Applied AI Letters

WILEY

PERSPECTIVE OPEN ACCESS

TL-GNN: Android Malware Detection Using Transfer Learning

Ali Raza[1] | Zahid Hussain Qaisar[2] | Naeem Aslam[1] | Muhammad Faheem[3] | Muhammad Waqar Ashraf[4] | Muhammad Naman Chaudhry[1]

[1]Department of Computer Science, NFC Institute of Engineering and Technology, Multan, Pakistan | [2]Department of Computing and Emerging Technologies, Emerson University, Multan, Pakistan | [3]Department of Computing Science, School of Technology and Innovations, University of Vaasa, Vaasa, Finland | [4]Department of Computer Engineering, Bahauddin Zakariya University, Multan, Pakistan

Correspondence: Muhammad Faheem (muhammad.faheem@uwasa.fi)

Received: 4 October 2023 | **Revised:** 13 January 2024 | **Accepted:** 8 April 2024

Funding: The authors received no specific funding for this work.

Keywords: Android malware detection | deep learning | graph neural network | malware classifier | transfer learning

ABSTRACT

Malware growth has accelerated due to the widespread use of Android applications. Android smartphone attacks have increased due to the widespread use of these devices. While deep learning models offer high efficiency and accuracy, training them on large and complex datasets is computationally expensive. Hence, a method that effectively detects new malware variants at a low computational cost is required. A transfer learning method to detect Android malware is proposed in this research. Because of transferring known features from a source model that has been trained to a target model, the transfer learning approach reduces the need for new training data and minimizes the need for huge amounts of computational power. We performed many experiments on 1.2 million Android application samples for performance evaluation. In addition, we evaluated how well our framework performed in comparison with traditional deep learning and standard machine learning models. In comparison with state-of-the-art Android malware detection methods, the proposed framework offers improved classification accuracy of 98.87%, a precision of 99.55%, recall of 97.30%, $F1$-measure of 99.42%, and a quicker detection rate of 5.14 ms using the transfer learning strategy.

1 | Introduction

With the release of the first Android smartphone in September 2008, the new open-source operating system–based smartphones quickly gained popularity [1]. With an 84% market share of smartphones worldwide, the most widely used mobile operating system in the world is Android [2, 3]. In 2022, 12 new upgraded versions of Android were released. Security attacks are becoming more common due to this level of adoption and the open-source nature of Android applications [4], which significantly compromise the integrity of such applications. According to statistics, there are more than 50 million instances of potentially unwanted applications (PUAs) and malware for Android, as shown in Figure 1.

There are already over 3 million apps available on Google Play. Unfortunately, these applications contain a significant amount of harmful malware [5, 6]. Attackers attempt to obtain people's money by stealing and monitoring their data and personal information [7, 8]. Due to the open-source nature of Android-based applications, hackers are able to easily upload malicious code programs to Google Play, including Trojan horses, adware, file infestations, riskware, backdoors, spyware, and ransomware [9, 10]. It is crucial to develop efficient malware detection techniques due to the present spread and rising complexity of malware in order to address this important issue [11].

There is complexity and uncertainty with traditional malware detection techniques [12]. The extensive use of deep learning

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2024 The Authors. *Applied AI Letters* published by John Wiley & Sons Ltd.

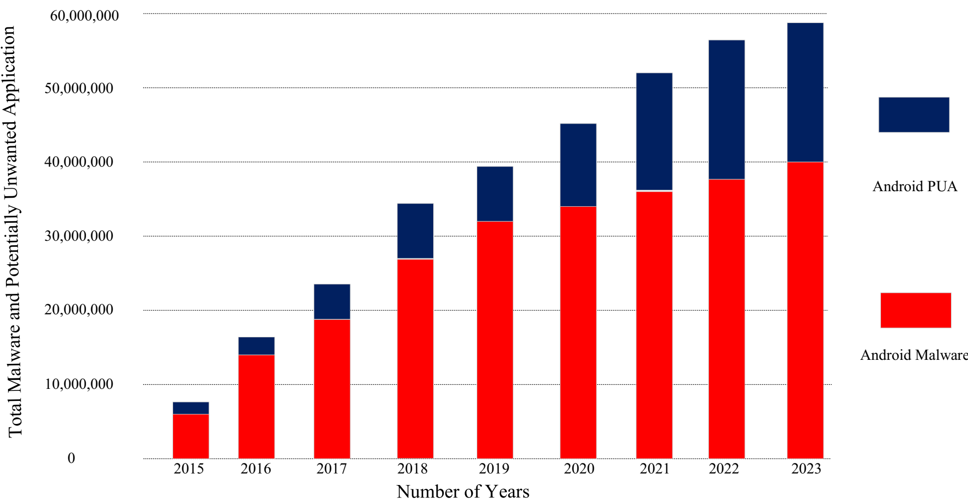

FIGURE 1 | Number of malware per year. *Source:* av-atlas.org.

and machine learning techniques in recent years [13] has significantly increased the accuracy of malware detection mechanisms, which has contributed to the development of Android malware detection utilizing these techniques [14].

An extensive amount of research has been done on methods for deep learning–based Android malware detection in response to the rapid growth of Android malware [15]. Using various types of deep and machine learning models [16, 17], researchers have put forth several methodologies and produced several research results.

Graph neural networks (GNNs) have been designed to interpret graph-structured data; they have demonstrated remarkable efficacy across a wide range of applications. Their remarkable accomplishments might be attributed to their ability to capture complex interdependencies and interactions among many components. Because of this, GNNs are especially well suited for jobs involving structured data, such as binary file representation. Neural networks used to graph data are referred to as GNNs. The nodes in a graph employed in GNN, such as a user in a social network, represent entities inside a particular issue area [18, 19]. Every node carries a feature vector and edge weights are used to quantify the relationship between nodes as shown by their edges.

To learn the representation vector of a node or the complete graph, GNNs take advantage of the graph structure and node properties. The neighborhood aggregation technique used by modern GNNs updates a node's representation iteratively by aggregating the representations of its neighbors. A node's representation captures the structural information in its network neighborhood after k iterations of aggregation [20].

In GNN models, there are two main types of operations. The first type of graph operation involves a node (commonly referred to as the center node) gathering the feature vectors of neighboring nodes. Subsequently, it performs other operations, such as calculations and reductions, and modifies its feature vector as necessary. This enables GNN to represent the data and graph structure using the updated node feature vectors. The second type involves neural operations, which can be executed individually among nodes or in center-neighbor patterns based on the graph topology. GNNs' [21] neural operations are applied in the latter scenario according to the neighborhood relationship.

Unfortunately, deep learning–based malware detection methods need a lot of labeled data points to accurately identify malicious threats [22, 23]. The size of the dataset needed to identify new malware threats is typically small, and finding new datasets takes more time [24, 25]. It takes a lot of time and resources to train deep learning models from scratch for a new dataset to identify a new malware threat [26]. Using the deep transfer learning technique is an efficient method for overcoming the challenges of model retraining and high computational complexity [27, 28]. The main approach adopted in our research to reduce computational complexity is to transfer well-known feature sets from a trained GNN model to a destination model with less training data [29–31].

In the modern world, antivirus software is no longer appropriate; instead, the Google Play Store runs a security check to stop the upload of harmful applications [32, 33]. Yet, despite the security check, the Play Store still has a large number of harmful applications [34, 35]. To counter these workarounds, numerous machine learning and deep learning techniques were developed. Most of the proposed deep learning methods require a significant amount of training time [36]. The proposed approach reduces the amount of time needed for training.

- Obfuscation techniques were used by malware developers to make it more challenging to detect their malware using conventional dynamic and static analysis approaches.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

- There are countless malware zero-days that are easy to prevent signature-based systems from detecting.
- Malware analysts are required to evaluate a significant amount of data, which takes time and might cause analysis fatigue. Malware analysis requires highly specialized knowledge.
- Research on classifying and identifying Android malware has not used transfer learning.

