Journal of Science and Technology
ISSN: 2362-1443 (Print) 3059-9490 (Online) 2026, Vol. 5, No. 1

Interactive Malware Analysis using RoBERTa based Model

Utkarsha Shukla
Institute of Engineering, Thapathali Campus
Tribhuvan University, Nepal
utkarsha.079msise18@tcioe.edu.np

Om Prakash Mahato*
Institute of Engineering, Thapathali Campus
TribhuvanUniversity,Nepal
om.mahato@tcioe.edu.np

*Corresponding author

***Abstract*—The rapid surge of malware presents significant threats to device security and user privacy, with traditional detection methods often struggling to keep pace with its evolving sophistication and variety. The escalating complexity and diversification of malware create a pressing need for advanced detection techniques that can effectively identify and mitigate these emerging cyber threats. This study addresses these challenges by employing advanced transformer-encoder models to create rich contextual representations from diverse malware datasets, aiming to significantly enhance the accuracy and performance of malware detection systems. To this end, relevant malware datasets are systematically collected, converted into text format and contextualized according to various malware labels to ensure a comprehensive analysis. The preprocessed data is then analyzed using the SecureBERT model, which is built upon the robust RoBERTa architecture, facilitates precise classification of malware types. The model's effectiveness is validated through interactive prompts, successfully categorizing input text messages into corresponding malware categories. Experimental results demonstrate a significant improvement in classification performance, highlighting the model's capability to analyze real-time texts related to malware effectively. This innovative approach offers substantial potential for enhancing cybersecurity measures and represents a breakthrough in the field of malware detection and analysis, paving the way for more effective strategies against emerging cyber threats.**

***Index Terms*—cybersecurity , malware, SecureBERT, transformer**

I. Introduction

Malware poses evolving threats, causing financial loss, data breaches, and information compromise. Techniques like polymorphism and obfuscation evade signature-based detection, which fails against zero-day variants. Behavioral analysis helps but often yields high false positives. Transformer-based models offer an adaptive solution, effectively detecting modern malware while reducing misclassification [1]. BERT (Bidirectional Encoder Representations from Transformers) is a powerful tool for malware detection due to its ability to capture complex relationships in data through bidirectional processing, which is crucial for identifying sophisticated malware behaviors in API calls or manifest files. It reduces the need for extensive feature engineering, learning directly from raw data and adapting to new malware variants. BERT has demonstrated high accuracy, achieving up to 98% in malware detection and 94% in classification by recognizing intricate patterns. Its versatility across different data types—such as API sequences, manifest files, and even visual representations of malware—combined with its fine-tuning capability, allows it to adapt to specific tasks and improve detection rates across various malware families. This paper presents a novel approach to malware classification by applying the SecureBERT transformer model, a RoBERTa-based architecture, which enhances threat detection by utilizing its deep contextual understanding of malware-related text for improved accuracy [2]. Unlike traditional signature-based methods, which struggle to detect new or evolving malware, SecureBERT analyzes large-scale cybersecurity texts to identify subtle patterns of malicious behavior. Its self-attention mechanism captures complex relationships, improving detection of sophisticated malware variants [3]. Fine-tuned for diverse malware categories, the model ensures robust performance against previously unseen threats. This work enhances real-world malware detection by integrating SecureBERT into cybersecurity frameworks, offering a proactive, transformer-based approach with improved adaptability, scalability, and precision.

DOI : https://doi.org/10.3126/jost.v5i1.XXXXX

Article History: Received: 13 Nov. 2025, Revised: 12 Jan. 2026, Accepted: 28 Feb. 2026.

II. LITERATURE REVIEW

Various static analysis techniques have been explored for malware detection. Antonellis et al. [4] and Zhang and Zhang [5] examined text representations like TDM, BoW, and TF-IDF, effective but dependent on training data quality. Huda et al. utilized static API calls, while Fan et al. leveraged instruction sequences [6]. Limitations of static analysis are addressed by dynamic techniques, analyzing malware behavior in execution environments. Memory analysis, highlighted by Monnappa [7], uncovers hidden activities, with certain API calls detected exclusively in memory [8]. Swati Nautiyal et al. [9] applied ANNs with Genetic Algorithms for cloud security to mitigate EDoS attacks. API call sequence and memory dump analyses proved resilient against obfuscation [10], [11]. Rahali and Akhloufi introduced MalBERTv2, a BERT-based malware detection model achieving accuracies of 0.8224–0.9376 [12], though computationally demanding. Mahalakshmi's MAD-NET using DBN reached 99.83% accuracy on CICAndMal2017 [13], and MADRAS-NET with LinkNet achieved 99.59% accuracy and 0.997 AUC, but struggles with generalization [14]. A transformer-based DPI algorithm outperformed CNN and LSTM with 79.57% accuracy on UNSW-NB15 and 79.07% on CIC-IOT23 [1]. Maniriho et al. proposed MeMalDet, a memory analysis framework using deep autoencoders and ensembles, achieving 99.78% accuracy with high precision and recall [15]. While effective against obfuscated malware, it is resource-intensive. Future work should focus on improving generalization, interpretability, and real-time adaptability. Despite advances, static analysis (e.g., TDM, BoW, TF-IDF, API calls) is prone to obfuscation and data quality issues, while dynamic analysis (e.g., memory forensics, API sequences) incurs high overhead and poor scalability. Deep learning models like MalBERTv2, MAD-NET, MADRAS-NET, and transformer-based DPI achieve strong accuracy but struggle with generalization, adaptability, and resource demands. This highlights the need for lightweight, interpretable, and adaptive frameworks. To address this, we use a RoBERTa-based model for malware classification.

