electronics

MDPI

Article

Android-SEM: Generative Adversarial Network for Android Malware Semantic Enhancement Model Based on Transfer Learning

Yizhao Huang, Xingwei Li, Meng Qiao, Ke Tang, Chunyan Zhang, Hairen Gui, Panjie Wang and Fudong Liu *

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China; huangyizhao1996@163.com (Y.H.); Tac1t0rnX@outlook.com (X.L.); kopo.code@gmail.com (M.Q.); tuck3r@foxmail.com (K.T.); iecyzhang@163.com (C.Z.); guihairen@163.com (H.G.); wangpanjie1995@163.com (P.W.)
* Correspondence: lwfydy@126.com

Abstract: Currently, among the millions of Android applications, there exist numerous malicious programs that pose significant threats to people's security and privacy. Therefore, it is imperative to develop approaches for detecting Android malware. Recently developed malware detection methods usually rely on various features, such as application programming interface (API) sequences, images, and permissions, thereby ignoring the importance of source code and the associated comments, which are not usually included in malware. Therefore, we propose Android-SEM, which is an Android source code semantic enhancement model based on transfer learning. Our proposed model is built upon the Transformer architecture to achieve a pretraining framework for generating code comments from malware source code. The performance of the pretraining framework is optimized using a generative adversarial network. Our proposed model relies on a novel regression model-based filter to retain high-quality comments and source code for feature fusion pertinent to semantic enhancement. Creatively, and contrary to conventional methods, we incorporated a quantum support vector machine (QSVM) for classifying malicious Android code by combining quantum machine learning and classical deep learning models. The results proved that Android-SEM achieves accuracy levels of 99.55% and 99.01% for malware detection and malware categorization, respectively.

Keywords: Android malware detection; transfer learning; quantum support vector machine; semantic enhancement; generative adversarial network

Citation: Huang, Y.; Li, X.; Qiao, M.; Tang, K.; Zhang, C.; Gui, H.; Wang, P.; Liu, F. Android-SEM: Generative Adversarial Network for Android Malware Semantic Enhancement Model Based on Transfer Learning. *Electronics* **2022**, *11*, 672. https://doi.org/10.3390/electronics11050672

Academic Editor: Vijayakumar Varadarajan

Received: 28 January 2022
Accepted: 16 February 2022
Published: 22 February 2022

1. Introduction

The number of Android applications has been increasing rapidly owing to the significantly increased use of Android-based mobile devices. Currently, Google Play has over three million applications. Unfortunately, there is also a significant amount of malicious code within these applications. Android-based systems and devices have become primary targets for attackers seeking access to other people's money by stealing their personal information and monitoring their data. Additionally, because of the open-source features associated with Android-based applications, attackers can easily upload programs running malicious code to Google Play, including programs running well-known malicious code categorized as Trojan horses, adware, file infectors, riskware, backdoors, spyware, and ransomware [1,2]. Because of the current proliferation and the increasing complexity of malware, it is imperative to develop efficient malware detection mechanisms to address this crucial issue [3,4].

Security personnel use various technological approaches to detect and classify Android malware. Such approaches can be categorized as static, dynamic, and mixed forms of analyses, which are performed with the aid of machine learning [5]. Over recent years, most

studies on static feature extraction focused on extracting features, such as bytecode, sound, images, log records, code execution paths, control flow graphs, and data flow graphs. Such studies also focused on using machine learning algorithms, such as decision trees, logistic regression, Naive Bayes, support vector machines (SVM), and random forests (RF), or deep neural network (DNN)-based models, such as convolutional neural networks, generative adversarial networks, and recurrent neural networks, to detect malicious code [6]. On the other hand, current research on source code machine translation is extensive and accurate [7]. To the best of our knowledge, there are no studies on malicious code detection using machine translation to enhance the information contained in the source code of malicious software. In addition to the use of classical machine learning algorithms, research on quantum machine learning has resulted in significant breakthroughs [8]. For example, it has been shown theoretically that exponential acceleration can be achieved using a quantum support vector machine (QSVM).

Currently, little research has been conducted on the use of decompiled source code as a machine learning feature, especially in detecting malicious code. Cen et al. [9] were the first to use decompiled source code, API call sequences, and permissions as features. They applied a probabilistic discriminant model based on regularized logistic regression to detect Android malware. Akram et al. [10] proposed DroidMD, which is an extensible tool based on self-improvement that detects malicious applications in the market at the source code level. DroidMD analyzes only the application code, thereby detecting malicious code, with an accuracy of 95.5%. The studies mentioned above have achieved initial results in the field of Android malicious code detection, but the features used in these previous studies are limited to common features, and the experiments' results are not satisfactory. However, there is currently no study on transferring comment information from Android source code to the malicious code domain as program features for semantic enhancement through pretrained models. This is because it is currently impossible to obtain comment information from malicious code. Therefore, we attempt to establish novel features at the source code level to better describe program functions and extract semantic information. On the other hand, the malware comments obtained through transfer learning may not be of good quality, and as a result, they cannot be used to express program semantics. Therefore, we propose novel optimization methods using adversarial generation and filtering.

Therefore, in this study, we propose Android-SEM, an Android source code semantic enhancement model based on transfer learning. Our proposed model can be used to detect Android malware effectively. Based on the principles of natural language processing (NLP), our proposed model (Android-SEM) first extracts the semantics of Android source code, after which it generates enhanced semantic information of the source code. Android-SEM mainly comprises two parts: a transfer learning-based semantic enhancement model and a quantum classifier-based QSVM. The first part pretrains the dataset [11] corresponding to more than two million source codes through a transformer-based deep learning model, which we then improve using a generative adversarial network. The generative network attempts to create comments samples that will be incorrectly identified as human-made, and the pretraining model learns to correctly identify such samples. Our proposed model then decompiles the samples contained in the CICMalDroid2020 dataset [12] to obtain the source code and normalize it, after which it performs semantic enhancement on a pretrained transformer to obtain the comments vector. The semantic enhancement vectors are then fused with the vectors of the decompiled source code. The feature fusion vector must pass through the newly proposed regression model-based filter. The second part of our model performs quantum feature mapping on the feature vectors using QSVM to classify malicious code. Through experiments, we demonstrate that our proposed model can detect malware at an accuracy level of 99.55%. We demonstrate the effectiveness and future applications of the semantic enhancement, hybrid classical, and quantum models proposed in this study through ablation experiments.

The main contributions of this study are as follows:

- To the best of our knowledge, we are the first to use feature fusion vectors obtained from comments and source code to enhance the semantic information contained in malware. We propose Android-SEM, which is a generative adversarial pre-training model based on the transfer learning method. We effectively use the model for malicious Android code detection and classification.
- We transfer comment information that hardly exists in Android malicious code from a large number of Android source code datasets to the malicious code domain through transfer learning to enhance program-related semantic information.
- We propose a transfer learning-based filter, which can filter out bad comments without labels, and we use generative adversarial networks to improve the quality of generated comments, thereby effectively improving the classification abilities of our proposed model.
- To the best of our knowledge, we are the first to develop a model that uses QSVM combined with classical deep learning to detect malicious code in Android-based applications, and Android-SEM is more accurate than other detection models proposed in recent studies.

2. Related Work

2.1. Android Malware Detection and Classification

This study focused on static analysis, and the relevant literature is reviewed here. Zhang et al. [13] examined the relationship between entities through API calls, exceptions, permissions, and edges. They constructed a private dataset containing 322,594 samples collected between 2012 and 2018. They used RF, model pool, SVM, and DNN as the four frameworks for their experiments [14–17]. However, they extracted fewer features, and it is difficult to fully describe the program semantics, thus failing to achieve high-precision detection. Coincidentally, Idrees et al. [18] used a dataset comprising 1745 programs. An unsupervised model was used to preprocess the features and generate vectors by extracting permissions and intent features. They also extracted only a few features for model training. Currently, at least six classifiers have been used for these experiments (RF [19], Naïve Bayes [20], decision tree [21], decisionTable [22], sequential minimal optimization [23], and multilayer perceptron (MLP) [24]). Wang et al. [25] implemented an Android package kit (APK) tool for extracting features from Smail files, and they used Androguard to build an API call graph to form an adjacency matrix, after which they checked for malware using operations-related calls. Rathore et al. [26] used reinforcement learning to develop malware variants, and they proposed two attack types: single-policy attack (white box) and multi-policy attack (gray-box), for which they obtained fooling rates of 44.28% and 29.44%, respectively.

