electronics

MDPI

Article

MalOSDF: An Opcode Slice-Based Malware Detection Framework Using Active and Ensemble Learning

Wenjie Guo [1], Jingfeng Xue [1], Wenheng Meng [1], Weijie Han [2], Zishu Liu [1], Yong Wang [1] and Zhongjun Li [1,*]

[1] School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China; 3120215536@bit.edu.cn (W.G.); xuejf@bit.edu.cn (J.X.); 3220221066@bit.edu.cn (W.M.); wangyong@bit.edu.cn (Y.W.)
[2] School of Space Information, Space Engineering University, Beijing 101416, China
* Correspondence: leezj@bit.edu.cn

Abstract: The evolution of malware poses significant challenges to the security of cyberspace. Machine learning-based approaches have demonstrated significant potential in the field of malware detection. However, such methods are partially limited, such as having tremendous feature space, data inequality, and high cost of labeling. In response to these aforementioned bottlenecks, this paper presents an Opcode Slice-Based Malware Detection Framework Using Active and Ensemble Learning (MalOSDF). Inspired by traditional code slicing technology, this paper proposes a feature engineering method based on opcode slice for malware detection to better capture malware characteristics. To address the challenges of high expert costs and unbalanced sample distribution, this paper proposes the SSEAL (Semi-supervised Ensemble Active Learning) algorithm. Specifically, the semi-supervised learning module reduces data labeling costs, the active learning module enables knowledge mining from informative samples, and the ensemble learning module ensures model reliability. Furthermore, five experiments are conducted using the Kaggle dataset and DataWhale to validate the proposed framework. The experimental results demonstrate that our method effectively represents malware features. Additionally, SSEAL achieves its intended goal by training the model with only 13.4% of available data.

Keywords: malware classification; opcode slice; active learning; ensemble learning

check for updates

Citation: Guo, W.; Xue, J.; Meng, W.; Han, W.; Liu, Z.; Wang, Y.; Li, Z. MalOSDF: An Opcode Slice-Based Malware Detection Framework Using Active and Ensemble Learning. *Electronics* **2024**, *13*, 359. https://doi.org/10.3390/electronics13020359

Academic Editor: Aryya Gangopadhyay

Received: 10 December 2023
Revised: 11 January 2024
Accepted: 12 January 2024
Published: 15 January 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In an era of unprecedented expansion in the digital landscape's gray areas, the proliferation of numerous threats has reached an alarming scale. According to Kaspersky, there has been a 20% increase in detected malware attack attempts, surpassing 74.2 million incidents in 2022 compared to 61.7 million in the previous year. By 2025, it is projected that the total volume of human-generated data will reach a staggering 175 ZB [1]. This surge in data and evolving digital threats underscores the growing significance of network security. The relentless pace of malware evolution, characterized by the continuous emergence of new versions and families, poses a formidable challenge for cybersecurity professionals.

Malware not only adapts to its environment but also enhances its anti-detection capabilities through advanced techniques such as obfuscation, encryption, and shell usage, thereby posing significant challenges for cybersecurity. Traditional methods, including classic virus signature databases and heuristic scanning, have encountered limitations due to their inability to effectively combat malware while exhibiting high false-positive rates. As a result, these methods have struggled to cope with the current state of network security. The evolving threat landscape has prompted the emergence of machine learning as a critical tool in malware detection [2]. Machine learning techniques have proven their ability to analyze known malware samples, extract discriminative features, and accurately classify these samples [3,4].

The essence of effective malware detection lies in feature engineering [5]. Feature engineering aims to extract the intrinsic attributes that are most likely to be used to distinguish malicious software from benign software in PE files, and then generate corresponding digital features for representation. Feature engineering generally includes the analysis, definition, extraction and other steps of features. With accurate feature selection, the detection engine can capture the deep and unique features of malicious software, providing strong support for subsequent malicious code detection. Features typically encompass both static and dynamic attributes. In the real physical world, initiating the dynamic analysis of unknown software can be challenging, rendering static analysis the most immediate and expeditious method for initial assessment. Moreover, from an efficiency perspective, static features remain the industry's preferred choice for detection [6]. While existing static structures such as Data Flow Graphs (DGs) [7], Control Flow Graphs (CFGs) [8], and Function Call Graphs (CGs) [9] can encompass a substantial amount of semantic information, they suffer from significant space consumption. Furthermore, even minor alterations in the source code can result in substantial variations in the extracted graph features [10]. Therefore, the extraction of features from opcodes remains the most common and efficient approach [11].

Machine learning-based methods not only require a large amount of computing resources to train models but also have high data requirements [12]. The distribution of benign and malware samples in the real world exhibits significant imbalances. Simultaneously, the rapid evolution of potential and unknown samples occurs at an extraordinary pace. Despite the existence of platforms such as VirusShare, which provides an extensive repository of malware samples, machine learning-based models for malware code detection may still result in false negatives when they fail to adequately learn the characteristics of malware samples. On the classic Win32 Platform, disassembling malware generates more than 800 different instructions, where there is significant semantic duplication and redundancy among them. Therefore, generating features solely based on all assembly instructions is not the most efficient method. Features corresponding to invalid instructions and redundant instructions can reduce detection efficiency and even lead to overfitting problems. Some existing methods [13] have performed feature selection during feature engineering to reduce dimensionality. However, this practice may result in the loss of some semantic information that could affect the accuracy of the detection results.

