Using DTL-MD with GANs and ResNet for Malicious Code Detection

Yiming Li*, Tao Xie, Dongdong Mei
Department of Computer Science and Technology, School of Computer Science and Engineering, Ningxia Institute of Science and Technology, Shizuishan 753000, China
Email: Liyiming0206@163.com
*Corresponding Author

Keywords: malicious code detection, transfer learning, feature selection, online learning, intelligent detection

Received: December 31, 2024

This study proposes a malicious code detection model DTL-MD based on deep transfer learning, which aims to improve the detection accuracy of existing methods in complex malicious code and data scarcity. In the feature extraction process, the weighted sum method of GIST and LBP features is used to combine the advantages of the two features. Online transfer learning is used to reduce the data distribution difference between the target domain and the source domain. The model uses ResNet50V2 as the backbone network and combines SimAM to enhance the feature extraction and representation capabilities. In addition, in order to further improve the robustness of detection, GAN is used to generate malicious code variants and expand the training data set. In the experiment, the public CICIDS 2017 data set is used for model training and testing. The performance test results show that when the threshold is 0.7, the accuracy of DTL-MD is 95.8% and the F1 score is 0.93. In a performance test involving 30,000 samples, the throughput of the DTL-MD model under Trojans, viruses, worms, and adware is 11, 12, 11, and 12 tasks/s, respectively, and the inference time is 211, 225, 239, and 234 samples/s, respectively. Compared with GAN, DTL-MD increases the throughput by about 10% and the inference speed by about 15%. The research aims to provide new ideas for improving the intelligence and automation level of malicious code detection technology, which has certain application value and practical significance.

Povzetek: A deep transfer learning-based model (DTL-MD) enhances malicious code detection using ResNet50V2, GAN-generated variants, and online learning, achieving 95.8% accuracy and improving detection speed and robustness against evolving threats.

1 Introduction

As the Internet becomes more widespread and information technology advances rapidly, the transmission routes of malicious code (MC) have become increasingly complex, and the impact of malicious software in modern society is becoming increasingly significant [1]. MC not only affects the security of personal information, but also poses a serious threat to the network environment of enterprises, government agencies, and society [2]. In recent years, the types of MC have increased exponentially, from traditional viruses, Trojans, and spyware to ransomware and worms in recent years, showing a trend of diversification and high complexity [3]. With the continuous advancement of MC attack technology, the existing signature matching-based detection methods have been unable to effectively cope with the challenges of variant MCs and new attacks [4]. Furthermore, as MC samples continue to pile up, the challenge lies in efficiently extracting discernible features from a vast array of samples and enhancing the model's ability to adapt to novel attack types via transfer learning. This has become crucial for boosting detection precision and tackling variant attacks effectively [5]. In this context, scholars have proposed various innovative methods for detecting MC to cope with the constantly changing MC environment. Kim proposed an MC detection technology combining dynamic and static analysis to address the diversification of MC propagation channels and the increasing intelligence of propagation technology. Through dynamic and static analysis of Trojan-type downloaders and MC of deliverers, the accuracy of detection was effectively improved [6]. Kim et al. raised an approach for detecting and classifying MC based on application programming interface sequences to address the problem of the proliferation and diversification of malware. Research showed that the proposed method showing high detection efficiency and accuracy [7]. Wang et al. proposed an efficient detection method combining CNNs and generative adversarial networks (GANs) to address the accuracy issues caused by the complexity of MC families and the rapid growth of variants in malware detection. Research showed that this method significantly improved detection accuracy by generating MC variants and performing lightweight classification [8]. Li et al. proposed an MC detection method based on feature fusion and machine learning. They extracted multidimensional features through static analysis and statistical analysis, extracted feature vectors using the n-gram model and TF-IDF, selected the best feature vectors with the classifier, and finally built an automatic detection model. The results showed that the recognition accuracy of this method could reach 98.0%, with an F1 score of 0.969 [9].

Online Transfer Learning (OTL) is a technique that combines the advantages of online learning and transfer learning. It improves the adaptability of models in dynamic environments by receiving new data in real time and using source domain knowledge to transfer to the target domain. Li et al. raised an evolutionary multi-objective Bayesian optimization algorithm combined with multi-source OTL to address the challenge of limited fitness evaluation in multi-objective optimization problems. Through the comparison of multiple multi-objective optimization benchmark problems and real-world problems, it was proved that transfer learning could effectively improve the optimization performance of the problem [10]. Cherifi et al. proposed an automatic classification method for chest CT scans based on machine learning and deep learning. CT images were classified into COVID-19 or non-COVID-19 categories, and different machine learning models were used. The results showed that the accuracy of the ResNet50V2 model was 86.67% on a small data set and 97.52% on a large data set, demonstrating the potential of OTL in rapid detection [11]. Cui proposed a performance test of target detection and motion recognition algorithms in combination with OTL. In addition, the study also compared the recognition accuracy of 3D-CNN and dual-resolution 3D-CNN models under different video frames. The results showed that when the number of video frames was 20, the accuracy of the two algorithms in recognizing basic basketball movements was 89.6% and 95.8%, respectively [12]. Qin et al. proposed an OTL-based estimation method to address the problem of data domain distribution differences caused by changes in temperature, aging, and other conditions in battery state of charge estimation. Through the design of a transfer conversion mechanism and a new Hoeffding-based extreme learning machine algorithm, research showed that this method could effectively reduce negative transfer and accurately estimate under complex conditions [13]. Table 1 summarizes the above related studies, including the proposed models, key indicators, and limitations.

Table 1: Literature review summary.

Reference	Method	Key Results	Limitations
Kim [6]	Combination of dynamic and static analysis	Solve the problem of the diversification of the malware propagation method.	Poor detection performance on complex malware.
Kim, Lee [7]	API aequence-based detection	The more complex the malicious behavior, the higher the detection efficiency.	Limited adaptability to large data sets
Wang et al. [8]	CNN + GAN	Classification accuracy: 97.78%	High computational complexity
Li et al. [9]	Feature fusion and machine learning	Recognition accuracy: 98.0%, F1 score: 0.969, AUC: 0.973.	More suitable for static samples
Li et al. [10]	Adaptive online multisource transfer learning method	The performance gains brought by transfer learning are demonstrated on multiple benchmarks and real-world problems.	Insufficient focus on the specific domain of malware detection
Cherifi et al. [11]	ResNet50V2 transfer learning	The accuracy is 86.67%, sensitivity is 93.94%, specificity is 81%, F1 score is 86% on the small dataset.	Affected by hyperparameter adjustment
Cui [12]	SSD with 3D-CNN architecture	The best recognition accuracy of 95.8% was achieved using 3D-CNN at 20 video frames.	Applicability needs to be verified
Qin et al. [13]	OTL framework and Hoeffding-based ELM	Accurate SOC estimation results can be obtained even under complex application conditions.	Target space differences lead to negative transfer

According to Table 1, although existing MC detection methods have made some progress in the fields of static analysis and signature matching, traditional methods still have insufficient adaptability and detection accuracy when faced with rapidly increasing MC variants, complex attack patterns, and scarce target domain samples. Specifically, existing methods usually rely on fixed feature extraction rules and cannot effectively deal with new MC variants. In addition, the model has poor data adaptability in the target domain, resulting in reduced accuracy when migrating to new data sets. Due to limited training data, many methods have insufficient generalization capabilities and are difficult to meet actual application needs. To this end, a Deep Transfer Learning-based Malware Detection Model (DTL-MD) is proposed. The novel aspects of this research include the following. Firstly, by employing a feature selection strategy, the process of extracting features from MC samples is refined, thereby enhancing the model's discriminatory capabilities. Secondly, through OTL, the model can effectively cope with the situation of insufficient MC samples in the target domain and adapt to

new MC variants. The objective of this research is to offer a more efficient, intelligent, and adaptable solution for detecting MC.