Protecting users from threats like ransomware, botnets, and spyware is the main objective, and deep transfer learning is being used to differentiate between benign and malicious applications.

The objectives and goals of the research are twofold:

- Employing a range of techniques to effectively identify malware from samples of both benign and malicious applications.
- To evaluate the results of various approaches and algorithms and offer recommendations for the most effective malware detection approach.

The following are the main goals of using transfer learning at the initial stage:

- **Model development time:** The time needed to develop and train a new model decreases significantly because the last few dataset-specific layers are required to be trained.
- **Knowledge utilization:** It is feasible to train a new target task using knowledge of the source model. Thus, there is no need to start from scratch while training the new target model.
- **Overfitting problems:** When conventional deep and machine learning approaches are developed on only a small dataset, overfitting problems arise. The problem is solved through transfer learning by fine-tuning the model layers.
- **Computation cost:** Using complex and hybrid datasets for training deep learning models requires a lot of computational power. The transfer learning approach can be used to reduce this high computational cost.

The following are the significant contributions of our work:

- We discuss the basic principles of transfer learning and deep learning that were used in developing the proposed approach.
- We propose TL-GNN, a new automatic malware detection approach for Android that precisely detects the malware and its type using a GNN.
- To avoid data bias, we evaluated TL-GNN on a wide range of public datasets. The results of the experiments show that TL-GNN has higher accuracy than other approaches.
- The training of the GNN model is accelerated through transfer learning. Transfer learning was used to transfer the model to the classification phase.

2 | Literature Review

In this part of the article, we discuss earlier frameworks or techniques for detecting malware. As we develop the model for malware detection, we also talk about the gaps in the literature that currently exist and how we can overcome them.

Mas'ud et al. [37] used a hybrid approach to analyze the behavior of mobile malware. A broad model of mobile malware behavior that the authors provided can help identify the key components for detecting Android malware in an Android application.

Su et al. [38] proposed a deep learning–based malware detection approach for the Android platform. They utilized static analysis approaches and reported detection accuracy of above 97%; however, continuously emerging attacks were not included in the analysis.

Wang, Liu, and Chang [39] evaluated the transferability of adversarial examples produced on a structured and sparse dataset as well as the resistance of malware detection classifiers trained using adversarial methods to adversarial examples. The decision tree, random forest (RF), SVM, CNN [40], and RNN machine learning classifiers can all be tricked by adversarial examples generated by DNN, according to the authors. They also point out how adversarial training can increase DNN's robustness in terms of resisting adversarial attacks.

Singh et al. [41] proposed a system based on machine learning to analyze Android applications. The authors exploited the APK files to gather manual features and extracted grayscale photos from a Drebin dataset file. The image processing-based algorithms are used to extract image files. The system can classify Android applications, and the authors were effective in achieving an accuracy of 93%, although overfitting issues can arise if an algorithm needs to be trained on a huge amount of data.

Pektaş and Acarman [29] uses the application programming interface (API) call graph to show all possible runtime execution paths used by malware. The API call graphs that have been converted into a low-dimensional numeric vector feature set can now be incorporated into deep neural networks. The *F*-measure, accuracy, recall, and precision metrics for the malware classification are each 98.86%, 98.65%, 98.47%, and 98.8%, respectively.

Cui et al. [1] put out a novel method for classifying and identifying malware and its variations using CNN-based deep learning classifiers. The BAT algorithm was also introduced in the paper for the goal of dataset equilibrium. Image data augmentation is also used during the training process to improve the model's effectiveness and accuracy. The model classified 9339 malware samples into 25 malware families with a 94.5% accuracy rate.

Kumar et al. [42] proposed a malware classification and detection approach based on CNN that he used to classify the malware image dataset, achieving 98% accuracy for the 25 malware family-representative 9339 samples.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 1 | Comparison among the recent related work.

Author	Models	Years	Accuracy (%)	Precision (%)	Recall (%)	$F1$ measure (%)
Zhang et al. [53]	RF	2019	96.00	97.00	95.00	96.00
Cui et al. [1]	CNN	2018	94.50	94.60	94.50	88.70
Go et al. [45]	ResneXt	2020	98.32	97.64	97.93	97.69
Smmarwar et al. [54]	OEL-AMD	2022	96.95	95.99	94.89	95.98

Kalash et al. [43] proposed a deep learning approach using CNN for classifying the malware dataset samples. There are two datasets (MalImg and Microsoft Malware challenge), and the model was implemented using these datasets. They attained 98.52% and 98.99% accuracy for both datasets, respectively.

Singh et al. [44] gathered and analyzed a dataset of 37,374 samples from 22 malware families. Additionally, he proposed a deep CNN-based classification method and obtained 98.98% accuracy when classifying the malware dataset samples.

Go et al. [45] proposed classifying malware images into specific families using a CNN-based model with three convolutional layers and a fully connected layer. Two publicly accessible datasets, Microsoft Malware and MalImg, were used to test and train the model. The model's accuracy levels are 98.48% and 97.49%, respectively.

In Visualdroid [46], security analysts must first obtain a sample of the malware, then create an appropriate signature (or make sure the sample can be correctly labeled based on existing signatures), and finally push the new signature to all antimalware tools, typically via an online update mechanism. A malware sample cannot be automatically or instantly used to generate a signature.

In StormDroid [47], dynamic and static analyses are merged. The .apk file is used to access static features, such as some API calls. Its dynamic features are derived from running log records that keep track of operating system interactions, network activity, and file system access. Opcode sequences are also considered as detection features.

Freitas et al. [48] presented MALNET, a sizeable malware for Android FCG dataset, and used new graph representation learning approaches for Android malware detection.

Xu et al. [49] presented DroidEvolver to identify Android malware that updates automatically and without user intervention. The model is updated using online learning techniques, eliminating the need for retraining and lowering the high computational cost. The authors assessed 34,722 malicious and 33,294 benign applications during a 6-year period. According to the authors, the model outperformed the state-of-the-art MamaDroid model in terms of the average $F1$-measure.

Fu and Cai [50] investigated Android malware detectors' concerns with deterioration. The authors analyzed the performance of four state-of-the-art detectors and found that the performance of the available solutions degrades over time. Additionally, the researchers proposed a new approach built on a long-term analysis of application characterization with a focus on runtime behaviors. In order to analyze the deterioration problem, a comparison between the proposed strategy and four state-of-the-art approaches was also made.

Suarez-Tangil et al. [51] proposed the DroidSieve technique for categorization based on static features [52]. The proposed approach assesses static features while using feature sets that obfuscate information. The model attained a 99.82% accuracy with no false positives and a 99.26% family categorization accuracy.

Table 1 provides details about the studies that were examined, including the authors' methodology and algorithmic choices. The papers' results are displayed in Table 1.

3 | Proposed Methodology

In this section, we discuss the dataset, implementation details, and workflow of our proposed model. In the proposed approach, the Android applications were collected from various sources, such as MALNET [48], MALNET-TINY [48], BIG [12], and Malicia [6]. The extracted sets of features are preprocessed and transformed into binary vectors after being extracted from the acquired APK files. From a category of Android applications, dynamic and static, the parameters are extracted to classify the applications into dangerous and benign apps. These apps are downloaded from publicly accessible sources, third-party app stores, as well as the official Google Play Store. Intent, version, system services, manifest permissions, strings, and components are examples of static parameters. These parameters include metadata that are kept in the application. In contrast to static parameters, dynamic parameters like logs, files, system calls, and network activities give information regarding an application's behavior and control flow. The source model is used as an input for classification and training purposes once the binary vectors have also been transformed into graphs.

The static and dynamic features are discussed below:

- **Manifest.permissions:** The manifest file, which contains every Android application, is used for providing details about resources, services, packages, strings, and so forth. The manifest.permissions file is one of the package files inside the manifest file. It is used to authorize applications. When the application is installed, this file is used to verify the rights. The ability to access SMS, location, and storage are crucial rights.
- **String value:** These string values are then applied to the text information of the resources. The Strings.xml file is often found where these files are stored. This file contains

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

details about the application, including its size, version, list of permissions, and history.