Additionally, current malware detection engines [14] demonstrate effective performance in detecting known sample types. However, they face challenges when dealing with emerging families and unknown types of malware. Moreover, the high cost associated with human judgment needs to be considered. Therefore, the current research focus lies in designing a detection model that can accurately operate with minimal known labeled samples. This model should possess the capability to withstand label scarcity and the unequal distribution of types. This challenge limits the effectiveness of existing methods, necessitating the exploration of solutions to overcome this bottleneck.

Why is the opcode slice defined by this work? Firstly, the concept of slicing, referred to as program slicing, was initially proposed by Mark D. Weiser in the 1980s for debugging and modifying source code [15]. With technological advancements, the scope of slicing has gradually expanded from static to dynamic analysis and from forward to backward traversal [16], encompassing a single process to multiple processes and non-distributed to distributed programs. Application scenarios also include software debugging, testing, maintenance, reconstruction, and security purposes [17]. Additionally, classic objects for slicing include data streams, information flows, and dependency graphs [18], which involve handling control flows, composite data types, and pointers. The ability to identify the behavioral points of malware, such as data transfers, process comparisons, flow control, program control, loop control, and other operations, offers malware analysts a clearer understanding of the intent behind the malware. On the other hand, detection engines primarily focus on feature engineering that can extract and quantify the critical behavioral points of each malware.

Despite extensive research on malware detection techniques based on semantic information, researchers continue to face several challenging issues:

- Existing deep learning methods based on opcodes demand embedding all instructions, which can be time consuming. Moreover, the extensive variety of opcodes, some of which lack meaningful semantic information, can protract model construction.
- Malware evolves at a rapid pace, making it challenging to obtain accurate labels for the latest, real-world malware samples. The central research concern revolves around employing a minimal set of labeled samples for effective detection—a pivotal issue for practical engineering applications.
- Current detection engines unavoidably grapple with false positives and false negatives when confronted with previously unknown types of malware. These challenges are intrinsically linked to human expert analysis and judgment. Consequently, it is imperative to consider the associated costs of manual detection.

This paper introduces the MalOSDF framework designed to address the need for efficient and rapid feature extraction from malware samples and develop a resilient malware detection engine capable of identifying unknown malware types. Specifically, this work presents an opcode slice-based feature engineering method and Semi-supervised Ensemble Active Learning (SSEAL) malware detection algorithm. The opcode slice-based feature engineering method conducts semantic aggregation, effectively reducing feature dimensionality. Simultaneously, malicious samples are embedded with semantic information to resolve the issue of sparse features and dimensionality explosion associated with the one-hot encoding of all opcodes [19]. The MalOSDF malware detection method employs the principles of semi-supervised learning, and utilizes active learning and ensemble learning techniques. This approach enhances the quality of knowledge extraction and learning for model training while addressing the limitations of classical machine learning models in detecting unknown categories of malware and their vulnerability to noisy data.

The contributions of this article include the following points:

- In this paper, the opcode slice-based feature engineering method is proposed to reduce dimensions efficiently.
- This work presents the SSEAL approach, which effectively addresses the limitations of having extensive dataset requirements and encourages a more comprehensive exploration of sample knowledge.
- This paper uses the Kaggle dataset for experiments and evaluates the effectiveness of the proposed framework.

The rest of the paper is organized as follows. Section 2 surveys the related work in the field of malware detection. Section 3 describes the framework of MalOSDF. Section 4 evaluates its performance and compares it with similar studies. Section 5 discusses this work. Section 6 concludes our findings in the paper.

2. Related Work

Malware detection can be considered a classic feature engineering process, which includes feature definition, feature extraction, and feature detection [20]. Malware detection methods can be categorized into syntax-based and semantic-based approaches. The semantic-based approach provides stable support for interpretability.

MalInsight [21] is proposed by profiling malware from three aspects, which are the basic structure, low-level behavior, and high-level behavior. And the importance of the three aspects is evaluated and sorted, quantitatively demonstrating that these aspects have the same effects with the optimal feature set. Han et al. proposed MalDAE [22], which correlates and fuses dynamic and static API sequences into one hybrid sequence based on semantics mapping and constructs the hybrid feature vector space. MalDAE gives an understandable explanation for common types of malware and provides predictive support for understanding and resisting malware. Inspired by this method, this work believes that we can focus on selecting opcode with strong maliciousness for slicing.

Huang et al. [23] proposed EAODroid, an approach based on the enhanced API order for Android malware detection, which learns the similarity of system APIs from a large number of API sequences and groups similar APIs into clusters. The extracted API clusters are further used to enhance the original API calls executed by an app to characterize the behaviors and perform classification. The method of clustering similar APIs provides us with inspiration, that is, we can define corresponding slices for similar opcode.

The issue of feature redundancy is addressed by Kong et al. [24] through the utilization of mutual information-based feature selection techniques. This approach effectively reduces over 900 features to 64 dimensions while incorporating sample row and size characteristics, thereby achieving efficient feature detection. However, extensive feature selection based on mutual information consumes a substantial amount of time. Our intuition is to directly extract features from a semantic perspective, which can offer a more rapid and potentially even more interpretable alternative. Therefore, this paper endeavors to define feature slices from a semantic standpoint as the basis for subsequent feature engineering.

In the feature detection phase, machine learning has been widely adopted by numerous scholars as the primary technical approach due to its powerful data mining capabilities [25]. The field of machine learning encompasses various subfields, including supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Both supervised and unsupervised learning require labeled data for model training.

The conventional approach in existing semi-supervised machine learning methods typically involves assigning pseudo-labels to unlabeled data that closely resemble the distribution of existing labeled data. However, this practice may lead to potential model detection failures when confronted with unknown samples. The acquisition of malicious samples varies across different scenarios, resulting in imbalances within the sample distribution. Addressing this challenge is crucial during the model training process, prompting researchers to contemplate effective strategies for training models using an unbalanced and limited amount of malware samples.