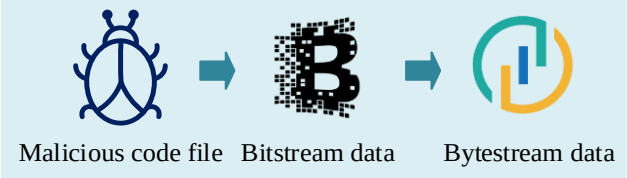
The contributions of the research are as follows: First, DTL-MD reduces the impact of sample scarcity in the target domain by introducing the OTL strategy. Second, an improved feature selection mechanism is designed to more effectively extract key features that are helpful for MC identification. Finally, DTL-MD improves detection speed and computational efficiency by optimizing the computational structure, combining a simplified attention mechanism and an efficient convolutional module.

2 Methods and materials

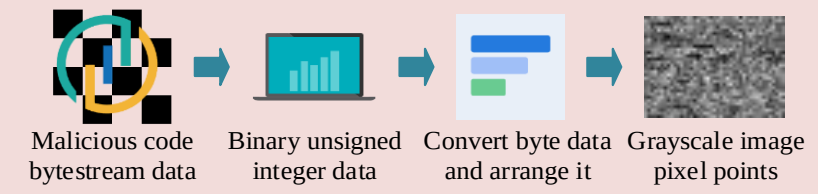
2.1 Design of visual texture feature extraction based on feature fusion

It is assumed that OTL can effectively reduce the data distribution differences between the source domain and the target domain, thereby improving the adaptability of the model in the target domain, especially in the case of insufficient data. Meanwhile, feature selection is assumed to optimize feature extraction, remove redundant features, and improve model accuracy and robustness. Based on these assumptions, the DTL-MD model is designed, and the model construction and optimization process will be introduced in detail below.

In MC detection, the visualization technology of MC based on image processing can provide powerful support for detection models by converting the binary data of MC into images and extracting texture features from them. Global Image Structure Feature (GIST) and Local Binary Pattern Feature (LBP) are two common texture features. GIST can capture the global structural information of an image, while LBP focuses on local texture details [14]. Therefore, a visual texture feature extraction scheme based on feature fusion is proposed, which combines GIST and LBP to better capture the multidimensional features of MC samples. First, the MC is converted from a byte stream to a visual image, as shown in Figure 1.



(a) Bitstream data extraction from malicious code file



(b) Visualization process of malicious code bytestream

Figure 1: Visualization of MC.

Figure 1(a) shows the extraction process of MC bit streams or byte streams. By extracting byte stream data from MC samples, their binary representations are obtained. These data reflect the basic structure and behavior patterns of the program. Figure 1(b) shows the process of further processing the byte stream data into a grayscale image. In the standardization process, each byte is mapped to a pixel value in the grayscale image, preserving the local features and global structural information in the MC. Therefore, the preprocessing process of MC detection combining the two is shown in Figure 2.

Figure 2: Detailed data preprocessing process in MC detection.

As shown in Figure 2, after inputting MC, noise or irrelevant information is first removed through data cleaning. Next, byte stream extraction and normalization are performed to make the data format consistent. Then, the standardized byte stream data is converted into a grayscale image, with each byte corresponding to a pixel's grayscale value, completing the graphical representation of the data. First, a grayscale image is generated from the MC file, and GIST extracts the global structural information of the image through a Gabor filter. Gabor filter can effectively capture the local structure and texture information of the image. The expression is shown in equation (1) [15].

$$\psi(x,y,\theta,\lambda,\gamma,\psi)=\exp\left(-\frac{x^2+\gamma^2y^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x}{\lambda}+\psi\right) \tag{1}$$

In equation (1), x and y are the image coordinates. θ is the the rotating angle of the filter. λ is the wavelength, and γ is the spatial aspect ratio. σ is the standard deviation, and ψ is the phase offset. Applying multi-scale and multi-directional Gabor filtering to an image results in response maps for different directions and frequencies, reflecting the comprehensive texture information of the image. Next, the response map is subjected to pooling processing to extract features representing the global structure, as shown in equation (2).

$$F_{\theta,\lambda}(x,y)=G(x,y)*\psi(x,y,\theta,\lambda,\gamma,\psi) \tag{2}$$

In equation (2), $F_{\theta,\lambda}(x,y)$ is the filter response diagram. $G(x,y)$ is a grayscale image. $*$ indicates a convolution operation. Furthermore, in LBP extraction, the local texture information of the image is extracted by calculating the relationship between the gray value of each pixel and its neighboring pixels, as shown in equation (3).

$$LBP(x,y)=\sum_{p=0}^{P-1}s(I_p-I_c)\cdot 2^p \tag{3}$$

In equation (3), I_p and I_c are the values of the neighboring pixels and the central pixel, respectively. $s(x)$ is a symbolic function. P is neighboring pixel. By encoding the texture features of local images, each local region of the image is converted into a binary number, thereby extracting local texture features. In addition, in order to reduce the dimensionality of features and enhance the discriminative power of features, the information gain and L1 regularization strategies are introduced to construct a feature selection strategy. Information gain measures the contribution of a feature to the target class information. The calculation of information gain $IG(x_i)$ is shown in equation (4).

$$IG(x_i)=H(y)-H(y|x_i) \tag{4}$$

In equation (4), $H(y)$ is the entropy of the target variable y. $H(y|x_i)$ is the entropy of the target variable y under the given feature condition x_i. Subsequently, Lasso regression selects features through the L1 regularization term, thereby avoiding the interference of redundant features. For example, after L1 regularization, features with higher scores include certain image texture features, while features with a score of zero are excluded. Here is an example: After L1 regularization feature selection, the model selected feature 1 (score: 0.85), feature 2 (score: 0.72), and feature 4 (score: 0.91), excluding features with lower scores (such as feature 5, score: 0.02). The optimization objective of Lasso regression is given in equation (5).

$$\hat{\beta}=\arg\min_{\beta}\left(\sum_{i=1}^{n}(y_i-x_i^T\beta)^2+\lambda'\sum_{j=1}^{p}|\beta_j|\right) \tag{5}$$

In equation (5), x_i is the eigenvector. β_j is the weight coefficient of the feature. λ' is the regularization parameter, which controls the strictness of feature selection. Through Lasso regression, the unimportant parts of the feature weight coefficients will be compressed to zero, automatically selecting features with high discriminative power. During the feature selection process, the threshold of information gain was set to 0.05, and only features with information gain greater than this threshold were retained to remove low-contribution features. Subsequently, L1 regularization (λ' =0.01) further compressed the feature space, reducing the number of features from 512 to 128.

After feature selection, GIST and LBP are fused. After feature fusion, the model selected fused features with high scores, such as GIST feature 1 (score: 0.88) and LBP feature 2 (score: 0.79), which played a key role in the classification of MC. To better combine the advantages of the two features, a weighted sum is used to obtain the final feature vector F_{fusion}, as shown in equation (6).

$$F_{fusion}=\omega_1\cdot F_{GIST}+\omega_2\cdot F_{LBP} \tag{6}$$

In equation (6), F_{GIST} and F_{LBP} represent the GIST and LBP vectors after feature selection. ω_1 and ω_2 are the feature weights, which are determined by cross-validation and manual tuning. In the cross-validation process, the data set is divided into multiple subsets, and the model is evaluated on different training and validation sets to select the best weight combination. For certain specific weights, manual tuning is performed to optimize the model's performance in MC detection. Therefore, after the above calculations, a general framework for MC detection based on combined features is finally obtained, as shown in Figure 3.

