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*
Shramik Shanti Campus
Pokhara University, Nepal
utkar24@gmail.com

Om Prakash Mahato
Nepal Telecommunications Authority
Kathmandu, Nepal
opmahato@nta.gov.np

*Corresponding author

***Abstract*—The rapid growth and increasing sophistication of malware pose significant threats to modern cybersecurity systems, where traditional signature-based and static analysis techniques often fail to detect evolving and zero-day attacks. This study proposes an interactive malware analysis framework leveraging a RoBERTa-based SecureBERT model to perform accurate and real-time classification of malware-related text. Diverse benchmark datasets are collected and transformed into textual representations, followed by extensive preprocessing, data balancing, and augmentation strategies to address class imbalance and improve generalization. Additionally, synthetic data generation is incorporated to enhance the detection of rare and emerging malware patterns. The SecureBERT model is fine-tuned using Low-Rank Adaptation (LoRA), enabling efficient training with reduced computational overhead while maintaining high performance. The system integrates an interactive interface that allows real-time user input and classification, improving practical usability. Experimental results demonstrate strong performance, achieving an overall accuracy of approximately 96% with high precision, recall, and F1-scores across multiple malware categories. Evaluation through confusion matrices, ROC curves, and precision-recall analysis further validates the robustness of the approach. Despite its effectiveness, the model exhibits limitations in handling highly obfuscated real-world malware due to its reliance on textual features. The proposed framework offers a scalable, adaptive, and efficient solution for malware classification, advancing intelligent cybersecurity systems.**

***Keywords*—cybersecurity,malware,LoRA,RoBERTa**

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

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

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

detection by integrating SecureBERT into cybersecurity frameworks, offering a proactive, transformer-based approach with improved adaptability, scalability, and precision.

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.

Naseeer et.al [16] proposed *Syn-detect*which is a hybrid Android malware detection framework that integrates LLM-based synthetic data generation with a fine-tuned BERT classifier. GPT-2 generates synthetic TCP traffic, validated it using KL-divergence, and is combined with real data. The proposed model was evaluated on CIC-AndMal2017 and CIC-AAGM2017 where it achieved up to 99.8% accuracy and an MCC of ~ 0.99.

Hussain et.al [17] presented a transfer learning framework for ransomware detection and family classification using fine-tuned BERT and RoBERTa on API call sequences extracted from PE files. Trained on a dataset of 3,300 samples, the models capture complex ransomware behaviors and achieve 95.6% (BERT) and 94.4% (RoBERTa) accuracy, outperforming traditional methods.

Obregon et.al [18] proposed a machine learning approach for ransomware detection using static PE header features, eliminating file execution. A dataset of 2,497 samples from 19 ransomware families is processed with feature extraction, selection, and class balancing. Ensemble models (Random Forest, XGBoost, LightGBM) achieve F1 $\approx 97.6\%$ and AUC ≈ 0.999, outperforming MLP.

Chen et.al [19] proposes a hybrid, metadata-driven framework for malicious URL detection combining a RoBERTa-Large transformer with multi-source network threat intelligence. It integrates contextual subword embeddings from URLs with lightweight structural metadata (e.g., length, slash count, entropy) via a dual-attention mechanism, enabling effective detection of lexical obfuscation and network-level threats.

Alhazmi et.al [20] proposed a hybrid robust model that is trained on a balanced dataset of benign, phishing, malware, and defacement URLs, achieving approximately 98% accuracy and outperforming SVM, XGBoost, and LSTM. Explainability techniques such as SHAP and LIME are used to interpret predictions, emphasizing the role of structural features and attention mechanisms.

Despite advancements, static analysis (e.g., TDM, BoW, TF–IDF, API calls) is vulnerable to obfuscation and data quality issues, while dynamic analysis (e.g., memory forensics, API sequences) suffers from high overhead and limited scalability. Although deep learning models such as MalBERTv2, MAD-NET, MADRAS-NET, and transformer-based DPI achieve high accuracy, they face challenges in generalization, adaptability, and resource efficiency. To address these issues, a RoBERTa-based model is employed for malware classification.

