DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware

Tiezhu Sun*
tiezhu.sun@uni.lu
University of Luxembourg
Kirchberg, Luxembourg

Nadia Daoudi
nadia.daoudi@list.lu
Luxembourg Institute of Science and Technology
Esch-sur-Alzette, Luxembourg

Kisub Kim*
falconlk00@gmail.com
Independent Researcher
Hongkong

Kevin Allix
kallix@kallix.net
Independent Researcher
Rennes, France

Tegawendé F. Bissyandé
tegawende.bissyande@uni.lu
University of Luxembourg
Kirchberg, Luxembourg

Jacques Klein
jacques.klein@uni.lu
University of Luxembourg
Kirchberg, Luxembourg

Abstract

Recent advancements in ML and DL have significantly improved Android malware detection, yet many methodologies still rely on basic static analysis, bytecode, or function call graphs that often fail to capture complex malicious behaviors. DexBERT, a pre-trained BERT-like model tailored for Android representation learning, enriches class-level representations by analyzing Smali code extracted from APKs. However, its functionality is constrained by its inability to process multiple Smali classes simultaneously. This paper introduces DetectBERT, which integrates correlated Multiple Instance Learning (c-MIL) with DexBERT to handle the high dimensionality and variability of Android malware, enabling effective app-level detection. By treating class-level features as instances within MIL bags, DetectBERT aggregates these into a comprehensive app-level representation. Our evaluation demonstrates that DetectBERT not only surpasses existing state-of-the-art detection methods but also adapts to evolving malware threats. Moreover, the versatility of the DetectBERT framework holds promising potential for broader applications in app-level analysis and other software engineering tasks, offering new avenues for research and development.

ACM Reference Format:
Tiezhu Sun, Nadia Daoudi, Kisub Kim, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2024. DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware. In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3674805.3690745

1 Introduction

The ubiquity of Android devices in today's digital ecosystem has made them a prime target for malicious actors. As the sophistication of Android malware continues to evolve, so does the need for effective detection mechanisms. Traditional malware detection approaches, while beneficial, often struggle to keep pace with the rapid development of new and complex malware apps [4, 15]. This challenge is exacerbated by the dynamic and open nature of the Android platform, which allows for the frequent introduction of new applications and updates.

Recent advancements in machine learning [2, 24] (ML) and deep learning [5, 7, 11] (DL) have offered promising directions for enhancing Android malware detection. These techniques, capable of learning from large datasets to identify subtle patterns and anomalies, have shown their potential across a broad range of software engineering applications. However, they largely depend on static analysis [8, 13, 18], extracting basic features or low-level bytecode from APK files [5, 22], and analyzing function call graphs [14, 27]. While these approaches showcase some effectiveness, they often fall short in capturing the nuanced and complex behaviors that characterize new emerging malware apps.

Recently, DexBERT [21] has been proposed as a pre-trained BERT [6]-like model specifically designed for class-level Android representation learning. By learning the Smali code disassembled from APKs, DexBERT has demonstrated its capability of significantly enhancing performance in class-level Android analysis tasks, such as malicious code localization. Nevertheless, the applications of DexBERT are limited by its input constraints, which makes it only able to analyze a single Smali class and cannot support an overall app-level understanding.

To address a similar issue in whole slide image classification and long document classification, TransMIL [19] and LaFiCMIL [23] leverage correlated Multiple Instance Learning (c-MIL). This enhances the ability to identify and learn valuable features from a vast array of cropped image patches and segmented document fragments, which are sourced from large medical images and lengthy documents, respectively. Inspired by previous studies and recognizing the typical presence of multiple classes within an APK, we employ c-MIL principles for full app-level representation learning to detect Android malware, which led to the development of our proposed model, DetectBERT. Specifically, DetectBERT sets itself apart from TransMIL and LaFiCMIL by:

- Employing Smali classes directly as natural instances within the c-MIL framework, avoiding the need for preliminary data processing such as image cropping or document splitting.

*Corresponding Authors.

ESEM '24, October 24–25, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-1047-6/24/10
https://doi.org/10.1145/3674805.3690745

- Omitting positional embeddings due to the non-sequential nature of Smali class interconnections, which are based on invocation links.
- Freezing DexBERT's weights during training to leverage the pre-trained model's original capabilities for generating class embeddings, thus conserving computational resources.

