. Designed within a realistic network setting, it captures an extensive range of attack scenarios, including DoS, DDoS, Man-in-the-Middle, ransomware, and password cracking. Data collection was carried out across multiple streams, encompassing network traffic, operating system logs, and telemetry data from IoT sensors, providing a holistic foundation for malware detection. The dataset comprises 223 million

Model	Distilled from	Hidden size	# Heads	# Layers	Vocab size	# Parameters	Speedup	Use case
distilGPT2	GPT-2	768	12	6	50257	81,912,576	2x faster	Text generation
distilBERT	BERT-base	768	12	6	30522	66,362,880	60% faster	NLP (classification, QA)
TinyBERT	BERT-base	312	12	4	30522	14,350,248	80% faster	NLP (lightweight BERT)
Llama-3.2-1B	Llama 3	2048	32	16	128256	1,235,814,400	N/A	General-purpose LLM
TinyT5	T5-base	256	4	4	32128	15,570,688	2x faster	Text-to-text (summarization, translation)

Table I
SLMs CHARACTERISTICS AND USE CASES.

records with 44 features, supporting a wide variety of analytical approaches. Its inclusion of multimodal data enables cross-layer analysis and the integration of multiple data streams, which are essential for developing advanced detection models tailored to complex and multi-vector attacks. The size and versatility of TON-IoT make it particularly valuable for training models designed for edge computing deployments, addressing the challenges of real-world IoT environments while enhancing the detection of sophisticated cyber threats.

- **CIC IoT Dataset 2023.** The CIC IoT dataset [7] is the largest discussed in this paper and spans 12.8 GB. It provides a detailed representation of IoT network traffic and malicious behaviours, including botnet activities, ransomware, and stealthy fileless malware. Collected from 105 IoT devices, the dataset encompasses approximately 548 GB of raw traffic data, capturing 33 distinct attack types, such as DDoS, DoS, Reconnaissance, Web-based attacks, Brute Force, Spoofing, and Mirai-based threats. Each of its 46.69 million records includes 47 features, such as flow duration, protocol type, packet size (minimum, average, and maximum), and packet capture timestamps. The dataset's comprehensive coverage and detailed labelling make it ideal for training high-capacity models. The inclusion of diverse attack scenarios and extensive features ensures that models trained on it can generalize effectively to a wide range of IoT security contexts. However, the dataset's large size poses challenges for direct edge deployment, making it more suitable for pre-training models in centralized environments. These models can then be distilled or fine-tuned for deployment on edge devices, enabling effective detection of complex and multi-vector attacks while addressing the constraints of resource-limited IoT environments.

III. RELATED WORK

Recent research has produced several approaches to secure edge computing environments through the usage of LLMs. For instance, *SecurityBERT* [22] is an approach for IoT/IIoT malware detection through a privacy-focused BERT-based lightweight model. The system can identify various malware types, including ransomware, spyware, and keyloggers. Its novel privacy-preserving fixed-length encoding (PPFLE)

technique converts unstructured network data into BERT-compatible text while using cryptographic hash functions to protect sensitive information. Trained on real network traffic, logs, and alerts, *SecurityBERT* achieves 98.2% accuracy in distinguishing between 14 malware types.

Zhang et al. (2024) [23] developed *EdgeShard*, a framework that optimizes LLM deployment on edge computing by enabling collaborative inference among heterogeneous edge devices and cloud servers. *EdgeShard*'s innovative approach involves breaking down LLMs into smaller components distributed across multiple devices while using optimization algorithms to determine optimal resource allocation and model partitioning. Experiments conducted with Llama2 models demonstrate significant improvements in both latency reduction and throughput compared to baseline methods (e.g., strategies to process all data on the edge device or split processing equally between the edge device and the cloud). However, *EdgeShard* is not used for malware detection, rather for text generation.

Additionally, [24] presents a novel framework designed to optimize the deployment of LLMs in edge computing settings through model-based reinforcement learning (MBRL). The approach focuses on determining the most effective splitting point within the LLM architecture to improve efficiency and performance under wireless network conditions. By modelling the problem as a Markov Decision Process and integrating a reward surrogate model, the framework adeptly balances inference performance with computational demands on end-user devices. Experimental results highlight the framework's success in achieving this balance. However, [24] has not been designed for malware detection tasks.

Ravi et al. (2023) [25] presents an approach to malware detection in resource-constrained edge devices by utilizing a lightweight vision transformer (ViT) model. Their method converts executable bytecodes into images for analysis, implementing the system on a PYNQ Z1 FPGA board with limited resources. The ViT model demonstrated superior accuracy to traditional CNN architectures, like InceptionV3 and ResNet-50, proving particularly effective for real-time edge device applications.

While the above-mentioned proposals present promising results for securing edge computing environments through LLMs, their approaches involve model optimization or mod-

Name	Description	Size (GB)	Records	Attack types	# of Features	Source	Year	Format
X-IIoTID	It contains attackers behaviors on Industrial IoT devices	0.38	0.82M	IG, IT, DS	65	Real devices	2022	CSV
EdgeIIoTset	Collection of IoT / IIoT device data	1.2	72M	IG, DS	46	Simulated	2022	PCAP, CSV
TON-IoT	Collection of network traffic data for IoT environment	3.68	223M	IT, DS	44	Real-world and simulated	2021	CSV, PCAP, BLG, Logs
CIC IoT 2023	A real-time dataset of large-scale attacks in IoT	12.8	46.69M	IG, IT, DS	47	Real-world devices, Simulation	2023	CSV, PCAP

Table II

PUBLIC DATASETS SUITABLE FOR EDGE COMPUTING MALWARE DETECTION. INFORMATION GATHERING (IG), INFORMATION THEFT (IT), DENIAL OF SERVICE (DS), MILLION RECORDS(M)

ification. Our proposal investigates the performance of LLM models as vanilla and after fine-tuning, which involves adjusting the model's pre-trained parameters without changing the model architecture.

Other approaches utilize LLMs for malware detection, though not specifically targeting edge computing. Most of them rely on BERT (see Section II) and its variants. For example, SecureBERT [26], a cybersecurity-specific model based on RoBERTa, is trained on a corpus of 1.1 billion words from various online sources. It automates malware and intrusion detection by interpreting cybersecurity texts, such as cyber threat intelligence (CTI) documentation, and enhances malware detection training through its domain-specific understanding. MalBERT [27] applies BERT for detecting Android malware via static source code analysis. Its extension, MalBERTv2 [28], refines malware identification using code-aware BERT models, pre-tokenization, and relevance-focused training. [29] combines textual and visual features for Android malware detection, using a BERT-based transfer learning approach and a novel malware-to-image transformation for enhanced analysis. AppPoet [30] leverages GPT-4 with multi-view prompt engineering to improve Android malware detection by incorporating detailed feature descriptions. [31] emphasizes pre-training's role in Windows malware detection through dynamic analysis, showcasing BERT-based models' potential in this area.