The studies mentioned above have contributed to the malicious code field from different aspects. Whether it is detecting malicious code or developing malicious code variants, good results have been achieved. However, there remains room for improvement in the accuracy of malicious code detection. Additionally, all the studies mentioned above involve the extraction of very general features, which cannot be used to further mine the information of the program itself. Therefore, the methods mentioned above cannot achieve multi-granularity program semantic extraction, cannot comprehensively extract the entire program semantic information, and cannot achieve a high accuracy rate.

2.2. Source Code Comment Generation

Source code comments are used to describe the functions and behaviors of the source code, thereby enabling programmers to understand the code. As a result, programmers can conveniently perform follow-up work on the code. However, most developers do not include comments in their code, even in software development environments that require cooperation among multiple stakeholders and rapid iterations. Additionally, there is an overall lack of comments in the source code of benign software as well as in malicious code, where the lack of comments is particularly evident [27]. Code comments are rarer in

malicious code than in benign software. Therefore, generating comments for malware is significantly challenging.

In 2019, LeClair et al. [7] combined seq2seq and the attention mechanism to develop the ast-attendgru method for generating code comments automatically [28]. They also integrated a graph convolutional neural network (GCN), based on which they proposed a method for processing abstract syntax tree (AST) information [29,30]. Their proposed solution was a graph neural network-based method that resulted in the enhanced generation of comments after transforming the AST into a graph. The machine translation was significantly enhanced because they used the transformer architecture. Ahmad et al. [31] and Wang et al. [32] also used transformer-based deep learning models to complete translation tasks, and relative to the findings of previous studies, they improved the accuracy of the generated results. By contrast, Dai et al. [33] discovered that when a transformer-based deep learning model processes long text, the text is truncated into multiple fixed-length fields. Therefore, the Transformer-XL method was developed, which is optimized for such cases and can model the relationship beyond a fixed length.

2.3. Quantum Machine Learning

The concept of combining machine learning and quantum physics involves using the superposition and entanglement of quanta to improve computing and storage capabilities. It is also used to reduce the complexities and training processing times of calculations involved in artificial intelligence. Compared with classical computing, quantum machine learning has many advantages. Because quantum states can be superimposed, quantum principles can be used to realize parallel computing without the addition of hardware and to accomplish the quadratic or exponential acceleration of solutions relative to those obtained through classical algorithms [34].

Relative to that obtained through classical SVMs, the acceleration effect of QSVMs is reflected mainly in tasks involving the determination of support vectors and the calculation of kernel function matrices [35]. For example, Park et al. [36] used quantum feature mapping to convert data into quantum states, and they constructed SVM cores from these quantum states, which were then evaluated against classical SVMs with radial basis function (RBF) cores.

As the name suggests, a quantum feature map $\phi(\mathbf{x})$ is a map from the classical feature vector $\mathbf{x}$ to the quantum state $|\Phi(\mathbf{x})\rangle\langle\Phi(\mathbf{x})|$. This is facilitated by applying the unitary operation $\mathcal{U}_{\Phi(\mathbf{x})}$ on the initial state $|0\rangle^n$, where n represents the number of qubits used for encoding. Feature maps [37] currently available in Qiskit include PauliFeatureMap, ZZFeatureMap, and ZFeatureMap. The PauliFeatureMap is defined as:

```
PauliFeatureMap(feature_dimension=None, reps=2,
        entanglement='full', paulis=None,
        data_map_func=None, parameter_prefix='x',
        insert_barriers=False)
```

In mathematics and mathematical physics, a Pauli matrix is a set of three 2 × 2 unitary Hermitian complex matrices. In quantum mechanics, they appear as a term in the Pauli equation that describes the interaction between magnetic field and spin. All Pauli matrices are Hermitian matrices, and they and the linear expansion of the identity matrix become the vector space of 2 × 2 Hermitian matrices.

A unitary operator of depth d is defined as follows:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \prod_{d} U_{\Phi(\mathbf{x})} H^{\otimes n}, U_{\Phi(\mathbf{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\mathbf{x}) \prod_{k \in S} P_i \right), \quad (1)$$

Which contains layers of Hadamard gates interleaved with entangling blocks, $U_{\Phi(\mathbf{x})}$, encoding the classical data, as shown in Figure 1 for $d = 2$.

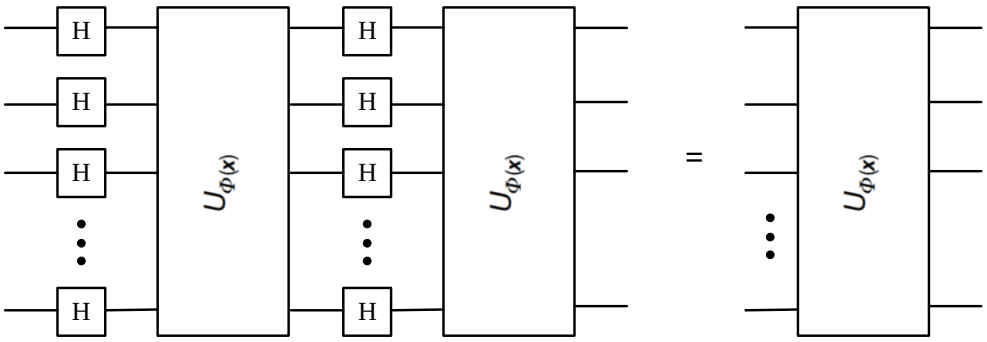


Figure 1. Circuit diagram.

Within the entangling blocks, $U_{\Phi(\mathbf{x})}$: $P_i \in \{I, X, Y, Z\}$ denotes the Pauli matrices. The index S describes connectivities between different qubits or data points, where $S \in \{\binom{n}{k}\ combinations,\ k = 1, \ldots n\}$, and by default, the data mapping function $\phi_S(\mathbf{x})$ is as follows:

$$\phi_S : \mathbf{x} \mapsto \begin{cases} x_i & \text{if } S = \{i\} \\ (\pi - x_i)(\pi - x_j) & \text{if } S = \{i, j\} \end{cases} \tag{2}$$

When $k = 2, P_0 = Z, P_1 = ZZ$, the following is the $ZZFeatureMap$:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \left(\exp\left(i \sum_{jk} \phi_{\{j,k\}}(\mathbf{x}) Z_j \otimes Z_k \right) \exp\left(i \sum_{j} \phi_{\{j\}}(\mathbf{x}) Z_j \right) H^{\otimes n} \right)^d \tag{3}$$

However, as discussed by Havlicek et al. [37], quantum kernel machine algorithms may exhibit quantum advantages over classical approaches only if the corresponding quantum kernel is difficult to estimate classically. Although necessary, the difficulty of estimating the kernel using classical resources is not always a sufficient condition for obtaining quantum advantages using quantum kernel machine algorithms. Nonetheless, it was proven recently by Liu et al. [38] that there are learning problems for which learners with access to quantum kernel methods will have a quantum advantage over learners based solely on classical algorithms.

Quantum computing has made a huge leap in processing power. Through quantum effects, such as interference or entanglement, quantum computing is expected to achieve quantum advantages, such as the exponential acceleration of computing performance, and effectively solve specific problems that are difficult for classical computers to solve. Additionally, the efficiency of conventional nuclear computing is not high. In the high case, quantum computing will grow exponentially with the size of the problem. In addition to increasing computing speed, quantum computing can improve model accuracy and model training speed. Previous studies [8] have demonstrated the exponential speedup potential of training SVMs using quantum computing, particularly using quantum techniques, to invert the kernel matrix from polynomial to logarithmic complexity. Currently, quantum computing is being gradually applied in image processing and other fields, and it has obtained significantly good results [39]. Therefore, quantum advantage is also required in the field of malicious code detection to improve and break the existing bottleneck. Although we do not currently have the hardware conditions to conduct a real experiment, we can model it using a classical computer.

3. Method

The overall architecture proposed in this study is presented in Figure 2. The architecture comprises three stages: feature engineering, a semantic enhancement model based on transfer learning, and classification. The third stage is based on classical and quantum classifiers.