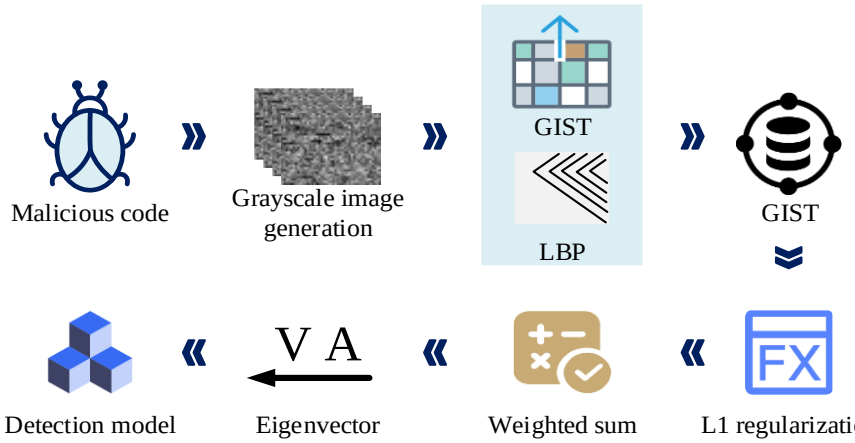
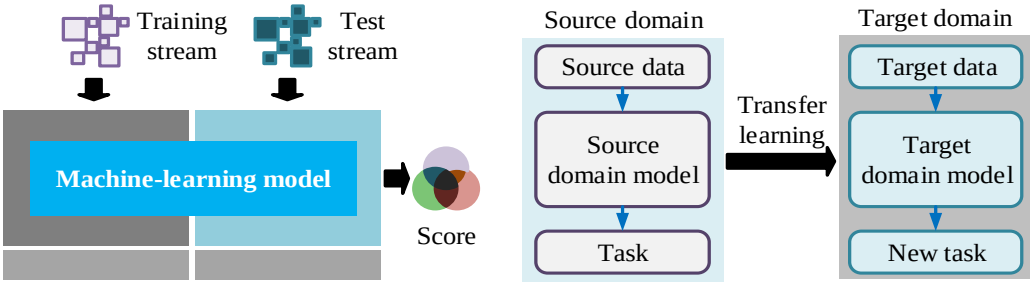


Figure 3: A common framework for MC detection based on combined features.

As shown in Figure 3, the first input MC file undergoes data cleaning and byte stream extraction, and after normalization processing, it is converted into a grayscale image, with each byte corresponding to a pixel's grayscale value. Subsequently, GIST and LBP are extracted from the generated grayscale image. Next, the study uses information gain screening and L1 regularization to further complete feature selection and remove redundant features. Subsequently, the GIST and LBP are fused through weighted summation to form the final feature vector, which is then input into the detection model for MC detection.

2.2 MC detection method based on OTL

In the previous section, the feature extraction and fusion methods of MC provide important input data for subsequent detection tasks. OTL combines the advantages of online learning and transfer learning. It can receive new data in real time and use the source domain knowledge to optimize the target domain model. Specifically, online learning enables the model to be updated in real time and adapt to changing attack patterns and MC variants. Meanwhile, transfer learning can transfer knowledge from the rich data in the source domain to solve the problem of scarce samples in the target domain. In this way, OTL not only improves the generalization ability of the model, but also enhances its ability to respond to new attacks in practical applications. To guarantee the effectiveness of OTL, feature selection is crucial. Through the feature selection method in Section 2.1, the GIST and LBP features are optimized to provide efficient input data. These refined features make the application of OTL in the target domain more efficient. Feature selection ensures that the OTL model can focus on the most discriminative features by removing redundant information, thereby improving detection accuracy and adaptability [16]. Therefore, the research attempts to propose an MC detection method based on OTL. By combining the advantages of online learning and transfer learning, it can achieve incremental updates when new samples arrive, and improve the detection accuracy of the target domain with the help of source domain knowledge. The framework of the two is shown in Figure 4 [17].



(a) Online learning (b) Transfer learning

Figure 4: Schematic diagram of online learning and transfer learning.

Figure 4(a) shows the online learning mechanism, which performs incremental updates by receiving training data streams in real time and performs real-time predictions on test data streams. In the transfer learning mechanism of Figure 4(b), the source domain model utilizes transfer learning to adapt to the target domain, addressing the issue of limited sample availability in the target domain. The essence of OTL lies in transferring knowledge from the source domain to the target domain. Assuming that the source domain data is $D_S = \left\{\left(x_i^S, y_i^S\right)\right\}_{i=1}^{n_S}$ and the target domain is $D_T = \left\{\left(x_i^T, y_i^T\right)\right\}_{i=1}^{n_T}$, where x_i^S and x_i^T represent the input data of the source and target domains, respectively. y_i^S

and y_i^T are the corresponding labels. The goal is to optimize model performance in the target domain by minimizing the discrepancy in distribution between the source and target domains. The measurement method is the Maximum Mean Discrepancy (MMD), which quantifies the distribution difference between the source domain and the target domain, as shown in equation (7) [18].

$$MMD(D_S, D_T) = \left\| \frac{1}{n_S} \sum_{i=1}^{n_S} \phi(x_i^S) - \frac{1}{n_T} \sum_{i=1}^{n_T} \phi(x_i^T) \right\|_H \quad (7)$$

In equation (7), $\phi(\cdot)$ is the mapping function. n_S and n_T are the numbers of samples for the source domain and target domain. By minimizing MMD, OTL can minimize the difference between the source domain and the target domain, ensuring that the model can be transferred to the target domain. In the incremental learning and online updating steps, the model needs to gradually receive new data and continuously update. Assuming that the parameters of the model are θ, in each step t, the goal of the model is to continuously update the parameters through incremental learning. Whenever new data arrives, assuming that the loss function of the current step is $L_t(\theta)$, the model is updated at time step t as shown in equation (8) [19].

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta L_t(\theta_t) \quad (8)$$

In equation (8), η_t is the learning rate, and $\nabla_\theta L_t(\theta_t)$ is the gradient of the loss function with respect to the parameter θ. Through this incremental update process, the model only updates the part related to the new data without retraining the entire model. This method ensures that OTL improves the adaptability of the target domain through local updates without retraining the entire model. The OTL framework is developed based on TensorFlow and Keras, combined with a custom gradient update rule to adapt to online incremental learning scenarios. In the domain adaptation process, OTL uses minimizing MMD as the objective function and adjusts model parameters in real time through back propagation. Existing experimental results showed that incremental updates had significant advantages in dynamic environments. Reference [20] proves that the incremental learning strategy can effectively improve the real-time update capability of the model and enhance performance and response speed. To further improve the performance of MC detection in the target domain within the OTL framework, a DTL-MD MC detection model is developed. It combines the GAN and the Residual Network (ResNet), while introducing the Group Convolution and the Simple Attention Module (SimAM). Group convolution improves computational efficiency by reducing model parameters, providing faster adaptation capabilities for fine-tuning in the target domain. SimAM further enhances the expression of key features and improves the detection performance on the target domain. Its structure is shown in Figure 5.

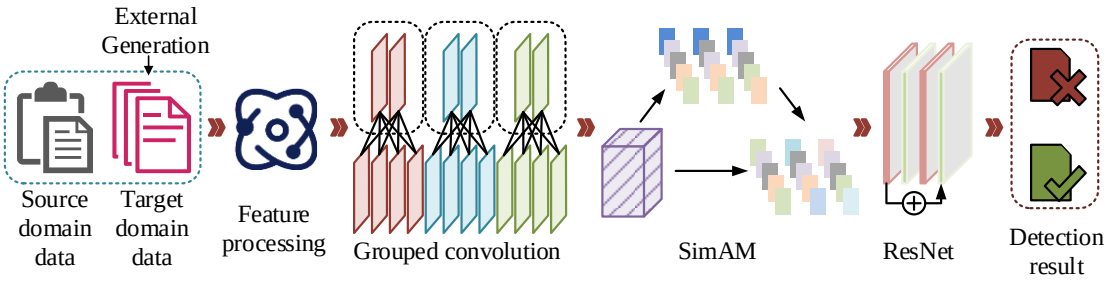


Figure 5 DTL-MD model structure.