III. METHODOLOGY

A. Architectures

BERT is an open-source machine learning framework for natural language processing (NLP) tasks. It enables computers to understand ambiguous text by analyzing the context of words relative to each other. Pretrained on extensive text from Wikipedia, BERT can be fine-tuned for various tasks, including question answering. The model is based on transformers, which dynamically calculate the relationships between inputs and outputs [16]. Figure 1 describes BERT architecture.

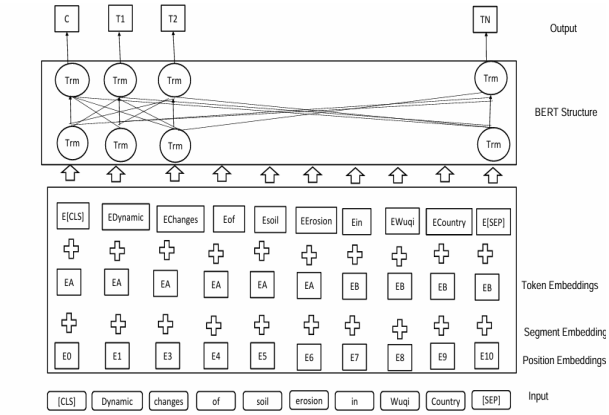

Fig. 1. Overall Structure of BERT Model

RoBERTa (Robustly Optimized BERT Pretraining Approach), developed by Facebook AI in 2019, is a BERT variant that focuses solely on Masked Language Modeling, removing the Next Sentence Prediction task. It uses dynamic masking, trains on a larger 160GB dataset with longer durations and bigger batch sizes, and applies hyperparameter optimizations to improve performance. [17].Figure 2 describes RoBERTa architecture.

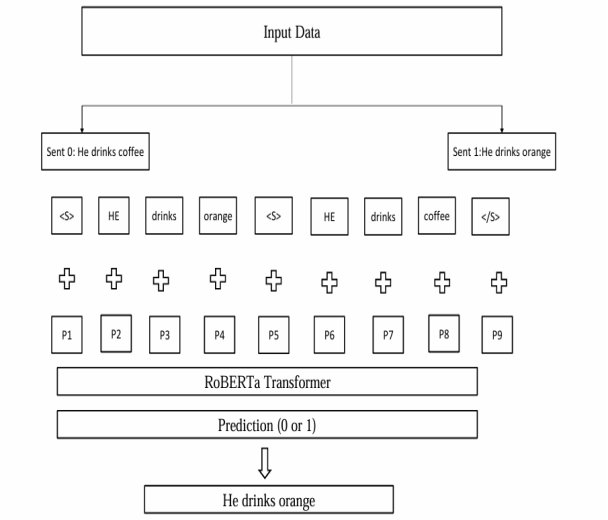

Fig. 2. RoBERTa Model [17]

SecureBERT, a security-focused variant of BERT based on RoBERTa-base, is fine-tuned for Android

malware analysis. It leverages bidirectional transformers to capture contextual patterns indicative of malicious activity. Using a custom tokenizer of 50,265 cybersecurity-specific tokens and trained on 98,411 security texts ($\sim$ 1 billion tokens) with Masked Language Modeling, SecureBERT has 123 million parameters and employs noise injection during training. It outperforms RoBERTa and SciBERT in threat detection and risk assessment [18]. Figure 3 illustrates its architecture.