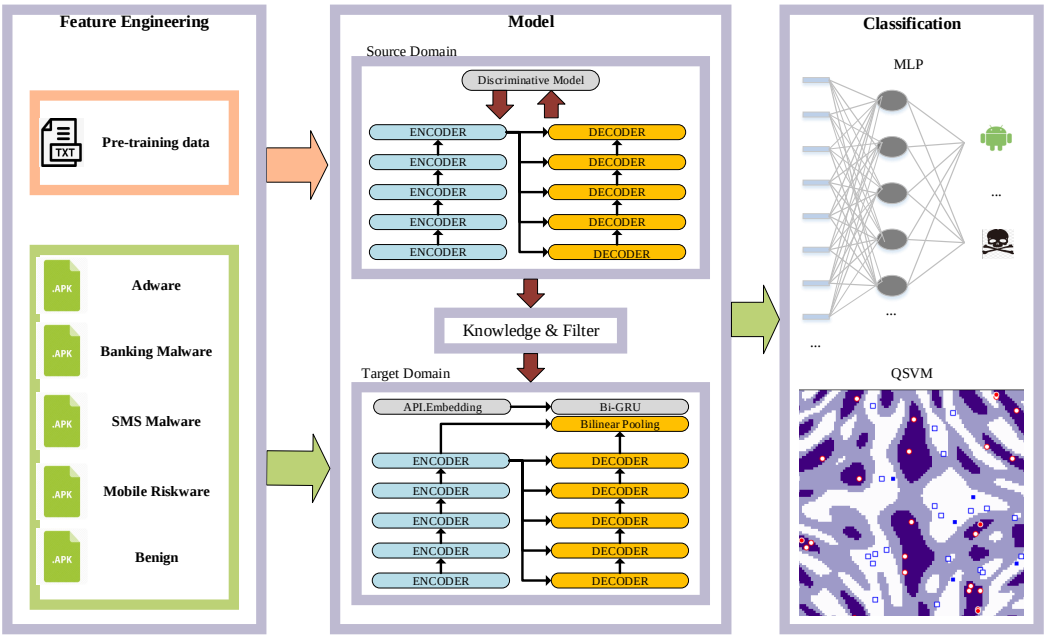


Figure 2. Model overview.

3.1. Feature Engineering

First, we obtained more than 2.1 million pairs of Java methods and one-sentence method comments from more than 28 thousand Java projects [11]. We further processed the sequence parts of the dataset, standardized logical symbols, numbers, and variables, and limited the vocabulary size. As shown in Figure 3, for the malware dataset, we obtained a total of 21,720 Android samples using CICMalDroid2020 [12] and Google Play. We implemented a multi-process batch program on the malware applications, unpacked the APKs to obtain the Dex files, and converted them into their intermediate representation (IR) format to obtain the class files, which we packaged into jar files. After reverse-cracking the applications to obtain their source codes, we extracted their functions through regular matching and subjected them to the same normalization process used on the pretraining data. Simultaneously, the Androguard tool was used to extract the API sequence, and all the information was stored in an SQLite database, which is convenient for management, postprocessing, and future use.

3.2. Model Design

Source code comments are essential in program development. From the perspective of program analysis, comments can further improve the semantic information of the code. However, determining the source code comments of malware is challenging. For this purpose, we propose a semantic enhancement model for malicious code based on migration learning, as shown in Figure 4.

According to recent NLP-related research, transfer learning models based on the Transformer architecture [40] combine two advantages: transfer learning and attention. Therefore, we used a transformer to develop our semantic enhancement model. First, we used the functions and comments in the pretraining dataset to pretrain the model. Second, we extracted methods from the malicious code dataset, performed word table mapping, and trained the fine-tuning model through word embedding and position embedding. The feature vector was obtained after passing through multiple encoders, and the matrix vector in the encoder was input into the decoder from which comment vectors were automatically generated. To perform the feature fusion of the two vectors, we used bilinear pooling [41] and obtained a high-dimensional semantically enhanced vector. Finally, we obtained a low-dimensional vector representation through a fully connected layer.

Because it is based on the transformer as an extension of the pre-trained model architecture, the $O(n^2)$ computational complexity of the Android-SEM model is also acceptable.

In the pretraining stage, we used a generative adversarial approach to optimize the generator effect. Generative adversarial networks are commonly used in the field of image processing, but we borrowed their core ideas and applied them to sequence structure processing. We treated the generator in the pretraining stage as a "fake coin maker", the comments it generated as "fake coins", and the human-labeled comments in the dataset as "real coins". In addition, we regarded the discriminator composed of the encoder and the fully connected layer as "police" and improved the quality of the comments generated by the generator through the mutual confrontation between the "police" and the "fake coin maker".

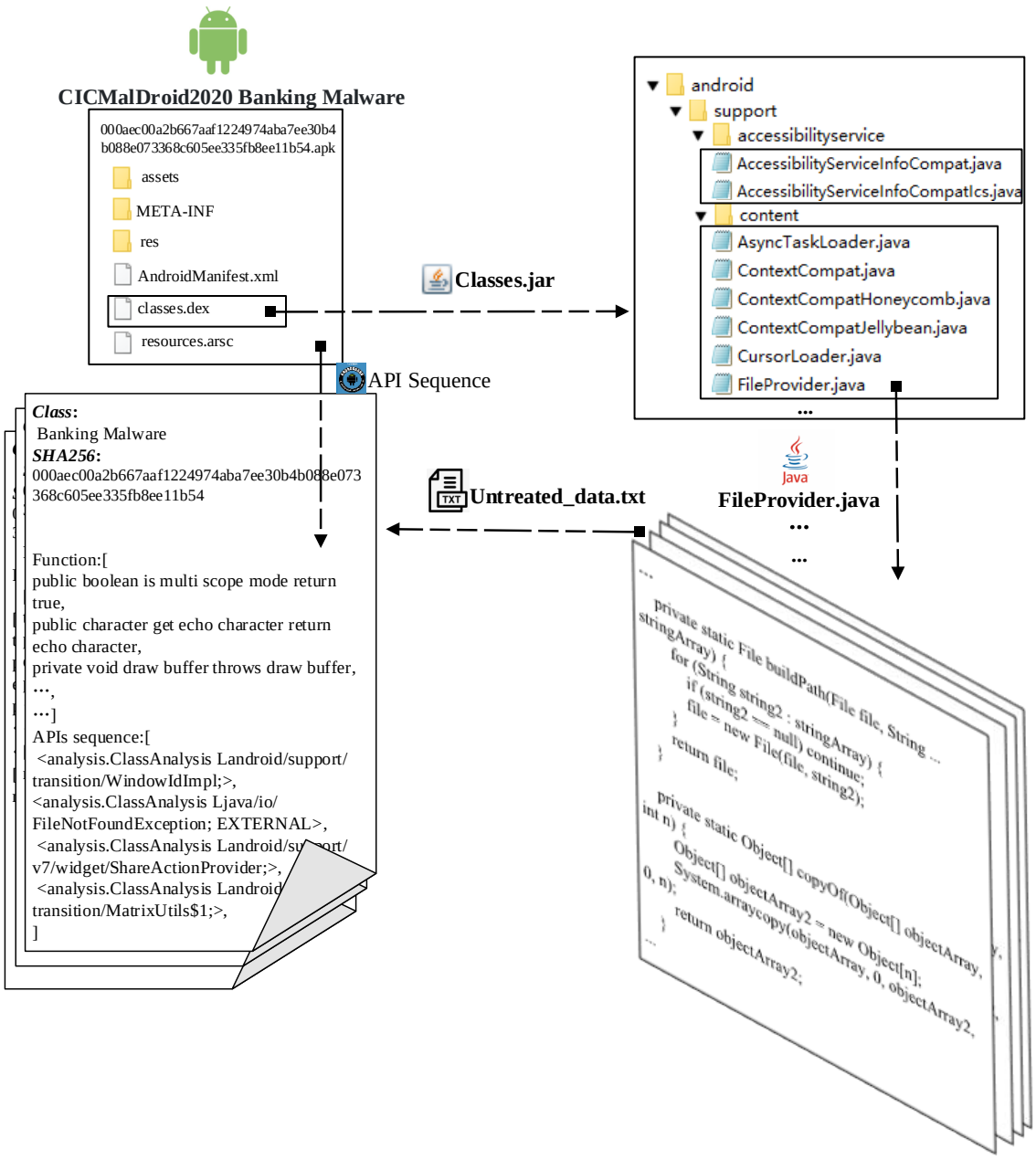


Figure 3. Extraction process for feature engineering.

