ORIGINAL ARTICLE

Journal Section

TL-GNN: Android Malware Detection Using Transfer Learning

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

[1]Department of Computer Science, NFC Institute of Engineering and Technology, Multan 60000, Pakistan.

[2]School of Technology and Innovations, University of Vaasa, 65200, Finland.

[3]Department of Computer Engineering, Bahauddin Zakariya University, Multan 60800, Pakistan.

Correspondence
Muhammad Faheem, School of Technology and Innovations, University of Vaasa, 65200, Finland.
Email: muhammad.faheem@uwasa.fi

Present address
[†]Department of Computer Science, NFC Institute of Engineering and Technology, Multan 60000, Pakistan.

Funding information
This research work was supported by the University of Vaasa and the Academy of Finland.

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 data sets 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 to traditional deep learning and standard machine learning models. In comparison to 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 by utilizing the transfer learning strategy.

KEYWORDS
Android malware detection; Malware classifier; Deep learning; Transfer learning: Graph neural network

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 percent market share of smartphones worldwide, the most widely used mobile operating system in the world is Android[[2][3]]. In 2022, around 12 new upgraded versions of Android will be 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 three 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 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],[15]].

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[16]. Using various types of deep and machine learning models, [[17],[18]] researchers have put forth a number of methodologies and produced a number of research results.

Unfortunately, deep learning-based malware detection methods need a lot of labeled data points in order to accurately identify malicious threats[[19] [20]]. The size of the data set needed to identify new malware threats is typically small, and finding new data sets takes more time [[21],[22]]. It takes a lot of time and resources to train deep learning models from scratch for a new data set in order to identify a new malware threat[23]. Using the deep transfer learning technique is one efficient method of overcoming the issues of model retraining and high computational complexity [[24],[25]]. 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 [[26],[27], [28]].

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[[29] [30]]. Yet, despite the security check, the Play Store still has a large number of harmful applications[[31],[32]]. 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[33]. 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.
- 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.

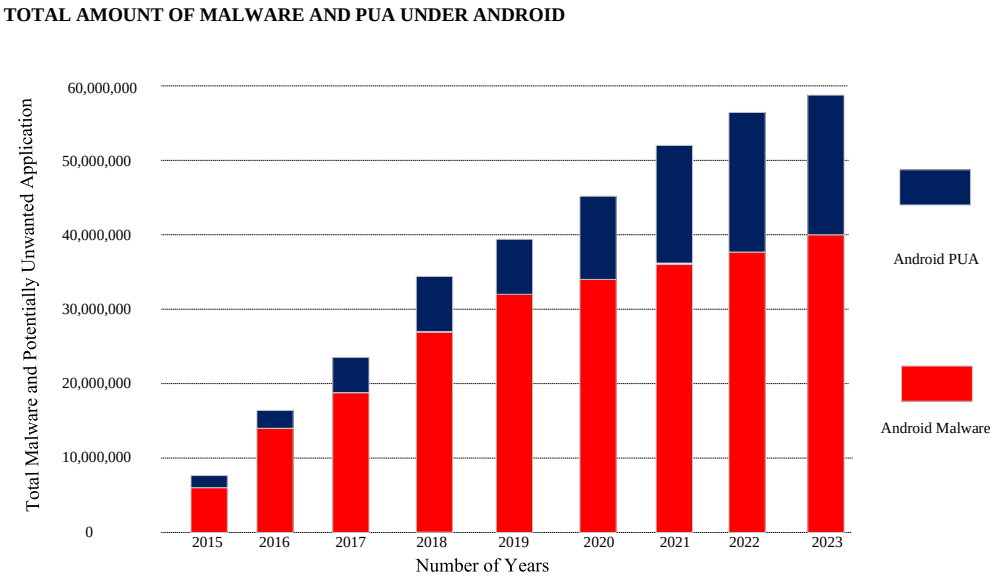

FIGURE 1 Number of Malware Per Year. (Source: av-atlas.org)

The objectives and goals of the research are two-fold:

- 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 data set-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.