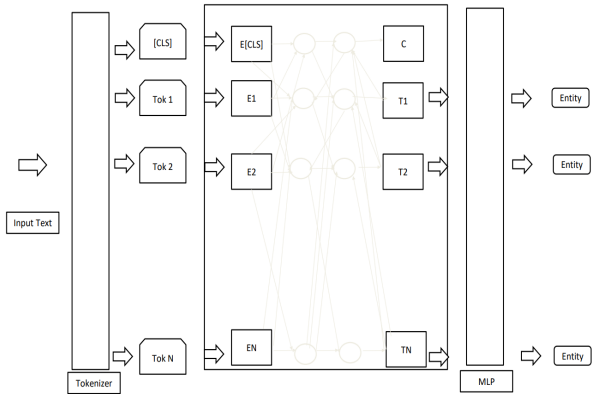

Fig. 3. SecureBERT Architecture [19]

SecureBERT excels in malware-related text classification by effectively understanding word context and providing robust performance on noisy, imbalanced data. Its fine-tuning capabilities make it adaptable for various cybersecurity tasks, while BERT's state-of-the-art performance ensures high accuracy in NLP tasks, including text classification.

B. Mathematical Modelling

1) CBOW (Continuous Bag of Words): In the CBOW model, the target word is predicted from surrounding context words within a given window. The input layer encodes the context, and the hidden layer represents the target word as a continuous vector in the embedding space, as formalized in equation 1. The output layer predicts the target word [20].

$$\text{Word2Vec}(w_i) = \sum_{\substack{j=1 \\ j \neq i}}^{n} P(w_j | w_i) \tag{1}$$

Here, w_i denotes the input word, and $P(w_j \mid w_i)$ denotes the conditional probability of word w_j occurring given word w_i.

2) Mathematical Foundation of SecureBERT: SecureBERT, a RoBERTa variant adapted for cybersecurity tasks, relies on key mathematical components. The self-attention mechanism computes attention scores as described in equation 2:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2}$$

where Q, K, and V are the query, key, and value matrices, and d_k is the key dimension. Input tokens t_i are embedded into dense vectors $\mathbf{e}_i$ via the embedding matrix $\mathbf{E}$:

$$\mathbf{E} = \text{Embedding}(T). \tag{3}$$

Multiple transformer encoder layers iteratively transform embeddings into contextualized representations in equation 4:

$$\mathbf{H}^{(l)} = \text{Transformer}^{(l)}(\mathbf{H}^{(l-1)}), \quad l = 1, \ldots, L, \tag{4}$$

with $\mathbf{H}^{(0)} = \mathbf{E}$ and final output $\mathbf{C} = \mathbf{H}^{(L)}$. For classification, the [CLS] token embedding $\mathbf{z}$ is extracted in equation 5:

$$\mathbf{z} = \text{CLS}(\mathbf{C}), \tag{5}$$

and logits are computed via a linear layer in equation 6:

$$\mathbf{o} = \mathbf{W}\mathbf{z} + \mathbf{b}, \tag{6}$$

where $\mathbf{W}$ and $\mathbf{b}$ are classifier weights and bias. This architecture enables SecureBERT to capture sequence-level context for accurate predictions.

Token Influence Analysis:

An input token sequence $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, is given where the model produces logits $\mathbf{y} = f(\mathbf{x})$. To measure the influence of token x_i, it is replaced with the `[MASK]` token to create a masked sequence $\mathbf{x}^{(i)}$, yielding the output $\mathbf{y}^{(i)} = f(\mathbf{x}^{(i)})$. The influence of x_i is computed as the L_1-norm difference between the original and masked outputs:

$$\text{Influence}(x_i) = \|\mathbf{y} - \mathbf{y}^{(i)}\|_1 = \sum_j |y_j - y_j^{(i)}|. \tag{7}$$

All token–influence pairs are collected as $\mathcal{S} = \{(x_i, \text{Influence}(x_i))\}_{i=1}^{n}$ and ranked in descending order of influence:

$$\text{RankedTokens} = \text{Sort}(\mathcal{S}, \text{key} = \text{Influence}, \text{descending}). \tag{8}$$

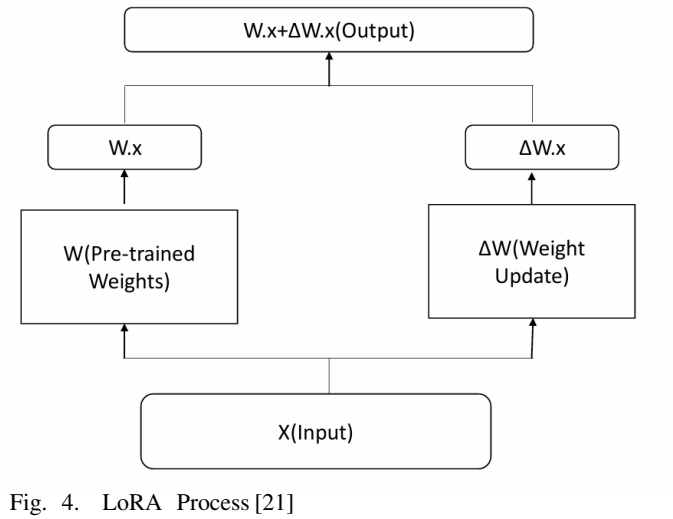

Fig. 4. LoRA Process [21]