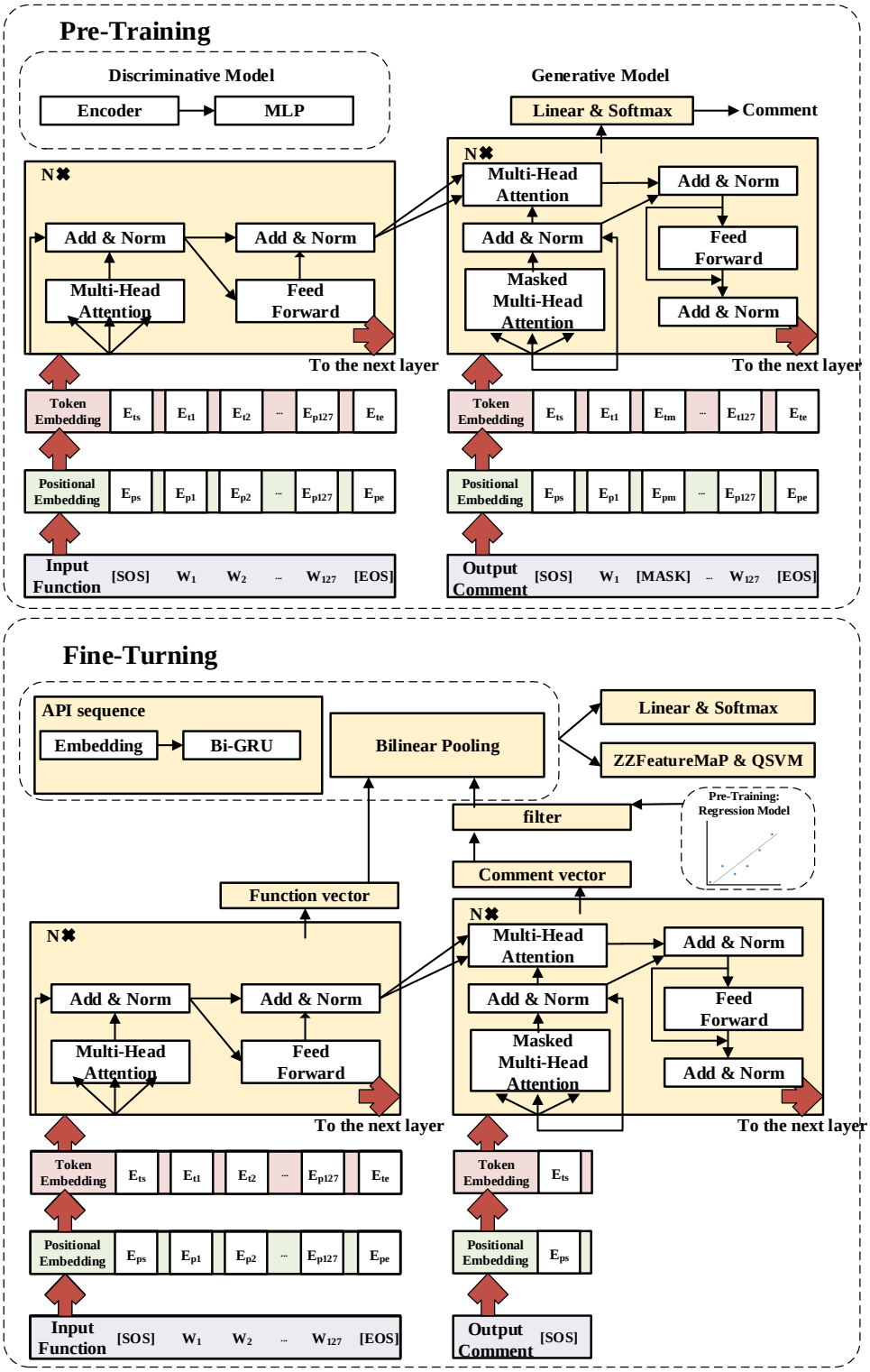


Figure 4. Overview of Android-SEM.

We propose a transfer learning-based scoring device for generated comments to avoid generating bad comment vectors that do not represent program semantics effectively. Because similar source code and comment vectors must have similar semantics, the Euclidean distance between the two is small. Therefore, during the pretraining stage, we used the function vector obtained from the encoder and the Euclidean distance between the comments vector obtained from the decoder and the BLUE value calculated from the generated comments and its label value, both of which were trained using a linear regression model.

After the model is saved, the bad comments are filtered by generating the Euclidean distance between the comments vector and the function vector obtained during the fine-tuning stage. Therefore, we could eliminate bad comments in the feature fusion stage without the need for annotated labels.

By contrast, we conducted fine-tuning training on the API sequences extracted from the malicious code dataset. We also used the concept of NLP to map the API sequences through the vocabulary, PyTorch.nn.embedding for word embedding, and the bidirectional gated recurrent unit (Bi-GRU) to learn the API semantics and context to obtain the vector representation of the API sequence. Finally, the feature fusion vector formed using the method, comments, and feature vector of the API sequence were spliced in preparation for malicious code detection and classification tasks. The tasks were performed using two methods: quantum feature mapping and MLP.

Currently, significant research has been conducted on CNN-based image processing of Android code and the subsequent analysis [42], and most studies involving sequence formats used significantly common features. In this study, we used migration learning to migrate the source code comments to the original uncommented malicious Android code. We obtained features that were not considered in previous studies through the semantic enhancement model. Follow-up experiments proved that the semantic enhancement model based on the transformer architecture can enhance malicious code classification.

3.3. Quantum Classifier

Nuclear methods have a wide range of applications in quantum machine learning [43]. In this study, we first used classical data and quantum feature mapping to encode the data into the quantum state space. The most important decision at this step involved selecting an appropriate and effective feature mapping method. The quantum feature mapping $\phi((x))$ was from the classical feature vector (x) to the quantum state $|\Phi(\mathbf{x})\rangle\langle\Phi(\mathbf{x})|$. This was facilitated by applying the unitary operation $\mathcal{U}_{\Phi(\mathbf{x})}$ in the initial state $|0\rangle^n$, where n represents the number of qubits used for encoding.

Therefore, we selected three different entangled quantum feature mapping methods to provide quantum advantages for the experiments. As mentioned in the Related work section, when $K = 2$, $P_0 = Z$, and $P_1 = ZZ$, the *ZZFeatureMap* is as follows, and its simple circuit simulation diagram is shown in Figure 5:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \left(\exp\left(i\sum_{jk}\phi_{\{j,k\}}(\mathbf{x})Z_j \otimes Z_k\right)\exp\left(i\sum_{j}\phi_{\{j\}}(\mathbf{x})Z_j\right)H^{\otimes n}\right)^d \tag{4}$$

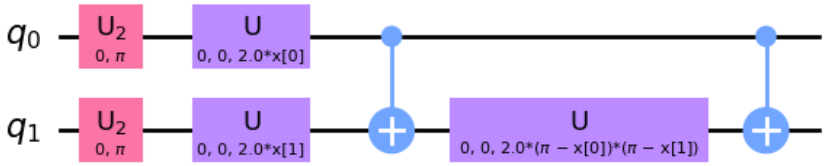


Figure 5. Schematic diagram of the analog circuit of ZZFeatureMap.

Among these, we used two different entanglement methods for *ZZFeatureMap*: *circular* and *linear*. On the other hand, we also customized the Pauli Gate in the feature map, similar to the method mentioned in the Related work section, such that when $P_0 = X$, $P_1 = Y$, and $P_2 = ZZ$, the *PauliFeatureMap* is as follows, and its simple circuit simulation diagram is shown in Figure 6:

$$v_{\Phi(\mathbf{x})} = \left(\exp\left(i\sum_{jk}\phi_{(j,k)}(\mathbf{x})Z_j \otimes Z_k\right)\exp\left(i\sum_{j}\phi_{[j]}(\mathbf{x})Y_j\right)\exp\left(i\sum_{j}\phi_{[j\}}(\mathbf{x})X_j\right)H^{\otimes n}\right)^d \tag{5}$$

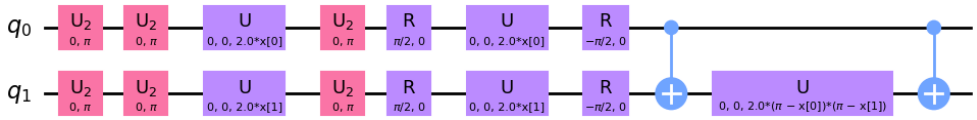


Figure 6. Schematic diagram of the analog circuit of PauliFeatureMap.