The framework tackles data imbalance and improves detection of evolving threats through synthetic data generation, enhancing the identification of rare malware patterns.

III. Methodology

A. Architectures

1) BERT: : 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 [21].

2) RoBERTa: : 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. [22].

3) SecureBERT: : SecureBERT is a security-focused variant of BERT built on RoBERTa-base, fine-tuned for Android malware analysis. It leverages bidirectional transformers to capture contextual patterns of malicious activity. Trained using Masked Language Modeling on 98,411 cybersecurity texts (∼1 billion tokens) with a custom vocabulary of 50,265 tokens, it has 123 million parameters and incorporates noise injection during training. The model outperforms RoBERTa and SciBERT in threat detection and risk assessment. [23] It is highly effective for malware-related text classification, demonstrating strong performance on noisy and imbalanced data while maintaining high accuracy across cybersecurity NLP tasks.

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 [24].

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

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:

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

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:

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

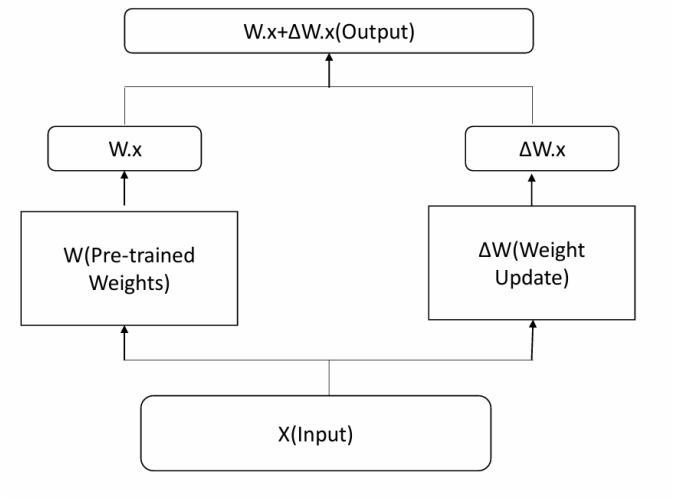

Fig. 1. LoRA Process

3) LoRA (Low-Rank Adaptation): Figure 1 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 $r \ll \min(d,k)$ [25]. 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 [25]. 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.

C. System Block Diagram

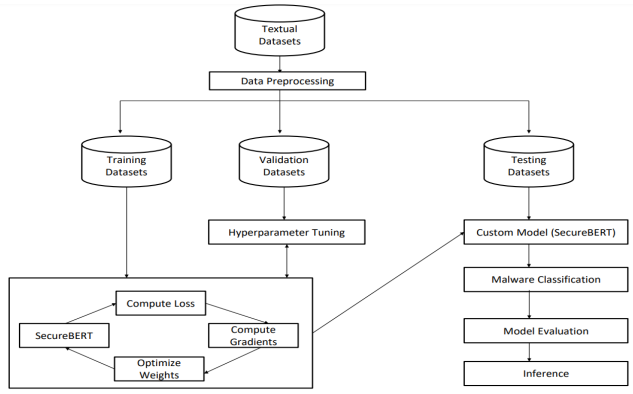

Fig. 2. Block Diagram for Android Malware Analysis using SecureBERT

Figure 2 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. Classes of dataset were initally unbalanced, hence they were balanced using certain methods. 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.

D. 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 [26]; Drebin includes extracted Android app features [27]; CICMalDroid2017 offers .pcap files and 80+ network features across 42 families in four categories [28];Tuandromd contains 4,465 instances with 241 attributes labeled as malware or goodware [29]; MalwareTextDB supplies NLP-based data on malware types (viruses, worms, trojans, ransomware, APTs) with metadata [30]; Ransomware and Trojan datasets emphasize malicious vs. benign network traffic [31], [32]; and APT Notes compiles documents on Advanced Persistent Threats [33].