Renato et al. [26] proposed an iterative data preprocessing method capable of increasing the separation between clusters. Unlike other methods, it iteratively favors more meaningful features. Wang et al. proposed SIMPLE [27], a few-shot malware classification approach that utilizes a multi-prototype modeling technique to generate multiple prototypes for each malware family, thereby enhancing its generalization capacity based on observations derived from dynamic analysis of API call sequences. Gao et al. proposed MaliCage, a packed malware family classification framework based on DNN and GAN. MaliCage consists of three core modules: a packer detector, a malware classifier, and a packer generative adversarial network (GAN). This method effectively overcomes the bottleneck caused by an insufficient sample size.

Numerous scholars have conducted research in the realm of the cost-effective acquisition of labeled data, employing technical and mathematical methods. This domain is referred to as active learning, which involves human intervention in the labeling process during the training of detection models. Annotated samples contain knowledge that is more amenable to exploration, thereby enhancing the model's capabilities.

The paper argues that samples which can be easily misclassified by existing detection engines actually contain more information that is helpful for improving the accuracy of the detector. In other words, these challenging-to-categorize samples may possess crucial features. If these features can be correctly identified and utilized, they have the potential to enhance the performance of the detector. The work presented in [28] focuses on the anomaly detection domain and introduces an active learning-based approach. The crucial aspect of this study lies in the fact that data points located at the classification boundaries of detection engines are likely to possess a wealth of untapped information, particularly when making determinations about unknown samples. In machine learning models that require extensive training data, accurately labeling these data points holds paramount importance for enhancing the model's detection performance. Consequently, active learning becomes exceptionally vital in such scenarios.

3. Methdology

To tackle the above challenges, this section provides an overview of the MalOSDF framework. Specifically, this paper presents a comprehensive assembly slicing approach to characterize malicious behavior and proposes a feature engineering method for the efficient embedding and optimal utilization of semantic information. Additionally, this section outlines the SSEAL detection algorithm that considers the real-world expert costs associated with addressing threats posed by rapidly malware evolving.

3.1. MalOSDF Overall Architecture

This work introduces a malware detection framework based on opcode slice as depicted in Figure 1. It consists of two main components: the feature engineering method and the SSEAL algorithm. In this section, we provide a brief description for each step as follows:

1. Disassembly: Decompiling binary code.
2. Feature Definition: Based on the semantic analysis of assembly instructions, we customarily define malware opcode slice. Assembly program slicing refers to the statements or expressions in assembly programs that affect specified variables within the sample, leading to the creation of a mapping dictionary.
3. Feature Engineering: Data preprocessing is performed on the samples, and based on predefined opcode slice types, opcode slice sequences are extracted from the assembly programs of malware.
4. Generating Feature Matrices: The statistical features are calculated for each malware sample based on the opcode slice sequences. Specifically, this method counts the occurrences of each slice and N-gram statistically, thereby generating the feature matrix of the set of malware samples.
5. Malware Detection: Using the proposed SSEAL algorithm, a classifier is trained based on the feature vectors of malware samples.

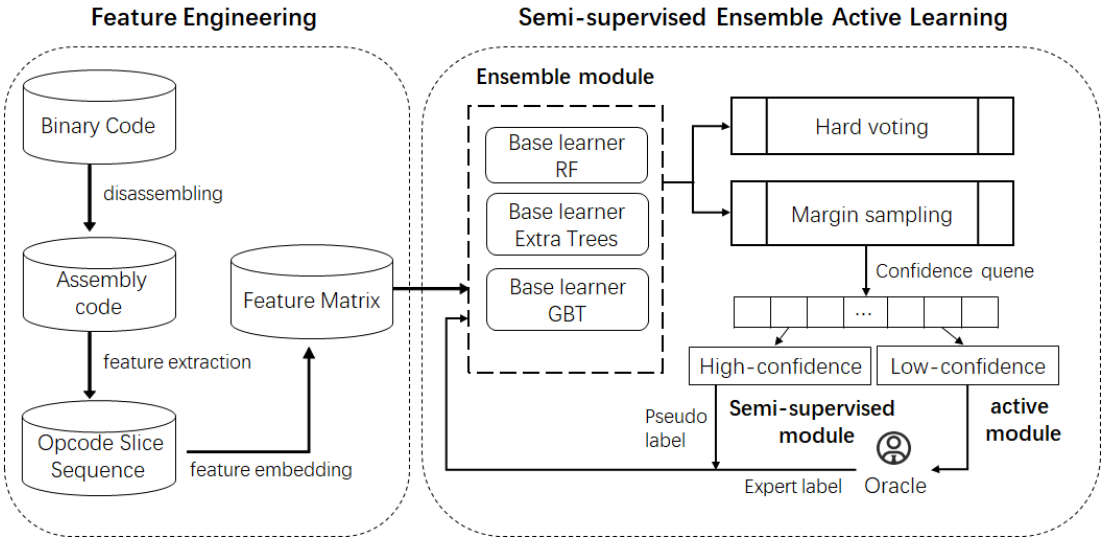

Figure 1. MalOSDF overall architecture.

The classifier utilized for malware detection consists of two key modules, which will be introduced in Section 3.3 in detail. For the Ensemble Learning Module, ensemble classifiers are generated through a hard voting mechanism, including Random Forest (RF), Extra Trees (ET), and Gradient Boosting Trees (GBT). And the Semi-Supervised Active Learning Sample Sampling Module handles the semi-supervised active learning process.