To implement c-MIL for our scenario, DetectBERT employs the Nyström Attention layer [25], a technique that efficiently learns the relationships among embedding vectors. Specifically, upon generating class embeddings via DexBERT, we introduce a learnable category vector that is initialized to conform to a normal distribution and is dimensionally compatible with the class embeddings. Subsequently, both the category vector and all class embeddings are processed through the Nyström Attention layer. This mechanism helps DetectBERT learn the intricate correlations among the vectors, enabling a dynamic exchange of information between the category vector and each class embedding. Such an exchange is pivotal for highlighting features essential to malware detection. The process enriches the category vector, which is then passed through a fully connected layer, serving as the foundation for the ultimate malware detection decision.

We conduct an extensive evaluation on a large dataset of 158 803 applications. The experimental results demonstrate that DetectBERT not only surpasses three basic feature aggregation methods but also outperforms two state-of-the-art Android malware detection techniques. Furthermore, we conduct a temporal consistency evaluation to assess how well DetectBERT performs over time, especially when confronted with newer malware samples absent in the training data. This assessment reaffirms the robustness of DetectBERT, demonstrating its sustained effectiveness against emerging threats and highlighting its relevance in dynamic real-world scenarios. This research bridges the gap between sophisticated representation learning models and the practical demands of app-level malware analysis. Looking forward, DetectBERT showcases significant potential to expand its utility in software engineering, particularly by using Multiple Instance Learning (MIL) to efficiently process and analyze large-scale inputs. This capability is crucial for advancing software quality and security assessments in complex environments like mobile and IoT ecosystems.

The contributions of our study are as follows:

- We introduce DetectBERT, an efficient and effective approach utilizing MIL to scale DexBERT for full app-level representation learning in Android malware detection.
- We perform a comprehensive evaluation, demonstrating DetectBERT's superiority over both basic feature aggregation methods and state-of-the-art malware detection techniques, including its robustness through a temporal consistency evaluation to verify performance against new, unseen malware samples.
- We highlight DetectBERT's potential to revolutionize software engineering, specifically through its use of MIL to manage large-scale data, offering significant improvements in software assessments.
- To facilitate replication, we have made the dataset and source code available at: https://github.com/Trustworthy-Software/DetectBERT

2 Background

To provide a foundation for understanding this study, this section delves into the background of three core concepts: Android malware detection, DexBERT, and multiple instance learning.

2.1 Android Malware Detection

Traditional Android malware detection methods have increasingly fallen short in addressing the complexity of malware varieties [4, 15]. The integration of machine learning [2, 24] and deep learning techniques [5, 7, 11] into the field represents a significant advancement, promising enhanced detection capabilities. Nonetheless, the application of these sophisticated approaches in Android malware detection encounters substantial challenges, as they largely rely on static analysis [8, 13, 18], low-level bytecode [5, 22], and function call graphs [14, 27], potentially overlooking the intricate behaviors inherent in modern malware. DetectBERT is the fist endeavor to enable DexBERT [21] to detect Android malware with a comprehensive app-level representation, from a novel perspective of multiple instance learning.

2.2 DexBERT

DexBERT [21] is a pretrained model based on the BERT architecture [6]. It is designed specifically for extracting class-level features from Android applications. It processes disassembled Smali code from Dalvik bytecode as input and learns to produce corresponding representations, i.e., the embedding of a Smali class. Smali serves as the textual representation of Dalvik bytecode, similar to how assembly code represents compiled code in a readable form. These class embeddings, or feature vectors, enable the execution of various class-level Android analysis tasks, including malicious code localization. DexBERT's ability to understand the intrinsic logic and behavior of applications represents a substantial step forward in the battle against malware. However, its input capacity is limited to 512 BERT tokens, which is only enough to process one single Smali class. Considering that an APK typically contains thousands of Smali classes, there is a clear need for an effective class embedding aggregation method. Our proposed method, DetectBERT, is designed to leverage DexBERT's representation learning capabilities for app-level tasks, such as Android malware detection.