Over-fitting problems: When conventional deep and machine learning approaches are developed on only a small data set, over-fitting problems arise. The problem is solved through transfer learning by fine-tuning the model layers.

Computation cost: Using complex and hybrid data sets 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 graph neural network.
- To avoid data bias, we evaluate 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 paper, 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.

Zaki et al. [34] 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. [35] 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 et al. [36] 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, SVM, CNN,[37] 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. [38] 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 data set 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.

Pektacs [26] uses the 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.

Zhihua 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 9,339 malware samples into 25 malware families with a 94.5% accuracy rate.

Kumar et al. [39] 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 9,339 samples.

Kalash et al. [40] proposed a deep learning approach using CNN for classifying the malware dataset samples. There are two datasets, and the model was implemented (MalImg and Microsoft malware challenge). They attained 98.52% and 98.99% accuracy for both datasets, respectively.

Singhet al. [41] gathered and analyzed a dataset of 37,374 samples from 22 malware families. Additionally, he proposed a deep CNN-based classification method and got 98.98% accuracy when classifying the malware dataset samples.

Gibert et al. [42] 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 [43], 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 [44], dynamic and static analysis 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 considered detection features as well.

Scott et al. [45] presented MalNet, a sizeable malware for Android FCG dataset, and used new graph representation learning approaches for Android malware detection.

Xu et al. [46] 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 six-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 [47] investigated Android malware detectors' concerns with deterioration. The authors analyze the performance of four state-of-the-art detectors and find 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. [48] proposed the DroidSieve technique for categorization based on static features [49]. 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[45], MALNET-Tiny[45], BIG [12], 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 in order 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 data that is 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 is contained in every Android application and used for providing details about resources, services, packages, strings, etc., The manifest.permissions file is one of the package files inside the manifest file. It is used to give applications authorization. When the application is installed, this file is used to verify 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 where these files are stored. This file contains details about the application, including its size, version, list of permissions, and history.
- **API Call:** Through application programming interfaces (APIs), runtime function calls are made. Applications are initiated when they require interaction with

TABLE 1 Comparison among the recent related work.

Author	Models	Years	Accuracy(%)	Precision(%)	Recall(%)	F1 Measure(%)
Zhang et al.[50]	RF	2019	96.00	97.00	95.00	96.00
Zhihua et al.[1]	CNN	2018	94.50	94.60	94.50	88.70
Go et al.[42]	ResneXt	2020	98.32	97.64	97.93	97.69
Smmarwar et al.[51]	OEL-AMD	2022	96.95	95.99	94.89	95.98

system resources.

- **Intent:** An application's activities are started by intents. Intents are triggered whenever an application wants to carry out a task for the purpose of offering runtime APK binding.
- **Services:** The components of Android services are in 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 ART (Android.runtime) 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 Android Runtime (ART).

Ranking was done after feature vectors had been 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 | 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. 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 data sets are extracted from the dataset APK archives. The APK data has been kept 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 to generate the graph:

Step-1: The files Android Manifest.XML, Resources.arsc, Classes.dex, and jar are extracted from data sets that contain APK archives.

Step-2: The generated files are disassembled into 8-bit binary files. When the data in the files is 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 data set.

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.