[32] presents an approach to malware detection with LLMs by examining the patterns of system calls generated by applications. The authors utilized a 1TB dataset of system calls collected from a Raspberry Pi 3-based spectrum sensor called MalwSpecSys. This dataset encompasses benign and malicious behaviours, providing a rich and diverse data source for the training and validation of the proposed models. The authors evaluate several LLMs, including BERT, DistilBERT, GPT-2, BigBird, Longformer, and Mistral, each characterized by different context sizes that influence their ability to process long sequences of syscalls. The performance of these models was assessed using various classification metrics, including accuracy, precision, recall, F1-score, Cohen's Kappa, and Matthews Correlation Coefficient (MCC). The results indicate a trend where models with larger context sizes (i.e., the amount of input data the model can process at once) generally exhibited superior performance, with BigBird and Longformer achieving the highest scores. However, while larger contexts

can capture more information, they may lead to overfitting and misclassification of certain behaviours, as evidenced by the performance decline of the Mistral model at a context size of 8192 tokens. The authors suggest that employing ensemble methods, which combine predictions from multiple models with varying context sizes, could further enhance detection accuracy by leveraging the strengths of each model. The work demonstrates the potential of LLMs in the context of malware detection through syscall analysis.

[33] presents a comparative analysis of LLMs, specifically BERT and GPT-2, for the detection of zero-day attacks on IoT devices. The adopted methodology involves preprocessing three datasets, ToN-IoT, NSL-KDD, and KDD99, before training and fine-tuning the models. While all the datasets are popular, NSL-KDD and KDD99 are outdated. The models are deployed for real-time anomaly detection, with performance evaluated using accuracy, precision, recall, and F1-score metrics. The analysis results indicate that BERT outperforms GPT-2 across multiple metrics, demonstrating its effectiveness in generalizing to various attack types, but does not provide results for the deployment of such models on the device.

[34] presents a comprehensive analysis of the effectiveness and limitations of LLMs for software automatic vulnerability detection in the context of small and medium-sized enterprises (SMEs) that may not have the resources to engage in professional security assessments. The paper evaluates open-source general-purpose models, such as LLAMA 3.2 8B [19], and commercial models, such as GPT-4o mini [15], noting that many of them are subject to overfitting, leading to unreliable performance metrics for the considered use case. The findings indicate that while LLMs are showing promising results, their precision in recognizing software vulnerabilities is inadequate. Additionally, the paper underscores the necessity for evaluation datasets that are disjoint from training data to measure model performance accurately.

Apart from [33], the above-mentioned proposals focus on software vulnerabilities, while we focus on malware detection on edge devices. Additionally, none of these proposals produces a comprehensive performance and feasibility analysis of deploying an LLM on an edge device, which is one of the key contributions of our paper. Finally, none of these solutions provides a comprehensive analysis considering multiple models and datasets simultaneously, as the one we conduct.

IV. DEPLOYING LARGE LANGUAGE MODELS FOR MALWARE DETECTION ON EDGE DEVICES

The key idea of our proposal is deploying LLMs on an edge devices to analyze network traffic in real-time and perform malware detection. We expect two main benefits from this solution. The first is an improved response time and reduced network latency due to the analysis being performed locally on the edge device without sending data to a remote server. The second is that having the model analyzing the local network brings the capability of rapid adaptation and response to evolving threats.

However, in designing LLM-based malware detection at the edge level, we have taken into account three main requirements that drove the final architecture.

- **Feasibility.** Using LLMs for malware detection requires significant computational and memory demands. Such requirements often exceed the capabilities of the majority of edge devices, typically constrained by limited computational power, storage, and energy resources. As such, the naive solution of deploying the model directly on edge devices cannot be applied. This requires *investigating alternative architectures/solutions, able to trade off the limited resources demands and the model's accuracy.*
- **Accuracy.** Although LLMs excel in natural language understanding and pattern recognition, maintaining consistently high accuracy under resource constraints poses a considerable challenge. A possible solution is to specialize the LLM with tailored training through fine-tuning. Fine-tuning [35] is a transfer learning technique where a pre-trained model is adapted to a specific task by further training it on a smaller, domain-specific dataset. Pre-trained models, such as those derived from large-scale datasets (e.g., BERT, distilBERT), already possess a generalized understanding of patterns and structures within data. Fine-tuning leverages this pre-existing knowledge by focusing on task-specific nuances, enabling the model to achieve better performance with fewer computational resources compared to training from scratch. During fine-tuning, the model weights are adjusted based on the new dataset using supervised learning methods. The process also involves adapting the model's architecture for the specific task. This can include adding a classification head or specialized final layers that transform the model's learned representation into the desired output format (e.g., probability scores for classification tasks). It is also possible to modify the model's architecture more extensively, such as adding pooling layers for text classification or cross-attention mechanisms for question answering. The model is then trained by defining an appropriate loss function (e.g., binary cross-entropy loss for binary classification) and optimizing the parameters using algorithms such as *AdamW* [36].
It is expected that a fine-tuned model obtains improved accuracy without increasing device resource consumption. However, fine-tuning requires significant resource

usage in the additional training phases. Thus, in designing an LLM-based malware detection solution for edge devices, a relevant challenge is to *decide if fine-tuning is needed and, in that case, how to perform it effectively.*

- **Adaptability.** Applying LLMs directly on edge devices brings the benefits of localized threat identification. However, models analyzing a single network flow may lead to poor adaptability to other network flows. This is particularly relevant in a malware detection scenario, where malware patterns and attack behaviors are not uniform across different network environments. This is further exacerbated if fine-tuning is adopted. Indeed, a model that has been fine-tuned on a dataset representing a particular network flow may become very good at detecting threats within that environment but struggle to recognize attacks in other contexts. Moreover, if the dataset used for fine-tuning lacks diversity, the model may overfit to specific traffic patterns or attack signatures rather than learning more generalizable features. In general, overfitting occurs when a model learns to perform exceptionally well on the training data but fails to generalize to unseen data. This happens when the model captures noise, outliers, or overly specific patterns in the training set, mistaking them for general trends. Overfitting typically arises from excessive training, insufficiently diverse datasets, or an overly complex model architecture that matches the training data too closely. The primary consequence of overfitting is poor performance on new data, making the model less reliable in real-world applications. In malware detection tasks, an overfitted model might excel at identifying known malware signatures but struggle to detect novel threats, thus undermining its utility.