2.3 Multiple Instance Learning

In this study, we adopt Multiple Instance Learning [10, 12, 20] (MIL) as our strategy for integrating class-level embeddings into an overall app-level representation. MIL is characterized by its ability to handle a collection of instances grouped into a "bag", where the bag is assigned a single label, but individual instance labels are not provided. A notable challenge within MIL is the variable number of instances each bag may contain, requiring a MIL model that can adeptly manage bags of diverse sizes. MIL's versatility has been demonstrated in various domains, especially in computer vision [19, 28] and in natural language processing [12, 23]. These successful applications inspired us to explore MIL's potential in enhancing Android malware detection using DexBERT. Specifically, by conceptualizing each APK as a "bag" and its constituent Smali classes as "instances" within this framework, we are able to

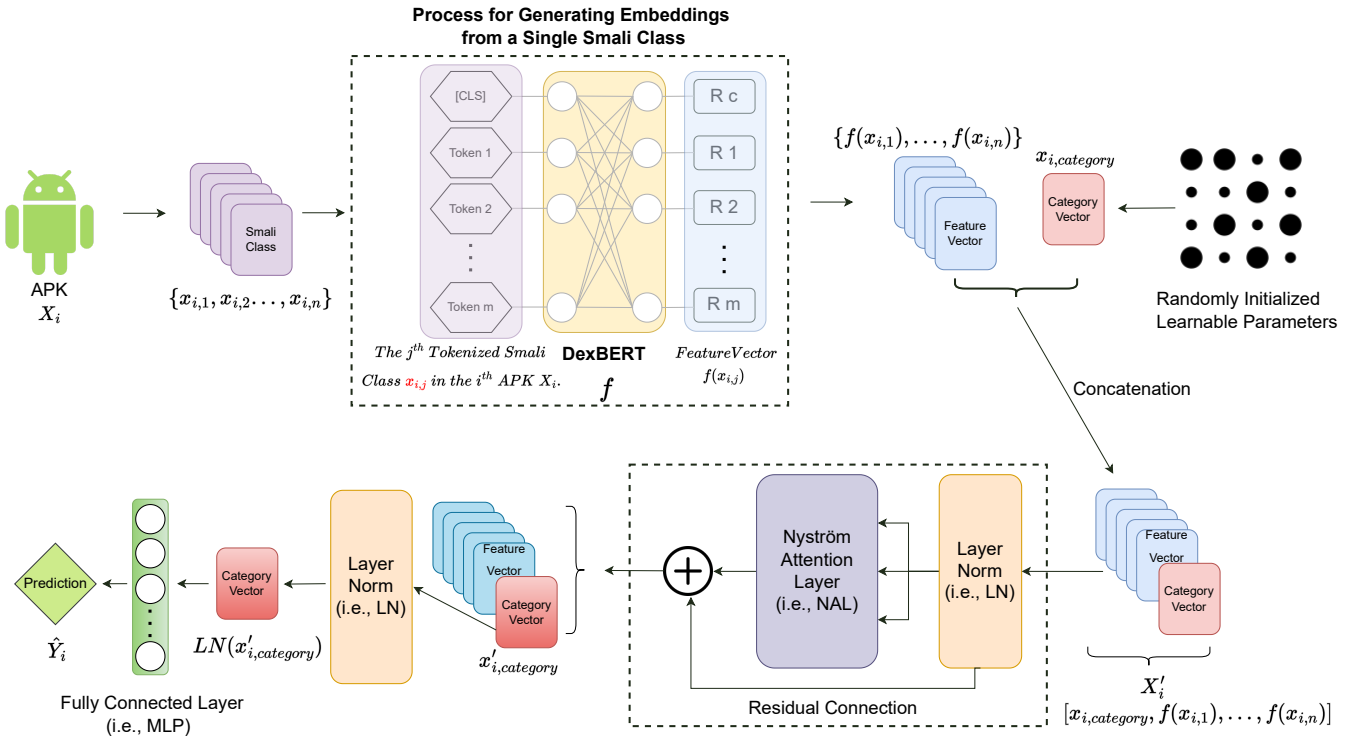

Figure 1: Overview of DetectBERT Workflow. First, DexBERT produces Smali class embeddings as c-MIL instances. A category vector of the same size is then introduced as an additional instance. The Nyström Attention layer helps DetectBERT find correlations among instances, allowing the category vector to capture key information from class embeddings for malware detection. Lastly, this vector is processed in a fully connected layer to make the detection decision.

establish a fitting model for detecting malware in Android applications. However, standard MIL may not fully capture the complex interconnections between Smali classes. This is where correlated Multiple Instance Learning (c-MIL) comes in, which considers the dependencies and interactions among instances within a bag. c-MIL enables DetectBERT to assess collective class behavior, crucial for identifying Android malware, by recognizing patterns that emerge from the invocation relationships among classes.

3 Approach

In this section, we start with the key theoretical foundations of our method, then detail the design of DetectBERT.