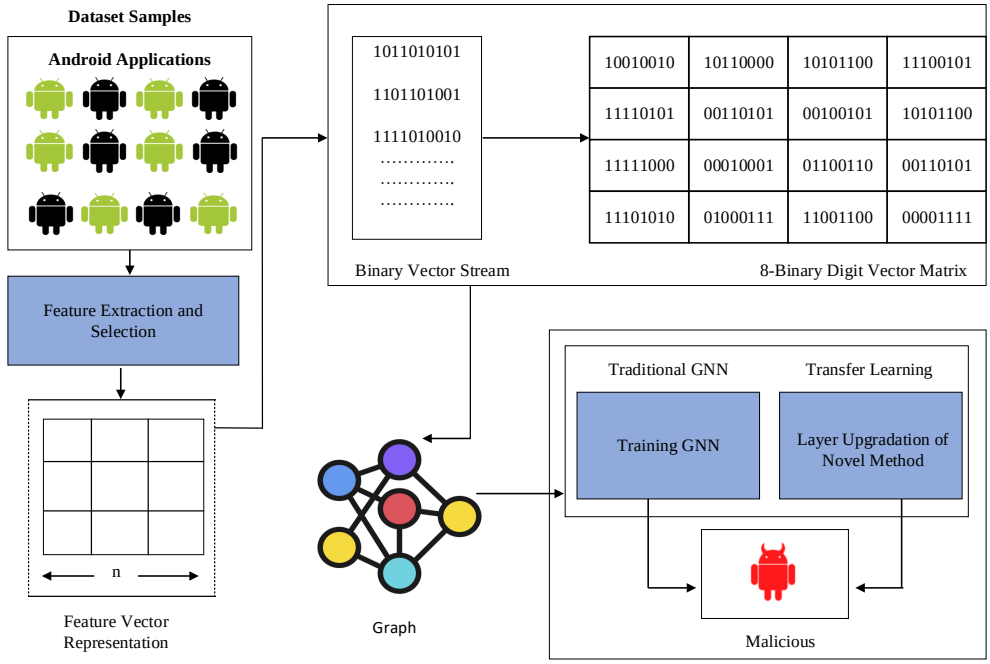

FIGURE 2 APK to Graph Conversion Process

3.2 | Proposed Work

Figure 3 shows the architecture of TL-GNN. APK graphs can be efficiently visualized by 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 data sets are extracted from the data set APK archives. The APK data has been kept in a binary array vector matrix and has 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 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.
- 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 invoca-

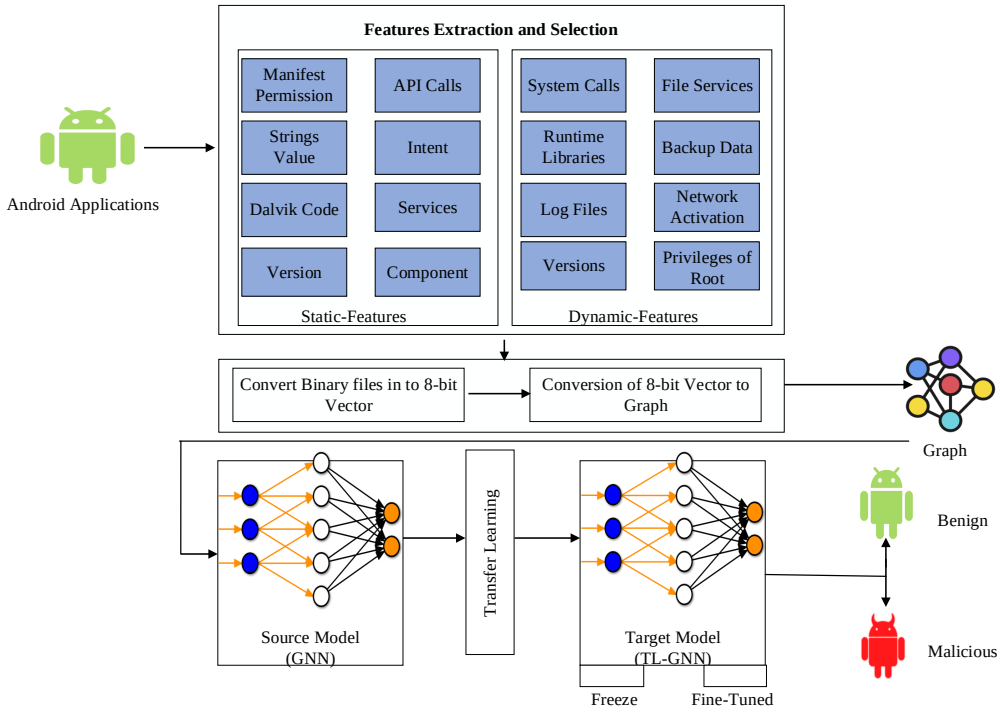

FIGURE 3 Proposed Work.

tion relationship within the entire application is represented by the approximation graph, which contains all methods defined in the Dalvik bytecode code.