This paper addresses the challenge of managing a large number of features in traditional machine learning-based malware detection. This work proposes a method based on opcode slice construction for generating static features of malware. This approach not

only conserves significant computational resources but also reduces the time required for malware detection, thereby providing a robust foundation for efficient detection.

Furthermore, this framework introduces a Semi-supervised Ensemble Active Learning algorithm that not only enhances the efficiency of training multi-class models but also reduces the training time, mitigates the impact of noisy data, tackles the issue of imbalanced sample distribution, and equips the system with the capability to identify unknown family samples of malware.

3.2. Feature Engineering

From the above analysis, this section proposes an opcode slice-based feature engineering method. Specifically, we give the definition of opcode slices and describe the method of features embedding.

3.2.1. Opcode Slice Definition

After disassembling the malware binary file, the assembly instruction is obtained. An assembly instruction generally consists of two parts: the operation code and the operation number. Building upon classical instructions that encompass data transfer, arithmetic operations, logical operations, string operations, program control, input–output operations, processor control, privilege instructions, and system function call instructions, this invention leverages the distinctive semantics of opcode in malware assembly instructions. Furthermore, we explore the inclusion of additional semantic information from the operands associated with opcodes, such as pointers, variable values, function addresses, and memory details, to enhance the feature engineering process.

Specifically, after extracting all the opcodes and operands, we first normalize and standardize the operands according to the method [29] shown in Table 1. (1) Immediate numbers are divided into function call, jump address, reference, and default classes based on their purpose. Calling functions are further classified as standard library function calls, recursive calls, internal calls (within the same file), and external calls (in different files). Jump addresses represent jumps between basic blocks within a function. References distinguish between string references, static variable references (in the program's global data area or static data area), and data references (in the program's data segment or stack). Other immediate numbers are not differentiated. (2) Registers are classified into the following categories: The flag registers, such as control registers, debugging registers, floating-point registers and segment registers, are classified into one class. In the special purpose register, the stack pointer register, base register and instruction pointer register are grouped into one class. Except for the aforementioned registers, all others are classified as general-purpose registers and further differentiated based on the number of bytes they can hold (such as 1 byte, 2 bytes, 4 bytes, 8 bytes, etc). (3) Pointers are divided into direct addressing and indirect addressing based on their addressing method. Direct addressing is further categorized into pointers less than 8 bytes in size and pointers greater than 8 bytes in size. Indirect addressing is divided into string pointers and other pointers according to their pointing object class. The second step is to sort the importance of operation instructions based on TF-IDF (term frequency–inverse document frequency) statistical analysis. Then, semantic analysis is performed on opcodes with a TF-IDF value greater than 1 to obtain the opcode slices. Our work customizes the corresponding opcode slices for frequently occurring opcodes within malware by categorizing 157 commonly appearing assembly instructions into 35 distinct opcode slices as shown in the Table 2.

Table 1. Normalization rules for instruction operands: immediate, register and pointer [29].

Operand	Categories	Normalized Form
Immediate	function call jump address reference default	libc[funcname], self, innerfunc, externfunc jmpdst dispstr, dispbss, dispdata immval
Register	size stack/base/instruction special purpose	reg[1 | 2 | 4 | 8] [s | b | i]p[1 | 2 | 4 | 8] reg[cr | dr | st], reg[c | d | e | f | s]s
Pointer	direct indirect	memptr [base + index×scale + dispstr]

Table 2. Opcode slice definition.

Slice Label	Opcode Contained	Slice Label	Opcode Contained
Data_Transfer	mov, movsx, movzx	Arithmetic_Div	div, idiv
Data_Swap	xchg, xlat, bswap	Logical_Operation	and, or, not, xor
Stack_Operation	push, pop, pusha, pushad, popa, popad	Test	test
Address_Transmission	lea, lds, lss, les, lfs, lgs	Bit_Test	bt, bts, btr, btc
Flag_Transfer_Ah	lahf, sahf	Bit_Scan	bsf, bsr
Flag_Transfer_Stack	pushf, pushfd, popf, popfd	Shift_Operation	shl, shr, sal, sar, rol, ror, rcl, rcr, shld, shrd
Type_Conversions	cbw, cwd, cwde, cdq, bswap	Unconditional_Tran	jmp
String_Operation	movs, movsb, movsw, movsd	Conditional_Tran	jz, jnz, je, jne, js, jns, jo, jno, jp, jpe, jnp, jpo, jc, jnc, jb, jnb, jae, jnae, jl, jnge, jnl, jge, jle, jng, jnle, jg, ja, jnbe, jna, jbe
String_Storage	stos, stosb, stosw, stosd	Conditional_Transfer	jcxz, jecxz
String_Reads	lods, lodsb, lodsw, lodsd	Loop_Control	loop, loopz, loope, loopnz, loopne
String_Comparison	cmps, cmpsb, cmpsw, cmpsd	Call	call
String_Scan	scas, scasb, scasw, scasd	Return	ret
Arithmetic_Add	add, adc, inc	Interrupt	int, iret
Arithmetic_Sub	sub, sbb, dec	Repeat	rep, repe, repz, repne, repnz, irp, irpc
Arithmetic_Neg	neg	Basic_Input_Output	in, out
Compare	cmp	String_Input_Output	ins, ins, insb, insw, insd, outs, outsb, outsb, outsw, outsd
Arithmetic_Mul	mul, imul	Flag	clc, cmc, stc, cld, stc, cli, sti
Processor	nop, hlt, wait	Privilege	sgdt, lsi, invd

3.2.2. The Process of Feature Engineering

Regarding the section on assembly program feature representation, after obtaining the opcode slices, our method computes their statistical features. This involves calculating the N-gram statistical features based on the opcode slices for each malware sample. These features serve as the feature vectors for the assembly program training samples. The specific workflow is shown in Figure 2:

1. Malware samples' disassembled opcodes are sequentially read, and opcode slices are extracted based on predefined opcode slice definitions, ignoring undefined opcodes.
2. The occurrences of opcode slice N-grams are counted and sorted for N = 1, 2, 3.
3. The top-k opcode slices N-grams are selected as features.
4. The N-gram statistical features are computed for each assembly sample, forming individual sample feature vectors.
5. A feature matrix is constructed for the code sample dataset.