TABLE I
LABEL DISTRIBUTION IN DATASET

Label	Number of Instances
Scareware	4952
Ransomware	8221
Adware	6865
SMSMalware	5098
Trojan	7167
Benign	5178
Spyware	5385
Polymorphic	1320
Downloader	6204
Cryptojacker	4897
Worm	6422
FakeApp	1811
Keylogger	2269
Other	10690

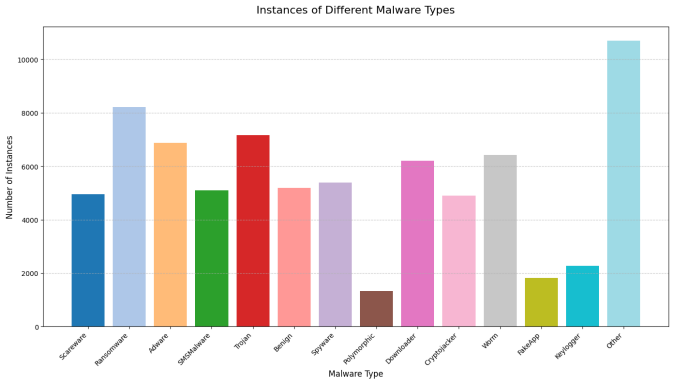

Fig. 3. Bar graph representation of different malware types

The dataset comprises 14 labels with varying instance counts, indicating class imbalance. The largest class is *Other* (10,690), followed by *Ransomware* (8,221), *Trojan* (7,167), and *Adware* (6,865). Moderate-frequency classes include *Worm* (6,422), *Downloader* (6,204), *Spyware* (5,385), *SMSMalware* (5,098), *Benign* (5,178), and *Scareware* (4,952). Less frequent classes are *Cryptojacker* (4,897), *Keylogger* (2,269), *FakeApp* (1,811), and *Polymorphic* (1,320). The distribution is highly skewed, which may affect model performance and requires strategies such as class weighting or resampling. Hence, the data balancing approach is applied which is discussed below.

E. Data Balancing Approach

To mitigate class imbalance, a hybrid strategy combining oversampling (via augmentation) and undersampling is employed in equation 13.

Imbalance Ratio:

$$IR = \frac{\max(N_c)}{\min(N_c)} \tag{13}$$

where N_c denotes the number of samples in class c.

Target Samples per Class:

$$T = \min\left(\frac{\sum_{i=1}^{3} N_{\text{top3}}}{3},\ 2000\right) \tag{14}$$

This sets a balanced upper limit based on dominant classes as described in equation 14.

1) Adaptive Balancing Strategy: Classes are adjusted relative to the target size T. *Minority Classes* ($N_c < T$): Samples are augmented as given below in equation 15:

$$k = \left\lceil \frac{T - N_c}{N_c} \right\rceil \tag{15}$$

Aggressive augmentation is applied when $N_c < 10$, moderate when $N_c < 0.3T$, and light otherwise. *Majority Classes* ($N_c > T$): In majority classes,random undersampling is performed which is described in equation 16:

$$N_{\text{sampled}} = T \tag{16}$$

2) Augmentation Strategies: It has been applied to increase diversity, multiple augmentation techniques are applied probabilistically which is described in equation 17.

Synonym Replacement:

$$P(\text{synonym}) = 0.3 \cdot 1(w \in \text{SynonymDict}) \tag{17}$$

Replaces words with their synonyms when available has been described in equation 18.

Random Deletion:

$$n_{\text{delete}} \sim \mathcal{U}(0.1L,\ 0.2L) \tag{18}$$

Randomly removes tokens, where L is sequence length which has been described in equation 18.

Random Swap:

$$n_{\text{swap}} = \max(1, \lfloor 0.1L \rfloor) \tag{19}$$

Swaps positions of randomly selected tokens which is shown in equation 19.

Character Noise:

$$n_{\text{replace}} = \max(1, \lfloor 0.05L \rfloor) \tag{20}$$

It Introduces minor character-level perturbations as described in equation 20 . Back-translation artifacts are also simulated with probability 0.3 to mimic linguistic variation.