3) LoRA (Low-Rank Adaptation): Figure 4 illustrates LoRA, an efficient fine-tuning technique that introduces a small number of trainable parameters into large pretrained models while keeping most weights frozen. For a pre-trained weight matrix $W \in \mathbb{R}^{d\times k}$, standard fine-tuning updates all $d \times k$ parameters, whereas LoRA decomposes as shown in equation 9

$$W = W_0 + \Delta W, \tag{9}$$

where W_0 is fixed and ΔW is a low-rank update in equation 10:

$$\Delta W = AB, \tag{10}$$

with $A \in \mathbb{R}^{d\times r}$, $B \in \mathbb{R}^{r\times k}$, and $r \ll \min(d, k)$ [22]. This reduces trainable parameters from $d \times k$ to $r(d+k)$. Training optimizes only A and B in equation 11

$$\mathcal{L} = \mathcal{L}_{\text{task}}(X, Y; W_0 + AB) + \lambda(\|A\|_F^2 + \|B\|_F^2), \tag{11}$$

where $\mathcal{L}_{\text{task}}$ is the task-specific loss, $\|\cdot\|_F$ the Frobenius norm, and λ a regularization parameter [22]. Thus, LoRA achieves memory-efficient training by adding only in equation 12

$$\text{Additional Parameters} = r(d + k), \tag{12}$$

keeping W_0 unchanged during optimization.

Figure 5 illustrates the system block diagram. The process begins with data preprocessing, where textual Android malware datasets are cleaned, normalized, numerically labeled, and split into training, validation, and test sets, creating structured data compatible with PyTorch. During training, the pretrained SecureBERT model is fine-tuned using techniques like LoRA, optimizing hyperparameters such as learning rate, batch size, and epochs. For evaluation and inference, the model predicts unseen data, assessed via accuracy, precision, recall, F1-score, and visualizations like confusion matrices, ROC-AUC, and PR curves. Input text is tokenized, and the model outputs a classification label.

C. System Block Diagram

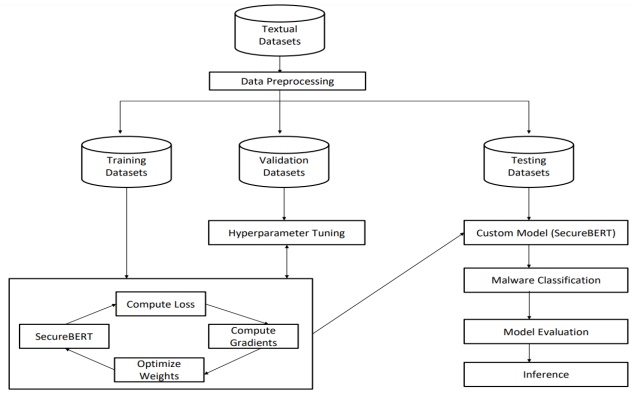

Fig. 5. Block Diagram for Android Malware Analysis using SecureBERT

D. Instrumentation Requirements

The paper uses a device with 16 GB DDR4 RAM, a 12th Gen Intel® Core™ i5-1235U processor (1.30 GHz), and Intel® Iris Xe Graphics for training and testing machine learning models. Python, along with TensorFlow, PyTorch, Scikit-learn, and Pandas, handles model development and data preprocessing. Hugging Face's Transformers library is used for fine-tuning SecureBERT. Cloud environments like Google Colab and Kaggle provide additional CPU/GPU resources, supporting complex model training beyond local hardware limits, ensuring efficient malware analysis with transformer models.

E. Dataset Explanation

This paper supports multiple benchmark datasets for Android malware and benign apps, ensuring diverse and representative coverage. **AndroZoo** provides APKs with SHA-256, permissions, and labels [23]; **Drebin** includes extracted Android app features [24]; **CICMalDroid2017** offers .pcap files and 80+ network features across 42 families in four categories [25]; **Tuandromd** contains 4,465 instances with 241 attributes labeled as

malware or goodware [26]; **MalwareTextDB** supplies NLP-based data on malware types (viruses, worms, trojans, ransomware, APTs) with metadata [27]; **Ransomware** and **Trojan** datasets emphasize malicious vs. benign network traffic [28], [29]; and **APT Notes** compiles documents on Advanced Persistent Threats [30].