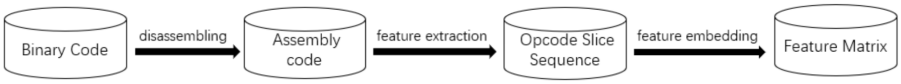

Figure 2. The process of feature engineering.

For example, we can use $s_1 \sim s_m$ to mark assembler training samples.

Define the top-k opcode slice n-gram as follows: $S_i, i \in (1\ n)$. Given the assembler sample s_p of a malware, count the number of Si occurrences as $N_{p,i}$. Then, the feature vector of sample s_p is defined as Equation (1), where $label_p$ is the label of the assembler sample $malware_p$:

$$\vec{F}_{pj} = \{\ N_{p,1} \quad \dots \quad N_{p,i} \quad \dots \quad N_{p,n} \quad label_p\ \} \tag{1}$$

Defining the feature matrix as a collection of feature vectors, then the feature matrix for p assembly program samples is shown as Equation (2):

$$FM = \begin{Bmatrix} N_{1,1} & \dots & N_{1,i} & \dots & N_{1,n} & label_1 \\ N_{p,1} & \dots & N_{p,i} & \dots & N_{p,n} & label_p \\ N_{m,1} & \dots & N_{m,i} & \dots & N_{m,n} & label_m \end{Bmatrix} \tag{2}$$

The various dimensions (columns) of N in the figure above represent the frequency of corresponding slice types. Subsequently, we can use the feature matrix FM derived from the assembly program samples to train the subsequent detection engine.

3.3. Semi-supervised Ensemble Active Learning

One of the challenges that semi-supervised learning algorithms need to address is the introduction of a significant amount of noisy samples during the training process, which can hinder the model from learning the correct information. SSEAL alleviates the issue of noisy samples by employing a collaborative training algorithm for multiple classifiers, which is an ensemble learning approach.

Active learning methods enable a more efficient identification of samples that contain valuable information within the dataset. Expert queries also equip the model with the capability to detect unknown samples. It is important to note that, for the purpose of analysis, we assume that all labels provided by experts are reliable.

This algorithm includes the following modules in sequence.

1. Ensemble Learning Module

   The integrated learning module includes multiple base learners. In this method, three base learners are designed, namely RF, Extra Trees, and GBT. Each sample is inputted to different base learners to obtain the probabilities of the test samples belonging to different categories. Simultaneously, the detection results of each base learner are outputted, and the final results of the test samples are obtained by hard voting from the three base learners.

   Let us delve deeper into the three base learners involved. Each of these base learners possesses unique characteristics and operating mechanisms, collectively forming the framework of our integrated learning module (Figure 3).

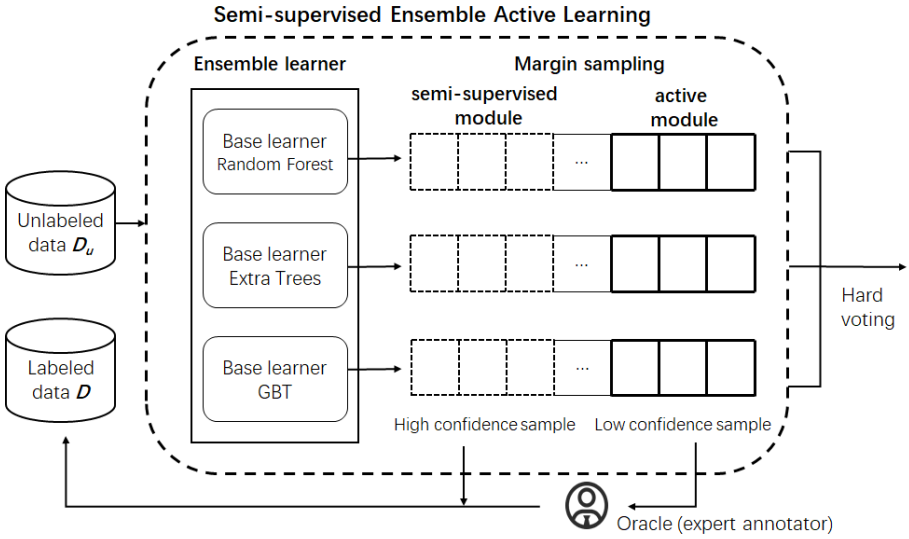

Figure 3. The process of feature engineering.

Random Forest (RF): Random Forest is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. It enhances the performance of a single decision tree by introducing randomness in the feature selection and bootstrap sampling, thereby improving robustness and generalization.

Extra Trees (ET): Extra Trees, or Extremely Randomized Trees, is another ensemble learning method that builds multiple decision trees and combines their predictions. Similar to Random Forest, it introduces randomness in the feature selection process, but Extra Trees takes it a step further by using random thresholds for each feature rather than searching for the best split points. This additional randomness can lead to increased diversity among the trees and potentially better generalization.