As shown in Figure 5, the DTL-MD model first generates target domain data through GAN and inputs it into the model for processing together with the source domain data. Next, ResNet extracts features, which are then further processed and enhanced through Group Convolution and SimAM to finally generate detection results. The data generated by GAN is generated outside the model and then input into the model together with the source domain data, thereby improving the adaptability and detection capabilities of the target domain. In DTL-MD, the study combines grouped convolution to further improve computational efficiency and model performance, as shown in equation (9).

$$Y_{i,j,k} = \sum_{p=0}^{K-1} \sum_{q=0}^{K-1} \sum_{r=0}^{C/G-1} X_{i+p,j+q,r} \cdot W_{p,q,r,k} \quad (9)$$

In equation (9), $Y_{i,j,k}$ is the k th channel at position (i, j) in the output feature map. $X_{i+p,j+q,r}$ is the convolution window value of the input feature map's r th channel. $W_{p,q,r,k}$ is the convolution and weight of the r th channel. C and G are the number of channels and the number of groups in the input feature map. Then, the SimAM attention mechanism is introduced to enhance the representation of important features by weighting the features of each channel. Furthermore, GAN enhances the diversity of target domain data by generating new MC samples. Compared with the original dataset, the samples generated by GAN are customized for specific MC categories, such as Trojans, viruses, and worms. The feature distribution of the generated samples is similar to that of the original dataset, aiming to supplement the lack of samples and improve the performance of the model in detecting specific categories of MC. OTL, by updating

model weights in real time, enables the model to quickly adapt to new feature distributions when receiving new target domain samples.

DTL-MD employs transfer learning techniques to pre-train on source domain data to capture its underlying features, followed by fine-tuning on the target domain to align with its specific characteristics. The comprehensive loss function is outlined in equation (10) [21].

$$L_{total} = L_S(\theta) + \lambda L_T(\theta) + \mu \cdot MMD(D_S, D_T) \quad (10)$$

In equation (10), $L_S(\theta)$ and $L_T(\theta)$ are the loss functions of the source domain and target domain, respectively. λ and μ are hyper-parameters. The errors of the source domain and the target domain are calculated as the prediction errors of the source domain data and the target domain data, respectively. The MMD term is used to measure the distribution difference between the source domain and the target domain features. By minimizing the loss function, the model optimizes the source domain task while minimizing the MMD to minimize the difference between the source domain features and the target domain features, helping the model to better adapt to the target domain data. Therefore, the final framework of the DTL-MD MC detection model is shown in Figure 6.

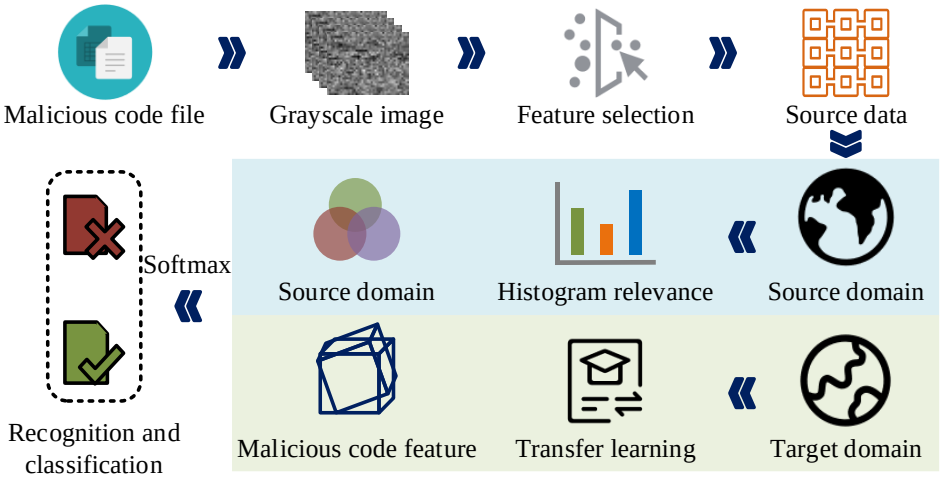


Figure 6: The framework of the DTL-MD MC detection model.

As shown in Figure 6, in the DTL-MD MC detection model, the source domain and target domain data are processed in a hierarchical and parallel manner. After the source domain data undergoes feature selection, the importance of the features is evaluated through histogram correlation analysis, and features with strong task relevance are screened out to provide support for subsequent feature extraction. The target domain data is combined with new samples generated by GAN, and the feature distribution is optimized through transfer learning. Finally, the feature extraction module processes the source domain and target domain features in parallel, and Softmax generates the detection results. Finally, the pseudo code of DTL-MD is shown in Figure 7.

Through this pseudo-code, the specific implementation methods of the model in data processing, feature extraction, feature fusion and OTL are more clearly understood, and the replicability and transparency of the model implementation are improved.

3 Results

3.1 Parameter impact analysis and ablation experiments

To evaluate the effectiveness of the raised DTL-MD MC detection model, the research built a hardware and software environment that meets the requirements of the experiment. The experimental platform uses the Ubuntu 20.04 operating system, the algorithm development language is Python, and the model construction and optimization are based on the TensorFlow and Keras frameworks. The hardware configuration includes an AMD Ryzen 7 5800H processor, an NVIDIA GeForce RTX 3070 graphics card, and 16 GB of memory. The experimental data is sourced from the publicly available MC dataset CICIDS 2017 dataset, which contains a variety of network attack types and malware samples and is suitable for malware classification and variant detection tasks. The dataset was divided into a training set (70%), a validation set (15%), and a test set (15%). The validation set was used to tune hyperparameters. There was no sample overlap between the training set and the test set to ensure the fairness of the model performance evaluation. In the experiment, MC samples and normal samples in the training data were evenly distributed. GAN-generated samples were used to enhance sample data of specific categories in the target domain, improving classification accuracy and the generalization ability of the model.

First, the hyperparameters and in the loss function were jointly tuned, and the results are shown in Table 2.

```
# Pseudocode for DTL-MD Model

# Step 1: Data Preprocessing
def preprocess_data(file):
    byte_data = extract_byte_data(file)  # Extract byte data from the file
    normalized_data = normalize(byte_data)  # Normalize the data
    grayscale_image = convert_to_grayscale(normalized_data)  # Convert to grayscale image
    return grayscale_image

# Step 2: Feature Extraction
def extract_features(image):
    gist_features = extract_gist(image)  # Extract GIST features
    lbp_features = extract_lbp(image)  # Extract LBP features
    return gist_features, lbp_features

# Step 3: Feature Selection
def select_features(gist_features, lbp_features):
    selected_features = select_important_features(gist_features, lbp_features)  # Feature selection
    optimized_features = apply_regularization(selected_features)  # Apply L1 regularization
    return optimized_features

# Step 4: Feature Fusion
def fuse_features(gist_features, lbp_features, weights):
    fused_features = weights[0] * gist_features + weights[1] * lbp_features  # Feature fusion
    return fused_features

# Step 5: Online Transfer Learning
def online_transfer_learning(model, source_data, target_data):
    mmd_value = compute_mmd(source_data, target_data)  # Calculate MMD
    model.update_parameters(mmd_value)  # Update model parameters with new data
    return model

# Step 6: Train and Detect
def train_and_detect(model, train_data, test_data):
    model.train(train_data)  # Train model on the data
    detection_results = model.detect(test_data)  # Detect using the trained model
    return detection_results

# Main Execution
def main():
    input_file = "malicious_code_sample"

    # Step 1: Data Preprocessing
    image = preprocess_data(input_file)

    # Step 2: Feature Extraction
    gist, lbp = extract_features(image)

    # Step 3: Feature Selection
    selected_features = select_features(gist, lbp)

    # Step 4: Feature Fusion
    fused_features = fuse_features(gist, lbp, [0.7, 0.3])

    # Step 5: Online Transfer Learning
    model = initialize_model()  # Initialize the model
    model = online_transfer_learning(model, source_data, target_data)

    # Step 6: Train and Detect
    detection_results = train_and_detect(model, train_data, test_data)

    # Output final results
    print("Detection Results: ", detection_results)

if __name__ == "__main__":
    main()
```