3.1 Theoretical Foundations

Theorem 3.1. *Suppose $S : \chi \to \mathbb{R}$ is a continuous set function w.r.t Hausdorff distance [16] $d_H(.,.)$. $\forall \varepsilon > 0$, for any invertible map $P : \chi \to \mathbb{R}^n$, $\exists$ function σ and g, such that for any set $X \in \chi$:*

$$|S(X) - g(P_{X \in \chi}\{\sigma(x) : x \in X\})| < \varepsilon \tag{1}$$

The proof of Theorem 3.1 can be found in [19]. This theorem shows that a Hausdorff continuous **set function** $S(X)$ can be arbitrarily approximated by a function in the form $g(P_{X \in \chi}\{\sigma(x) : x \in X\})$. This concept applies to MIL, where the theorem's sets correspond to MIL's bags. Consequently, the theorem provides a foundation for approximating bag-level predictions in MIL using instance-level features. In the context of Android malware detection, this principle instructs us to achieve app-level predictions through the utilization of class-level embeddings produced by DexBERT.

Theorem 3.2. *The instances in the bag are represented by random variables $\theta_1, \theta_2, ..., \theta_n$, the information entropy of the bag under the correlation assumption can be expressed as $H(\theta_1, \theta_2, ..., \theta_n)$, and the information entropy of the bag under the i.i.d. (independent and identical distribution) assumption can be expressed as $\sum_{t=1}^{n} H(\theta_t)$, then we have:*

$$\begin{aligned} H(\theta_1, \theta_2, ..., \theta_n) &= \sum_{t=2}^{n} H(\theta_t | \theta_1, \theta_2, ..., \theta_{t-1}) + H(\theta_1) \\ &\leq \sum_{t=1}^{n} H(\theta_t) \end{aligned} \tag{2}$$

The proof of Theorem 3.2 can be found in [19]. This theorem demonstrates that a bag under the correlation assumption has lower information entropy than that under the i.i.d. assumption (i.e., in standard MIL). The lower information entropy in c-MIL suggests reduced uncertainty and the potential to provide more valuable information for bag classification tasks than the standard MIL. In the context of Android malware detection, we claim that the Smali classes from the same APK are correlated in some way (e.g., invocation relationships between classes). This implies that the presence

or absence of a malicious class in a bag can be influenced by the other classes contained within the APK. Therefore, c-MIL seems a perfect choice to leverage the class embeddings of DexBERT for the app-level task of Android malware detection.

Given an app X_i composed of Smali classes $\{x_{i,1}, x_{i,2}, ..., x_{i,n}\}$, for $i = 1, ..., N$, that exhibit correlation among each other. The app-level label is Y_i, yet the class-level labels $\{y_{i,1}, y_{i,2}, ..., y_{i,n}\}$ are not accessible. Then, detecting Android malware in the context of c-MIL can be defined as:

$$Y_i = \begin{cases} 0, & if \sum y_{i,j} = 0 \quad y_{i,j} \in \{0, 1\}, j = 1, ..., n \\ 1, & otherwise \end{cases} \tag{3}$$

$$\hat{Y}_i = S(X_i), \tag{4}$$

where S is a scoring function and $\hat{Y}$ is the predicted score indicating the likelihood of the analyzed application being malware. N is the total number of applications, and n is the number of Smali classes in the ith application. The number n generally varies for different applications.

3.2 DetectBERT

According to Theorem 3.1, we leverage a series of **sub-functions to approximate** the c-MIL score function S defined in Equation 4. Given a set of APKs $\{X_1, ..., X_N\}$, where each APK X_i contains multiple Smali classes $\{x_{i,1}, ..., x_{i,n}\}$ and an APK label Y_i, a corresponding category vector $x_{i,category}$ is randomly initialized. We represent DexBERT [21] as a feature extracting function f, Multilayer Perceptron [17] as *MLP*, Nyström Attention Layer [25] as *NAL* and Layer Normalization [3] as *LN*. The c-MIL score function S can be approximated as follows:

$$X'_i = [x_{i,category}, f(x_{i,1}), ..., f(x_{i,n})] \tag{5}$$

$$x'_{i,category} = (NAL(LN(X'_i)) + X'_i)^{(0)} \tag{6}$$

$$\hat{Y}_i = MLP(LN(x'_{i,category})) \tag{7}$$

The selection of the Nyström Attention layer in DetectBERT's architecture is driven by its capability to efficiently handle large sequences of data. First, as an attention mechanism, Nyström Attention naturally facilitates the learning of correlations among class embeddings, treating them akin to tokens in natural language processing. This enables DetectBERT to dynamically assess and prioritize the relevance of each Smali class' features based on their contextual relationship, crucial for accurate malware detection. Secondly, when confronted with a vast number of tokens—typical in complex Android applications—the Nyström Attention layer proves significantly more efficient than traditional attention mechanisms. Its design reduces computational complexity from quadratic to linear, making it especially suitable for large-scale datasets where efficiency is paramount. These characteristics make Nyström Attention an ideal choice for DetectBERT.