The risk of overfitting during the fine-tuning process can make LLMs susceptible to biases and inaccuracies, and this could result in biased predictions or an inability to identify novel threats. It is thus mandatory to *experimentally evaluate whether LLM-based malware detection can adapt to new network flows*, that is, if it is able to transfer knowledge learned from one network flow to another. To mitigate overfitting, techniques such as k-fold cross-validation, regularization, and the use of diverse training datasets can be employed [37], [38], to ensure the model retains its generalizability and robustness across various scenarios.

In our proposed architecture, to address the feasibility requirement, rather than LLM models, we focus on small language models (SLMs) [17].

Selecting the SLM to be adopted is not an easy task since several factors need to be considered (e.g., available resources, desired level of accuracy). To better understand which variables impact the model selection, we experimentally evaluate several SLMs simulating different device settings. In particular, we consider the distilled and tiny models reported in Table I. Additionally, as mentioned in Section II, related literature [39], [40] claims that models from the encoder-only

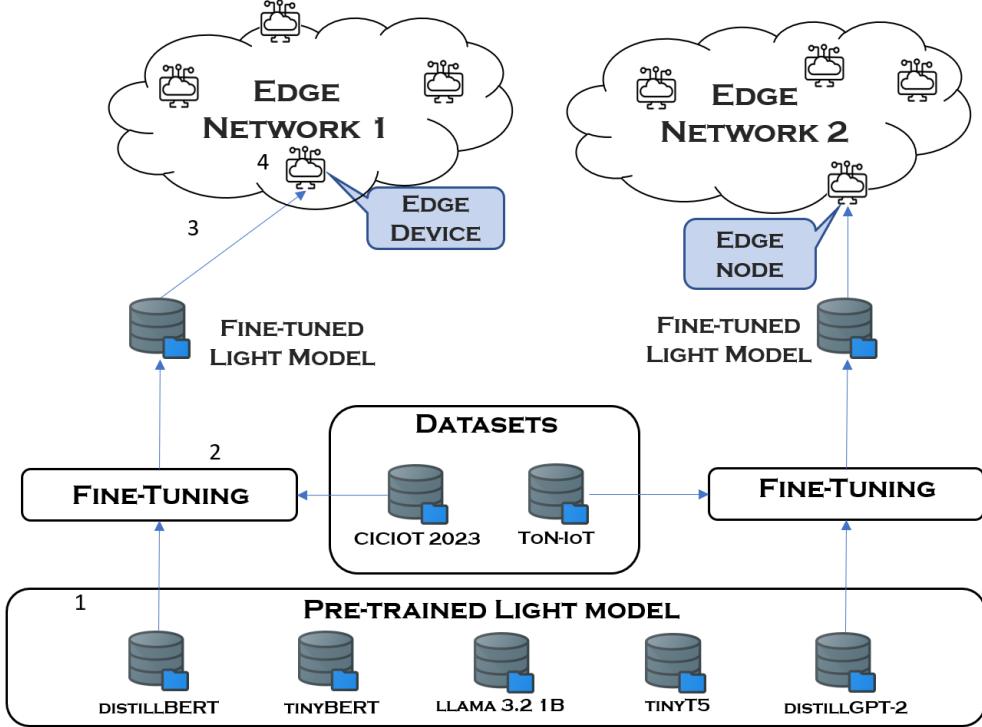


Figure 1. Proposed architecture

category should be preferred over decoder-only and encoder-decoder, as they are specifically tailored for classification tasks. Therefore, we experimentally evaluate if this holds also for malware detection. As such, we select at least one representative from each model category (i.e., encoder-only, decoder-only and encoder-decoder), considering DistilGPT-2, DistilBERT, TinyBERT, LLaMA 3.2 1B, and TinyT5 (see Figure 1 (1)), to evaluate which is the most suitable for malware detection.

However, even if we leverage on distilled/tiny models, we need to carefully evaluate whether these models can be efficiently deployed on edge devices that are very heterogeneous in computational power and memory size. Thus, we conduct extensive experiments to determine if edge devices with different capabilities, varying from simple devices (e.g., CCTV cameras) to edge nodes (e.g., Jetson Nano), can support efficient SLM-based real-time malware detection.

To properly address the accuracy requirement, we first run a set of experiments to verify if fine-tuning is required. In particular, we evaluate the SLM models' accuracy under zero-shot settings, that is, when fine-tuning is not applied to the model, and under the few-shot learning setting, that is, when only a few dataset items are used to train the model. Initial results (see Section V) show that while SLMs can still perform basic tasks (e.g., keyword matching), their accuracy is not optimal for detecting complex malware patterns.

Based on these results, we propose a fine-tuned SML-based malware detection approach, graphically depicted in Figure

1. Adopting fine-tuning requires the selection of a suitable dataset for training. With this respect, we opt to select different training datasets, each one able to tailor the model to specific malware patterns. This allows to have edge devices in the architecture fine-tuned to detect specific types of malware, allowing specialized and targeted threat identification.

Selecting which malware type to associate with each edge device considers the device's context and functionality. For instance, if the edge device manages network traffic, it might be more susceptible to botnet attacks, or if it handles sensitive financial or personal data, it may need enhanced protection against ransomware (see Figure 1 (2)).

On one hand, this specialization is expected to improve the overall system's ability to detect a wide range of malware efficiently and enable faster response times to potential security incidents. On the other hand, this could increase the risk of poor adaptability of the model to network flows that differ from the training data (i.e., **adaptability** requirement).

To evaluate this risk, we rigorously assess the models' ability to adapt to other network flows. In particular, we adopt a *cross-dataset validation* strategy, where we train the models on one dataset (e.g., EdgeIoTset) and test them on a different dataset (e.g., Ton-IoT). Also, to ensure the robustness and generalizability of the fine-tuned models and to minimize the risk of overfitting, we employed the k-fold cross-validation technique. The results of the tests are detailed in Section V.

V. EXPERIMENTAL RESULTS

In this section, we provide the results of our extensive experimental evaluation, with the aim of verifying whether LLMs can be effectively used for malware detection at the edge. In particular, the experiments have been design to evaluate the **accuracy** (see Sections V-B and V-C), **adatability** (see Section V-D), and **feasability** (see Section V-E).

A. Settings

Our experiments were conducted on two different machines: a high-performance system equipped with an AMD EPYC 7742 64-core Processor, 32GB RAM, and an NVIDIA A100-SXM4-80GB GPU, and a laptop featuring an Intel i7-10710U 6-core CPU (4.7 GHz), 16GB DDR4 RAM, and an NVIDIA GTX 1650 Max-Q 4GB GPU. In the following, we do not specify the machine used in each experiment, as it only impacts the training time for fine-tuning, while the other performance indicators remain identical. The only exception is for training large datasets (e.g., CIC IoT 2023), where we had to use the high-performance system, as the laptop lacked sufficient memory to complete the process.