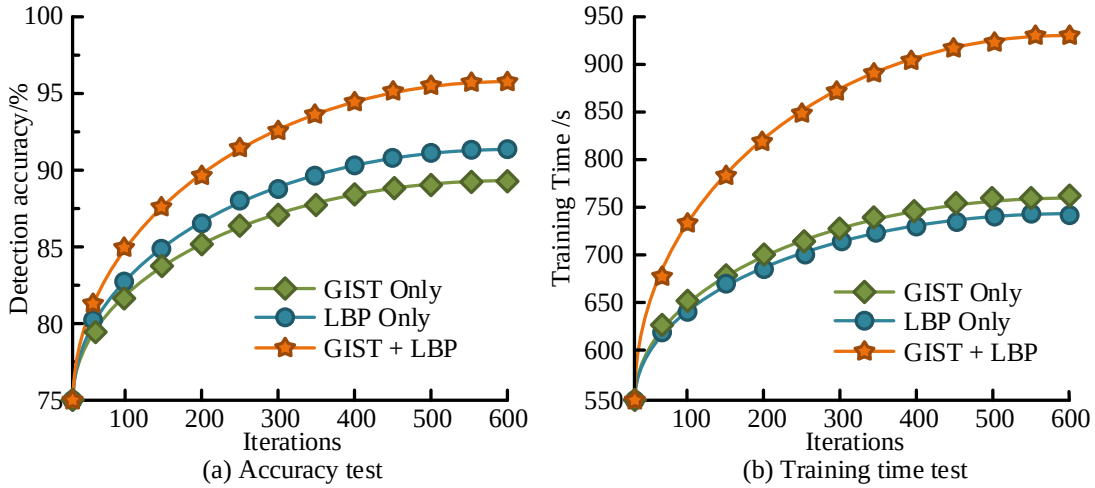
Figure 7: Pseudocode of DTL-MD

Table 2 Hyperparameter joint tuning experiment.

λ	μ	**Accuracy (%)**	**F1 Score (%)**	**Training time (s)**
0.1	**0.01**	90.3	88.4	1117
0.1	**0.05**	91.8	89.2	1162
0.1	**0.1**	93.1	90.5	1213
0.1	**0.5**	92.6	89.7	1263
0.1	**1.0**	91.9	89.1	1318
0.5	**0.01**	92.7	90.8	1214
0.5	**0.05**	94.2	91.8	1267
0.5	**0.1**	95.7	93.4	1316
0.5	**0.5**	94.8	92.3	1374
0.5	**1.0**	94.1	91.9	1427
1.0	**0.01**	92.4	90.1	1263
1.0	**0.05**	93.4	91.3	1311
1.0	**0.1**	93.3	90.9	1373
1.0	**0.5**	92.8	90.5	1426
1.0	**1.0**	91.7	89.8	1482

From Table 2, when λ was 0.5 and μ was 0.1, the model had the best performance on the verification set, with an accuracy of 95.7%, an F1 of 93.4%, and a training time of 1316 s. Meanwhile, an excessively high μ significantly increased the training time, while a low λ might weaken the utilization efficiency of source domain features.

Secondly, by testing the performance of a single feature and the fusion of the two features at different iteration times, the research analyzed the independent contribution of each feature and its performance improvement after fusion. The results are shown in Figure 8.



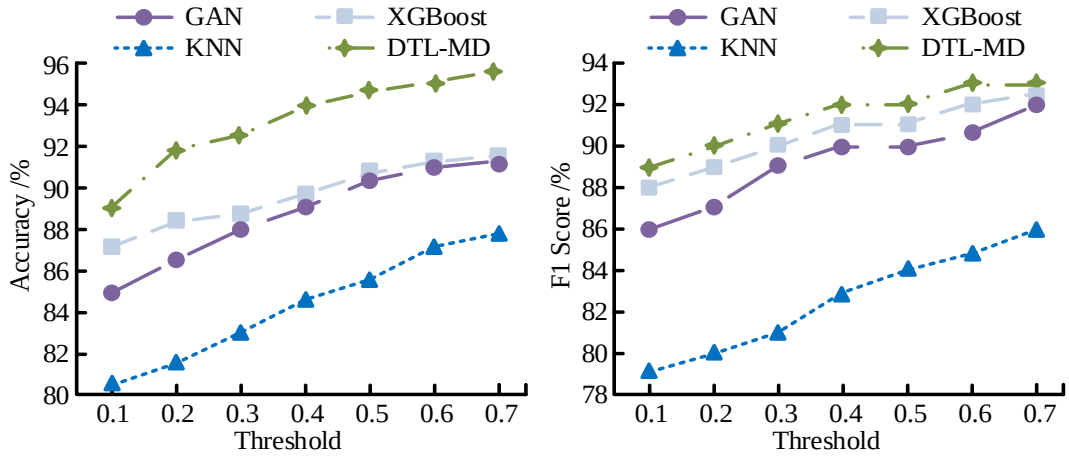
(a) Accuracy test (b) Training time test

Figure 8: Ablation experiment.

As shown in Figure 8(a), when the number of iterations was 600, the detection accuracy of the fused feature reached 95.8%. The fused feature can more comprehensively characterize the characteristics of the MC sample. In Figure 8(b), when the number of iterations was 600, the training time of the fused feature was 932 s, which was about 200 s longer than that of the GIST and LBP features. Although the fused feature increased the training time, the improvement in its detection accuracy showed that this computational overhead was reasonable in MC detection tasks that required high precision.

3.2 Performance test of DTL-MD MC detection model

GAN, Extreme Gradient Boosting (XGBoost), and K-Nearest Neighbors (KNN) are selected as comparison algorithms. First, the classification ability of the MC detection model was evaluated, and the results are shown in Figure 9.



(a) Accuracy test (b) F1 test

Figure 9: Classification performance test results.