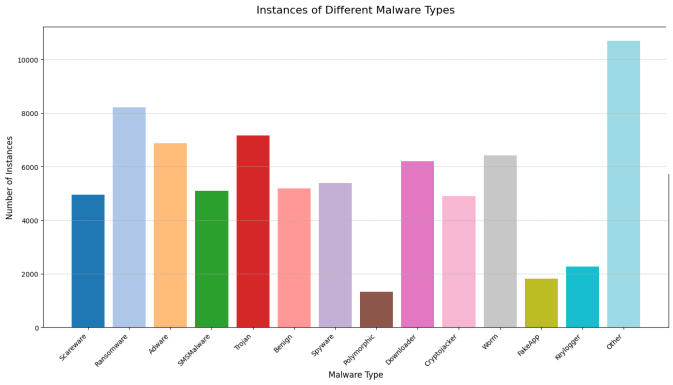

Fig. 6. Bar graph representation of different malware types

IV. Result and Discussion

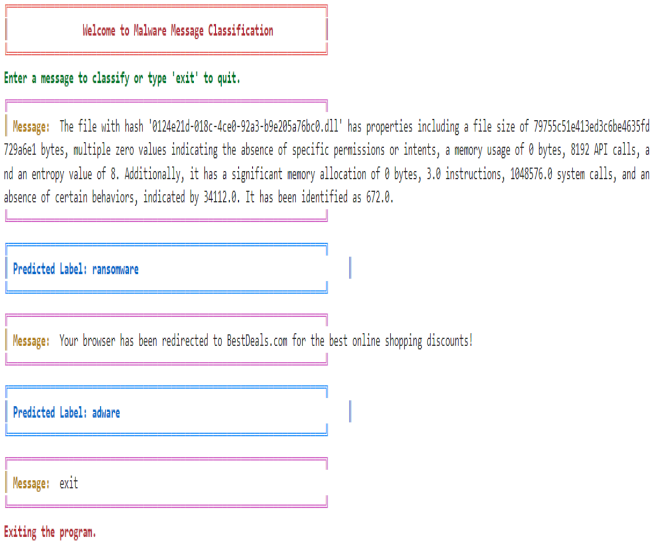

Fig. 7. Interactive Classification

The SecureBERT model in Figure 7 is fine-tuned for cybersecurity tasks and uses RoBERTaTokenizer to convert text into numerical tokens. Malware descriptions are tokenized into subwords, mapped to token IDs, and padded with special tokens ([CLS], [SEP]) for consistent length. SecureBERT's self-attention layers capture contextual relationships, producing a probability distribution to predict the most likely Android malware type.

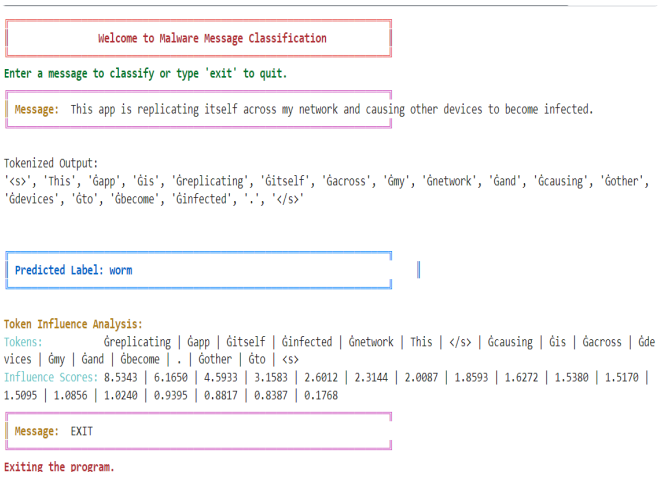

Fig. 8. Message Tokenization and classification

The diagram illustrates the output of a malware message classification system, where textual descriptions of files are analyzed and classified as "worm." Each input message contains file attributes, including size, memory usage, and system calls. The model first tokenizes the input, segmenting the text into smaller units, or "tokens," that can be processed by the model. In this example, the tokenizer produces subword units, such as "GApp" and "Greplicating," along with special tokens, e.g., "¡s¿". Influence scores are computed as the difference between the original logits and the logits obtained after masking a token, indicating the contribution of each token to the model's prediction. By ranking tokens according to their influence scores, one can identify the tokens most critical in shaping the classification outcome, highlighting those that substantially affect the model's decision, whether it results in a correct or incorrect prediction.

TABLE I
Hyperparameters for Model Tuning

Parameter	Values
Learning Rate	1e-4, 1e-5, 2e-5, 3e-4, 3e-5
Batch Size	16,32,64,128
Epochs	3,4,5,10,15

Hyperparameter tuning optimizes model performance by balancing accuracy and generalization. Learning rates ($1e$-4 to $3e$-5), batch sizes (16–128), and epochs (3–15) were tested, with early stopping to prevent over/underfitting (Table I). Expanding the test set from 4,000 to 13,000 instances and adjusting epochs improved evaluation while controlling overfitting.