Gradient Boosting Trees (GBT): Gradient Boosting is an ensemble technique where weak learners (typically shallow decision trees) are combined to create a strong learner. GBT builds trees sequentially, with each tree correcting the errors made by the previous ones. It minimizes a loss function by adding weak learners, which allows it to capture complex relationships in the data and achieve high predictive accuracy. Gradient Boosting Trees are particularly effective in handling diverse and non-linear patterns in the data.

2. Semi-supervised active learning module

The integrated learning module calculates the detection confidence value for each sample by determining the probability that it belongs to different categories as output by the three classifiers. Samples with high and low confidence levels are marked according to preset screening criteria, which will be described in detail below. The active learning component transfers low-confidence samples to an expert marker, who inputs labeled codes into the tagged dataset. In the semi-supervised learning part, firstly, count the maximum value Num_max of different types of samples in the labeled sample set. Then obtain the corresponding pseudo tags for these high-confidence samples and calculate the difference Num_aug between this category and Num_max. Finally, select the high-confidence samples with the maximum Num_aug from this category to add them to the labeled dataset.

Uncertainty sampling involves the extraction of samples that are challenging for the model to distinguish, which are then provided for expert annotation. These hard-to-distinguish samples contain valuable knowledge that can significantly enhance algorithmic detection. The key here is quantifying the model's difficulty in differentiation. Classic approaches include least confident, margin sampling, and entropy methods. Margin sampling selects samples that are almost equally likely to be classified into two categories, meaning

the difference in the model's probabilities for these data points is minimal. Specifically, margin sampling chooses samples with the smallest difference between the highest and second-highest predicted probabilities. In the context of multi-class malware detection in this paper, margin sampling is found to be more effective for model training.

The newly added dataset with tags needs to be removed from the original test set, which is the unlabeled set. Then, it should be re-entered into Module 1 to update the model until reaching the specified number of iterations or when the overall tag ratio reaches the threshold. At that point, the loop stops, and we obtain the final training model and classification results.

The overall algorithm is shown in Algorithm 1.

Algorithm 1: Semi-supervised Ensemble Active Learning (SSEAL).

Input: unlabeled data D_u, labeled data D, classifiers *RF,Extra Trees and GBT*, confidence threshold θ.
Output: trained models *RF,Extra Trees,GBT.*

```
// Ensemble learning
foreach x_i in D_u do
    prob_rf ← RF(x_i)
    prob_et ← ExtraTrees(x_i)
    prob_gbt ← GBT(x_i)
end
pseudo label pl_i ← vote(prob_rf, prob_et, prob_gbt)
// Semi-supervised active sampling
foreach x_i in D_u do
    conf_rf ← RF(x_i)
    conf_et ← ExtraTrees(x_i)
    conf_gbt ← GBT(x_i)
    if conf<θ then
        // Active learning
        get expert label el_i
        add <x_i, el_i> to data Temp
        add data Temp to labeled data D
        remove data Temp from unlabeled data D_u
    else
        // Semi-supervised Learning
        num_max ← max(num_(class_1), num_(class_2), ..., num_(class_n))
        if num(pl_i) < num_max then
            add <x_i, pl_i> to data Temp
            add data Temp to labeled data D
            remove data Temp from unlabeled data D_u
        else
        end
    end
end
```

In particular, each base learner outputs the probability of classification results for each sample. Then, the edge sampling margin sampling is carried out to calculate the probability difference between the largest and second categories. It is called reliability, shown in the following formula:

$$Conf_x = margin(x) = P_{y_1} - P_{y_2} \tag{3}$$

The classifier has a low confidence in the sample, which indicates that the sample contains more mining knowledge and is more useful for model training. Then, we select M samples with a mining value in turn as shown in the following formula:

$$x_M^* = \text{argmin}_x(P_\theta(\hat{y}_1 \mid x) - P_\theta(\hat{y}_2 \mid x)) \quad (4)$$

The algorithm utilizes the pool-based active learning method. Pool-based active learning allows for labeling by experts, enabling the algorithm framework to identify unknown samples. Specifically, the tested sample queues are sorted from high to low to obtain a confidence queue. Experts are then contacted to manually label samples with low confidence levels. Samples with higher reliability can be used to enrich and balance datasets. The classification results of all classifiers are unified, that is, the confidence level is high, and pseudo tags are directly added. After the newly obtained pseudo-label sample is selected, it is initially qualified to be added to the training set. However, the sample selection strategy also needs to consider the sample equilibrium situation. This algorithm sets that if the malware sample type already occupies the largest distribution, it will not be added to the training set.

The above measures make the training dataset of the model balanced and enable the model to obtain the ability to resist noise data.

4. Experiment and Analysis

In this section, we provide details about the experiments conducted to evaluate the proposed method for disassembling binary code and its application in malware detection. We start by introducing the dataset used in the experiments and then proceed to discuss the experimental setup.

4.1. Dataset

To disassemble binary code, this work employs IDA Pro 6.4 for the disassembly of binary source files. This paper uses Microsoft's Kaggle dataset [30] and Intel's DataWhale dataset for experiments. The distribution of various types in the dataset is shown in the following Table 3.

Table 3. Distribution of samples in experimental dataset Kaggle.

Family	Kaggle	DataWhale
1 (Ramnit)	1541	385
2 (Lollipop)	2478	598
3 (Kelihos_ver3)	2942	784
4 (Vundo)	475	6641
5 (Simda)	42	5676
6 (Tracur)	751	7563
7 (Kelihos_ver1)	398	7560
8 (Obfuscator.ACY)	1228	11,368
9 (Gatak)	1013	9425
Total	10,868	50,000

4.2. Experiment Setup

4.2.1. Experimental Environment

The runtime environment of the experiment is (1) Intel(R) Core(TM) i7-10870H CPU @ 2.20 GHz, 16 GB memory, (2) Ubuntu 18.04 (64 bit).