Figures 9(a) and (b) show the accuracy and F1 score of each model as a function of the threshold value. The F1 score can balance the precision and recall, and is particularly suitable for MC detection in unbalanced data sets, ensuring fewer missed detections and false positives. In Figure 9(a), when the threshold was 0.7, the accuracy of GAN, XGBoost, KNN, and DTL-MD were 91.2%, 91.3%, 87.9%, and 95.8%, respectively. In Figure

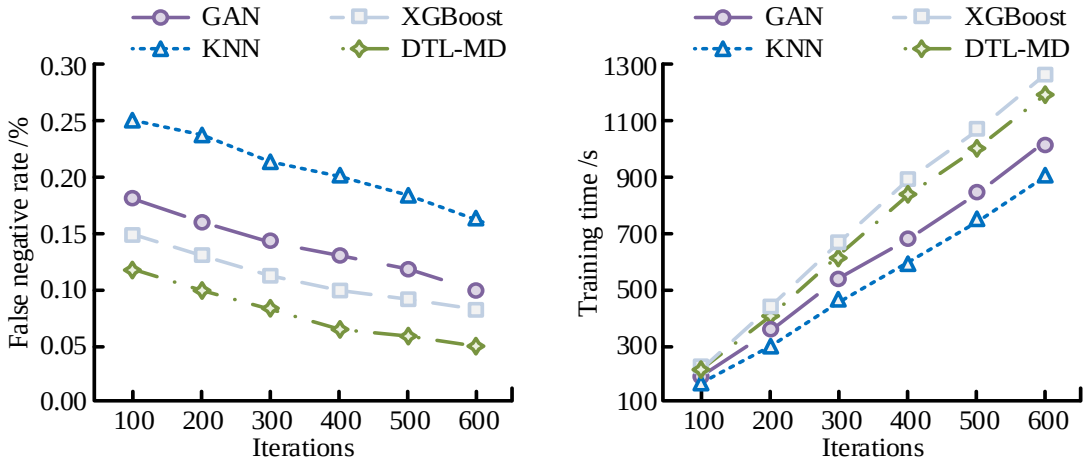
9(b), when the threshold value was 0.7, the F1 scores of each model were 92.1%, 91.9%, 85.9%, and 93.2% respectively. DTL-MD had the highest accuracy and F1 score at high thresholds, effectively reducing false positives through strict classification criteria and avoiding erroneous classification of MC. In contrast, the accuracy and F1 score of XGBoost and GAN were similar, but slightly lower than that of the DTL-MD model, which was due to their conservative decision boundary at high thresholds. Although the number of false positives was reduced, some more complex malicious samples was missed. To ensure the reliability of the results, the standard deviation and 95% confidence interval of the accuracy and F1 score of each model at a threshold of 0.7 were calculated, see Table 3.

Table 3: Standard deviation and confidence interval of accuracy and F1 score.

Model	Accuracy /%	F1 Score /%	Accuracy Std Dev	Accuracy confidence interval	F1 Score Std Dev	F1 Score confidence interval
GAN	91.2	92.1	±0.3	[90.9, 91.5]	±0.2	[91.8, 92.4]
XGBoost	91.3	91.9	±0.2	[91.1, 91.5]	±0.3	[91.6, 92.2]
KNN	87.9	85.9	±0.4	[87.5, 88.3]	±0.3	[85.6, 86.2]
DTL-MD	95.8	93.2	±0.2	[95.6, 96.0]	±0.2	[93.0, 93.4]

According to Table 3, the DTL-MD model showed smaller fluctuations in the standard deviation and confidence interval of the accuracy and F1 score, indicating that it had higher stability under different experimental conditions, higher accuracy and smaller fluctuation range.

Subsequently, the False Negative Rate (FNR) and training time of each model as a function of the number of iterations are shown in Figure 10.



(a) FNR test (b) Training time test

Figure 10: FNR and training time test results.

In Figures 10, the training time reflects the deployment efficiency of the model in practical applications, ensuring high accuracy while having good real-time and operability. In Figure 10 (a), when the number of iterations was 600, the false detection rates of each model were 0.10%, 0.08%, 0.16%, and 0.05%, respectively. In Figure 10 (b), when the number of iterations was 600, the training time of each model was 1049 s, 1282 s, 901 s, and 1257 s respectively. As the number of iterations increased, the false detection rate of the DTL-MD model decreased from 0.12% to 0.05%. In terms of training time, the DTL-MD model required 1257 seconds for training. More iterations of training improved the model's accuracy, but also led to an increase in computational overhead. Finally, the test results of each model under various sample sizes are in Table 4.

Table 4: Test outcomes with various sample sizes.

Model	Sample size	Samples per second	Memory usage /MB	Model size /MB	Computational complexity /GFLOPS
GAN	5000	319.7	1603.5	134.8	47.2
	10000	310.3	1645.7	139.1	48.1
	15000	299.6	1697.4	144.5	50.2
	20000	289.8	1746.2	149.3	52.3

XGBoost	5000	329.1	1449.3	124.6	42.8
	10000	314.7	1497.2	129.3	43.6
	15000	308.2	1547.6	134.9	45.1
	20000	299.4	1598.4	139.8	46.9
KNN	5000	229.4	1202.6	109.5	28.3
	10000	219.8	1246.5	113.6	29.5
	15000	209.3	1299.4	117.9	30.8
	20000	199.6	1349.8	122.1	31.9
DTL-MD	5000	199.7	1702.5	160.3	39.9
	10000	209.4	1804.6	164.2	41.8
	15000	219.8	1906.1	168.9	44.2
	20000	229.3	2008.3	173.4	47.1

In Table 4, computational complexity measures the computational efficiency of the model in processing data in GFLOPS (billion floating-point operations per second), reflecting the computational resources required by the model to complete a specific task. This value is related to the model architecture and the size of the dataset. The computational complexity reflects the computing resources required by the model to process each task. A lower FLOPs value means that the model has better scalability and can run efficiently on large-scale datasets or real-time applications. The processing speed of XGBoost reached 299.4 samples/second with a sample size of 20,000. In contrast, the processing speed of DTL-MD was relatively slow, especially at 20,000 samples, which was only 229.3 samples/s. The complex deep learning structure required more computing time and resources to complete the detection of MC. In terms of memory usage, DTL-MD consumed 2008.3 MB with a sample size of 20,000. In terms of model size, the size of DTL-MD remained at 160 MB. In terms of computational complexity, GAN reached 52.3 GFLOPS at 20,000 samples, while DTL-MD had a computational complexity of 47.1 GFLOPS at 20,000 samples, which is suitable for deployment in environments with sufficient computing resources. As the dataset increased, the processing speed of DTL-MD increased. For small datasets (such as 5,000 samples), its processing speed was slow, but the memory usage and computational complexity were low. As the sample size increased, although the training time and memory usage increased, DTL-MD still maintained high accuracy, especially in the 20,000 sample dataset, where it performed well and showed good generalization ability.

3.3 MC detection simulation experiment based on DTL-MD model

Furthermore, the research conducted simulation experiments on DTL-MD to test its practical application effect. The comparison models were selected from the more advanced models in the field, namely Malware GAN-enhanced Network (MGANet), Sequence GAN for Malware Detection (SeqGAN-Malware), and Deep Reinforcement Learning for Malware Detection (DRL-Malware). The research team constructed a self-built MC dataset containing 30,000 samples, of which 20,000 malware samples covered various types such as Trojans, viruses, and ransomware, and 10,000 normal software samples were from commonly-used applications. Firstly, the throughput and inference speed results of each model under the detection of four types of MC, namely Trojans, viruses, worms, and adware, are shown in Figure 11.

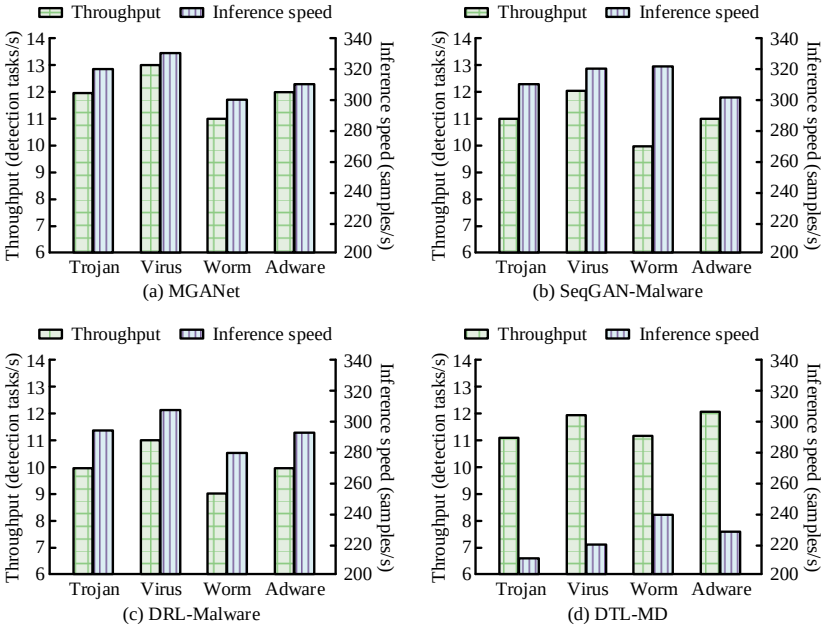


Figure 11: Throughput and inference time tests under different attack types.