- **API call:** Through APIs, runtime function calls are made. Applications are initiated when they require interaction with system resources.
- **Intent:** An application's activities are started by intents. Intents are triggered whenever an application wants to carry out a task to offer runtime APK binding.
- **Services:** The components of Android services are responsible for the background tasks carried out while an application is open. Users do not need to get involved with services.
- **Version:** This provides information about the APK file's most recent version. Typically, when an application is upgraded, versions change.
- **Dalvik code:** These Java bytecodes were converted into executable code. In more recent Android versions, the Android Runtime (ART) ART library has replaced Dalvik code. Debugging tools are provided by the ART library to help identify application issues.
- **Component:** Android APKs are divided into components for better storage. The components store permissions, activities, intents, and resource files.
- **System call:** To access the resources of the Android operating system, applications use system calls. They serve as system-level APIs that can communicate with system files.
- **Runtime libraries:** Debugging and diagnostic tools are made available via the ART.

Ranking was performed after feature vectors were generated from the feature sets. For instance, the parameter "permission" is more important than the parameter "version." In a way similar to this, the importance of each feature parameter is ranked. By ranking the parameters, it is possible to filter out insignificant features. The significant feature sets have been converted into binary graphs once the feature sets have been filtered.

3.1 | Feature Extraction

Hybrid features analyze applications from multiple perspectives; they are the most comprehensive. In this research work, we use both static and dynamic features that are important for malware detection.

- **Static features:** Android files are analyzed for information such as requested permissions, opcode sequences, API calls, and so forth to conduct static analysis. For many optional features that are simple to extract, static detection is widely used in the field of Android malware detection.
- **Dynamic features:** To monitor the data flow during the program's execution and track the behavior of the program processing closer to the real behavior, dynamic analysis involves running the program in a sandbox environment and tracking the behavior of the program's API call sequence, system call, network traffic, and CPU data.

We utilized Check Point Software Technologies' CuckooDroid to automate the feature extraction procedure. CuckooDroid, an automated cross-platform framework for emulation and analysis, is based on the well-known Cuckoo sandbox and several other open-source projects. It provides both static and dynamic APKs inspection, including the ability to avoid certain VM-detection techniques, extract encryption keys, inspect SSL traffic, trace API calls, perform basic behavioral signatures, and many more features.

One of the key components of creating a strong, successful, and efficient machine learning model is identifying the features that have an important impact on the outcome. The computational power and time needed for a model to run and generate results increase with the number of features. We started with 714 features (from both static and dynamic analyses), but we could only take into consideration the 357 most crucial aspects for our study after performing feature selection on our data. Three goals are achieved by feature selection: improving the prediction performance of the predictors, producing faster and less costly predictors, and enhancing the understanding of the underlying mechanism that produced the data. Enhanced interpretability, superior generalization, and excellent accuracy are some of the advantages of embedded approaches, thereby obtaining a total of 357 significant features. Table 2 displays the 20/357 most significant features that we took into consideration.

3.2 | Raw Feature Data Preprocessing

Once the raw feature is obtained from the feature extraction phase, data preprocessing becomes necessary and typically involves the following stages:

- **Data cleaning:** After obtaining the original data through feature extraction, data cleaning eliminates unnecessary characteristics directly. For example, it is important to remove any unnecessary permissions that malicious and benign Android apps may have.
- **Data integration:** Data integration is the process of combining information from two or more features to enable comprehensive detection. Effective data fusion techniques are required when combining features from many sources for input to the classifier during the detection stage.
- **Data reduction:** To address problems with the high dimensionality of the Android feature vector, data reduction mostly refers to the technique of dimensionality reduction of the data involving complex algorithms.
- **Data transformation:** Transforming data from one form to another is known as data transformation. The most often used study technique is to generate graphs from the retrieved data and send them into a deep neural network.

3.3 | Generating Graph From Android Applications

The research criteria state that static features of an APK file, like the string XML files, resource files, Android Manifest.xml, and Dalvik files, can be used to efficiently visualize an APK graph.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 | Most significant features.

Android feature	Total weights
android-content-ContentValues-put	0.021404
android.permission. READ-PHONE-STATE	0.064252
android.permission.SEND-SMS	0.060294
android.permission.GET-TASKS	0.016792
getDeviceId	0.035959
android.permission.GET-ACCOUNTS	0.012937
android.intent.action. BATTERY-CHANGED	0.023639
android-util-Base64-encode	0.074646
android-app-SharedPreferencesImpl-EditorImpl-putLong	0.011981
android-util-Base64-encodeToString	0.020104
android.intent.action. BATTERY-CHANGED	0.040793
android.permission.WAKE-LOCK	0.017911
android.permission. ACCESS-WIFI-STATE	0.016962
dalvik-system-DexFile-dalvik-system-DexFile	0.023681
android-telephony-TelephonyManager-getSubscriberId	0.013778
java-net-ProxySelectorImpl-select	0.013303
android.permission. MOUNT-UNMOUNT-FILESYSTEMS	0.013049
dalvik-system-DexFile-loadDex	0.018581
getSubscriberId	0.021915
android-telephony-TelephonyManager-getDeviceId	0.012892

These files were used to extract the malware graph from the malicious APKs. By converting files into binary vectors, graphs can be produced. The components required to generate graph datasets are extracted from the dataset APK archives. The APK data have been stored in a binary array vector matrix and can be interpreted as a binary stream. The 8-bit binary files generated by disassembling the APK files are then mapped to the grayscale range of the graph. The vector array matrix generated from the binary streams is used to construct the graph, as shown in Figure 2.

The following are the steps used to generate the graph:

- **Step 1:** The files Android Manifest.XML, Resources.arsc, Classes.dex, and jar are extracted from datasets that contain APK archives.
- **Step 2:** The generated files are disassembled into 8-bit binary files. When the data in the files are transformed into binary data, binary vector streams are generated.
- **Step 3:** From the binary vector streams, an 8-bit array vector matrix is generated.
- **Step 4:** The graph is generated using the array vector matrix, which is then stored in a graph dataset.

The generated graph serves as an input for both the transfer learning and conventional GNN models. The last few layers of the transfer learning model are updated, while the first few layers are left unchanged because they provide generalized feature sets.

Certain calculations must be made to transform application files into graph representations. The complexity of the conversion process is dependent upon several variables, such as the size and complexity of the application, the selected graph representation, and the specific features extracted. During the conversion process, larger and more complex applications could have higher computation costs.

During the conversion process, we explored parallelization and optimization strategies to reduce computational costs. More effective resource usage can be achieved by dividing the workload among several processors or by improving the conversion step's algorithms.

We performed a trade-off analysis in our study between the classification performance that was obtained and the computational cost of converting applications to graphs. Although there is some overhead associated with the conversion process, the advantages in terms of increased efficiency and accuracy of classification must be evaluated against this cost.

3.4 | Proposed Work

Figure 3 shows the architecture of TL-GNN. APK graphs can be efficiently visualized using the static features of an APK file, such as the resource files, Dalvik files, Android Manifest.xml, and string XML files. Graphs can be obtained by transforming the files into binary vectors. The components required to create graph datasets are extracted from the dataset APK archives. The APK data were stored in a binary array vector matrix and have been interpreted as a binary stream. The 8-bit binary files generated by disassembling the APK files are then mapped to the grayscale range of the graph. The binary streams are converted into a vector array matrix and were then used to construct a graph.

In TL-GNN, the GNN model's first layers remain unchanged, and its last layers are modified. It is performed by altering the features in the final layers while keeping the initial layer alone in Figure 3. The final layer was changed because it utilized the majority of the dataset, while the first layer dealt with common features like the metadata, version, strings, AndroidManifest file, and various other static features. The first few layers can be frozen, which drastically reduces the amount of computational power needed to train the final few layers.