Experiments were conducted starting from multiple pre-trained language models. Each pre-trained model was retrieved for the Pytorch framework from the official repository on Huggingfaces², which can be considered a de facto standard for the open-source code of NLP and machine learning models. The models considered in our experiments include: DistilGPT2,³ DistilBERT,⁴ TinyBERT,⁵ LLama 3.2 1B⁶, and TinyT5.⁷ The models were trained on the four selected datasets, namely X-IIoTID, CIC IoT 2023, Edge-IIoTset, and TON-IoT, described in Section II.

Datasets undergo a preparation phase that involves loading data from CSV files, encoding target labels using Scikit-learn’s LabelEncoder, creating multilabel targets by aggregating encoded label columns, and concatenating feature columns for model input. The resulting dataset was split into training (60%) and test (40%) subsets. Model initialization utilized pre-trained models from the Hugging Face repository, configuring classification heads according to the number of distinct classes (e.g., *GPT2ForSequenceClassification* for distillGPT-2). Models and tokenizers are then loaded onto the target computing devices. Training configuration employed the *AdamW* optimizer with a learning rate of 2×10^{-5} and *BCEWithLogitsLoss* as the loss function, combining sigmoid activation with binary cross-entropy. The training procedure is executed on at least four epochs, with each epoch comprising forward propagation, loss computation, back-propagation, and optimization steps. Validation occurred after each epoch to monitor performance metrics.

²<https://huggingface.co/>

³<https://huggingface.co/distilbert/distilgpt2>

⁴<https://huggingface.co/distilbert/distilbert-base-uncased>

⁵https://huggingface.co/huawei-noah/TinyBERT_General_4L_312D

⁶<https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct>

⁷<https://huggingface.co/t5-efficient-tiny>

Table III reports all the performance metrics used in our experiments. The model’s performance is evaluated using the following metrics: accuracy, precision, recall and F1-score. In their definitions, TP (true positives) denotes data items (i.e., network flow) correctly labeled as positive, that is, with malware; FN (false negatives) denotes the data items incorrectly labeled as negative when they are actually positive; FP (false positives) denotes those labeled as positive when they are actually negative; whereas TN (true negatives) denotes the number of those correctly labeled as negative, i.e., without malware. Moreover, in the table, batch size refers to the number of samples processed simultaneously during training, floating-point accuracy (FPA) indicates the precision level of numerical computations, whereas sequence length (S_lengthS) defines the maximum number of tokens the model can process in a single forward pass. The literature provides formulas that can be exploited to calculate the model performances on the device theoretically [12], [41], [42].

To mitigate the risk of overfitting during model training, we apply k-fold cross-validation. Specifically, the process was conducted with $k = 5$, ensuring a robust evaluation by splitting the dataset into five distinct folds, each serving as a validation set. In contrast, the remaining folds were used for training. The application of this technique demonstrated consistency in the results, as the performance metrics obtained with k-fold cross-validation were identical to those achieved without its application. This outcome suggests that overfitting was effectively prevented, ensuring the model’s ability to generalize well to unseen data. Incorporating this approach further validated the training process, reinforcing the model’s reliability for deployment in real-world scenarios.

Moreover, we applied feature extraction techniques to identify whether relevant attributes within the dataset were present. Those techniques included lasso, recursive feature elimination (RFE), principal component analysis (PCA), random forest and correlation matrix. For instance, when applying the lasso technique to the X-IIoTID dataset, the analysis identified only 6 out of 65 features as significant for the task. Instead, the correlation matrix and Random Forest methods removed 20 features, whereas RFE and PCA retained 25. However, to ensure a comprehensive evaluation, the test sets were run using the full datasets without applying any feature pruning.

To cope with the challenges presented in Section IV, we design and perform three sets of experiments that are described in what follows.

B. Zero-shot classification

In the first set of experiments, we evaluate the model’s performance using a zero-shot classification approach. Zero-shot classification is a scenario where the model is applied to a task without any task-specific fine-tuning. The model is expected to generalize from its pre-trained knowledge to perform the classification task. In our context, the task is to classify whether the data represents benign or malicious behaviour. The outcomes of this evaluation are summarized in

Metric	Formula	Description
Model performance metrics		
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Evaluate the overall correctness of the model by comparing the number of correctly classified instances (both positive and negative) to the total number of instances
Precision	$\frac{TP}{TP + FP}$	Measures the proportion of true positives among all instances predicted as positive (proportion of estimated positives that are actually positive)
Recall	$\frac{TP}{TP + FN}$	Proportion of real positives that are estimated positive, known as True Positive Rate
F1-score	$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$	Is the harmonic mean of precision and recall, balancing both false positives and false negatives
Models performance on device		
Memory for input tensors	Batch size \times S_length \times FPA	Memory consumed by the input data (tokens or sequences) before being processed by the model
Memory for activations	Batch size \times S_length \times Hidden size \times N_layers \times FPA	Memory required for storing intermediate activations (hidden states) during forward and backward passes
Memory for output tensors	Batch size \times S_length \times Vocab. size \times FPA	Memory allocated for the output of the model
Latency (prediction time)	$\frac{\text{Model total FLOPs}}{\text{Hardware FLOPs}}$	Time required for the model to generate predictions on given input data, based on the available computational resources
FLOPS(head)	$4 \times (\text{S_length})^2 \times \text{Hidden size} \times \text{N_heads}$	Computational complexity of the self-attention mechanism in the transformer model
FLOPS(feedforward)	$8 \times \text{S_length} \times (\text{Hidden size})^2$	Computational cost of the feedforward layers, which transform the embeddings within each transformer block
Total FLOPS	$\text{N_layers} \times (\text{FLOPs (head)} + \text{FLOPs (feedforward)})$	Total number of FLOPs required to process a single forward pass through the model

Table III

PERFORMANCE METRICS USED IN THE EVALUATION. TOTAL FLOPS (FLOATING-POINT OPERATIONS) REFER TO THE NUMBER OF ARITHMETIC OPERATIONS INVOLVING FLOATING-POINT NUMBERS THAT THE MODEL PERFORMS. N_HEAD IS THE NUMBER OF ATTENTION HEADS, FPA DENOTES FLOATING-POINT ACCURACY, S_LENGTH IS THE SEQUENCE LENGTH.