4.2.2. Experimental Design

In order to verify the effectiveness of the proposed method, we designed experiments to verify the effectiveness of the proposed feature engineering and SSEAL. Specifically, the following five types of experiments are designed in this paper. Reducing the cost of

data labeling is one of the core focuses of our research. Therefore, out of a total of over 10,000 samples, we select only 100 labeled samples as initial data. The model obtains 50 semi-supervised learning samples and 50 active learning samples, respectively, in each iteration as supplements to the training set.

(1) To demonstrate that feature engineering can effectively reflect the characteristics of different malicious code families, we compare the performance of traditional machine learning methods using features of varying dimensions.
(2) To verify that SSEAL is more robust than a single classifier, we compare SSEAL with a single classifier. This work observes an accuracy trend of SSEAL and single classifiers as the number of iterations increases. The single classifiers includes Random Forest, GBT and Extra Trees. There is no difference in the sampling strategy between SSEAL and base classifiers except that the base classifier only uses its own queue to filter low-confidence samples and high-confidence samples.
(3) To verify the effect of SSEAL under different dimensional features, we compare SSEAL algorithms in different N-gram dimensions.
(4) To verify the ability of SSEAL to detect unknown malicious samples, this paper observes its performance on unknown malware that was not included in the initial labeled dataset.
(5) To verify the ability of SSEAL to detect unbalanced malware datasets, this study compared the performance of SSEAL with different numbers of labeled samples for the initial training. We set 50, 100, and 200 samples in the initial labeled sample set, and each experiment performed 20 iterations with 50 samples queried in each iteration to evaluate the performance of SSEAL by observing the accuracy of the model in each iteration.

4.3. Results and Analysis

4.3.1. Comparison of Machine Learning Methods Across Varying Feature Dimensions

In order to evaluate the feature engineering method proposed in this paper, this paper adopts classifiers such as Random Forest, decision tree, nearest neighbor classification and extreme gradient lifting tree to carry out the experiments. The results of the classification detection based on traditional classifiers are shown in Table 4.

Table 4. Detection results of traditional classifiers.

		Random Forest	**Decision Tree**	**KNN**	**XGBoost**
Kaggle	Accuracy	97.93%	96.23%	96.04%	97.38%
	precision	97.38%	94.32%	95.93%	95.19%
	Recall	93.12%	91.52%	91.39%	91.43%
	F1 score	97.59%	92.52%	92.96%	92.69%
DataWhale	Accuracy	98.11%	97.28%	96.19%	97.13%
	precision	97.70%	94.81%	93.06%	92.74%
	Recall	94.34%	94.06%	90.58%	91.55%
	F1 score	95.36%	94.02%	91.39%	92.38%

As shown in Table 4, the accuracy of using the Random Forest classification algorithm is relatively higher, and the experimental results show that 97.93% accuracy can be obtained based on 37-dimensional feature vectors.

Further, in order to prove whether this feature engineering method has space for further optimization, this paper further selects top-k (k = 37, 21, 15) dimension slice features for classification according to the occurrence frequency of these features. The experimental results are shown in Table 5.

Table 5. Classification results of different dimension of opcode slice features.

Feature Dimension	Metric	Random Forest	Decision Tree	KNN	XGBoost
N = 37	Accuracy	97.93%	96.23%	96.04%	97.38%
	Precision	97.38%	94.32%	95.93%	95.19%
	Recall	93.12%	91.52%	91.39%	91.43%
	F1 Score	94.59%	92.52%	92.96%	92.69%
N = 21	Accuracy	97.79%	96.55%	96.04%	97.33%
	Precision	97.22%	95.77%	95.93%	96.62%
	Recall	93.07%	91.84%	91.39%	91.27%
	F1 Score	94.47%	93.10%	92.96%	92.89%
N = 15	Accuracy	97.47%	95.63%	96.14%	97.06%
	Precision	97.03%	92.42%	96.03%	96.40%
	Recall	92.81%	91.02%	91.47%	92.49%
	F1 Score	94.25%	91.47%	93.04%	93.77%

The experimental results demonstrate that the Random Forest algorithm still achieves higher detection rates, and the impact is not significantly reduced when reducing the slice dimension. The accuracy rate reaches 97.79% when selecting 21-dimensional slice features and 97.47% when selecting 15-dimensional slice features.

This experiment proves that by only selecting key opcode slice features with richer semantics, the desired classification effect can be achieved.

It can be observed that the feature engineering designed in this paper yields superior results compared to other methods, as it saves time for feature preprocessing and training while also reducing the space complexity.

4.3.2. Comparison between SSEAL and Single Classifiers

In this experiment, SSEAL is compared with single classifiers with the BSS strategy to show the impact of ensemble classifiers. Figure 4 shows that the accuracy of SSEAL is higher than other single classifiers. When the iteration rounds 20 times, the accuracy of the approaches gradually becomes stable, and the accuracy of SSEAL is slightly higher than the other approaches; when the iteration round is less than 20, SSEAL has better performance than the other approaches in most iterations.