Through any one of the three aforementioned types of quantum feature mapping $\phi((x))$, a quantum nucleus, $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_j)^{\dagger}\phi(\mathbf{x}_i)$, can be naturally generated. This quantum core can be used as a measure of similarity. When the value of $k(\mathbf{x}_i, \mathbf{x}_j)$ was larger, $\mathbf{x}_i$ and $\mathbf{x}_j$ were closer. We represented the quantum nucleus as a matrix, $K_{ij} = \left|\langle\phi^{\dagger}(\mathbf{x}_j)|\phi(\mathbf{x}_i)\rangle\right|^2$, and we calculated each element in the kernel matrix by calculating the transition amplitude as follows:

$$\left|\left\langle\phi^{\dagger}(\mathbf{x}_j) \mid \phi(\mathbf{x}_i)\right\rangle\right|^2 = \left|\left\langle 0^{\otimes n}\middle|\mathbf{U}_{\phi}^{\dagger}(\mathbf{x}_j)\mathbf{U}_{\phi}(\mathbf{x_i})\middle|0^{\otimes n}\right\rangle\right|^2 \tag{6}$$

Next, classification was performed using the SVM. Figure 7 below represents a schematic diagram of the analog circuit. Owing to the limitations of the existing hardware, we could only perform classical simulation. Therefore, the acceleration may not be reflected accordingly. In the follow-up experiments, we compared the accuracy of the feature maps of different encoding methods to those of the classical SVM, thereby proving the effectiveness of QSVM in detecting malicious Android code.

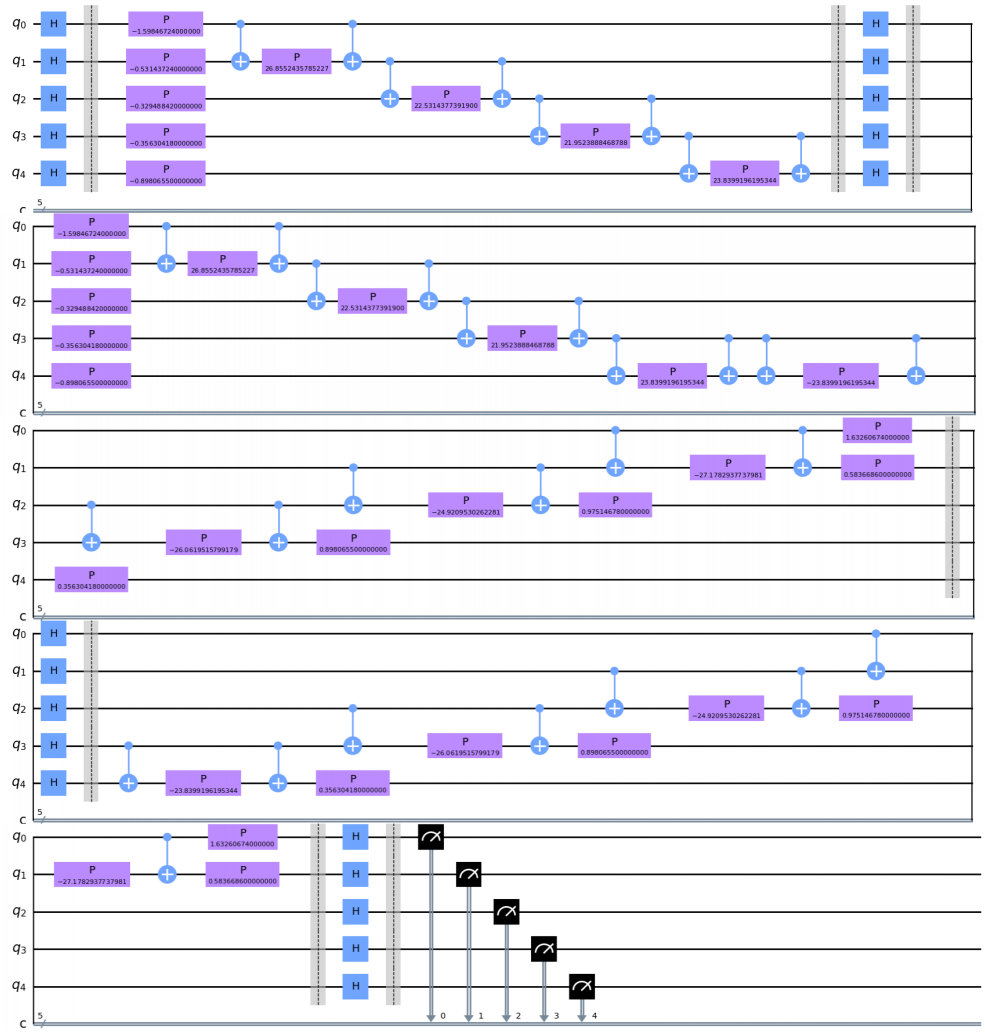


Figure 7. Schematic diagram of analog circuit.

4. Evaluation

4.1. Datasets

Our experimental dataset included data from CICMalDroid2020 and Google Play. The details are presented in Figure 8.

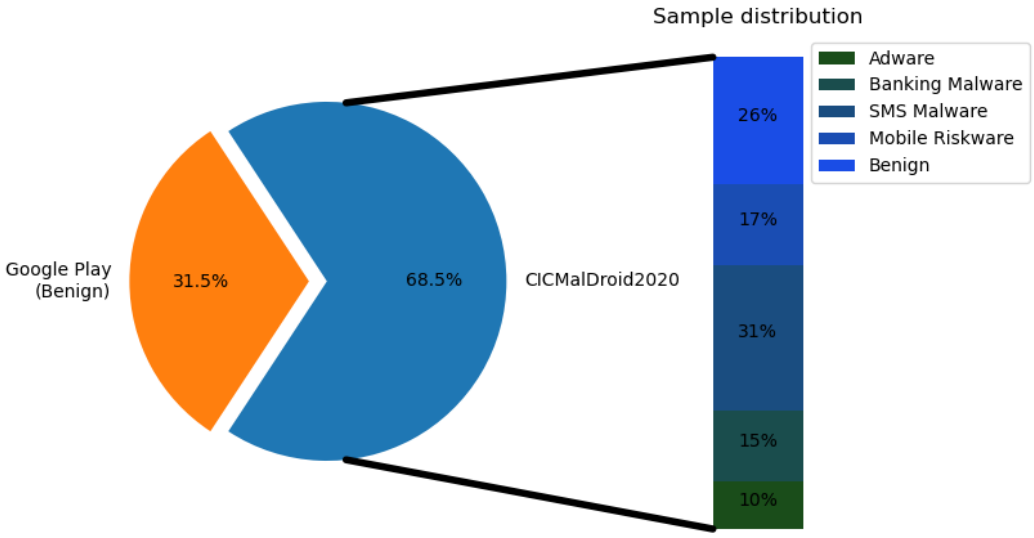


Figure 8. Dataset distribution.

First, we obtained more than 16,000 APK files of five types from the CICMalDroid2020 dataset. After complete feature engineering and extraction, 14,873 samples comprising 1500 types of adware, 2204 types of banking malware, 4622 types of SMS malware, 2534 types of riskware, and 4013 types of benign software were obtained. We used these samples as the dataset for classifying Android malware. To balance the number of benign and malicious samples, we obtained 6847 types of benign software from Google Play. To further ensure the effectiveness of the experiment, we obtained software from multiple categories, including business, games, and education, to prevent the emergence of a single type of benign software. We balanced the numbers of benign and malicious software samples in preparation for the malware detection tasks.

4.2. Experimental Setup

This section describes the implementation rules and evaluation indicators applied in this study. The model used in the experiment utilized the dataset introduced in Section 4.1. This dataset was divided into a training set and a test set based on a ratio of 7:3. In addition, we evaluated the performance of the model in terms of accuracy, precision, and recall. Because the task we needed to solve was more sensitive to false positives than negatives, we used F1 as an evaluation indicator.

The calculation method is shown in Table 1, where true positive TP_i denotes the number of c_i samples correctly classified as such. False positive FP_i denotes the number of c_i samples incorrectly classified as non-c_i samples. False negative FN_i represents the number of non-c_i samples incorrectly classified as c_i samples. True negative TN_i represents the number of non-c_i samples correctly classified as non-c_i samples. Table 2 shows the starting values of the configuration parameter set for the Android-SEM model.

Table 1. Performance Metrics.