DetectBERT distinguishes itself from TransMIL and LaFiCMIL in three key aspects. Firstly, unlike the image cropping in TransMIL or document splitting in LaFiCMIL, DetectBERT directly utilizes Smali classes as natural instances for c-MIL without requiring any preprocessing steps. Secondly, our model does not employ positional embeddings, as outlined in Equation 5. This decision is based on the observation that Smali classes are interconnected through invocation relationships rather than a sequential order. Lastly, to enhance efficiency, the weight parameters of DexBERT are frozen during training. DetectBERT leverages the DexBERT's original capacity to generate meaningful class embeddings without the need for further fine-tuning, thus avoiding additional computational costs.

The overall process of DetectBERT is outlined in Figure 1 and can be detailed as follows: given an APK X_i containing n Smali classes $\{x_{i,1}, ..., x_{i,n}\}$, we use the pretrained DexBERT model f to generate the corresponding embedding vectors $\{f(x_{i,1}), ..., f(x_{i,n})\}$. Then, we initialize a **learnable** category vector $x_{i,category}$ that conforms to a normal distribution and has the same shape as each class embedding. By considering the category vector as an additional embedding vector, we concatenate it with the class embeddings as $[x_{i,category}, f(x_{i,1}), ..., f(x_{i,n})]$, and learn the correlation between each embedding vector using the Nyström Attention layer. Layer normalization and residual connection [9] are applied to standardize the input and output data of the Nyström Attention layer and facilitate gradient backpropagation, enhancing model stability and performance. With the help of the attention mechanism, the category vector exchanges information with each class embedding and extracts necessary features for malware detection. Finally, the category vector is fed into a fully connected layer (i.e., MLP) to finalize the detection task.

4 Study Design

This section outlines the design of our study, detailing the research questions, dataset, empirical setup, and evaluation metrics used to assess the performance of DetectBERT.

4.1 Research Questions

We mainly investigate three research questions, each aimed at exploring different aspects of DetectBERT's performance:

- **RQ1**: How does DetectBERT perform compared to basic feature aggregation methods in detecting Android malware?
- **RQ2**: How does DetectBERT perform compared to state-of-the-art Android malware detection models?
- **RQ3**: How does DetectBERT maintain its detection effectiveness over time in the face of evolving Android malware?

These questions are designed to comprehensively evaluate the effectiveness, competitiveness, and robustness of DetectBERT.

4.2 Dataset

For our evaluation, we utilize the large-scale benchmark dataset from DexRay [5], which contains a total of 158 803 apps. This dataset facilitates a direct comparison of DetectBERT against basic aggregation methods and the state-of-the-art Android malware detection approaches. The malware and benign apps in the DexRay dataset are selectively collected from the popular and continually growing Android repository named AndroZoo [1]. Specifically, the DexRay dataset contains 96 994 benign and 61 809 malware apps. Benign apps are defined as the apps that have not been detected by any antivirus from VirusTotal [1], while the malware collection contains the apps that have been detected by at least two antivirus engines.

[1] https://www.virustotal.com/

4.3 Empirical Setup

In our empirical setup, we employ the DexBERT model to generate a feature vector for each Smali class. Considering the extensive size of the dataset, we optimize our process by storing these generated feature vectors on disk, thereby significantly reducing computational costs by preventing the need to regenerate feature vectors during each training phase. Consequently, the DetectBERT model requires only 2 GB of GPU memory for training and achieves an average inference time of just 0.005 seconds per app, underscoring its computational efficiency.

To enhance the model's capacity without altering the underlying DexBERT architecture, we freeze DexBERT's parameters and incorporate dual Nyström Attention layers. This approach allows us to expand the model's capabilities while maintaining a stable structure. For the optimization process, we employ the Lookahead optimizer [29] to fine-tune the model. The training is conducted for 20 epochs with a learning rate set to `1e-4`.