IV. Result and Analysis

TABLE II
MALWARE MESSAGE CLASSIFICATION RESULTS

Field	Content
Title	Malware Message Classification Interface
Instruction	Enter a message to classify or type `exit` to quit.
Message 1	`File with hash 0124e21d...a76bc0.dll showing abnormal system-level attributes and behavior patterns.`
Prediction 1	**Ransomware**
Message 2	`Browser redirected to BestDeals.com for online shopping discounts.`
Prediction 2	**Adware**
Message 3	`exit`
Status	Program terminated successfully.

Table II presents the outputs of the Malware Message Classification Interface. The table consists of two columns: *Field*, indicating the type of information, and *Content*, providing the corresponding details. It includes sample input messages and their predicted malware categories. Message 1, describing abnormal system behavior, is classified as *Ransomware*, while Message 2, involving browser redirection, is classified as *Adware*. Message 3 represents the exit command, after which the system terminates successfully. The table demonstrates the model's ability to classify messages and handle user interaction.

TABLE III
TOKEN INFLUENCE ANALYSIS FOR MALWARE MESSAGE

Rank	Token	Influence Score
1	replicating	8.5343
2	app	6.1650
3	itself	4.5933
4	infected	3.1583
5	network	2.6012
6	This	2.3144
7	causing	2.0087
8	is	1.8593
9	across	1.6272
10	devices	1.5380

Table III presents the most influential tokens from the message *This app is replicating itself across my network and causing other devices to become infected.*, which the model classifies as a *worm*. The influence scores indicate each token's contribution to the prediction. Tokens such as *replicating*, *itself*, and *infected* have the highest impact, reflecting key characteristics of self-propagation and infection. Terms like *network* and *devices* further emphasize the spread of the malware, while common words such as *is* and *this* show minimal influence. Overall, the model effectively focuses on behavior-related terms for accurate classification.

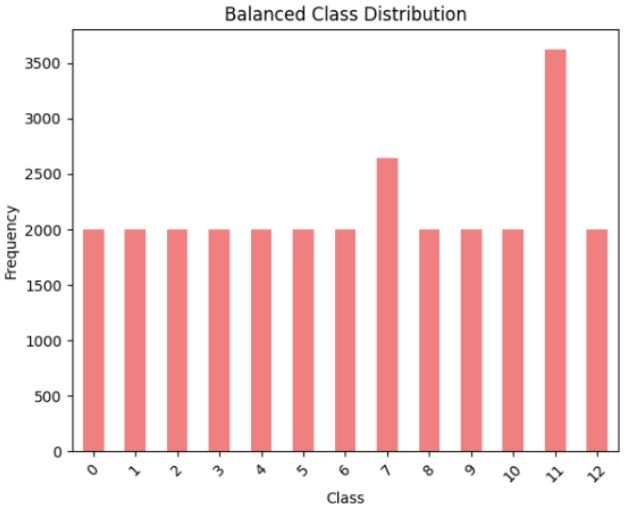

Fig. 4. Balanced Class Distribution

.

TABLE IV
BALANCED CLASS DISTRIBUTION WITH CORRESPONDING WEIGHTS

Label	Count	Weight
0	2000	1.0870
1	2000	1.0870
2	2000	1.0870
3	2000	1.0870
4	2000	1.0870
5	2000	1.0870
6	2000	1.0870
7	2640	0.8235
8	2000	1.0870
9	2000	1.0870
10	2000	1.0870
11	3622	0.6003
12	2000	1.0870

Table IV and Figure 4 illustrates a largely balanced dataset where most classes contain 2000 samples, while label 7 and label 11 are relatively overrepresented with 2640 and 3622 samples, respectively. To compensate for this imbalance, class weights are assigned inversely to the number of samples, with balanced classes receiving a higher weight (1.0870) and the overrepresented classes

receiving lower weights (0.8235 and 0.6003). The weighting strategy helps to ensure that the model treats all classes more fairly and reduces bias toward classes with more data.