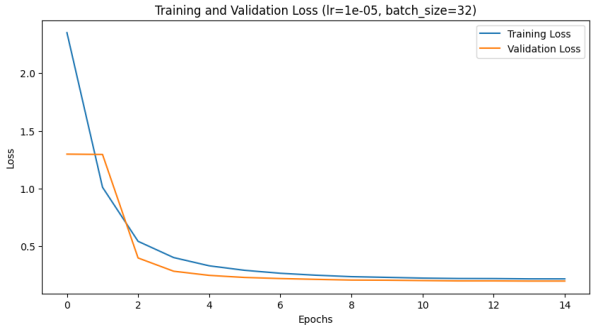

Fig. 9. Line Plot describing training/validation plot

Figure 9 shows training and validation loss curves. Training loss drops rapidly after the first epoch, and by the second epoch, the model begins to converge, demonstrating effective learning and generalization. Losses flatten after six epochs, indicating minimal gains from further training. The convergence of validation loss toward training loss confirms efficient learning and good generalization.

TABLE II
CLASSIFICATION REPORT FOR MALWARE TYPES

Malware Type	Precision	Recall	F1-Score	Support
Scareware	0.91	0.95	0.93	1000
Ransomware	0.95	0.96	0.95	1635
Adware	0.91	0.96	0.93	1365
SMS Malware	0.94	0.94	0.94	1047
Trojan	0.97	0.95	0.96	1377
Benign	0.97	0.94	0.96	1062
Spyware	0.97	0.94	0.95	1077
Polymorphic	0.98	0.78	0.87	247
Downloader	1.00	0.95	0.97	1260
Cryptojacker	1.00	1.00	1.00	1269
Worm	1.00	1.00	1.00	356
Fake App	1.00	1.00	1.00	463
Keylogger	1.00	1.00	1.00	463
Accuracy			0.96	13158
Macro Avg	0.96	0.95	0.96	13158
Weighted Avg	0.96	0.96	0.96	13158

Table II presents precision, recall, and F1-scores for all 13 classes. Cryptojacker, Worm, Fakeapp, and Keylogger achieve perfect scores of 1.0, while Polymorphic shows lower recall (0.78) and F1-score (0.87), indicating some classification difficulty. Overall, the model performs strongly, with weighted averages of 0.96 for precision, recall, and F1-score, and macro averages of 0.96, 0.95, and 0.95, respectively. Support values indicate class instance counts.

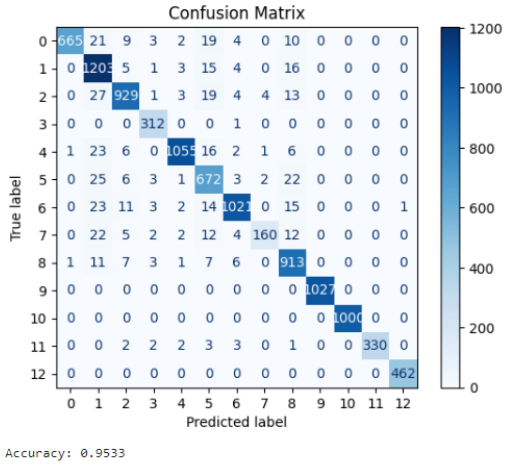

Fig. 10. Confusion Matrix

Figure 10 displays the confusion matrix heatmap which illustrates the relationship between true labels (y-axis) and predicted labels (x-axis), where high values along the diagonal represent correct predictions. Off-diagonal cells indicate misclassifications, with notable misclassifications occurring for classes 0 to 8. Despite these off-diagonal errors, the model has performed well overall, demonstrating high accuracy and minimal misclassifications.

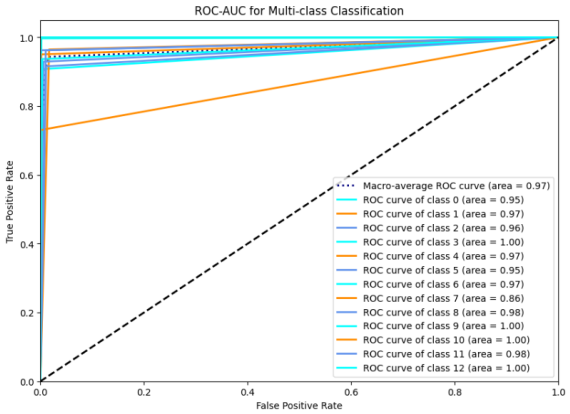

Fig. 11. Plot for ROC Curve

Figure 11 illustrated ROC curve for the multi-class classification model to distinguish the model's ability to between different classes. The x-axis represents the False Positive Rate (FPR), while the y-axis represents the True Positive Rate (TPR), with each colored line corresponding to a specific class. The closer a curve is to the top-left corner, the better the model's performance for that class. The AUC values, ranging from 0.97 to 1.00, indicate excellent classification performance across all classes. The macro-average ROC curve, with an AUC of 0.97, provides a strong summary of the

model's overall effectiveness. This high AUC across classes suggests that the model effectively distinguishes between them, demonstrating strong classification accuracy with minimal misclassification.