4.4 Evaluation Metrics

For comparison with basic aggregation methods in Section 5.1 and comparison with state-of-the-art approaches in Section 5.2, we follow the baseline DexRay [5] by shuffling the dataset and split it into 80% for training, 10% for validation, and 10% for test. This process is repeated 10 times to ensure robustness, with the average results reported. In Section 5.3, to assess temporal consistency, we temporally partition our dataset, using apps from 2019 for training (90%) and those from 2020 for testing (10%), based on the date information in the app's DEX file. Results are reported using the four metrics: accuracy (%), precision (%), recall (%), and F1 Score (%), with precision maintained to two decimal places for consistency with baseline results.

5 Experimental Results

In this section, we evaluate DetectBERT's performance by addressing the three research questions outlined in Section 4.1, aiming to provide insights into distinct aspects of its effectiveness.

5.1 RQ1: How does DetectBERT perform compared to basic feature aggregation methods in detecting Android malware?

In this subsection, we evaluate DetectBERT's performance in detecting Android malware by comparing it against three fundamental techniques for aggregating DexBERT embeddings: Random Selection, Element-wise Addition, and Element-wise Average. Each technique compresses information from multiple class embeddings into a single representative vector, which is then used as input to a fully connected layer for final prediction.

- **Random Selection**: It randomly selects a single class embedding to represent the entire APK, testing the efficacy of individual class features in malware detection.
- **Element-wise Addition**: It aggregates class embeddings by adding corresponding elements together, aiming to capture the cumulative effect of features across all classes.
- **Element-wise Average**: Similar to addition, it calculates the average of corresponding elements, offering a normalized representation of class features.

Table 1: Performance comparison with basic feature aggregation approaches.

Model	Accuracy	Precision	Recall	F1 Score
Random Selection	0.82	0.82	0.82	0.82
Element-wise Addition	0.86	0.87	0.84	0.86
Element-wise Average	0.92	0.92	0.92	0.92
DetectBERT	**0.97**	**0.98**	**0.95**	**0.97**

Table 1 shows that while the Element-wise Average method achieves an F1 score of 0.92, indicating the effectiveness of DexBERT embeddings, DetectBERT significantly outperforms this and other basic methods. Notably, DetectBERT attains superior precision and F1 scores of 0.98 and 0.97, respectively, improving performance by 6 and 5 percentage points over the Element-wise Average method.

This substantial improvement is attributed to DetectBERT 's sophisticated architecture, which more effectively captures and utilizes the intricate relationships among class embeddings. Unlike basic aggregation methods that compress embeddings into a singular vector, DetectBERT employs Nyström Attention layers. These layers dynamically adjust and integrate information from different classes based on their contextual relevance, facilitating a more nuanced aggregation. This advanced capability proves crucial in detecting Android malware, where subtle interactions among app features often signify malicious activity.

RQ1 Answer: DetectBERT significantly outperforms basic aggregation methods in detecting Android malware, emphasizing the importance of advanced embedding processing techniques for handling complex Android APKs, thus enhancing the reliability and accuracy of malware detection systems.

5.2 RQ2: How does DetectBERT perform compared to state-of-the-art Android malware detection models?

In this subsection, we evaluate the competitive performance of DetectBERT against two established state-of-the-art models in the field of Android malware detection: Drebin [2] and DexRay [5]. Drebin is known for its application of machine learning techniques to hand-crafted features, while DexRay utilizes a deep learning framework to analyze low-level bytecode images.

Table 2: Performance comparison with existing state-of-the-art approaches.

Model	Accuracy	Precision	Recall	F1 Score
Drebin	0.97	0.97	0.94	0.96
DexRay	0.97	0.97	0.95	0.96
DetectBERT	**0.97**	**0.98**	**0.95**	**0.97**

According to the findings in Table 2, DetectBERT not only matches but also exceeds the precision and F1 scores of the well-established Drebin and DexRay models on the challenging DexRay dataset. It elevates the precision score to 0.98, improving upon the already high score of 0.97 achieved by both Drebin and DexRay while maintaining high recall. These slight yet significant improvements in precision and F1 scores are critical as they highlight DetectBERT 's exceptional ability to pinpoint malware accurately without missing genuine threats. This performance enhancement is primarily due to DetectBERT's effective use of sophisticated c-MIL mechanisms. These mechanisms dynamically adjust based on the relationships among class embeddings, enabling DetectBERT to capture subtle nuances and connections within the Smali classes. This capability distinguishes it from other models that rely on conventional feature analysis techniques and may overlook such intricate relationships. This nuanced detection capability is key to DetectBERT's success, offering deeper insights into the complex behaviors typical of modern malware.