Model	Dataset	Accuracy	Precision	Recall	F1-Score
distilGPT2	X-IIoTID	0.4950	0.2676	0.4950	0.3460
	Edge-IIoTset	0.7164	0.5020	0.7164	0.6164
	TON-IoT	0.7189	0.5124	0.7189	0.5985
	CIC IoT 2023	0.7290	0.5230	0.7290	0.6089
distilBERT	X-IIoTID	0.5456	0.2427	0.5135	0.3484
	Edge-IIoTset	0.7193	0.5305	0.7284	0.6025
	TON-IoT	0.7189	0.5124	0.7189	0.5985
	CIC IoT 2023	0.7290	0.5230	0.7290	0.6089
TinyBERT	X-IIoTID	0.5135	0.2637	0.5062	0.3514
	Edge-IIoTset	0.7284	0.5305	0.7284	0.6139
	TON-IoT	0.7230	0.5195	0.7230	0.6048
	CIC IoT 2023	0.7344	0.5289	0.7344	0.6152
Llama 3.2 1B	X-IIoTID	0.5070	0.2626	0.5070	0.3460
	Edge-IIoTset	0.5833	0.5981	0.5833	0.5906
	TON-IoT	0.4522	0.2456	0.7009	0.3607
	CIC IoT 2023	0.4410	0.2378	0.6947	0.3624
TinyT5	X-IIoTID	0.5135	0.2425	0.5135	0.3484
	Edge-IIoTset	0.7164	0.5346	0.7274	0.6223
	TON-IoT	0.7180	0.5158	0.7180	0.6005
	CIC IoT 2023	0.7208	0.5175	0.7208	0.6028

Table IV

RESULTS FOR ZERO-SHOT TESTS WITH BINARY CLASSIFICATION

Table IV, which reports accuracy, precision, recall, and F1-score for each considered model and dataset.

The results in Table IV indicate that the models achieve a maximum accuracy of 72%, leading to suboptimal performance. Among the models, we see how encoder-only models, namely distilBERT and distilGPT2, perform slightly better than the others. This aligns with our prediction and results reported in related literature (e.g., [34]) and demonstrates that considered LLM models are not ready to be used without fine-tuning for the malware detection task.

C. Fine-tuning classification

In the second set of experiments, we fine-tune the models and measure their accuracy under two distinct scenarios.

The first scenario, referred to as the same-dataset approach, involves training the model on a specific dataset (D1) and evaluating its accuracy on the same dataset. We evaluate the accuracy of this approach by exploiting two approaches for the dataset samples, one named few-shot and the other complete. The few-shot approach involves training the model with a small number of labelled examples. In our experiments, we use 30000 lines of the datasets (~ 30 MB) for the few-shot. The key difference is that in the few-shot approach, the model adapts to the task with limited samples, unlike zero-shot, where no task-specific data is provided. The test we run uses a binary classification approach. Instead, for the fine-tuning case, the model undergoes more extensive training using a

Model	Dataset	Epochs	Train. Time	Train.Loss	Train. Accuracy	Test Loss	Test Accuracy	Precision	Recall	F1-score
distilGPT2	X-IIoTID	4	2	0.0730	97.57%	0.0561	98.05%	0.9790	0.9760	0.9775
	Edge-IIoTset	4	2	0.0391	99.12%	0.0257	99.22%	0.9920	0.9900	0.9910
	TON-IoT	4	2	0.0788	98.55%	0.0543	98.57%	0.9855	0.9835	0.9845
	CIC IoT 2023	4	2	0.0866	97.45%	0.0754	97.55%	0.9740	0.9720	0.9737
distilBERT	X-IIoTID	4	2	0.0572	99.20%	0.0477	99.14%	0.9910	0.9912	0.9911
	Edge-IIoTset	4	2	0.0500	99.07%	0.0376	99.21%	0.9912	0.9900	0.9906
	TON-IoT	4	2	0.0688	98.55%	0.0529	98.40%	0.9835	0.9830	0.9836
	CIC IoT 2023	4	2	0.1048	96.73%	0.0925	96.88%	0.9555	0.9850	0.9852
TinyBERT	X-IIoTID	4	0.5	0.3075	86.70%	0.2988	87.30%	0.8450	0.8350	0.8390
	Edge-IIoTset	4	0.5	0.1927	93.16%	0.1823	94.07%	0.9400	0.9300	0.9350
	TON-IoT	4	0.5	0.3086	88.42%	0.2843	88.92%	0.8890	0.8780	0.8815
	CIC IoT 2023	4	0.5	0.1617	96.30%	0.1534	96.30%	0.9615	0.9600	0.9607
Llama 3.2 1B	X-IIoTID	4	3	0.3060	89.16%	0.2917	87.40%	0.9702	0.9689	0.9640
	Edge-IIoTset	4	3	0.1716	94.17%	0.1701	93.88%	0.9913	0.9885	0.9755
	TON-IoT	4	3	0.2988	91.98%	0.2980	90.01%	0.9556	0.9508	0.9462
	CIC IoT 2023	4	3	0.1514	96.01%	0.1412	96.25%	0.9667	0.9675	0.9681
TinyT5	X-IIoTID	50	8.3	0.0635	96.67%	0.0501	97.00%	0.9703	0.9680	0.9692
	Edge-IIoTset	15	2.5	0.1409	97.50%	0.0269	97.12%	0.9757	0.9740	0.9748
	TON-IoT	15	2.5	0.1292	95.85%	0.0554	95.76%	0.9369	0.9350	0.9359
	CIC IoT 2023	100	166	0.1834	90.00%	0.2421	91.25%	0.9125	0.9100	0.9112

Table V
RESULTS FOR THE FEW-SHOT TEST. TRAINING TIME IS MINUTES

significantly larger set of labelled data. In our case, we exploit the full size of the target datasets.

We show the results for the few-shot scenario in Table V, whereas Table VI reports the result for the complete case. The table lists the number of epochs needed to achieve relevant accuracy and the time needed to complete the training phase. At the end of each epoch, the training loss and accuracy are calculated; we list only the result of the last epoch. The test results are calculated after the training phase, including precision, recall, and f1-score. We recall that loss expresses how far the model's predictions are from the actual labels, quantifying the error. A lower loss and higher accuracy indicate better performance.

The fine-tuning results show a significantly increased accuracy moving from the few-shot to the complete approach. The results indicate that all the considered models achieve an accuracy on average of 94% with few shots and 99% for complete. However, we note how TinyBERT achieves the best performance, taking into account the training time and number of epochs. The reason for the worst performance of TinyT5 could be related to its lesser number of heads. Although such a result may indicate overfitting, we can exclude it as the difference between train and test loss is minimal, and the results from the k-fold cross-validation are aligned.