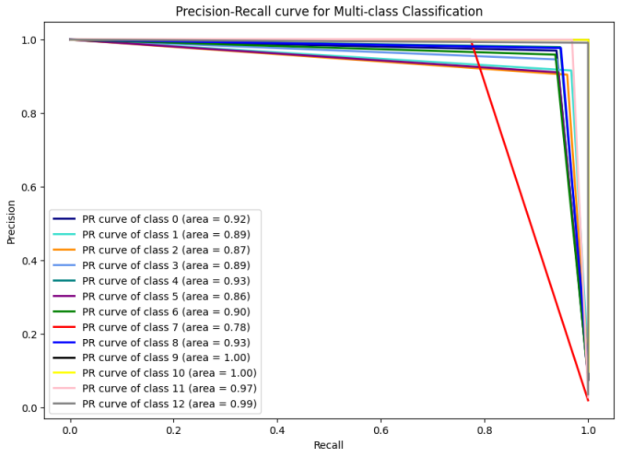

Fig. 12. Plot for Precision-Recall Curve

.

The Precision-Recall curve in figure 12 illustrates the classification performance for multiple classes, where each curve represents the trade-off between precision and recall for a specific class. Most classes show high precision and recall, with AUC values varying from 0.78 to 1.0, signifying generally strong classification performance. Class 9 and Class 10 have perfect AUC scores (1.0), while Class 7 has the lowest AUC (0.78), suggesting the model struggles more with this class. Overall, the model performs well across most classes, maintaining high precision even as recall increases, but shows some variability in performance depending on the class.

TABLE III
TEXT LENGTH CHARACTERISTICS

Characteristic	Value
Minimum Length	0
Maximum Length	32,759
Mean Length	160.21
Median Length	110.0
Standard Deviation of Length	581.51

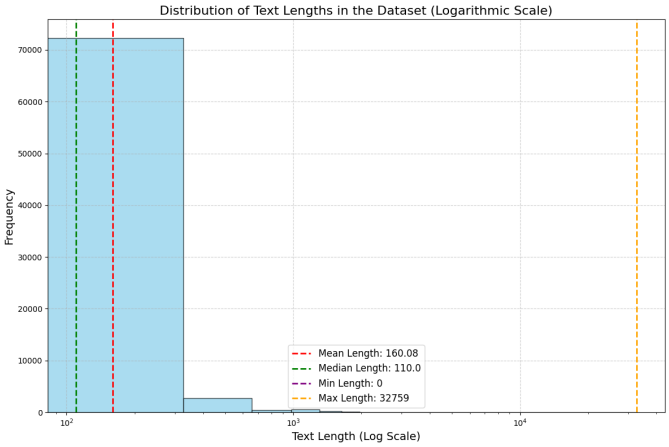

Fig. 13. Distribution of Text Lengths

Table IV and figure 13 shows how the lengths of texts in the dataset are distributed. The x-axis, which uses a log scale to make the wide range of lengths easier to see, shows the length of each text, while the y-axis shows how often texts of each length appear i.e frequency. The histogram bars represent these frequencies, with taller bars meaning more texts of that length. There are also dashed lines which mark important points: the mean (red) shows the average text length, the median (green) is the middle value, and the minimum (purple) and maximum (orange) mark the shortest and longest texts. Labels, a title, and a grid make the plot easy to understand, and the legend explains what each line represents. Overall, the plot helps us see the spread of text lengths in the dataset clearly.

V. CONCLUSION AND FUTURE WORK

This work classified Android malware using SecureBERT fine-tuned with LoRA, achieving up to 95.33% accuracy. Data augmentation and synthetic data improved performance, but detecting real-world obfuscated malware remained challenging due to reliance on textual features. Class imbalance caused discrepancies in some categories. An interactive tool was developed for real-time predictions, enhancing practical applicability.

Future work focuses on enhancing the dataset with diverse, balanced, and augmented malware samples, along with metadata such as behavior logs and network traffic to improve model performance. Applying ensemble methods, advanced feature engineering, and techniques like fine-tuning, transfer learning, and

adversarial training can further increase robustness. Ensuring interpretability through Explainable AI (XAI) and reducing bias is essential for trust, while deployment with sandbox-generated behavioral data and continuous updates with new malware variants will improve real-world effectiveness and enable better detection of evolving malware generations.

REFERENCES

[1] K. Stein, A. Mahyari, G. F. III, and E. Alzahrani, "A transformer-based framework for payload malware detection and classification," *arXiv preprint arXiv:2403.18223*, 2024, arXiv:2403.18223v1. [Online]. Available: https://arxiv.org/abs/2403.18223

[2] Einfochips, "Malware detection using machine learning techniques," *Einfochips Blog*, 2023, accessed: 2025-08-20. [Online]. Available: https://www.einfochips.com/blog/malware-detection-using-machine-learning-techniques/

[3] N. Sahin, "Malware detection using transformers-based model gpt-2," in *Proceedings of the 2021 International Conference on Cybersecurity*, 2021.