RQ2 Answer: DetectBERT slightly surpasses the performance of state-of-the-art models Drebin and DexRay. This superior performance still demonstrates the effectiveness of its architecture in leveraging MIL to enhance detection accuracy in complex malware datasets.

5.3 RQ3: How does DetectBERT maintain its detection effectiveness over time in the face of evolving Android malware?

Model aging [26, 30] is a significant challenge in machine learning, where a model's performance tends to degrade when applied to new, previously unseen samples. In this section, we assess the robustness of DetectBERT against model aging by evaluating its temporal consistency, a measure of how well a model trained on historical data performs against recent threats. The methodology for this temporal consistency evaluation is detailed in Section 4.4, where the dataset is divided into training sets from 2019 and testing sets from 2020 to accurately emulate real-world applications against evolving malware. The performance of DetectBERT and other state-of-the-art models is compared using these temporally distinct datasets.

Table 3: Temporal consistency performance comparison with state-of-the-art approaches.

Model	Accuracy	Precision	Recall	F1 Score
Drebin	0.96	0.95	0.98	0.97
DexRay	0.97	0.97	0.98	0.98
DetectBERT	**0.99**	**0.99**	**0.99**	**0.99**

Table 3 showcases the evaluation results, indicating that all models tested exhibit commendable adaptability to new malware samples. Notably, DetectBERT excels, achieving superior accuracy, precision, recall, and F1 scores of 0.99 across all metrics. This performance not only surpasses the other state-of-the-art models but also demonstrates significant improvements over the scores reported in Table 2, where the earlier data sample composition varied. This discrepancy highlights the exceptional capability of DetectBERT to generalize from past data to future scenarios without significant performance degradation. The increased proportion of training data from 2019 in this setup (90% compared to the previous 80%) likely played a significant role in enhancing the robustness of DetectBERT. This broader learning base enabled it to more effectively identify malware characteristics that persist or evolve over time, contributing to its heightened effectiveness.

DetectBERT 's standout performance in this temporal consistency evaluation confirms its effectiveness in handling new malware threats, a key advantage for maintaining relevance in the rapidly evolving landscape of Android security. This high level of adaptability stems from DetectBERT 's sophisticated representation learning capabilities, which extract and leverage the high-level semantics from Smali code in Dex files. Unlike simpler models that might rely on surface-level features, DetectBERT deeply understands the underlying behaviors and patterns encoded in the application's code, allowing it to detect even subtly disguised malware. This adaptability is critical for practical deployment, where the ability to perform consistently over time is paramount.

RQ3 Answer: DetectBERT exhibits exceptional temporal consistency in malware detection, effectively mitigating model aging issues and demonstrating robust generalization capabilities to new and evolving threats. These qualities position our model as an invaluable asset in the ongoing battle against Android malware.

6 Conclusion

In this paper, we introduce DetectBERT, a novel framework that leverages the capabilities of DexBERT for app-level analysis, specifically tailored for the complex task of Android malware detection. By employing a correlated Multiple Instance Learning (c-MIL) strategy, DetectBERT not only surpasses traditional feature aggregation methods but also outperforms current state-of-the-art malware detection models. The performance improvement suggests that DetectBERT could set a new standard for Android security analysis, representation learning, and other software engineering tasks.

Future Work: We aim to further validate DetectBERT by comparing it with additional malware detection models and enhancing its ability to analyze a wider range of malware behaviors. This includes integrating continuous learning mechanisms and incorporating more comprehensive contextual data from APKs. Furthermore, we are excited about the prospects of adapting our c-MIL-based framework for other software engineering tasks that handle large-scale data inputs, potentially broadening its applicability and impact in the field.

Acknowledgments

This research was funded in whole, or in part, by the Luxembourg National Research Fund (FNR), grant references 16344458 (REPROCESS), 18154263 (UNLOCK), and 17046335 (AFR PhD grant).

References

[1] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th international conference on mining software repositories*. 468–471.