D. Cross-dataset validation

To evaluate the **adatability** of the proposed architecture, we run an experiment named cross-dataset validation, where

the model is fine-tuned on one dataset (D1) but evaluated on a completely different dataset (D2). This enables us to evaluate the model's generalisation ability to unseen data. We show the fine-tuning results in Table VII for both the few-shot and complete approach. The table reports the fine-tune model accuracy and shows that the complete approach performs better than the few-shot. Even if the models mainly achieve satisfactory performance, it is not universally true. Performance remains suboptimal for some combinations, such as CIC IoT 2023 and X-IIoTID. The reasons can be attributed to differences in feature distributions in those datasets. This problem is still an open issue in the literature (see domain shift [43]) and needs further evaluation.

Overall, experiments show the importance of fine-tuning, as the results differences between the zero-shot approach and the fine-tuning are significant for all the models. Also, training the model on a huge dataset is not fully necessary to obtain good results, as we can see from comparing the results between the few-shot and complete approach. Moreover, effective fine-tuning does not necessarily require access to extensive computational resources. Even accessible hardware setups can achieve robust performance when sufficient training iterations are conducted. For instance, the few-shot results were obtained using both of the previously mentioned hardware setups, achieving close results.

Model	Dataset	Epochs	Train Time	Train Loss	Train Accuracy	Test Loss	Test Accuracy	Precision	Recall	F1-score
distilGPT2	X-IIoTID	4	1	0.0005	99.97%	0.0010	98.05%	0.9980	0.9805	0.9891
	Edge-IIoTset	4	4	0.0012	99.97%	0.0002	99.97%	0.9990	0.9997	0.9993
	TON-IoT	4	24	0.0009	99.98%	0.0007	99.98%	0.9991	0.9998	0.9994
	CIC IoT 2023	4	80	0.0008	99.96%	0.0009	99.96%	0.9989	0.9996	0.9992
distilBERT	X-IIoTID	4	1	0.0007	99.95%	0.0012	98.98%	0.9985	0.9898	0.9941
	Edge-IIoTset	4	4	0.0015	99.94%	0.0005	99.95%	0.9988	0.9995	0.9991
	TON-IoT	4	24	0.0011	99.96%	0.0009	99.97%	0.9990	0.9997	0.9993
	CIC IoT 2023	4	80	0.0010	99.95%	0.0011	99.94%	0.9987	0.9994	0.9990
TinyBERT	X-IIoTID	4	0.8	0.0009	99.93%	0.0015	98.85%	0.9982	0.9885	0.9933
	Edge-IIoTset	4	2	0.0018	99.92%	0.0008	99.93%	0.9985	0.9993	0.9989
	TON-IoT	4	10	0.0013	99.94%	0.0011	99.95%	0.9988	0.9995	0.9991
	CIC IoT 2023	4	40	0.0012	99.93%	0.0013	99.92%	0.9985	0.9992	0.9988
Llama 3.2 1B	X-IIoTID	4	1	0.0011	99.91%	0.0018	98.75%	0.9980	0.9875	0.9927
	Edge-IIoTset	4	4	0.0020	99.90%	0.0010	99.91%	0.9983	0.9991	0.9987
	TON-IoT	4	20	0.0015	99.92%	0.0013	99.93%	0.9986	0.9993	0.9989
	CIC IoT 2023	4	60	0.0014	99.91%	0.0015	99.90%	0.9983	0.9990	0.9986
TinyT5	X-IIoTID	25	5	0.1525	98.50%	0.2125	98.75%	0.9275	0.9275	0.9275
	Edge-IIoTset	8	3.2	0.1650	98.75%	0.2250	99.00%	0.9200	0.9200	0.9200
	TON-IoT	8	19	0.1750	98.25%	0.2350	98.50%	0.9150	0.9150	0.9150
	CIC IoT 2023	40	152	0.1834	99.00%	0.2421	99.25%	0.9125	0.9125	0.9125

Table VI
RESULTS FOR THE FINETUNING TEST SET. TRAINING TIME IS HOURS

E. On device performance

The last set of experiments evaluates the **feasibility** of our approach on existing edge devices in terms of resource usage and classification time. For the tests, we select Raspberry Pi 3 and Jetson Nano devices, two widely used devices for resource-constrained environments. The first features a CPU with four cores (Cortex-A53, ARMv8, 64-bit) running at 1.4GHz and 1GB of LPDDR2 SDRAM. In contrast, Jetson Nano has a quad-core ARM Cortex-A57 CPU, a 128-core Maxwell GPU, and 2GB of LPDDR4 RAM. To setup the test environment, we deploy the fine-tuned models on a virtual edge device environment. We exploit Balena,⁸ which offers base images tailored to various edge devices.⁹ The corresponding device images are therefore retrieved from Balena’s repository and deployed into a Docker container. The container is configured to emulate the specifications of the selected edge device, including its memory, processing power, and architecture. Also, the necessary dependencies for model execution are installed, including libraries for managing model inference, such as the Hugging Face Transformers library, and hardware-specific optimizations like NVIDIA TensorRT for the Jetson Nano. The tests include classification time, resource usage (i.e., CPU, GPU, and memory), and model size. To perform the tests, we deploy an application to simulate incoming traffic to the edge device, sending a packet every

second. On the device, we execute another application consisting of two Python threads. One thread is entitled to model classification tasks, whereas the other handles the device’s primary function, such as processing CCTV camera streams or monitoring temperature sensors.

We exploited the formulas retrieved from the literature and listed in Table III to calculate the theoretical performance for the model deployed on the device. Considering the model characteristics listed in Table I and the PyTorch Overhead of 100 MB, we calculate performance for model size, RAM usage, and latency (classification time) as depicted in IX. In the table, model weights refer to the parameters learned during training, stored as tensors, and determine the model’s size. The input tensor represents the data fed into the model. Intermediate activations store temporary results during forward propagation through the model’s layers. The output tensor contains the model’s logits. In contrast, Table VIII, shows the results of our experiments.