Metric	Formula	Metric	Formula
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$	Recall	$\frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c}(TP_i+FN_i)}$
Precision	$\frac{\sum_{i=1}^{c} TP_i}{\sum_{i=1}^{c}(TP_i+FP_i)}$	F1	$\frac{2\times \text{ Precision } \times \text{ Recall}}{\text{Precision+Recall}}$

Table 2. Parameter configuration of the model.

Parameter	Value	Parameter	Value
Batch Size	64	Optimization Algorithm	Adam
Epoch	60	Word embedding size	512
Learning	Dynamic change	Full connection layer dimension	1024
dropout	0.1	Multiple attention number	8
Loss Function	CrossEntropyLoss	decoder(encoder) layers	4

4.3. Implementation

In this section, we discuss the experiment based on several questions:

1. Is the semantic enhancement model based on transfer learning effective in solving the problem of malicious code classification?
2. Regarding the semantic enhancement model proposed in this study, how do we prove that it results in significant benefits and improvements for solving the problem of malicious code detection and classification?
3. Under different feature mapping methods, can QSVM complete the task of malicious code detection and classification like SVM?

4.3.1. Model Classification Capabilities

We validated the model against the dataset described in the previous section. We also compared our proposed method with existing methods, specifically those proposed by Haq et al. [44] and Sihag et al. [45]. Their datasets were significantly similar to ours and are publicly available. Therefore, it is suitable to compare their findings to the results of our newly proposed model.

To detect malicious behavior, we used 6847 benign files from Google Play and 6847 malicious files from CICMalDroid2020. Table 3 and Figure 9 show the precision, recall, accuracy, F1, and ROC/AUC values of the model based on the dataset. For the binary classification problem, the model proposed in this study achieved an F1 score of 0.9943, and the AUC value was close to 0.9996. We also performed a multi-classification task of malicious files for malware on a separate CICMalDroid2020 dataset. In the five categories, the AUC values of each category were very close. The AUC value of Riskware is the lowest at 0.9992, whereas the AUC value of Adware is the highest at 0.9999, which is close to 1. The semantic enhancement model proposed in this study achieved high-precision classification. The aforementioned results show that the semantic enhancement model based on transfer learning is effective for Android malware detection and classification tasks.

For comparison, Sihag et al. [45] used a CNN engine to determine the maliciousness and type of malware by capturing network data packets to generate images, and they achieved accuracy levels of 99.12% and 98.91%, respectively. By contrast, Haq et al. [44] used a method for generating images based on entire APK files and applied a CNN model based on transfer learning for training, obtaining an accuracy level of 96.4%. However, the former used images to express only network interactions (streams and sessions generated through packet inspection) and did not involve other crucial information, such as system calls, APIs, or permissions. Therefore, there remain opportunities to improve the accuracy. Meanwhile, the latter generated grayscale images directly from entire APK files. Although grayscale

images are often used for malicious code detection, this method is evidently insufficient for the intended purpose, which was also reflected in the achieved accuracy level.

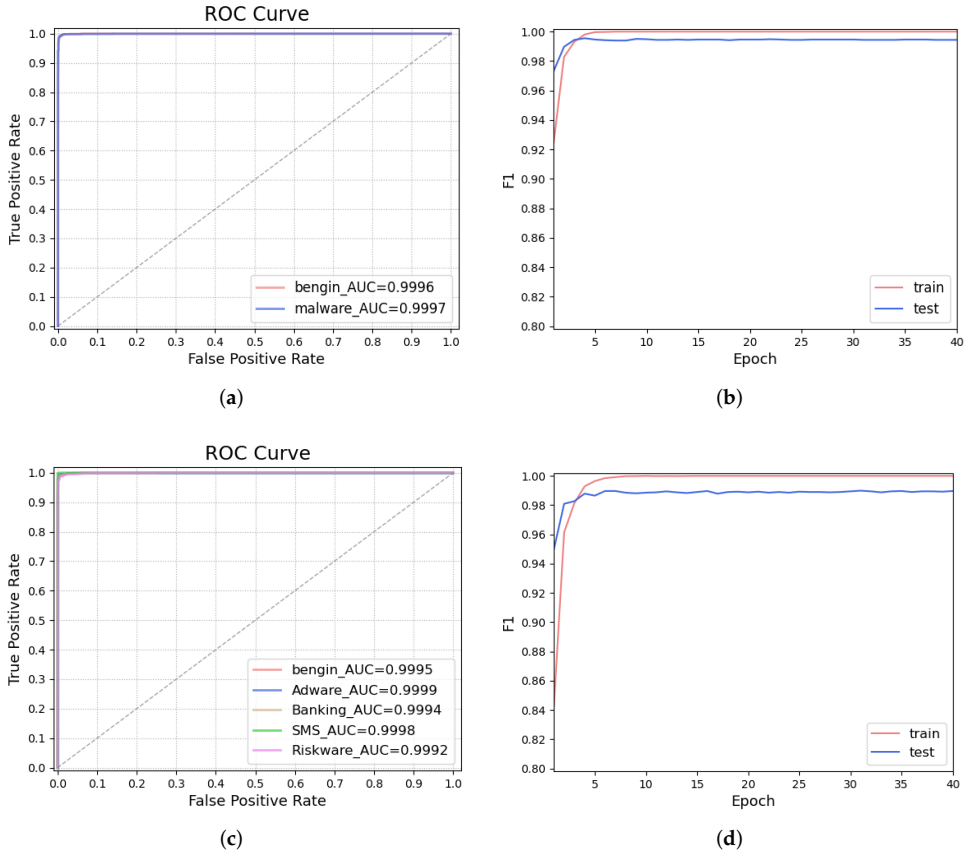


Figure 9. (**a**,**c**) ROC/AUC diagrams for detection and classification tasks; (**b**,**d**) F1 scores for detection and classification tasks.

Table 3. Malware detection and classification results

Classification Type	Work	Precision	Recall	F1	Accuracy
2 class	Vikas Sihag	98.97	96.59	97.76	99.12
	Our	99.50	99.35	99.43	99.55
5 class	Haq	-	-	-	96.4
	Vikas Sihag	98.51	98.46	98.49	98.91
	Our	98.90	98.78	98.84	99.01

In contrast, this study does not rely on analyzing images. Instead, it features the first semantic enhancement model that uses source code and transfer learning to generate comments automatically. Large-scale data were used for pretraining through a transformer-based deep learning model, and comments were generated from the source code to enhance program semantics. After the vector generated by the encoder and decoder parts during the pretraining process was filtered, feature fusion was performed through bilinear pooling. Finally, the context relationship was obtained through Bi-GRU training using the API sequence. This study is the first to use a pretraining model to generate comments automatically, which were then used as semantic enhancement features after feature fusion. It is also the first to use a generative adversarial network to improve the quality of generated semantics and migration learning to significantly save training time and achieve significantly high levels of accuracy.

4.3.2. Model Comparison Experiment

To prove the contribution of the proposed semantic enhancement model to the accuracy of malicious Android code detection, we conducted experiments to verify the mutual combination of different modules in the training model. The results are shown in Table 4, proving the necessity and effectiveness of the proposed model's complexity.

First, it is worth noting that, as shown in Table 4, the semantic enhancement model based on migration learning achieved the best performance after feature fusion through filters and bilinear pooling, and it achieved accuracy levels of 99.55% and 99.01% in two and five categories, respectively. Second, the method of fusing features using bilinear pooling was replaced with simple splicing, which resulted in an approximately 0.2% lower accuracy than that of the former. We believe that the semantic information obtained only via simple feature vector splicing may not be as rich as that obtained via mapping the features in two different vector spaces into the same dimension after mapping from high latitudes. The inclusion of a filter resulted in an improvement of approximately 5% owing to the removal of the bad comment vectors. This proves the significant effect of the filter proposed in this study.

Table 4. Model ablation experiments.