[2] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android

malware in your pocket.. In *Ndss*, Vol. 14. 23–26.
[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).
[4] Naseef-Ur-Rahman Chowdhury, Ahshanul Haque, Hamdy Soliman, Mohammad Sahinur Hossen, Tanjim Fatima, and Imtiaz Ahmed. 2023. Android malware Detection using Machine learning: A Review. In *Intelligent Systems Conference*. Springer, 507–522.
[5] N Daoudi, J Samhi, A K Kabore, K Allix, T F Bissyandé, and J Klein. 2021. Dexray: A simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. In *International Workshop on Deployable ML for Security Defense*.
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[7] Yuxin Ding, Xiao Zhang, Jieke Hu, and Wenting Xu. 2020. Android malware detection method based on bytecode image. *Journal of Ambient Intelligence and Humanized Computing* (2020).
[8] Hossein Fereidooni, Mauro Conti, Danfeng Yao, and Alessandro Sperduti. 2016. ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. In *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 1–5.
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[10] R Hebbar, P Papadopoulos, R Reyes, A F Danvers, A J Polsinelli, S Moseley, D Sbarra, M R Mehl, and S Narayanan. 2021. Deep multiple instance learning for foreground speech localization in ambient audio from wearable devices. *Audio, Speech, and Music Processing* (2021).
[11] TonTon Hsien-De Huang and Hung-Yu Kao. 2018. R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections. In *2018 IEEE International Conference on Big Data*.
[12] Yunjie Ji, Hao Liu, Bolei He, Xinyan Xiao, Hua Wu, and Yanhua Yu. 2020. Diversified multiple instance learning for document-level multi-aspect sentiment classification. In *EMNLP*.
[13] Ya Pan, Xiuting Ge, Chunrong Fang, and Yong Fan. 2020. A systematic literature review of android malware detection using static analysis. *IEEE Access* 8 (2020), 116363–116379.
[14] Abdurrahman Pektaş and Tankut Acarman. 2020. Deep learning for effective Android malware detection using API call graph embeddings. *Soft Computing* 24 (2020), 1027–1043.
[15] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, and Yang Xiang. 2020. A survey of android malware detection with deep neural models. *ACM Computing Surveys (CSUR)* 53, 6 (2020), 1–36.
[16] Günter Rote. 1991. Computing the minimum Hausdorff distance between two point sets on a line under translation. *Inform. Process. Lett.* 38, 3 (1991), 123–127.
[17] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.
[18] HR Sandeep. 2019. Static analysis of android malware detection using deep learning. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. IEEE, 841–845.
[19] Zhuchen Shao, Hao Bian, Yang Chen, Yifeng Wang, Jian Zhang, Xiangyang Ji, et al. 2021. Transmil: Transformer based correlated multiple instance learning for whole slide image classification. *Advances in neural information processing systems* 34 (2021), 2136–2147.
[20] K Song, L Bing, W Gao, J Lin, L Zhao, J Wang, C Sun, X Liu, and Q Zhang. 2019. Using customer service dialogues for satisfaction analysis with context-assisted multiple instance learning. In *EMNLP*.
[21] Tiezhu Sun, Kevin Allix, Kisub Kim, Xin Zhou, Dongsun Kim, David Lo, Tegawendé F Bissyandé, and Jacques Klein. 2023. Dexbert: effective, task-agnostic and fine-grained representation learning of Android bytecode. *IEEE Transactions on Software Engineering* (2023).
[22] Tiezhu Sun, Nadia Daoudi, Kevin Allix, and Tegawendé F Bissyandé. 2021. Android malware detection: looking beyond dalvik bytecode. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 34–39.
[23] Tiezhu Sun, Weiguo Pian, Nadia Daoudi, Kevin Allix, Tegawendé F Bissyandé, and Jacques Klein. 2023. LaFiCMIL: Rethinking Large File Classification from the Perspective of Correlated Multiple Instance Learning. *arXiv preprint arXiv:2308.01413* (2023).
[24] Yueming Wu, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. 2019. Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 139–150.
[25] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. 2021. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 14138–14148.
[26] Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. 2019. Droidevolver: Self-evolving android malware detection system. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 47–62.
[27] Yang Yang, Xuehui Du, Zhi Yang, and Xing Liu. 2021. Android malware detection based on structural features of the function call graph. *Electronics* 10, 2 (2021), 186.
[28] Hongrun Zhang, Yanda Meng, Yitian Zhao, Yihong Qiao, Xiaoyun Yang, Sarah E Coupland, and Yalin Zheng. 2022. Dtfd-mil: Double-tier feature distillation multiple instance learning for histopathology whole slide image classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18802–18812.
[29] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. 2019. Lookahead optimizer: k steps forward, 1 step back. *Advances in neural information processing systems* 32 (2019).
[30] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. 2020. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 757–770.