3.3 | Dataset

In this section, we've discussed well-known malware datasets for model training and classification. Datasets are listed in Table 2.

TABLE 2 Datasets List.

Dataset	Type	Family	Samples
MALNET [45]	47	696	1.2 Milliion
Malicia [6]	-	51	9339
BIG [12]	-	9	10868
MALNET-Tiny [45]	5	-	5000

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 9 families in the collection. The dataset's description is in the Table 3. The dataset was tested using both conventional and transfer learning methods.

In a hierarchy of 47 kinds and 696 families, MALNET [45] contains 1.2 million function call graphs with over 35k edges and 15k nodes, as shown in Table 4.

MALNET-TINY[45], which has 5,000 graphs in 5 different types. In order to keep the dataset truly "tiny," we also set a 5K node limit for each network. The purpose

TABLE 3 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

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.

Before we can apply graph-based analysis, the Malicia[6] samples must be converted from binaries into graphs. We found that 1,192 samples from the Malicia dataset didn't have a family label, and 581 samples from the dataset were not executable files. After eliminating these samples, we were left with 9,895 binaries from 51 families in the Malicia dataset, listed in Table 5.

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 in accordance with the classifiers' results from training. Until classifiers are trained, both operations are carried out.

Algorithm 1 GNN Algorithm's Framework.

Input: Training dataset $L = \{(e_1, f_1), ..., (e_N, f_N)\}$;

Output: Combination of classifiers $E_N(e)$;

Ensure:

Procedure: GNN Algorithm

Initialising: $w_i^1 = 1/M$ for all $1 \leq i \leq M$

for $x = 1, 2, 3, ..., M$ **do**;

  if $x = 1$ **then**

    GNN classifier training at weighted sample sets $\{L, S_1\}$;

  else

    Transfer the $(x-1)^{th}$ GNN's learning parameters to the x^{th} GNN classifier;

    The x^{th} GNN classifier is trained by the weighted sample set;

  end if

  Determine the predicted output category for the P classes of the xth GNN classifier $p_k x(e)$, where k = 1, 2,..., P;

  Calculate the x^{th} classifier's training error, ε_x according to (8);

  Based on ε_x, assign the classifier the weight α_x using (11);

  Normalise the sample weight S_{x+1} and modify the sample weight S_{x-1} in accordance with $p_k^x(e)$;

end for

4 | RESULTS AND DISCUSSION

A comparison analysis of the results of the several experiments we performed was done. Here, we examine the results of the experiments that were performed, as shown in Table 6.

4.1 | Performance Measurement

To evaluate a classification algorithm, the confusion matrix has to be visualized, and specific performance met-

TABLE 4 MALNET Dataset's statistical description for the nine major graph types.

			Node				Degree				Edge			
Type	**Family**	**Graph**	mean	min	std	max	mean	min	std	max	mean	min	std	max
Downloader	7	5k	20K	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

TABLE 5 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

rics must be calculated. These will make it easier to analyze the effectiveness of different methods and compare each one's performance.

4.1.1 | 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.

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.