Experimental results are aligned with the expected theoretical results. The results show that all models require high CPU usage ($\sim 70\%$) on both devices. LLama results are the most demanding, as expected, due to the model’s complexity. The size and RAM usage remain the same for each device. As expected, tiny models require fewer resources for size and RAM usage than the distilled version, with TinyT5 as the smallest. Similar behaviour is shown for the classification time, with TinyT5 being the fastest. The models perform

⁸<https://www.balena.io/>

⁹<https://docs.balena.io/reference/base-images/base-images-ref/>

Model	Dataset	Few-shot				Complete			
		CICIoT2023	TON-IoT	Edge-IIoTset	X-IIoTID	CICIoT2023	TON-IoT	Edge-IIoTset	X-IIoTID
distilGPT2	X-IIoTID	15.56%	48.54%	96.39%	—	16.28%	49.76%	97.10%	—
	Edge-IIoTset	15.56%	71.35%	—	68.44%	16.68%	72.42%	—	69.88%
	TON-IoT	15.44%	—	61.97%	68.44%	17.10%	—	62.34%	69.78%
	CIC IoT 2023	—	52.05%	93.44%	68.44%	—	53.78%	94.53%	70.14%
distilBERT	X-IIoTID	15.56%	57.31%	96.39%	—	16.88%	58.35%	97.20%	—
	Edge-IIoTset	15.30%	24.56%	—	19.63%	16.82%	25.78%	—	20.91%
	TON-IoT	15.56%	—	96.39%	68.44%	17.14%	—	97.12%	70.35%
	CIC IoT 2023	—	55.56%	96.39%	68.44%	—	56.87%	97.12%	69.25%
TinyBERT	X-IIoTID	15.56%	55.56%	96.39%	—	16.84%	56.88%	97.20%	—
	Edge-IIoTset	15.56%	55.56%	—	68.44%	17.25%	57.20%	—	70.28%
	TON-IoT	57.78%	—	14.43%	68.44%	59.18%	—	15.94%	69.78%
	CIC IoT 2023	—	19.88%	12.30%	24.25%	—	21.26%	14.16%	25.68%
Llama 3.2 1B	X-IIoTID	15.84%	50.48%	96.44%	—	17.12%	51.68%	96.96%	—
	Edge-IIoTset	15.28%	52.18%	—	65.32%	16.72%	54.24%	—	66.45%
	TON-IoT	15.56%	—	68.46%	68.28%	16.84%	—	69.78%	69.94%
	CIC IoT 2023	—	53.72%	94.48%	67.99%	—	54.88%	95.69%	68.74%
TinyT5	X-IIoTID	84.44%	19.88%	12.30%	—	85.80%	21.24%	13.98%	—
	Edge-IIoTset	84.44%	19.88%	—	24.25%	85.58%	21.20%	—	25.78%
	TON-IoT	10.01%	—	10.01%	28.57%	11.94%	—	11.85%	30.14%
	CIC IoT 2023	—	55.56%	41.31%	68.44%	—	56.58%	43.16%	70.24%

Table VII
FINE-TUNING RESULTS FOR CROSS-VALIDATION WITH FEW-SHOT AND COMPLETE APPROACH.

better on the Jetson Nano rather than the Raspberry for both classification time (-23 sec) and CPU usage (-60%), thanks to GPU support and better hardware. Overall, TinyBERT and TinyT5 show the best performance, making them the best choice for the edge scenario. The Jetson Nano's superior performance makes it a better choice for edge nodes, where models that require higher resources can be deployed. For example, in the case of Llama, which requires at least 5 GB of RAM, the Jetson Nano would be the ideal device thanks to its 8 GB RAM.

VI. CONCLUSIONS

In recent years, malware's growing sophistication and adaptability have highlighted the urgent need for innovative detection methods, particularly in resource-constrained edge computing environments. Traditional approaches have proven insufficient in addressing the challenges posed by modern malware, necessitating the exploration of advanced techniques. Lightweight Large Language Models (LLMs) have emerged as promising candidates for enhancing malware detection in edge scenarios.

In this paper, we tackled the issue of deploying LLMs in edge environments, investigating through a comprehensive experimental evaluation and careful design choices several challenges, such as maintaining high accuracy, mitigating over-

fitting, and addressing the computational constraints of edge devices. To address inherent limitations of LLMs applied in an edge scenario, we proposed an SML-based malware detection approach, where different edge devices in the architecture are fine-tuned to detect specific types of malware, allowing specialized and targeted threat identification. By considering several available datasets, we evaluated the effectiveness of this architecture by considering the most popular lightweight LLMs, and highlighted their strengths in detecting malware while adapting to diverse scenarios.

Future research will focus on refining the design of our architecture, exploring advanced regularization techniques to mitigate biases, and optimizing deployment strategies for real-world edge environments. We also plan to complement our architecture with a central model to improve the detection performance of new threats by specialized edge devices along the lines presented in [21]. The central model will enable the integration of feedback from diverse edge devices to improve overall threat detection accuracy and maintain consistency in response strategies across the network.

ACKNOWLEDGMENT

This research work has been partially supported by Cisco Research under the research grant award: "Malware Detection for Edge-based Computing" (Id 88749009).

Performance Results for Jetson Nano and Raspberry Pi 3									
Model	Dataset	Size (MB)	RAM (MB)	CPU Usage	Class. Time CPU (ms)	Class. Time GPU (ms)	Class. Time (sec)	CPU Usage	
distilGPT2	X-IIoTID	320	750	~10%	~750	~150	~24	~70%	
	Edge-IIoTset	320	750	~10%	~750	~150	~24	~70%	
	TON-IoT	320	750	~10%	~750	~150	~24	~70%	
	CIC IoT 2023	320	750	~10%	~750	~150	~24	~70%	
distilBERT	X-IIoTID	260	600	~10%	~750	~150	~24	~70%	
	Edge-IIoTset	260	600	~10%	~750	~150	~24	~70%	
	TON-IoT	260	600	~10%	~750	~150	~24	~70%	
	CIC IoT 2023	260	600	~10%	~750	~150	~24	~70%	
TinyBERT	X-IIoTID	55	380	~10%	~150	~30	~5	~70%	
	Edge-IIoTset	55	380	~10%	~150	~30	~5	~70%	
	TON-IoT	55	380	~10%	~150	~30	~5	~70%	
	CIC IoT 2023	55	380	~10%	~150	~30	~5	~70%	
Llama 3.2 1B	X-IIoTID	4600	5500	~10%	~15000	~3000	~400	~70%	
	Edge-IIoTset	4600	5500	~10%	~15000	~3000	~400	~70%	
	TON-IoT	4600	5500	~10%	~15000	~3000	~400	~70%	
	CIC IoT 2023	4600	5500	~10%	~15000	~3000	~400	~70%	
TinyT5	X-IIoTID	60	400	~10%	~60	~12	~2	~70%	
	Edge-IIoTset	60	400	~10%	~60	~12	~2	~70%	
	TON-IoT	60	400	~10%	~60	~12	~2	~70%	
	CIC IoT 2023	60	400	~10%	~60	~12	~2	~70%	