Figures 11 (a)-(d) show the throughput and inference time test results under different attack types. Throughput measures the task processing capability of the model, while inference speed reflects the single-sample processing efficiency. Throughput is calculated as the number of tasks completed per second, while inference speed represents the number of samples processed per second. The two are closely related, because a task usually contains detection operations for multiple samples, so throughput is usually lower than inference speed.

In Figure 11 (a), the throughput of MGANet under Trojans, viruses, worms, and adware was 12, 13, 11, and 12 tasks/s, respectively, with an inference speed of 320 to 330 samples/s. In Figure 11 (b), the throughput of SeqGAN-Malware under Trojan, virus, worm, and adware was 11, 12, 10, and 11 tasks/s, respectively. In Figure 11 (d), the throughput of the DTL-MD model under Trojan, virus, worm, and adware was 11, 12, 11, and 12 tasks/s, respectively, and the inference time was 211, 225, 239, and 234 samples/s, respectively. The large computational resources and complex network structure resulted in a heavy computational burden during the inference process, and the throughput performance was moderate. In Figure 11 (c), the inference speed and throughput of DRL-Malware were slightly lower. The training process of deep reinforcement learning required the model to optimize performance through continuous policy updates, and the limitations of the learning strategy led to a decrease in inference speed. Subsequently, the same test dataset was used, containing five main types of MC (Trojans, viruses, worms, adware, and ransomware), and 1,000 samples of each type were randomly selected. The corresponding robustness and processing delay results are shown in Figure 12.

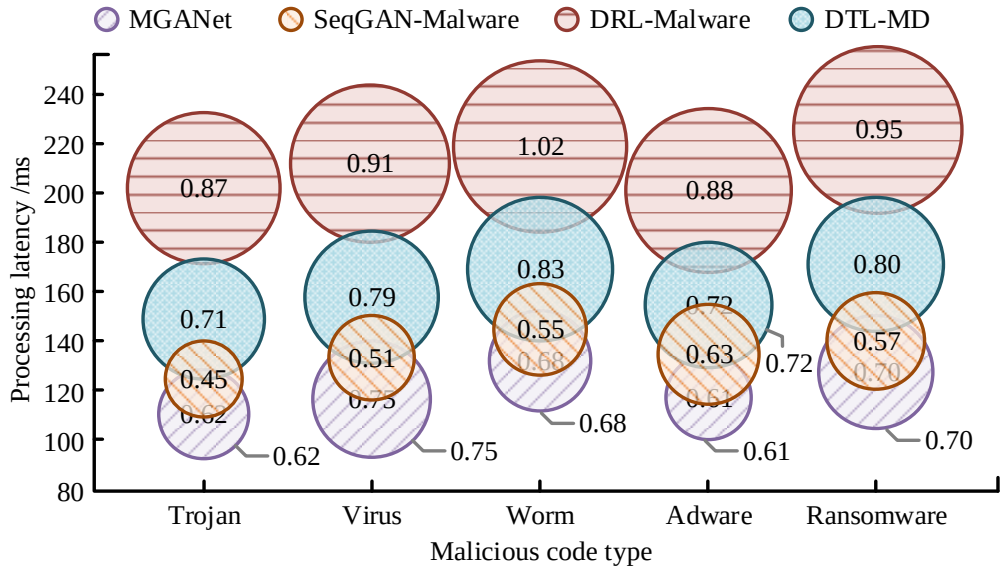


Figure 12: Robustness and processing delay results.

In Figure 12, the robustness of each type of MC was calculated through 10 experiments, and the standard deviation reflects the deviation between the experimental results and the average value. The smaller the standard deviation, the smaller the volatility of the model's detection results on the MC type, and the stronger the robustness. The robustness of DTL-MD under Trojan, virus, worm, adware, and ransomware code was 0.71, 0.79, 0.83, 0.72, and 0.80, respectively, indicating strong adaptability in the face of diverse MC. In terms of processing delay, the DTL-MD model had a longer inference time, with a response time of 172 ms under the worm type. Finally, the detection results of each model under different network bandwidths are shown in Table 5.

Table 5: Detection effect under different network bandwidths.

Network bandwidth	Model	Throughput (tasks/s)	Latency variance /ms	Computational efficiency (FLOPs/task)	Stability /SD
Low (100 Mbps)	MGANet	11	23	1.5	0.7
	SeqGAN-Malware	10	27	1.8	0.8
	DRL-Malware	9	30	2.0	0.9
	DTL-MD	10	29	1.9	1.1
Medium (500 Mbps)	MGANet	13	18	1.3	0.5
	SeqGAN-Malware	11	21	1.6	0.6

	DRL-Malware	10	23	1.8	0.7
	DTL-MD	11	22	1.7	0.8
High bandwidth (1 Gbps)	MGANet	14	15	1.2	0.4
	SeqGAN-Malware	12	17	1.5	0.5
	DRL-Malware	11	19	1.7	0.6
	DTL-MD	12	17	1.6	0.7

In Table 5, the performance of the model in various practical applications can be evaluated by the throughput, latency fluctuation, computational efficiency, and stability under different bandwidths. Stability is measured by calculating the standard deviation of the inference latency. A lower SD value indicates a more stable performance of the model, especially under low bandwidth conditions. DTL-MD was 10 tasks/s at low bandwidth. In terms of latency fluctuation rate, the latency fluctuation rate of MGANet under high bandwidth conditions was only 15 ms. The delay fluctuation rate of DTL-MD was 17 ms. In terms of computational efficiency, DTL-MD had a computational efficiency of 1.6 FLOPs/task under low bandwidth conditions. Complex models can lead to higher computational costs and longer processing times.

Finally, the study introduced the Deep Malware Detection Network (DMDN), Light Gradient Boosting Machine (LightGBM), and Adversarial Malware Detection Network (AMDN). The study also introduced the MalwareBazaar dataset and designed a cross-domain migration experiment. The results are shown in Table 6.

Table 6: Performance comparison in diverse datasets and cross-domain migration tests.

Dataset	Model	Precision (%)	Recall (%)	Robustness (Standard deviation)	Detection rate (Tasks/s)
MalwareBazaar	DTL-MD	91.8	92.3	0.73	12
	DMDN	89.7	90.5	0.81	10
	LightGBM	87.4	88.2	0.86	14
	AMDN	88.9	89.8	0.79	11
CICIDS 2017	DTL-MD	90.2	91.0	0.75	11
	DMDN	87.6	88.8	0.84	9
	LightGBM	85.3	86.5	0.89	13
	AMDN	86.7	87.9	0.82	10
Cross-domain Test	DTL-MD	88.3	89.7	0.75	10
	DMDN	85.9	87.0	0.81	8
	LightGBM	84.1	85.2	0.85	12
	AMDN	86.7	87.9	0.80	9

In Table 6, the Precision and Recall of DTL-MD on the MalwareBazaar dataset reached 91.8% and 92.3% respectively. Meanwhile, in the cross-domain migration experiment, the Recall of DTL-MD remained at 89.7% with a standard deviation of 0.75, which proved its robustness in dealing with changes in data distribution. In contrast, DMDN and AMDN performed second best in robustness, and although LightGBM had an advantage in detection rate, its detection accuracy was relatively low.