The following steps demonstrate the algorithm's workflow:

- Generating Dalvik bytecode files by decompiling the applications.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

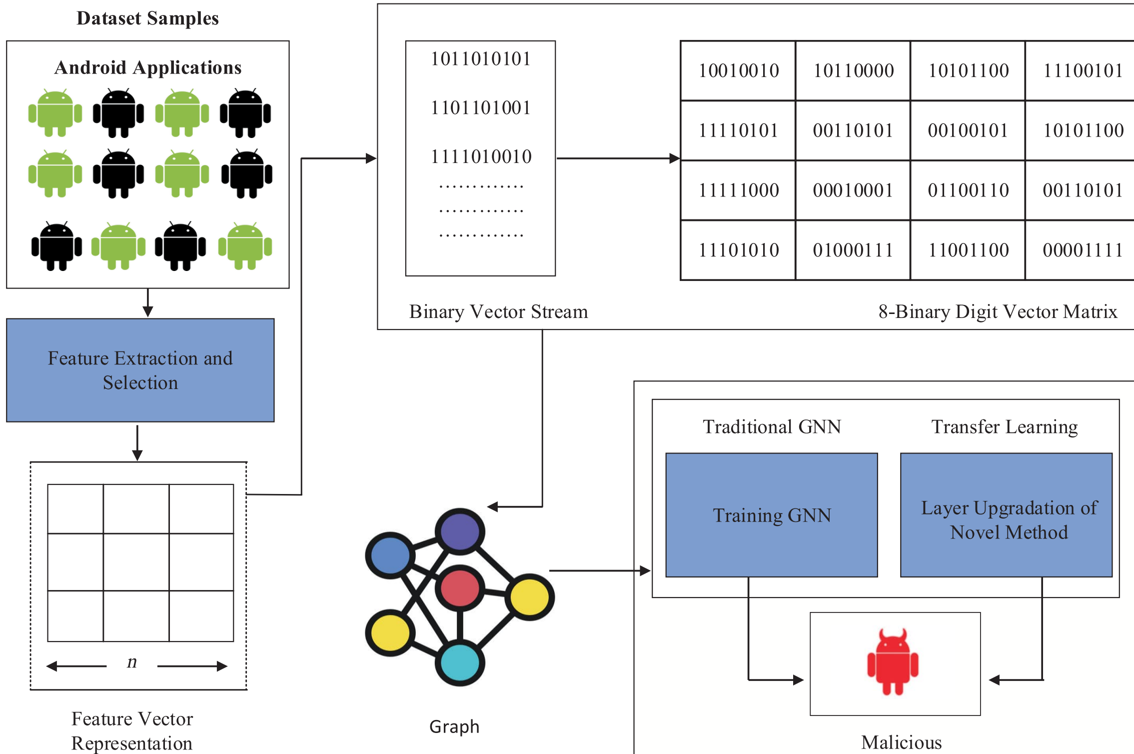

FIGURE 2 | APK to graph conversion process.

- Extracting all defined methods from each Dalvik bytecode file by scanning it, and finally constructing a node for every method.
- Building an edge between the caller and caller nodes based on the call relationship by iteratively traversing each method's call statement (such as "invoke-*") to identify the call interaction. The method invocation relationship within the entire application is represented by the approximation graph, which contains all methods defined in the Dalvik bytecode code.

3.5 | Dataset

In this section, we have discussed well-known malware datasets for model training and classification. Datasets are listed in Table 3. After converting application files into graphs, all the datasets are combined and stored in a graph dataset. From the dataset, 80% are used for training the model, while the remaining 20% are used for testing. We used the same datasets and feature sets with a GNN classifier to replicate their findings. Because the collection is composed of multiple datasets, benign applications that emerged with various behavioral patterns over time can be included. They created a more accurate and generalizable model by applying the data selection technique.

Microsoft provided the dataset [12] as part of the Malware Classification Challenge (BIG 2015) competition at the WWW2015/BIG 2015. The Kaggle platform makes the real dataset easily accessible. Microsoft provided a sizable malware dataset that was almost 0.5 gigabytes in size. The dataset includes more than 20,000 malware samples in .asm (disassembly code) and .byte (byte code) files. Bytecode files can be transformed into graphs using conversion methods. There are 10,868-byte file samples in nine families in the collection. The dataset's description is shown in Table 4. The dataset was tested using both conventional and transfer learning methods.

The Microsoft Malware Classification Challenge dataset introduces a distinct set of challenges due to the varied structure of its application files. Significantly, the structure of application files within the Microsoft Malware dataset differs substantially from standard APK files. Consequently, our methodology underwent specific adaptations to accommodate these differences. Customized preprocessing steps were implemented to ensure the accurate representation of applications as graphs, considering the unique characteristics of the Microsoft Malware dataset.

The inclusion of the Microsoft Malware Classification Challenge dataset stems from its distinctive attributes. The dataset provides a diverse set of challenges not encountered in traditional Android malware detection scenarios. Analyzing this dataset allows us to assess the adaptability of our methodology to different malware contexts.

Our decision to incorporate the Microsoft Malware dataset enables a comparative analysis, offering insights into the performance of our methodology across various malware types. This comparative approach enhances the robustness of our findings and contributes to a more comprehensive understanding of the proposed method's effectiveness.

In a hierarchy of 47 kinds and 696 families, MALNET [48] contains 1.2 million function call graphs with over 35K edges and 15K nodes, as shown in Table 5.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

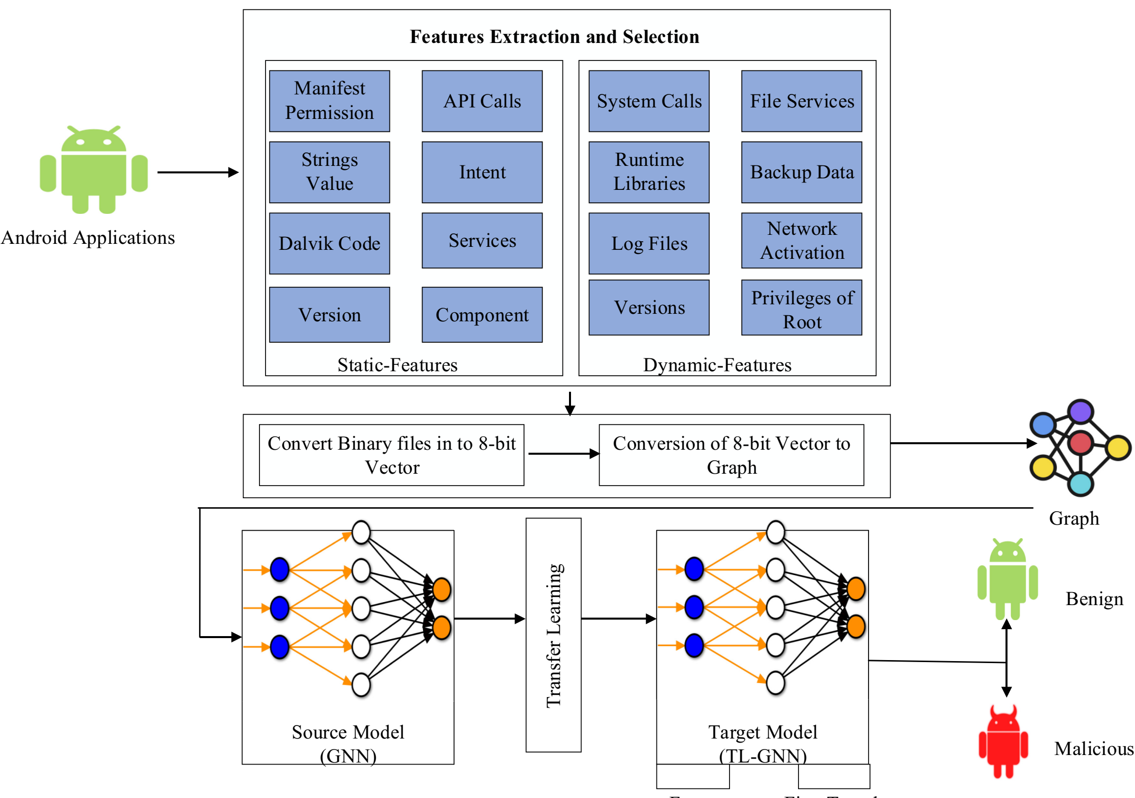

FIGURE 3 | Proposed work.