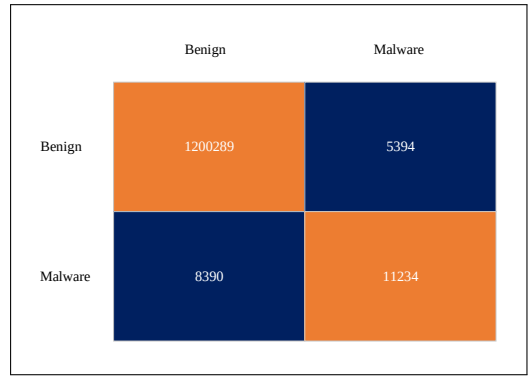

FIGURE 4 Prediction Matrix.

TABLE 6 Comparison among the recent related work.

Models	Technique	Accuracy	Precision	Recall	F1 Measure	Predict Time
		(%)	(%)	(%)	(%)	(ms)
CNN[1]	Deep Learning	94.50	94.60	94.50	88.70	20.00
RF[50]	Deep Learning	96.00	97.00	95.00	96.00	16.00
ResneXt[42]	Deep Learning	98.32	97.64	97.93	97.69	11.19
OEL-AMD[51]	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

4.1.2 | 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.

Accuracy: The entire percent of the dataset's instances for which a prediction was accurate. The mathematical formula is shown in the equation 1.

$$Accuracy = \frac{TN + TP}{TP + TN + FN + FP} \tag{1}$$

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

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall: The ratio of instances that were accurately predicted to all instances. The mathematical formula is shown in the equation 3.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

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

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

4.2 | Performance of Models

Compared to the conventional GNN model, the transfer learning approach performs better. The table 7 presents the performance results. According to table 7, with better accuracy, lower computational costs, and no overfitting issues, the transfer learning approach is better than other methods. Even if the entire model doesn't have to be trained from scratch, the transfer learning model's rate of convergence is quick.

TABLE 7 Performance Comparison of the Two Models

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

We compare 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 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 are able to 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 6 shows the experimental results of RF and TL-

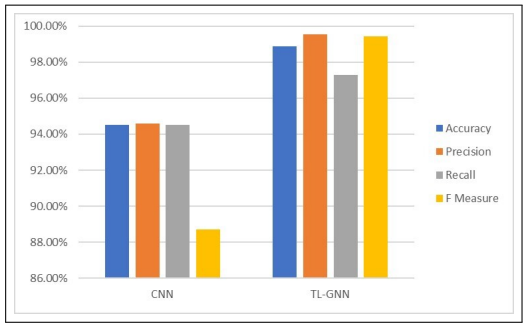

FIGURE 5 Comparison Between CNN and TL-GNN.

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%.

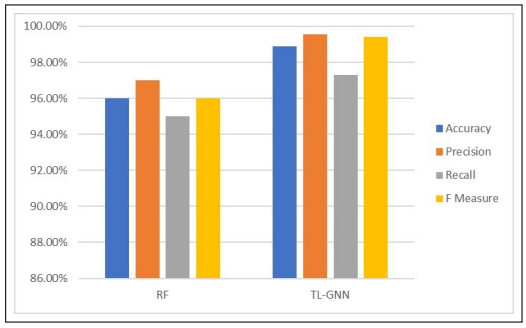

FIGURE 6 Comparison Between RF and TL-GNN.

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 to 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 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

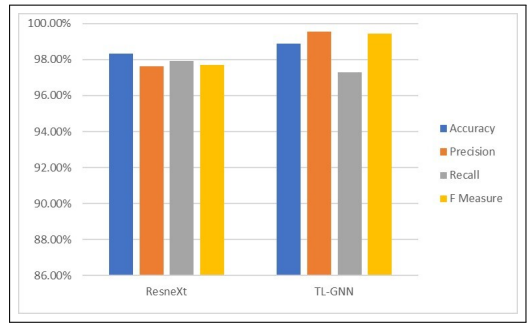

FIGURE 7 Comparison Between Resnext and TL-GNN.

achieved an accuracy of 96.96% with the precision of 95.99%, recall of 94.89%, and F-measure 95.98%.