[4] I. Antonellis and E. Gallopoulos, "Exploring term-document matrices from matrix models in text mining," *arXiv preprint cs/0602076*, 2006. [Online]. Available: https://arxiv.org/abs/cs/0602076

[5] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: A statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[6] S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, "Hybrids of support vector machine wrapper and filter based framework for malware detection," *Future Generation Computer Systems*, vol. 55, pp. 376–390, 2016. [Online]. Available: https://doi.org/10.1016/j.future.2014.06.001

[7] K. Monnappa, *Learning Malware Analysis: Explore the Concepts, Tools, and Techniques to Analyze and Investigate Windows Malware*. Packt Publishing Ltd., 2018.

[8] R. Sihwail, K. Omar, K. A. Z. Ariffin, and S. A. Afghani, "Malware detection approach based on artifacts in memory image and dynamic analysis," *Applied Sciences*, vol. 9, no. 18, p. 3680, 2019. [Online]. Available: http://dx.doi.org/10.3390/app9183680

[9] S. Nautiyal, C. R. Krishna, and S. Wadhwa, "Mitigating economic denial of sustainability (edos) in cloud environment using genetic algorithm and artificial neural network," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 8, no. 10, pp. 1993–1998, Aug 2019.

[10] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

[11] Y. A. Ahmed, B. Koçer, S. Huda, B. A. S. Al-rimy, and M. M. Hassan, "A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection," *Journal of Network and Computer Applications*, vol. 167, p. 102753, 2020.

[12] A. Rahali and M. A. Akhloufi, "Malbertv2: Code aware bert-based model for malware identification," *Big Data and Cognitive Computing*, vol. 7, no. 2, p. 60, Mar 2023.

[13] S. Poornima and R. Mahalakshmi, "Automated malware detection using machine learning and deep learning approaches for android applications," *Measurement: Sensors*, 2024.

[14] Y. Wang and S. Jia, "Madras-net: A deep learning approach for detecting and classifying android malware using linknet," *Measurement: Sensors*, 2024.

[15] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "Memaldet: A memory analysis-based malware detection framework using deep autoencoders and stacked ensemble under temporal evaluations," *Computers & Security*, 2024.

[16] TechTarget, "What is the bert language model?" https://www.techtarget.com/, 2024, accessed: 2024-07-20.

[17] DS Stream, "Roberta vs bert: Exploring the evolution of transformer models," https://dsstream.com/roberta-vs-bert-exploring-the-evolution-of-transformer-models/, 2024, accessed: 2024-08-20.

[18] E. Aghaei, "Securebert," https://huggingface.co/ehsanaghaei/SecureBERT, 2023, accessed: 2024-08-20.

[19] E. E. Aghaie and W. Shadid, "Securebert: A domain specific language model for cybersecurity," 2022, accessed: 2024-08-20.

[20] "Word2vec," https://en.wikipedia.org/wiki/Word2vec, May 2024, accessed: 2024-05-29.

[21] B. Jawade, "Understanding lora: Low-rank adaptation for finetuning large models," Jun. 2023, accessed: 2024-08-20.

[22] Y. Shizuya, "Understanding lora with python implementation," Medium, 2023, accessed: 2024-08-20.

[23] B. Liang, C. Hlauschek, Y. Zhou, X. Wang, and Y. Xue, "Androzoo: Collecting millions of android apps for the research community," https://androzoo.uni.lu/, 2016.

[24] S. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Efficient and explainable detection of android malware," https://drebin.mlsec.org/, 2024.

[25] H. Shiravi, A. Shiravi, and A. A. Ghorbani, "A realistic dataset for anomaly-based network intrusion detection," https://www.unb.ca/cic/datasets/andmal2017.html, Jul. 2017, accessed: 2024-06-04.

[26] J. Beach, "Tuandromd dataset," https://www.kaggle.com/datasets/joebeachcapital/tuandromd, 2023, accessed: 2024-07-28.

[27] B. Catak and Y. Yazi, "A benchmark api call dataset for windows pe malware classification," in *Proceedings of the 2020 IEEE European Symposium on Security and Privacy (EuroSP)*. IEEE, 2020, pp. 411–425.

[28] amdj3dax, "Ransomware detection data set," 2023, accessed: 2024-07-28.

[29] subhajournal, "Trojan detection," 2024, accessed: 2024-07-28.

[30] K. Bandla, "Aptnotes," https://github.com/kbandla/APTnotes, 2024, accessed: 2024-08-18.