TABLE V
HYPERPARAMETERS FOR MODEL TUNING

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

Hyperparameter tuning was conducted to optimize model performance by balancing accuracy and generalization. Learning rates ($1e$-4 to $3e$-5), batch sizes (16–128), and epochs (3–15) were evaluated, as shown in Table V. The best configuration was a learning rate of $1e$-5, batch size of 32, and 15 epochs. Early stopping was applied to prevent overfitting. Additionally, expanding the test set from 4,000 to 13,000 samples improved evaluation robustness while maintaining generalization.

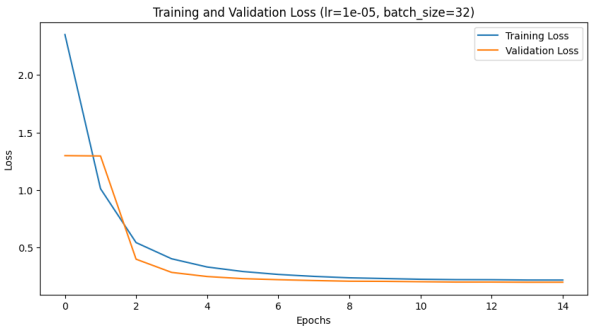

Fig. 5. Line Plot describing training/validation plot

Figure 5 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 VI
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 VI 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.

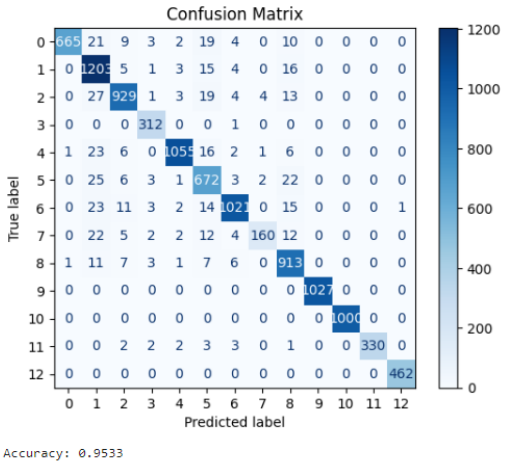

Fig. 6. Confusion Matrix

Figure 6 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.

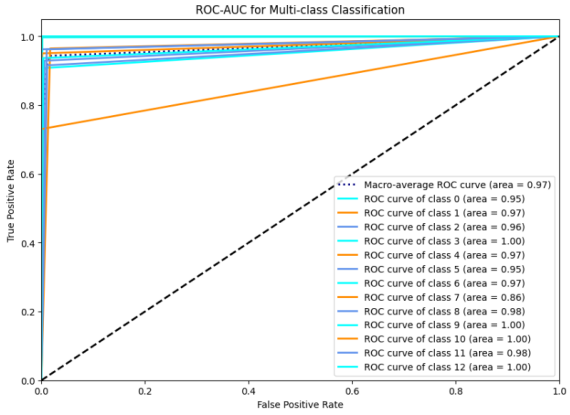

Fig. 7. Plot for ROC Curve

Figure 7 shows the ROC curves for the multi-class classification model, illustrating its ability to distinguish between classes. The x-axis represents the False Positive Rate (FPR) and the y-axis the True Positive Rate (TPR), with each colored line corresponding to a class. Curves closer to the top-left indicate better performance. The AUC values (0.97–1.00) reflect excellent classification across all classes, while the macro-average AUC of 0.97 summarizes strong overall performance with minimal misclassification.