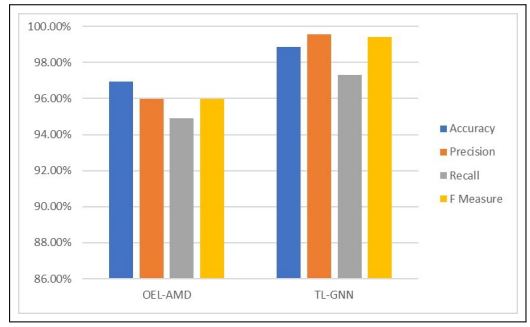

FIGURE 8 Comparison Between OEL-AMD and TL-GNN.

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.

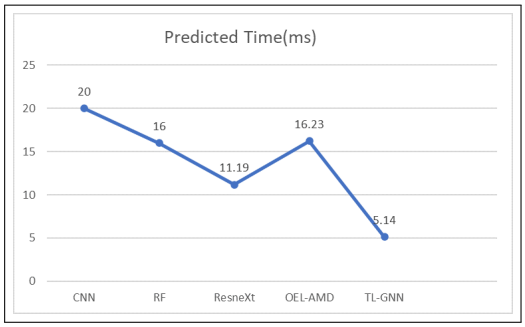

FIGURE 9 Comparison Between TL-GNN and other State-of-the-art Models.

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 pre-trained model. The evaluation results show that the transfer learning strategy offers enhanced accuracy of 98.87%, precision of 99.55%, recall of 97.30%, f1 measure of 99.42% and a quicker detection rate of 5.14 ms with extremely few false positives when compared to 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 data sets, 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're going to get 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.

6 | CONFLICT OF INTERESTS

The authors have no conflicts of interest.

7 | DATA AVAILABILITY

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

8 | CODE AVAILABILITY

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

9 | AUTHORS' CONTRIBUTIONS

All authors contributed equally to accomplishing this study. In addition, all authors read and approved the final manuscript.

10 | ETHICAL APPROVAL

Not applicable.

11 | CONSENT TO PARTICIPATE

Not applicable.

12 | CONSENT FOR PUBLICATION

Not applicable.

references

[1] Cui Z, Xue F, Cai X, Cao Y, Wang Gg, Chen J. Detection of malicious code variants based on deep learning. IEEE Transactions on Industrial Informatics 2018;14(7):3187–3196.

[2] Yu W, Ge L, Xu G, Fu X. Towards neural network based malware detection on android mobile devices. Cybersecurity systems for human cognition augmentation 2014;p. 99–117.

[3] Gao J, Li L, Kong P, Bissyandé TF, Klein J. Understanding the evolution of android app vulnerabilities. IEEE Transactions on Reliability 2019;70(1):212–230.

[4] Zhang X, Zhang Y, Zhong M, Ding D, Cao Y, Zhang Y, 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; 2020. p. 757–770.

[5] AlSobeihy M, Altamimi S, Salem E, Alhazzani H, Alhjaile E. Using Machine Learning to Classify Android Application Behavior. In: 2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE) IEEE; 2020. p. 1–4.

[6] Bhodia N, Prajapati P, Di Troia F, Stamp M. Transfer learning for image-based malware classification. arXiv preprint arXiv:190311551 2019;.

[7] Wang W, Zhao M, Gao Z, Xu G, Xian H, Li Y, et al. Constructing features for detecting android malicious applications: issues, taxonomy and directions. IEEE access 2019;7:67602–67631.

[8] Feng P, Ma J, Li T, Ma X, Xi N, Lu D. Android Malware Detection Based on Call Graph via Graph Neural Network. In: 2020 International Conference on Networking and Network Applications (NaNA) IEEE; 2020. p. 368–374.

[9] Kural OE, Kiliç E, Aksaç C. Apk2Audio4AndMal: Audio Based Malware Family Detection Framework. IEEE Access 2023;11:27527–27535.