Types	Model	Precision	Recall	F1	Accuracy
2 class	All modules have	99.50	99.35	99.43	99.55
	Not included Bilinear pooling	99.32	99.21	99.26	99.42
	Not included filter	94.75	94.69	94.71	94.87
	Not included Decoder	92.79	92.54	92.66	93.28
	Embedding and LSTM	87.26	87.43	87.35	87.97
5 class	All modules have	98.90	98.78	98.84	99.01
	Not included Bilinear pooling	98.69	98.66	98.67	98.85
	Not included filter	94.01	94.18	94.01	94.21
	Not included Decoder	92.57	91.96	92.26	92.78
	Embedding and LSTM	87.25	87.14	87.20	87.34

We removed the semantic enhancement module of the part that generates source code comments, and we retained only the pretrained encoder for position information, word embedding, and the multi-head attention mechanism in the encoder. Simultaneously, the semantic vector obtained using the API sequence through word embedding and Bi-GRU was retained. The two were vectorized and classified. The accuracy obtained in this experiment was approximately 2% lower than that obtained in the previous experiment. However, compared with the enhanced semantic module, the difference was 6%. This shows that the proposed semantic enhancement model can obtain more semantic information and improve classification accuracy. Finally, when we eliminated the entire pretraining module and used word embedding and a long short-term memory (LSTM) model for semantic extraction and feature classification, the accuracy dropped by approximately 5% compared with that of the former. This shows that the transformer-based pretraining model can obtain additional semantic information from a large Android program source code corpus in advance to complete the malicious code detection task, achieve a satisfactory classification result, and significantly reduce training time.

4.3.3. Quantum Classifiers with Different Encoding Methods

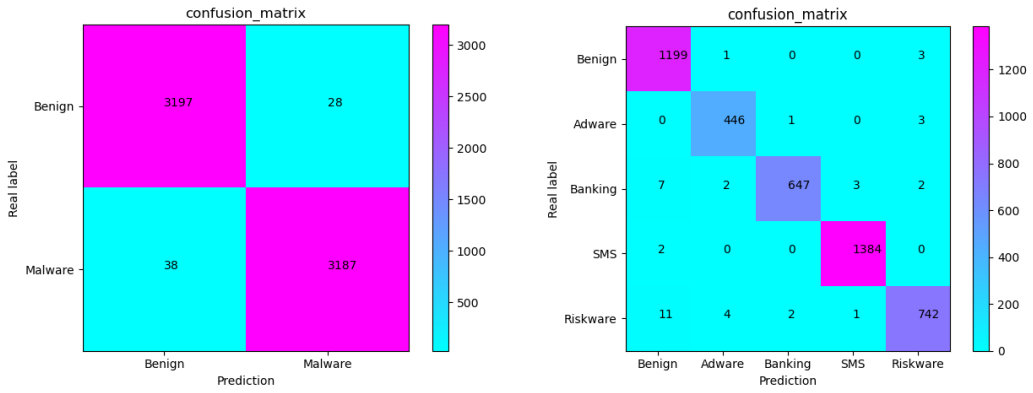
To prove that QSVM was comparable to classical SVMs in detecting and classifying malicious Android code, or better in certain circumstances, the hyperparameters were varied, and the classification performance and decision boundary were observed to select the best data fitting method and make slight adjustments. The objective was for the SVM kernel to produce better classification results than those of the QSVM.

Subsequently, we constructed the classification accuracy and confusion matrices of different types of quantum cores in the QSVM. The function of the SVM kernel is to transform data into a more suitable modeling space. Different algorithms use different types of kernel functions. It is well known that the most commonly used functions are linear and RBF, which we used as our comparison methods. Specifically, we used the linear and RBF methods in the classical SVM, after which we compared the scores of different quantum kernel functions in the QSVM with the results outlined in Table 5.

Table 5. Comparison of accuracy between quantum and classical classifiers using different coding methods.

Classifier	Feature Maps (Function)	2 Classification Score	5 Classification Score
SVM	Linear	0.96	0.94
	RBF	0.99	0.97
QSVM	ZZFeatureMap linear	0.96	0.94
	ZZFeatureMap circular	0.99	0.94
	PauliFeatureMap	0.99	0.98

We used the Android-SEM model to enhance the semantic information of Android source code and embed it in a vector space. We also used MLP to compress the feature vector from high latitude to low latitude, so that quantum classifiers can easily support it, after which we implemented different kernel functions for the ablation experiments. We then conducted comparative experiments on the classification of malicious Android code. Based on the test dataset, the accuracy of QSVM is comparable to that of classic SVM, and the detection accuracy of malicious code can reach 99%. Compared to the performance on simple datasets, QSVM can maintain satisfactory results on complex datasets. In other words, QSVM can provide enhanced performance if the dataset is a complex boundary that cannot be captured using conventional classical kernels. It is also undeniable that achieving such a high accuracy rate is related to the previous classic deep learning models. The semantic enhancement model based on transfer learning pre-trains feature vectors, thereby significantly improving the accuracy of QSVM. In addition, according to the confusion matrix presented in Figure 10, the results listed in Table 5 show that the precision of QSVM is significantly improved. This experiment proves that QSVM is effective in detecting and classifying malicious code.



(**a**) (**b**)

Figure 10. Quantum classifier confusion matrix: (**a**) 2-category classification; (**b**) 5-category classification.

5. Conclusions and Future Work

Based on the concept of transfer learning, in this study, we focused on obtaining comment information not initially present in Android malware through migration. We inte-

grated conventional deep learning models with quantum machine learning and proposed Android-SEM, which is a transfer learning-based model for the semantic enhancement of Android malicious code. To improve the effectiveness of our proposed model in detecting malicious code, we optimized the performance of the comment vectors generated by the semantic enhancement module through a generative adversarial approach. We also proposed a comment filter based on a linear regression model that retains high-quality comment vectors and combines them with other features to detect and classify Android malicious code. Experiments on the CICmalDroid2020 dataset showed that our proposed method can detect malicious code in 13,694 benign and malicious samples with an accuracy level of 99.55%. Moreover, 14,873 samples of different types of malicious code in the dataset can be classified with 99.01% accuracy. We conducted ablation experiments for the proposed model, which illustrated the necessity of each module, demonstrated the effectiveness of our proposed method, and showed that our proposed method achieved enhanced accuracy in the detection and classification of malicious code. In addition, this study combined quantum and classical machine learning methods. Simulation experiments showed that quantum and classical machine learning models achieved exceptional accuracy levels in malicious code detection tasks, with QSVM resulting in a theoretical enhancement.

In our future studies, we shall first add data flow information and structural features to the existing basis to extend the scope of our proposed system and further explore the problem of classifying Android malicious code. Second, we shall consider quantum state processing of the model's classical part to realize a full quantum model. Finally, we intend to enable the efficient analysis of obfuscated malicious code to improve the effectiveness of our proposed method in the real world.

Author Contributions: Conceptualization, Y.H., X.L. and F.L.; Data curation, Y.H., C.Z. and H.G.; Formal analysis, Y.H., K.T. and F.L.; Investigation, Y.H., X.L., M.Q. and F.L.; Methodology, Y.H., M.Q., K.T. and P.W.; Project administration, Y.H. and F.L.; Resources, Y.H., X.L. and F.L.; Software, Y.H.; Supervision, F.L.; Validation, Y.H. and M.Q.; Visualization, Y.H.; Writing—original draft, Y.H.; Writing—review & editing, Y.H. and F.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Natural Science Foundation of China under Grant No.61802435.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, Z.; Wang, R.; Japkowicz, N.; Tang, D.; Zhang, W.; Zhao, J. Research on unsupervised feature learning for Android malware detection based on Restricted Boltzmann Machines. *Future Gener. Comput. Syst.* **2021**, *120*, 91–108. [CrossRef]
2. Chen, D.; Wawrzynski, P.; Lv, Z. Cyber security in smart cities: A review of deep learning-based applications and case studies. *Sustain. Cities Soc.* **2021**, *66*, 102655. [CrossRef]
3. Shang, Y. Consensus of Hybrid Multi-Agent Systems With Malicious Nodes. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 685–689. [CrossRef]
4. Fragkos, G.; Minwalla, C.; Plusquellic, J.; Tsiropoulou, E.E. Artificially Intelligent Electronic Money. *IEEE Consum. Electron. Mag.* **2021**, *10*, 81–89. [CrossRef]
5. Wu, Q.; Zhu, X.; Liu, B. A Survey of Android Malware Static Detection Technology Based on Machine Learning. *Mob. Inf. Syst.* **2021**, *2021*, 8896013. [CrossRef]
6. Kouliaridis, V.; Kambourakis, G. A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. *Information* **2021**, *12*, 185. [CrossRef]
7. LeClair, A.; Jiang, S.; McMillan, C. A neural model for generating natural language summaries of program subroutines. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 25–31 May 2019; pp. 795–806.
8. Rebentrost, P.; Mohseni, M.; Lloyd, S. Quantum support vector machine for big data classification. *Phys. Rev. Lett.* **2014**, *113*, 130503. [CrossRef]
9. Cen, L.; Gates, C.S.; Si, L.; Li, N. A probabilistic discriminative model for android malware detection with decompiled source code. *IEEE Trans. Dependable Secur. Comput.* **2014**, *12*, 400–412. [CrossRef]
10. Akram, J.; Mumtaz, M.; Jabeen, G.; Luo, P. DroidMD: An efficient and scalable android malware detection approach at source code level. *Int. J. Inf. Comput. Secur.* **2021**, *15*, 299–321. [CrossRef]
11. LeClair, A.; McMillan, C. Recommendations for datasets for source code summarization. *arXiv* **2019**, arXiv:1904.02660.