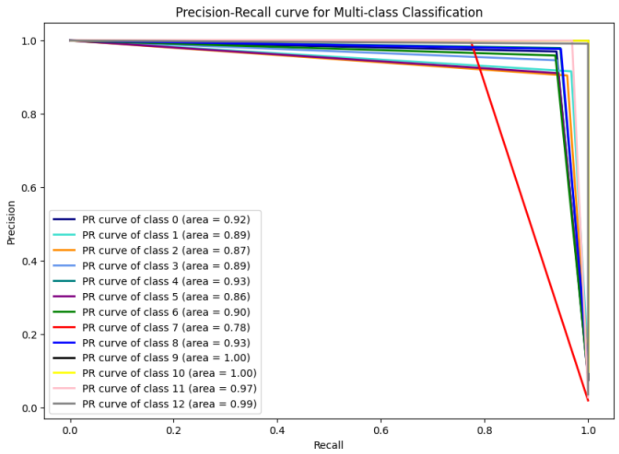

Fig. 8. Plot for Precision-Recall Curve

The Precision-Recall curve in Figure 8 illustrates the model's performance across multiple classes, with each curve showing the trade-off between precision and recall. Most classes achieve high precision and recall, with AUC values ranging from 0.78 to 1.0, indicating strong overall performance. Classes 9 and 10 reach perfect AUC (1.0), while Class 7 has the lowest AUC (0.78), suggesting relatively weaker performance. Overall, the model maintains high precision as recall increases, with slight variability across classes.

TABLE VII
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

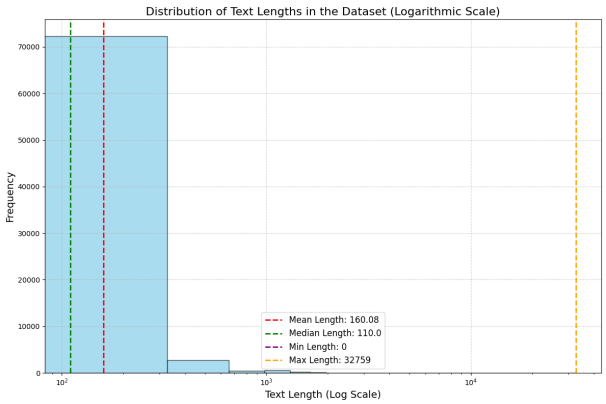

Fig. 9. Distribution of Text Lengths

Table VII and Figure 9 illustrates the distribution of text lengths in the dataset. The x-axis (log scale) represents text length, while the y-axis shows frequency. Histogram bars indicate how often each length occurs. Dashed lines highlight key statistics: mean (red), median (green), minimum (purple), and maximum (orange). Overall, the plot clearly shows the spread and variation of text lengths.

V. CONCLUSION

This work presented a SecureBERT-based approach for malware classification, enhanced with LoRA fine-tuning and supported by diverse datasets, data balancing, and augmentation techniques. The model achieved strong performance, with an overall accuracy of 96% and high precision, recall, and F1-scores across most classes. The results highlight the effectiveness of transformer-based models in capturing contextual patterns for multi-class malware detection. The inclusion of synthetic data and token influence analysis further improved generalization and interpretability. Additionally, the interactive system demonstrates the model's applicability for real-time malware classification.

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] M. Naseer, F. Ullah, S. Ijaz, H. Naeem, A. Alsirhani, G. N. Alwakid, and A. Alomari, "Obfuscated malware detection and classification in network traffic leveraging hybrid large language models and synthetic data," *Sensors*, vol. 25, no. 1, p. 202, 2025.

[17] A. Hussain, A. Saadia, and F. M. Alserhani, "Ransomware detection and family classification using fine-tuned bert and roberta models," *Egyptian Informatics Journal*, vol. 30, p. 100645, 2025.

[18] V. A. M. Obregon, J. P. R. Diaz, and C. M. T. Paredes, "Ransomware detection using executable header features and machine learning techniques," in *Proceedings of the 8th International Conference on Systems Engineering - Cybersecurity & AI: Building a Reliable Digital Future (CIIS 2025)*. ACM, 2025, pp. 1–11.

[19] L. Chen and L. Meng, "Metadata driven malicious url detection using roberta large and multi source network threat intelligence," *Scientific Reports*, vol. 16, p. 5449, 2026.

[20] A. H. Alhazmi, "A robust and dynamic malware detection and classification model using behavioral-based analysis and bert technique," *PLOS One*, vol. 20, no. 9, p. e0327604, 2025. [Online]. Available: https://doi.org/10.1371/journal.pone.0327604

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

[22] 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.

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

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

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

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

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

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