Table VIII
PERFORMANCE RESULTS FOR JETSON NANO AND RASP PI 3

Model	Model Total FLOPs (GFLOPs)	Model weights (MB)	Input Tensors (Billion)	Intermediate Activations (MB)	Output Tensors (MB)	RAM usage (MB)	Latency (Classification Time)		
							Raspb Pi 3	Jetson CPU	Jetson GPU
distilGPT2	7.25	327.65	4096.00	18.87	205.85	750	24.16s	724.78ms	144.96ms
distilBERT	7.25	265.45	4096.00	18.87	125.02	600	24.16s	724.78ms	144.96ms
TinyBERT	1.38	57.40	4096.00	5.11	125.02	380	4.60s	138.02ms	27.60ms
Llama-3.2-1B	137.44	4943.26	4096.00	134.22	525.34	5500	458.13s	13.74s	2.75s
TinyT5	0.54	62.28	4096.00	4.19	131.60	400	1.79s	53.69ms	10.74ms

Table IX
THEORETICAL PERFORMANCE OF THE SLM MODELS DEPLOYED ON THE DEVICE

REFERENCES

- [1] O. Friha, M. A. Ferrag, B. Kantarci, B. Cakmak, A. Ozgun, and N. Ghoulami-Zine, “Llm-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness,” *IEEE Open Journal of the Communications Society*, 2024.
- [2] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.
- [3] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, “Tinybert: Distilling bert for natural language understanding,” *arXiv preprint arXiv:1909.10351*, 2019.
- [4] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, “Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning,” *IEEE Access*, vol. 10, pp. 40 281–40 306, 2022.
- [5] M. Al-Hawawreh, E. Sitnikova, and N. Abutorab, “X-iiotid: A connectivity-agnostic and device-agnostic intrusion data set for industrial internet of things,” *IEEE Internet of Things Journal*, 2022.
- [6] N. Moustafa, “A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets,” *Sustainable Cities and Society*, vol. 72, 2021.
- [7] E. C. P. Neto, S. Dadkhah, R. Ferreira, A. Zohourian, R. Lu, and A. A. Ghorbani, “CICIoT2023: A real-time dataset and benchmark for large-scale attacks in iot environment,” *Sensor*, 2023.
- [8] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, “A survey of large language models,” *arXiv preprint arXiv:2303.18223*, 2023.
- [9] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, and R. L. Mercer, “Class-based n-gram models of natural language,” *Computational linguistics*, 1992.
- [10] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký, “Strategies for training large scale neural network language models,” in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, 2011.
- [11] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *Advances in neural information processing systems*, 2014.
- [12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez,

- L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [13] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering*, 2024.
- [14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [15] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," *Hayate-lab*, 2018.
- [16] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, 2020.
- [17] X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao, and T. Zhou, "A survey on knowledge distillation of large language models," *arXiv preprint arXiv:2402.13116*, 2024.
- [18] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.
- [19] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.
- [20] Y. Tay, M. Dehghani, J. Rao, W. Fedus, S. Abnar, H. W. Chung, S. Narang, D. Yogatama, A. Vaswani, and D. Metzler, "Scale efficiently: Insights from pre-training and fine-tuning transformers," *arXiv preprint arXiv:2109.10686*, 2021.
- [21] C. Rondonini, B. Carminati, E. Ferrari, A. Kundu, and A. Jajoo, "Large language models to enhance malware detection in edge computing," in *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-JSA)*. IEEE, 2024, pp. 1–10.
- [22] M. A. Ferrag, M. Ndhlovu, N. Tihanyi, L. C. Cordeiro, M. Debbah, T. Lestable, and N. S. Thandi, "Revolutionizing cyber threat detection with large language models: A privacy-preserving bert-based lightweight model for iot/iiot devices," *IEEE Access*, 2024.
- [23] M. Zhang, J. Cao, X. Shen, and Z. Cui, "Edgeshard: Efficient llm inference via collaborative edge computing," *arXiv preprint arXiv:2405.14371*, 2024.
- [24] Y. Chen, R. Li, X. Yu, Z. Zhao, and H. Zhang, "Adaptive layer splitting for wireless llm inference in edge computing: A model-based reinforcement learning approach," *arXiv preprint arXiv:2406.02616*, 2024.
- [25] A. Ravi, V. Chaturvedi, and M. Shafique, "Vit4mal: Lightweight vision transformer for malware detection on edge devices," *ACM Transactions on Embedded Computing Systems*, 2023.
- [26] E. Aghaei, X. Niu, W. Shadid, and E. Al-Shaer, "Securebert: A domain-specific language model for cybersecurity," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2022.
- [27] A. Rahali and M. A. Akhloufi, "Malbert: Malware detection using bidirectional encoder representations from transformers," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021.
- [28] Rahali, Abir and Akhloufi, Moulay A, "Malbertv2: Code aware bert-based model for malware identification," *Big Data and Cognitive Computing*, 2023.
- [29] F. Ullah, A. Alsirhani, M. M. Alshahrani, A. Alomari, H. Naeem, and S. A. Shah, "Explainable malware detection system using transformers-based transfer learning and multi-model visual representation," *Sensors*, 2022.
- [30] W. Zhao, J. Wu, and Z. Meng, "Appoet: Large language model based android malware detection via multi-view prompt engineering," *arXiv preprint arXiv:2404.18816*, 2024.
- [31] Z. Xu, X. Fang, and G. Yang, "Malbert: A novel pre-training method for malware detection," *Computers & Security*, 2021.
- [32] P. M. S. Sánchez, A. H. Celadrán, G. Bovet, and G. M. Pérez, "Transfer learning in pre-trained large language models for malware detection based on system calls," *arXiv preprint arXiv:2405.09318*, 2024.
- [33] M. Omar, H. M. Zangana, J. N. Al-Karaki, and D. Mohammed, "Harnessing llms for iot malware detection: A comparative analysis of bert and gpt-2," in *2024 8th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. IEEE, 2024, pp. 1–6.
- [34] F. Panebianco, A. Isgrò, S. Longari, S. Zanero, and M. Carminati, "Guessing as a service: Large language models are not yet ready for vulnerability detection," in *Proceedings of ITASEC25 (Joint National Conference on Cybersecurity (ITASEC & SERICS))*, 2025.
- [35] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.
- [36] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.
- [37] C. F. G. D. Santos and J. P. Papa, "Avoiding overfitting: A survey on regularization methods for convolutional neural networks," *ACM Computing Surveys (CSUR)*, vol. 54, no. 10s, pp. 1–25, 2022.
- [38] B. Ghoghoj and M. Crowley, "The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial," *arXiv preprint arXiv:1905.12787*, 2019.
- [39] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 6, pp. 1–32, 2024.
- [40] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, 2024.
- [41] T. Wolf, "Transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2020.
- [42] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Comput. Surv.*, vol. 55, no. 6, Dec. 2022. [Online]. Available: <https://doi.org/10.1145/3530811>
- [43] W. M. Kouw and M. Loog, "An introduction to domain adaptation and transfer learning," *arXiv preprint arXiv:1812.11806*, 2018.