[10] Talha KA, Alper DI, Aydin C. APK Auditor: Permission-based Android malware detection system. Digital Investigation 2015;13:1–14.

[11] Zhang C, Zhou Q, Huang Y, Tang K, Gui H, Liu F. Automatic Detection of Android Malware via Hybrid Graph Neural Network. Wireless Communications and Mobile Computing 2022;2022.

[12] Farhat H, Rammouz V. Malware classification using transfer learning. arXiv preprint arXiv:210713743 2021;.

[13] Iadarola G, Martinelli F, Mercaldo F, Santone A. Call graph and model checking for fine-grained android malicious behaviour detection. Applied Sciences 2020;10(22):7975.

[14] Saracino A, Sgandurra D, Dini G, Martinelli F. Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing 2016;15(1):83–97.

[15] Ni S, Qian Q, Zhang R. Malware identification using visualization images and deep learning. Computers & Security 2018;77:871–885.

[16] Levie R, Monti F, Bresson X, Bronstein MM. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. IEEE Transactions on Signal Processing 2018;67(1):97–109.

[17] Mahindru A, Sangal A. MLDroid—Framework for Android malware detection using machine learning techniques. Neural Computing and Applications 2021;33(10):5183–5240.

[18] Ham HS, Choi MJ. Analysis of android malware detection performance using machine learning classifiers. In: 2013 international conference on ICT Convergence (ICTC) Ieee; 2013. p. 490–495.

[19] Urooj B, Shah MA, Maple C, Abbasi MK, Riasat S. Malware detection: a framework for reverse engineered android applications through machine learning algorithms. IEEE Access 2022;10:89031–89050.

[20] Molina-Coronado B, Mori U, Mendiburu A, Miguel-Alonso J. Towards a fair comparison and realistic evaluation framework of android malware detectors based on static analysis and machine learning. Computers & Security 2023;124:102996.

[21] Alzaylaee MK, Yerima SY, Sezer S. DL-Droid: Deep learning based android malware detection using real devices. Computers & Security 2020;89:101663.

[22] Haq IU, Khan TA, Akhunzada A. A dynamic robust DL-based model for android malware detection. IEEE Access 2021;9:74510–74521.

[23] Fu Z, Ding Y, Godfrey M. An LSTM-Based Malware Detection Using Transfer Learning. Journal of Cybersecurity 2021;3(1):11.

[24] Qaisar ZH, Li R. Multimodal information fusion for android malware detection using lazy learning. Multimedia Tools and Applications 2022;81(9):12077–12091.

[25] Rammouz V. Using transfer learning for malware detection. PhD thesis, Notre Dame University-Louaize; 2021.

[26] Pektaş A, Acarman T. Deep learning for effective Android malware detection using API call graph embeddings. Soft Computing 2020;24:1027–1043.

[27] Xu P, Eckert C, Zarras A. Detecting and categorizing Android malware with graph neural networks. In: Proceedings of the 36th annual ACM symposium on applied computing; 2021. p. 409–412.

[28] Bakour K, Ünver HM, Ghanem R. The Android malware detection systems between hope and reality. SN Applied Sciences 2019;1(9):1–42.

[29] D'Angelo G, Palmieri F, Robustelli A, Castiglione A. Effective classification of android malware families through dynamic features and neural networks. Connection Science 2021;33(3):786–801.

[30] Bhat P, Behal S, Dutta K. A system call-based android malware detection approach with homogeneous & heterogeneous ensemble machine learning. Computers & Security 2023;130:103277.

[31] Xu K, Li Y, Deng RH. Iccdetector: Icc-based malware detection on android. IEEE Transactions on Information Forensics and Security 2016;11(6):1252–1264.

[32] Mirzaei O, Suarez-Tangil G, de Fuentes JM, Tapiador J, Stringhini G. Andrensemble: Leveraging api ensembles to characterize android malware families. In: Proceedings of the 2019 ACM Asia conference on computer and communications security; 2019. p. 307–314.