12. Mahdavifar, S.; Kadir, A.F.A.; Fatemi, R.; Alhadidi, D.; Ghorbani, A.A. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. In Proceedings of the 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 17–22 August 2020; pp. 515–522.
13. Zhang, X.; Zhang, Y.; Zhong, M.; Ding, D.; Cao, Y.; Zhang, Y.; Zhang, M.; Yang, M. 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, Virtual, 9–13 November 2020; pp. 757–770.
14. Onwuzurike, L.; Mariconti, E.; Andriotis, P.; Cristofaro, E.D.; Ross, G.; Stringhini, G. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Trans. Priv. Secur.* **2019**, *22*, 1–34. [CrossRef]
15. Xu, K.; Li, Y.; Deng, R.; Chen, K.; Xu, J. DroidEvolver: Self-evolving Android malware detection system. In Proceedings of the 2019 IEEE European Symposium on Security and Privacy (EuroS&P), Stockholm, Sweden, 17–19 June 2019; pp. 47–62.
16. Arp, D.; Spreitzenbarth, M.; Hubner, M.; Gascon, H.; Rieck, K.; Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket. *NDSS* **2014**, *14*, 23–26.
17. Suciu, O.; Coull, S.E.; Johns, J. Exploring adversarial examples in malware detection. In Proceedings of the 2019 IEEE Security and Privacy Workshops (SPW), San Francisco, CA, USA, 19–23 May 2019; pp. 8–14.
18. Idrees, F.; Rajarajan, M.; Conti, M.; Chen, T.M.; Rahulamathavan, Y. PIndroid: A novel Android malware detection system using ensemble learning methods. *Comput. Secur.* **2017**, *68*, 36–46. [CrossRef]
19. Pal, M. Random forest classifier for remote sensing classification. *Int. J. Remote Sens.* **2005**, *26*, 217–222. [CrossRef]
20. Wang, Q.; Garrity, G.M.; Tiedje, J.M.; Cole, J.R. Naive Bayesian classifier for rapid assignment of rRNA sequences into the new bacterial taxonomy. *Appl. Environ. Microbiol.* **2007**, *73*, 5261–5267. [CrossRef]
21. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
22. Wang, G.y.; Yu, H.; Yang, D. Decision table reduction based on conditional information entropy. *Chin. J. Comput. Chin. Ed.* **2002**, *25*, 759–766.
23. Platt, J. Sequential minimal optimization: A fast algorithm for training support vector machines. *Microsoft Res.* **1998**, *3*, 88–95.
24. Zanaty, E. Support vector machines (SVMs) versus multilayer perception (MLP) in data classification. *Egypt. Inform. J.* **2012**, *13*, 177–183. [CrossRef]
25. Wang, Z.; Li, C.; Yuan, Z.; Guan, Y.; Xue, Y. DroidChain: A novel Android malware detection method based on behavior chains. *Pervasive Mob. Comput.* **2016**, *32*, 3–14. [CrossRef]
26. Rathore, H.; Sahay, S.K.; Nikam, P.; Sewak, M. Robust Android Malware Detection System Against Adversarial Attacks Using Q-Learning. *Inf. Syst. Front.* **2020**, 1–16. [CrossRef]
27. Marquis-Boire, M.; Marschalek, M.; Guarnieri, C. *Big Game Hunting: The Peculiarities in Nation-State Malware Research*; Black Hat: Las Vegas, NV, USA, 2015.
28. LeClair, A.; Haque, S.; Wu, L.; McMillan, C. Improved Code Summarization via a Graph Neural Network. In Proceedings of the ICPC '20: 28th International Conference on Program Comprehension, Seoul, Korea, 13–15 July 2020; pp. 184–195. [CrossRef]
29. Xu, K.; Wu, L.; Wang, Z.; Feng, Y.; Sheinin, V. Graph2Seq: Graph to Sequence Learning with Attention-based Neural Networks. *arXiv* **2018**, arXiv:1804.00823.
30. Xu, K.; Wu, L.; Wang, Z.; Yu, M.; Chen, L.; Sheinin, V. Exploiting Rich Syntactic Information for Semantic Parsing with Graph-to-Sequence Model. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 918–924. [CrossRef]
31. Ahmad, W.U.; Chakraborty, S.; Ray, B.; Chang, K. A Transformer-based Approach for Source Code Summarization. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, 5–10 July 2020; pp. 4998–5007. [CrossRef]
32. Wang, R.; Zhang, H.; Lu, G.; Lyu, L.; Lyu, C. Fret: Functional Reinforced Transformer With BERT for Code Summarization. *IEEE Access* **2020**, *8*, 135591–135604. [CrossRef]
33. Dai, Z.; Yang, Z.; Yang, Y.; Carbonell, J.G.; Le, Q.V.; Salakhutdinov, R. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, 28 July–2 August 2019; pp. 2978–2988. [CrossRef]
34. Biamonte, J.; Wittek, P.; Pancotti, N.; Rebentrost, P.; Wiebe, N.; Lloyd, S. Quantum machine learning. *Nature* **2017**, *549*, 195–202. [CrossRef]
35. Anguita, D.; Ridella, S.; Rivieccio, F.; Zunino, R. Quantum optimization for training support vector machines. *Neural Netw.* **2003**, *16*, 763–770. [CrossRef]
36. Park, J.E.; Quanz, B.; Wood, S.; Higgins, H.; Harishankar, R. Practical application improvement to Quantum SVM: Theory to practice. *arXiv* **2020**, arXiv:2012.07725.
37. Havlíček, V.; Córcoles, A.D.; Temme, K.; Harrow, A.W.; Kandala, A.; Chow, J.M.; Gambetta, J.M. Supervised learning with quantum-enhanced feature spaces. *Nature* **2019**, *567*, 209–212. [CrossRef]
38. Liu, Y.; Arunachalam, S.; Temme, K. A rigorous and robust quantum speed-up in supervised machine learning. *Nat. Phys.* **2021**, *17*, 1013–1017. [CrossRef]

39. Wang, Z.; Liang, Z.; Zhou, S.; Ding, C.; Shi, Y.; Jiang, W. Exploration of Quantum Neural Architecture by Mixing Quantum Neuron Designs: (Invited Paper). In Proceedings of the IEEE/ACM International Conference On Computer Aided Design, ICCAD 2021, Munich, Germany, 1–4 November 2021; pp. 1–7. [CrossRef]
40. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.; Polosukhin, I. Attention is All you Need. In Proceedings of the Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 5998–6008.
41. Lin, T.; RoyChowdhury, A.; Maji, S. Bilinear CNN Models for Fine-Grained Visual Recognition. In Proceedings of the 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, 7–13 December 2015; pp. 1449–1457. [CrossRef]
42. Al-Fawa'reh, M.; Saif, A.; Jafar, M.T.; Elhassan, A. Malware Detection by Eating a Whole APK. In Proceedings of the 15th International Conference for Internet Technology and Secured Transactions, ICITST 2020, London, UK, 8–10 December 2020; pp. 1–7. [CrossRef]
43. Blank, C.; Park, D.K.; Rhee, J.K.K.; Petruccione, F. Quantum classifier with tailored quantum kernel. *NPJ Quantum Inf.* **2020**, *6*, 1–7. [CrossRef]
44. Haq, I.U.; Khan, T.A.; Akhunzada, A. A Dynamic Robust DL-Based Model for Android Malware Detection. *IEEE Access* **2021**, *9*, 74510–74521. [CrossRef]
45. Sihag, V.; Choudhary, G.; Vardhan, M.; Singh, P.; Seo, J.T. PICAndro: Packet InspeCtion-Based Android Malware Detection. *Secur. Commun. Netw.* **2021**, *2021*, 9099476. [CrossRef]