4 Discussion

In order to improve the accuracy of MC detection, the study designed a detection model DTL-MD based on OTL and tested its performance. Compared with the model proposed by Kim et al. in the literature [6], although it performed well in the accuracy of MC detection, it was relatively slow in processing speed and was suitable for relatively static scenarios. In the DTL-MD performance test, when the threshold was 0.7, the accuracy of DTL-MD was 95.8% and the F1 score was 93.2%. Although DTL-MD was slower than XGBoost and KNN in inference speed, its high accuracy made it more advantageous in MC detection tasks that require high reliability. In particular, on the 20,000 sample dataset, the memory usage of DTL-MD was 2008.3 MB and the computational complexity was 47.1 GFLOPS, showing its ability in computationally intensive tasks. The feature fusion-based method proposed by Wang et al. in the literature [8] showed a high accuracy in MC detection, but its computational complexity was high and the training and inference speeds were slow. In contrast, DTL-MD optimized computational complexity while maintaining high accuracy, making it still scalable in large data sets and real-time detection scenarios. In application tests, DTL-MD also performed well in the robustness of MC types such as Trojans, viruses, worms, and adware. Especially in low-bandwidth environments, DTL-MD had a throughput of 10 tasks/s and a latency fluctuation of only 15 ms, which was suitable for real-time MC detection.

The advantage of DTL-MD is that it is highly adaptable and can handle small data sets. It can also maintain good detection performance when there are fewer samples, showing strong generalization ability.

5 Conclusion

The study proposed an MC detection model DTL-MD that combines OTL and optimized feature selection strategies, and verified its effectiveness in MC detection. However, the model's throughput and latency volatility are high, and its real-time performance is poor in low-resource environments. In addition, its high computing requirements make its application on large-scale data sets face computing cost issues. In the future, the study will explore lightweight models, optimize the calculation process, and improve its computing efficiency through efficient feature extraction and pruning techniques to solve the problem of high computing overhead in large-scale data sets. Meanwhile, the experiment used a data set containing many known malicious samples. In the future, the study will introduce new MC samples to further verify the model's capabilities, especially its performance when detecting new samples.

Funding

This work was supported by the Natural Science Foundation of Ningxia Hui Autonomous Region in 2022, project number: 2022AAC03345.

References

[1] Wang R, Gao J, Huang S. AIHGAT: A novel method of malware detection and homology analysis using assembly instruction heterogeneous graph. International Journal of Information Security, 2023, 22(5): 1423-1443. DOI: 10.1007/s10207-023-00699-7

[2] Li F, Ren J. Suppression of MC Propagation in software-defined networking. Wireless Personal Communications, 2024, 135(1): 493-516. DOI: 10.1007/s11277-024-11065-8

[3] Liu T, Neware R, Bhatt M W, Shabaz M. A study on detection and defence of MC under network security over biomedical devices. The Journal of Engineering, 2022, 2022(11): 1041-1049.

[4] Dam K H T, Touili T. Extracting malicious behaviours. International Journal of Information and Computer Security, 2022, 17(3): 365-404. DOI: 10.1049/tje2.12153

[5] Cui Z, Zhao Y, Cao Y, Cai X, Zhang W, Chen J. Malicious code detection under 5G HetNets based on a multi-objective RBM model. IEEE Network. 2021, 35(2): 82-87. DOI:10.1109/MNET.011.2000331.

[6] Kim H W. A study on countermeasures by detecting trojan-type downloader/dropper MC. International Journal of Advanced Culture Technology, 2021, 9(4): 288-294. DOI: 10.17703/IJACT.2021.9.4.288

[7] Kim J, Lee S. Malicious behavior detection method using API sequence in binary execution path. Tehni Čki Vjesnik, 2021, 28(3): 810-818. DOI: 10.17559/TV-20210202132203

[8] Wang Z, Wang W, Yang Y, Han Z, Xu D, Su C. CNN-and GAN-based classification of MC families: a code visualization approach. International Journal of Intelligent Systems, 2022, 37(12): 12472-12489. DOI: 10.1002/int.23094

[9] Li S, Jiang L, Zhang Q, Wang Z, Tian Z, Guizani M. A malicious mining code detection method based on multi-features fusion. IEEE Transactions on Network Science and Engineering. 2022, 10(5):2731-2739. DOI:10.1109/TNSE.2022.3155187.

[10] Li H, Jin Y, Chai T. Evolutionary multi-objective Bayesian optimization based on multisource online transfer learning. IEEE Transactions on Emerging Topics in Computational Intelligence, 2023, 8(1): 488-502. DOI: 10.1109/TETCI.2023.3306351

[11] Cherifi D, Djaber A, Guedouar M E, Feghoul A, Chelbi Z Z, Ouakli A A. Covid-19 detecting in computed tomography lungs images using machine and transfer learning. Informatica. 2023, 47(8). DOI: 10.31449/inf.v47i8.4258

[12] Cui Z. Combining the SSD target identification algorithm with the 3D-CNN architecture for transfer learning research in basketball training. Informatica. 2024, 48(18). DOI: 10.31449/inf.v48i18.6454

[13] Qin P, Zhao L. An online transfer learning framework for cell SOC online estimation of battery pack in complex application conditions. IEEE Transactions on Transportation Electrification, 2023, 10(3): 5974-5986. DOI: 10.1109/TTE.2023.3324822

[14] Lu H, Jin C, Helu X, Du X, Guizani M, Tian Z. DeepAutoD: Research on distributed machine learning oriented scalable mobile communication security unpacking system. IEEE Transactions on Network Science and Engineering, 2021, 9(4): 2052-2065. DOI: 10.1109/TNSE.2021.3100750

[15] Khan S, Nauman M. Interpretable detection of malicious behavior in windows portable Executables using Multi-Head 2D transformers. Big Data Mining and Analytics, 2024, 7(2): 485-499. DOI: 10.26599/BDMA.2023.9020025

[16] Gurjar A, Voditel P. Transfer learning: a paradigm for machine assisted knowledge transfer. ECS Transactions, 2022, 107(1): 7179-7188. DOI: 10.1149/10701.7179ecst

[17] Dai S, Meng F. Addressing modern and practical challenges in machine learning: A survey of online federated and transfer learning. Applied Intelligence, 2023, 53(9): 11045-11072. DOI: 10.1007/s10489-022-04065-3

[18] Zhu Z, Lin K, Jain A K, Zhou J. Transfer learning in deep reinforcement learning: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2023, 45(11): 13344-13362. DOI: 10.1109/TPAMI.2023.3292075

[19] Solís M, Calvo-Valverde L A. Performance of deep Learning models with transfer learning for multiple-step-ahead forecasts in monthly time series. Inteligencia Artificial-Iberoamerical Journal of

Artificial Intelligence, 2022, 25(70): 110-125. DOI: 10.48550/arXiv.2203.11196

[20] Belouadah, E, Adrian P, Ioannis K. A comprehensive study of class incremental learning algorithms for visual tasks. Neural Networks, 2021, 135: 38-54. DOI: 10.1016/j.neunet.2020.12.003

[21] Minoofam S A H, Bastanfard A, Keyvanpour M R. TRCLA: a transfer learning approach to reduce negative transfer for cellular learning automata. IEEE transactions on neural networks and learning systems, 2021, 34(5): 2480-2489. DOI: 10.1109/TNNLS.2021.3106705