MALNET-TINY [48] has 5000 graphs in five different types. To keep the dataset truly "tiny," we also set a 5K node limit for each network. The purpose of MALNET-TINY is to enable users to quickly prototype new ideas because it takes a tiny fraction of the time to train a new model.

TABLE 3 | Datasets list.

Dataset	Type	Family	Samples
MALNET [48]	47	696	1.2 million
Malicia [6]	—	51	9339
BIG [12]	—	9	10,868
MALNET-TINY [48]	5	—	5000

Before we can apply graph-based analysis, the Malicia [6] samples must be converted from binaries into graphs. We found that 1192 samples from the Malicia dataset did not have a family label, and 581 samples from the dataset were not executable files. After eliminating these samples, we were left with 9895 binaries from 51 families in the Malicia dataset, listed in Table 6.

In Algorithm 1, a pseudo-code for a GNN is displayed. The classifiers are initially trained with relevant training data and weights in every iteration of sequential learning. The data weights will be modified for the next iteration by the classifiers' results from training. Until classifiers are trained, both operations are carried out.

TABLE 4 | Dataset for the Microsoft Malware Classification Challenge (BIG) description.

Class #	Type	Family	Samples
1	Worm	Ramnit	1013
2	Backdoor	kelihos_ver 3	2942
3	Backdoor	Gatak	1013
4	Adware	Lollipop	2478
5	Backdoor	kelihos_ver 1	398
6	Obfuscated malware	Obfuscator.ACY	1228
7	Trojan Downloader	Tracur	751
8	Backdoor	Simda	42
9	Trojan	Vundo	475

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 5 | MALNET dataset's statistical description for the nine major graph types.

Type	Family	Graph	Node				Degree				Edge			
			Mean	Min	Std	Max	Mean	Min	Std	Max	Mean	Min	Std	Max
Downloader	7	5K		37	28K	107K	46K	37	63K	321K	1.68	0.96	0.66	3.53
Addisplay	38	17K	13K	37	15K	98K	28K	37	34K	246K	1.87	0.92	0.37	4.38
Spr	46	14K	28K	12	21K	169K	67K	7	52K	369K	2.27	0.58	0.44	4.70
Trojan	441	179K	15K	5	18K	228K	34K	4	42K	530K	2.05	0.58	0.52	6.74
Benign	1	79K	3K5	5	30K	552K	79K	3	74K	2M	2.13	0.58	0.31	5.30
Spyware	19	7K	5K	12	6K	55K	11K	7	14K	121K	1.95	0.58	0.46	4.27
Adware	250	884K	14K	7	16K	221K	31K	4	38K	605K	2.21	0.50	0.36	6.24
Riskware	107	32K	12K	5	16K	173K	30K	4	39K	334K	2.16	0.50	0.56	5.42
Exploit	13	6K	24K	19	14K	102K	45K	14	30K	250K	1.88	0.74	0.33	3.34

4 | Results and Discussion

This section describes the experimental setup and evaluation metrics and compares our proposed approach results with the state-of-the-art.

4.1 | Experimental Environment

We used the Jupyter Notebook development environment to perform the research. Google Colaboratory, or "Colab" for short, was used to develop and train the models for the dataset preprocessing. Furthermore, the Anaconda IDE (Integrated Development Environment) was used. This version of the scientific computer programming languages R and Python aims to make package deployment and administration easier.

- For the preprocessing, a system with the following specifications was used: 64-bit operating system, specifically utilizing an Intel(R) Core(TM) i5 CPU and operating under Windows 11 [55,56].
- 16 GB RAM and 1 TB Solid State Disk Drive.

Colab was used for the model's development and testing because it provided free access to computing resources and the following benefits:

- It has preinstalled machine learning libraries such as PyTorch Geometric (PyG) library. PyG, built upon PyTorch.
- It allows cloud storage of work.
- For individual machine learning projects, Google Research offers specialized graphical processing units (GPUs) and tensor processing units (TPUs).

4.2 | Performance Measurement

To evaluate a classification algorithm, the confusion matrix has to be visualized, and specific performance metrics must be calculated. These will make it easier to analyze the effectiveness of different methods and compare each one's performance.

4.2.1 | Confusion Matrix

A table called the confusion matrix compared the actual class with the predicted class. It displays the number of samples in each quadrant. It aids in evaluating the model's predicted true positives, false positives, false negatives, and true negatives. This makes it easier to evaluate how effectively the model processed the classification.

The prediction matrix for the approach we propose is displayed in Figure 4. It helps determine how well the model performed the classification.

- **TP—True positive:** An effectively classified malware application.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 6 | Description of Malicia dataset.

Family	Size	Samples
cleanman	Small	32
CLUSTER:46.105.131.121	Small	20
securityshield	Large	150
CLUSTER:85.93.17.123	Small	45
zbot	Large	2167
CLUSTER:astaror	Small	24
CLUSTER:newavr	Small	29
winwebsec	Large	5852
CLUSTER:positivtkn.in.ua	Small	14
cridex	Small	74
harbot	Small	53
smarthdd	Small	68
Other (38 families)	Small	93

ALGORITHM 1 | GNN algorithm's framework.

Input: Training dataset $L = \{(e_1, f_1), \ldots, (e_N, f_N)\}$;
Output: Combination of classifiers $E_N(e)$;
Ensure:
1: **Procedure:** GNN algorithm
2: Initializing: $w_i^1 = 1/M$ for all $1 \leq i \leq M$
3: **for** $x = 1,2,3, \ldots, M$ **do**;
4: **if** $x = 1$ **then**
5: GNN classifier training at weighted sample sets $\{L, S_1\}$;
6: **else**
7: Transfer the $(x-1)$th GNN's learning parameters to the xth GNN classifier;
8: The xth GNN classifier is trained by the weighted sample set;
9: **end if**
10: Determine the predicted output category for the P classes of the xth GNN classifier $p_k x(e)$, where $k = 1, 2, \ldots, P$;
11: Calculate the xth classifier's training error, ϵ_x according to (8);
12: Based on ϵ_x, assign the classifier the weight α_x using (11);
13: Normalize the sample weight S_{x+1} and modify the sample weight S_{x-1} in accordance with $p_k^x(e)$;
14: **end for**

- **FP—False positive:** A benign application that was misclassified.
- **TN—True negative:** A benign application that was accurately classified.
- **FN—False negative:** An incorrectly classified malware application.

4.2.2 | Evaluation Matrix

Here, we calculate the following evaluation metrics along with the confusion matrix and evaluate various models using these metrics to determine which model works best.

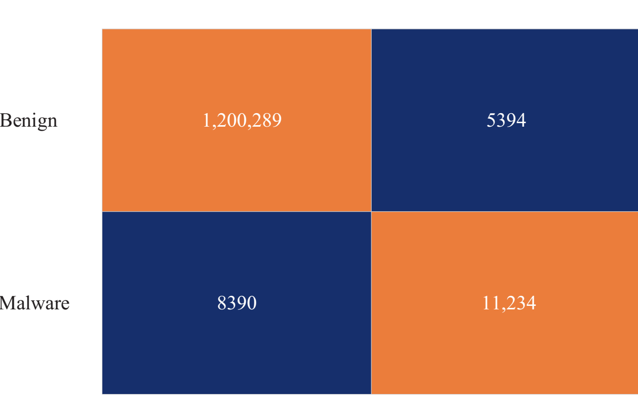

FIGURE 4 | Prediction matrix.