[33] Mercaldo F, Santone A. Deep learning for image-based mobile malware detection. Journal of Computer Virology and Hacking Techniques 2020;16(2):157–171.

[34] Mas' ud MZ, Sahib S, Abdollah MF, Selamat SR, Yusof R, Ahmad R. Profiling mobile malware behaviour through hybrid malware analysis approach. In: 2013 9th International Conference on Information Assurance and Security (IAS) IEEE; 2013. p. 78–84.

[35] Su X, Zhang D, Li W, Zhao K. A deep learning approach to android malware feature learning and detection. In: 2016 IEEE Trustcom/BigDataSE/ISPA IEEE; 2016. p. 244–251.

[36] Wang Y, Liu J, Chang X. Assessing transferability of adversarial examples against malware detection classifiers. In: Proceedings of the 16th ACM international conference on computing frontiers; 2019. p. 211–214.

[37] El-Shafai W, Almomani I, AlKhayer A. Visualized malware multi-classification framework using fine-tuned CNN-based transfer learning models. Applied Sciences 2021;11(14):6446.

[38] Singh J, Thakur D, Gera T, Shah B, Abuhmed T, Ali F. Classification and analysis of android malware images using feature fusion technique. IEEE Access 2021;9:90102–90117.

[39] Kumar R, Xiaosong Z, Khan RU, Ahad I, Kumar J. Malicious code detection based on image processing using deep learning. In: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence; 2018. p. 81–85.

[40] Kalash M, Rochan M, Mohammed N, Bruce ND, Wang Y, Iqbal F. Malware classification with deep convolutional neural networks. In: 2018 9th IFIP international conference on new technologies, mobility and security (NTMS) IEEE; 2018. p. 1–5.

[41] Singh A, Handa A, Kumar N, Shukla SK. Malware classification using image representation. In: International Symposium on Cyber Security Cryptography and Machine Learning Springer; 2019. p. 75–92.

[42] Go JH, Jan T, Mohanty M, Patel OP, Puthal D, Prasad M. Visualization approach for malware classification with resnext. In: 2020 IEEE Congress on Evolutionary Computation (CEC) IEEE; 2020. p. 1–7.

[43] Casolare R, De Dominicis C, Martinelli F, Mercaldo F, Santone A. Visualdroid: automatic triage and detection of android repackaged applications. In: Proceedings of the 15th International Conference on Availability, Reliability and Security; 2020. p. 1–7.

[44] Chen S, Xue M, Tang Z, Xu L, Zhu H. 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; 2016. p. 377–388.

[45] Freitas S, Dong Y, Neil J, Chau DH. A large-scale database for graph representation learning. arXiv preprint arXiv:201107682 2020;.

[46] Xu K, Li Y, Deng R, Chen K, Xu J. Droidevolver: Self-evolving android malware detection system. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P) IEEE; 2019. p. 47–62.

[47] Fu X, Cai H. On the deterioration of learning-based malware detectors for Android. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) IEEE; 2019. p. 272–273.

[48] Suarez-Tangil G, Dash SK, Ahmadi M, Kinder J, Giacinto G, Cavallaro L. Droidsieve: Fast and accurate classification of obfuscated android malware. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy; 2017. p. 309–320.

[49] Huang Y, Li X, Qiao M, Tang K, Zhang C, Gui H, et al. Android-SEM: Generative adversarial network for Android malware semantic enhancement model based on transfer learning. Electronics 2022;11(5):672.

[50] Zhang H, Luo S, Zhang Y, Pan L. An efficient Android malware detection system based on method-level behavioral semantic analysis. IEEE Access 2019;7:69246–69256.

[51] Smmarwar SK, Gupta GP, Kumar S, Kumar P. An optimized and efficient android malware detection framework for future sustainable computing. Sustainable Energy Technologies and Assessments 2022;54:102852.