Accuracy: This is defined as an entire percent of the dataset's instances for which a prediction is accurate. The mathematical formula is shown in Equation (1).

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} \quad (1)$$

Precision: From all the predicted values, it is a fraction of the relevant prediction. The mathematical formula is shown in Equation (2).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

Recall: The ratio of instances that were accurately predicted to all instances. The mathematical formula is shown in Equation (3).

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

***F*-measure:** We can calculate the *F*-measure with a combination of two measurements (precision and recall). The mathematical formula is shown in Equation (4).

$$F\text{-measure} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

4.3 | Performance of Models

A comparison analysis of the results of the several experiments we performed was performed. Here, we examined the results of the experiments that were performed, as shown in Table 7.

Compared with the conventional GNN model, the transfer learning approach performs better. Table 8 presents the performance results. According to Table 8, with better accuracy, lower computational costs, and no overfitting issues, the transfer learning approach is better than other methods. Even if the entire model does not have to be trained from scratch, the transfer learning model's rate of convergence is quick.

We compared the suggested model to the baseline approach using a variety of evaluation metrics to evaluate the model's performance. In terms of precision, recall, precision, accuracy, and *F*-measure, the results of the experiment with CNN and the proposed

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 7 | Comparison among the recent related work.

Models	Technique	Accuracy (%)	Precision (%)	Recall (%)	*F*1 measure (%)	Predict time (ms)
CNN [1]	Deep learning	94.50	94.60	94.50	88.70	20.00
RF [53]	Deep learning	96.00	97.00	95.00	96.00	16.00
ResneXt [45]	Deep learning	98.32	97.64	97.93	97.69	11.19
OEL-AMD [54]	Deep learning	96.95	95.99	94.89	95.98	16.23
TL-GNN	Deep learning	98.87	99.55	97.30	99.42	5.14

TABLE 8 | Performance comparison of the two models.

Approach	Computation cost	Accuracy
Conventional GNN	High	96.20
TL-GNN	Low	98.87

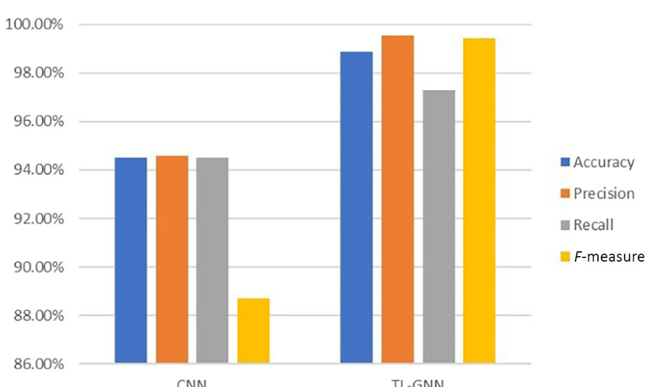

FIGURE 5 | Comparison between CNN and TL-GNN.

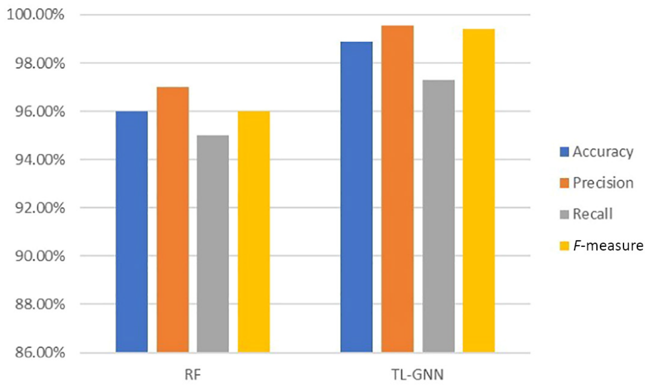

FIGURE 6 | Comparison between RF and TL-GNN.

approach are displayed in Figure 5. The graph shows the algorithm with the most accurate predicted frequency of use. This graph is generated after the algorithms have been trained on the datasets to see whether they can correctly detect the application's features.

The TL-GNN achieved a higher accuracy of 98.87%, with precision of 99.55%, recall of 97.30%, and *F*-measure of 99.42% than the CNN model, which achieved 94.50% accuracy, 94.60% precision, 94.50% recall, and 88.70% *F*-measure.

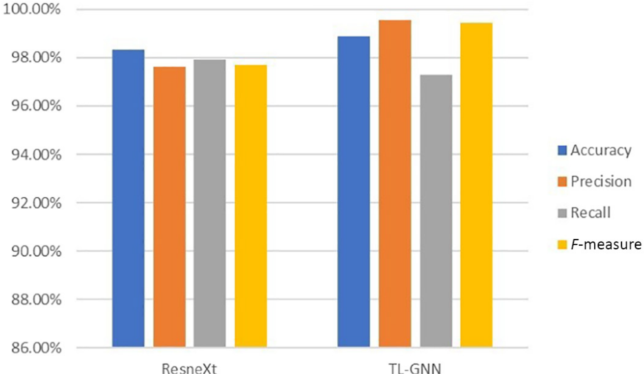

FIGURE 7 | Comparison between ResneXt and TL-GNN.

RF, a classical machine learning algorithm, was employed for comparative analysis in our study. This ensemble learning method is particularly effective in handling diverse and high-dimensional datasets, making it a suitable candidate for our graph-based feature vectors. The input for the RF algorithm was formed using feature vectors that corresponded to the graphs. These feature vectors encompass specific characteristics and properties extracted from each graph, thereby contributing to the comprehensive representation of the application. To determine how well different ways of representing features work, we conducted experiments using a variety of feature sets that were created based on the graphs. We carefully selected these sets to encompass various aspects of the application's behavior and structure. The diversity of feature vectors allows us to estimate the impact of feature selection on the performance of the RF algorithm.

Figure 6 shows the experimental results of RF and TL-GNN in terms of precision, accuracy, *F*-measure, and recall. The TL-GNN achieved a higher accuracy of 98.87% with a precision of 99.55%, a recall of 97.30%, and an *F*-measure of 99.42% than the RF model, which achieved an accuracy of 96.00%, with a precision of 97.00%, a recall of 95.00%, and an *F*-measure of 96.00%.

Figure 7 shows the experimental results of ResneXt and TL-GNN, which achieved high accuracy of 98.87% with precision of 99.55%, recall of 97.30%, and *F*-measure of 99.42% compared with the ResneXt model, which achieved accuracy of 98.32% with precision of 97.64%, recall of 97.93%, and *F*-measure of 97.69%.

Figure 8 shows the experimental results of RF and TL-GNN in terms of precision, accuracy, *F*-measure, and recall. The

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

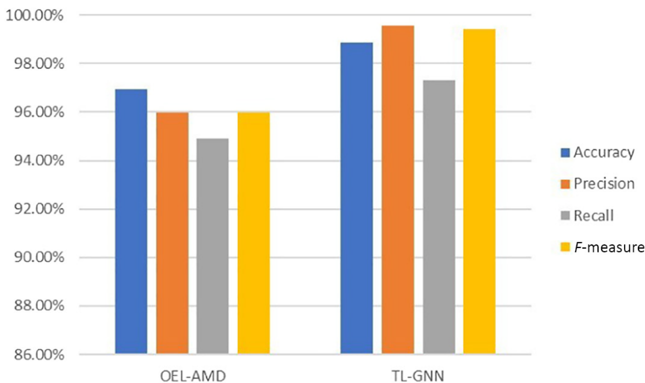

FIGURE 8 | Comparison between OEL-AMD and TL-GNN.

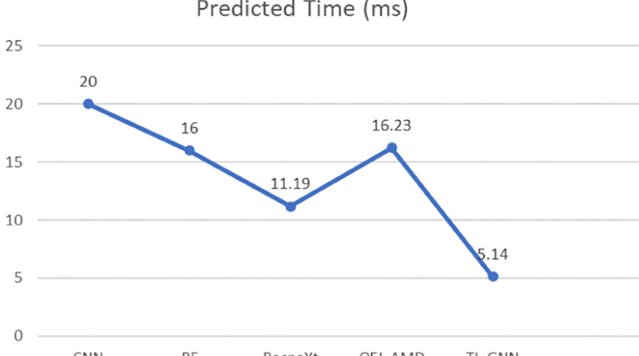

FIGURE 9 | Comparison between TL-GNN and other state-of-the-art models.

TL-GNN achieved a higher accuracy of 98.87% with the precision of 99.55%, recall of 97.30%, and *F*-measure of 99.42% than the OEL-AMD model, which achieved an accuracy of 96.96% with the precision of 95.99%, recall of 94.89%, and *F*-measure 95.98%.

Figure 9 compares the outcomes of our approach with those of the four models: CNN, RF, ResneXt, and OEL-AMD. As can be seen, our approach had a quicker detection rate of 5.14 ms. Due to the other methods' use of time-consuming, highly complex approaches, their performance was a little lower. Our method reduces the requirement for huge amounts of computation power as well as the need for new training data.

5 | Conclusion and Future Work

We conclude all of our work in this section and provide suggestions for the future.

Mobile malware has been available since the arrival of smartphones. Malware applications continued to be successful in escaping security models as Android increased in popularity. We addressed using traditional GNN and transfer learning methods to categorize and detect Android malware. A two-stage system that transforms Android applications into binary graphs was proposed. These graphs serve as the input for the conventional GNN model. We addressed the issues of complexity, overfitting, and computation cost by applying the transfer learning method to the trained model by freezing the starting layers of the pretrained model. The evaluation results show that the transfer learning strategy offers enhanced accuracy of 98.87%, precision of 99.55%, recall of 97.30%, *F*1 measure of 99.42%, and a quicker detection rate of 5.14 ms with extremely few false positives when compared with the conventional GNN model. We also compared the evaluation results with those of other approaches. It was shown that transfer learning outperforms conventional methods while also lowering computation costs.

Future research should provide us with thorough, fine-grained feature sets for enhanced outcomes. We also tried to reduce the requirement for high RAM and GPUs, as well as the issue of overfitting in the event of smaller datasets, while attempting to overcome the constraints of the proposed framework employed in our study. The most important reason for this is that, in our study, we considered both static and dynamic feature sets. Because static features lack attributes for runtime behavior, new malware strains dynamically change their behavior and form to avoid detection methods. The proposed approach is successful in detecting existing malware, but to maintain the detection approach, fresh sets of features as well as training layers must be chosen and transferred to the targeted model. The transfer learning approach has to be modified for new malware samples, in contrast to some of the earlier detectors described above, even though the training time will be reduced due to the lower computation cost. Novel malware behavior, dynamic permissions, resource obfuscation, and system call obfuscation are just a few of the factors that affect model updating. The problem of retraining the target model can be overcome by examining overall behavioral features instead of static features. Although our proposed approach offers good detection accuracy, we will go beyond those limitations in our next research to increase the detector's effectiveness. The transfer learning approach's issues with sustainability and performance deterioration will be the subject of our next phase.

Acknowledgments

The authors express their gratitude to their affiliated universities for their support in this research work.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

The data and code will be available upon request from the corresponding author.

References

1. Z. Cui, F. Xue, X. Cai, Y. Cao, G.-G. Wang, and J. Chen, “Detection of Malicious Code Variants Based on Deep Learning,” *IEEE Transactions on Industrial Informatics* 14, no. 7 (2018): 3187–3196.

2. W. Yu, L. Ge, G. Xu, and X. Fu, “Towards Neural Network Based Malware Detection on Android Mobile Devices,” in *Cybersecurity Systems for Human Cognition Augmentation* (Cham, Germany: Springer, 2014), 99–117.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

3. J. Gao, L. Li, P. Kong, T. F. Bissyandé, and J. Klein, "Understanding the Evolution of Android App Vulnerabilities," *IEEE Transactions on Reliability* 70, no. 1 (2019): 212–230.

4. X. Zhang, Y. Zhang, M. Zhong, et al., "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* (ACM Digital Library, 2020), 757–770.

5. M. AlSobeihy, S. Altamimi, E. Salem, H. Alhazzani, and E. Alhjaile, "Using Machine Learning to Classify Android Application Behavior," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (Gold Coast, Australia: IEEE, 2020), 1–4.

6. N. Bhodia, P. Prajapati, F. Di Troia, and M. Stamp, "Transfer Learning for Image-Based Malware Classification," 2019, arXiv Preprint arXiv: 190311551.

7. M. Bibi, Z. Hussain Qaisar, N. Aslam, M. Faheem, and P. Akhtar, "TL-PBot: Twitter Bot Profile Detection Using Transfer Learning Based on DNN Model," *Engineering Reports V8* (2024): e12838.

8. M. Luqman, M. Faheem, W. Y. Ramay, M. K. Saeed, and M. B. Ahmad, "Utilizing Ensemble Learning for Detecting Multi-Modal Fake News," *IEEE Access* 12 (2024): 15037–15049.

9. O. E. Kural, E. Kiliç, and C. Aksaç, "Apk2Audio4AndMal: Audio Based Malware Family Detection Framework," *IEEE Access* 11 (2023): 27527–27535.

10. K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-Based Android Malware Detection System," *Digital Investigation* 13 (2015): 1–14.

11. C. Zhang, Q. Zhou, Y. Huang, K. Tang, H. Gui, and F. Liu, "Automatic Detection of Android Malware via Hybrid Graph Neural Network," *Wireless Communications and Mobile Computing* 2022 (2022): 1–11.

12. H. Farhat and V. Rammouz, "Malware Classification Using Transfer Learning," 2021, arXiv Preprint arXiv: 210713743.

13. G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, "Call Graph and Model Checking for Fine-Grained Android Malicious Behaviour Detection," *Applied Sciences* 10, no. 22 (2020): 7975.

14. S. Ni, Q. Qian, and R. Zhang, "Malware Identification Using Visualization Images and Deep Learning," *Computers & Security* 77 (2018): 871–885.

15. R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "CayleyNets: Graph Convolutional Neural Networks With Complex Rational Spectral Filters," *IEEE Transactions on Signal Processing* 67, no. 1 (2018): 97–109.

16. A. Mahindru and A. Sangal, "MLDroid—Framework for Android Malware Detection Using Machine Learning Techniques," *Neural Computing and Applications* 33, no. 10 (2021): 5183–5240.

17. H. S. Ham and M. J. Choi, "Analysis of Android Malware Detection Performance Using Machine Learning Classifiers," in *2013 International Conference on ICT Convergence (ICTC)* (Jeju: IEEE, 2013), 490–495.

18. B. Raza, A. Aslam, A. Sher, A. K. Malik, and M. Faheem, "Autonomic Performance Prediction Framework for Data Warehouse Queries Using Lazy Learning Approach," *Applied Soft Computing* 91 (2020): 106216.

19. A. I. Kawoosa, D. Prashar, M. Faheem, N. Jha, and A. A. Khan, "Using Machine Learning Ensemble Method for Detection of Energy Theft in Smart Meters," *IET Generation, Transmission and Distribution* 17, no. 21 (2023): 4794–4809.

20. P. Xu, C. Eckert, and A. Zarras, "Detecting and Categorizing Android Malware With Graph Neural Networks," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing* (New York, NY: Association for Computing Machinery, 2021), 409–412.

21. W. Weng Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann, "Graph Neural Network-Based Android Malware Classification With Jumping Knowledge," 2022, arXiv e-Prints arXiv: 2201.07537.

22. B. Urooj, M. A. Shah, C. Maple, M. K. Abbasi, and S. Riasat, "Malware Detection: A Framework for Reverse Engineered Android Applications through Machine Learning Algorithms," *IEEE Access* 10 (2022): 89031–89050.

23. B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Towards a Fair Comparison and Realistic Evaluation Framework of Android Malware Detectors Based on Static Analysis and Machine Learning," *Computers & Security* 124 (2023): 102996.

24. M. K. Alzaylaee, S. Y. Yerima, and S. Sezer, "DL-Droid: Deep Learning Based Android Malware Detection Using Real Devices," *Computers & Security* 89 (2020): 101663.

25. I. U. Haq, T. A. Khan, and A. Akhunzada, "A Dynamic Robust DL-Based Model for Android Malware Detection," *IEEE Access* 9 (2021): 74510–74521.

26. Z. Fu, Y. Ding, and M. Godfrey, "An LSTM-Based Malware Detection Using Transfer Learning," *Journal of Cybersecurity* 3, no. 1 (2021): 11.

27. Z. H. Qaisar and R. Li, "Multimodal Information Fusion for Android Malware Detection Using Lazy Learning," *Multimedia Tools and Applications* 81, no. 9 (2022): 12077–12091.

28. V. Rammouz, "Using Transfer Learning for Malware Detection" (PhD thesis, Notre Dame University-Louaize, 2021).

29. A. Pektaş and T. Acarman, "Deep Learning for Effective Android Malware Detection Using API Call Graph Embeddings," *Soft Computing* 24 (2020): 1027–1043.

30. M. Faheem, H. Kuusniemi, B. Eltahawy, M. S. Bhutta, and B. Raza, "A Lightweight Smart Contracts Framework for Blockchain-Based Secure Communication in Smart Grid Applications," *IET Generation, Transmission and Distribution* 18, no. 3 (2024): 625–638.

31. M. Faheem, M. A. Al-Khasawneh, A. A. Khan, and S. H. H. Madni, "Cyberattack Patterns in Blockchain-Based Communication Networks for Distributed Renewable Energy Systems: A Study on Big Datasets," *Data in Brief* 53 (2024): 110212.

32. G. D'Angelo, F. Palmieri, A. Robustelli, and A. Castiglione, "Effective Classification of Android Malware Families Through Dynamic Features and Neural Networks," *Connection Science* 33, no. 3 (2021): 786–801.

33. P. Bhat, S. Behal, and K. Dutta, "A System Call-Based Android Malware Detection Approach With Homogeneous & Heterogeneous Ensemble Machine Learning," *Computers & Security* 130 (2023): 103277.

34. K. Xu, Y. Li, and R. H. Deng, "ICCdetector: ICC-Based Malware Detection on Android," *IEEE Transactions on Information Forensics and Security* 11, no. 6 (2016): 1252–1264.

35. O. Mirzaei, G. Suarez-Tangil, J. M. de Fuentes, J. Tapiador, and G. Stringhini, "AndrEnsemble: Leveraging API Ensembles to Characterize Android Malware Families," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security* (Auckland, New Zealand: ACM Digital Library, 2019), 307–314.

36. F. Mercaldo and A. Santone, "Deep Learning for Image-Based Mobile Malware Detection," *Journal of Computer Virology and Hacking Techniques* 16, no. 2 (2020): 157–171.

37. M. Z. Masud, S. Sahib, M. F. Abdollah, S. R. Selamat, R. Yusof, and R. Ahmad, "Profiling Mobile Malware Behaviour Through Hybrid Malware Analysis Approach," in *2013 9th International Conference on Information Assurance and Security (IAS)* (Gammarth, Tunisia: IEEE, 2013), 78–84.

38. X. Su, D. Zhang, W. Li, and K. Zhao, "A Deep Learning Approach to Android Malware Feature Learning and Detection," in *2016 IEEE Trustcom/BigDataSE/ISPA* (Tianjin, China: IEEE, 2016), 244–251.

39. Y. Wang, J. Liu, and X. Chang, "Assessing Transferability of Adversarial Examples Against Malware Detection Classifiers," in *Proceedings of*

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

the 16th ACM International Conference on Computing Frontiers (Alghero, Italy: ACM Digital Library, 2019), 211–214.

40. W. El-Shafai, I. Almomani, and A. AlKhayer, "Visualized Malware Multi-Classification Framework Using Fine-Tuned CNN-Based Transfer Learning Models," *Applied Sciences* 11, no. 14 (2021): 6446.

41. J. Singh, D. Thakur, T. Gera, B. Shah, T. Abuhmed, and F. Ali, "Classification and Analysis of Android Malware Images Using Feature Fusion Technique," *IEEE Access* 9 (2021): 90102–90117.

42. R. Kumar, Z. Xiaosong, R. U. Khan, I. Ahad, and J. Kumar, "Malicious Code Detection Based on Image Processing Using Deep Learning," in *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence* (Chengdu, China: ACM Digital Library, 2018), 81–85.

43. M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware Classification With Deep Convolutional Neural Networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (Paris, France: IEEE, 2018), 1–5.

44. A. Singh, A. Handa, N. Kumar, and S. K. Shukla, "Malware Classification Using Image Representation," in *International Symposium on Cyber Security Cryptography and Machine Learning* (Cham, Germany: Springer, 2019), 75–92.

45. J. H. Go, T. Jan, M. Mohanty, O. P. Patel, D. Puthal, and M. Prasad, "Visualization Approach for Malware Classification With ResNeXt," in *2020 IEEE Congress on Evolutionary Computation (CEC)* (Glasgow, UK: IEEE, 2020), 1–7.

46. R. Casolare, C. De Dominicis, F. Martinelli, F. Mercaldo, and A. Santone, "VisualDroid: Automatic Triage and Detection of Android Repackaged Applications," in *Proceedings of the 15th International Conference on Availability, Reliability and Security* (Ireland: ACM Digital Library, 2020), 1–7.

47. S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A Streaminglized Machine Learning-Based System for Detecting Android Malware," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (Xi'an, China: ACM Digital Library, 2016), 377–388.

48. S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A Large-Scale Database for Graph Representation Learning," 2020, arXiv Preprint arXiv: 201107682.

49. K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-Evolving Android Malware Detection System," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)* (Stockholm, Sweden: IEEE, 2019), 47–62.

50. X. Fu and H. Cai, "On the Deterioration of Learning-Based Malware Detectors for Android," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)* (Montreal Quebec, Canada: ACM Digital Library, 2019), 272–273.

51. G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy* (Scottsdale, AZ: ACM Digital Library, 2017), 309–320.

52. Y. Huang, X. Li, M. Qiao, et al., "Android-SEM: Generative Adversarial Network for Android Malware Semantic Enhancement Model Based on Transfer Learning," *Electronics* 11, no. 5 (2022): 672.

53. H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An Efficient Android Malware Detection System Based on Method-Level Behavioral Semantic Analysis," *IEEE Access* 7 (2019): 69246–69256.

54. S. K. Smmarwar, G. P. Gupta, S. Kumar, and P. Kumar, "An Optimized and Efficient Android Malware Detection Framework for Future Sustainable Computing," *Sustainable Energy Technologies and Assessments* 54 (2022): 102852.

55. M. W. Ashraf, S. M. Idrus, F. M. Iqbal, R. A. Butt, and M. Faheem, "Disaster-Resilient Optical Network Survivability: A Comprehensive Survey," *Photonics* 5, no. 4 (2018): 35.

56. R. A. Butt, M. W. Ashraf, M. Faheem, and S. M. Idrus. "A Survey of Dynamic Bandwidth Assignment Schemes for Tdm-based Passive Optical Network," *Journal of Optical Communications* 41, no. 3 (2020): 279–293.

26895595, 2024, 3, Downloaded from https://onlinelibrary.wiley.com/doi/10.1002/ail2.94 by University Of Vaasa, Wiley Online Library on [05/05/2026]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License