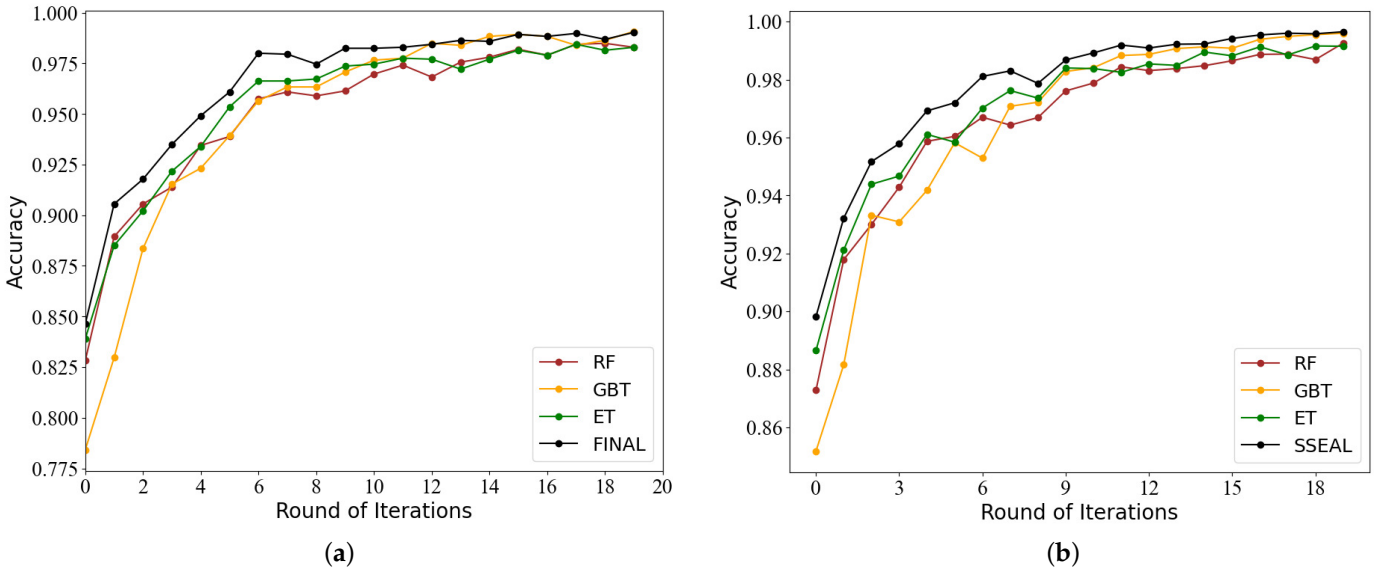

Figure 4. Comparison between SSEAL and single classifier. (**a**) Kaggle, (**b**) DataWhale.

Our advantage is that we use a small part of the data. The total sample size is 10,212, but our initial data size is 100, and after 20 iterations, we use a total of 1098 samples, which

accounts for 1098/8169 = 13.4% of the unlabeled data, and our work still achieves over 99 percent accuracy.

4.3.3. Comparison of SSEAL Algorithms in Different Feature Dimensions

In this experiment, we compared the effect of SSEAL algorithm using different feature dimensions. Figure 5 shows the running results of the 956-dimensional feature (left) and the 4250-dimensional feature (right), which take 7 min 25 s and 13 min 54 s, respectively. Although the final accuracy of using the 956-dimensional characteristic matrix is slightly lower than the result of the 4250-dimensional one, it still reaches 98.9%. On the basis of maintaining high accuracy, the model with low feature dimension converges faster and takes less training time, showing excellent performance.

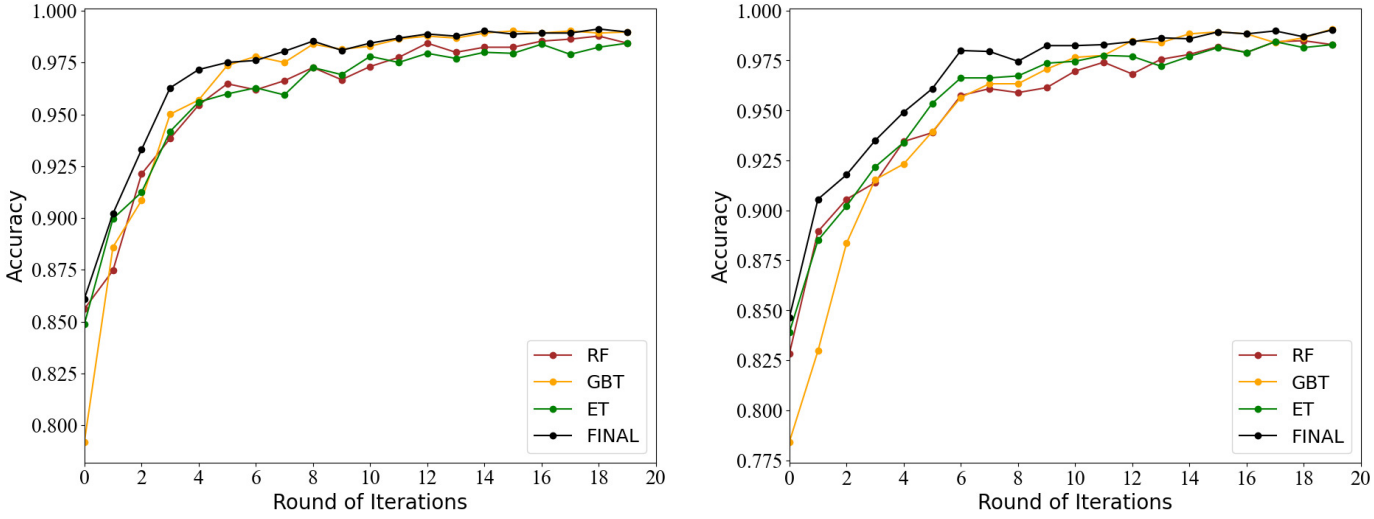

Figure 5. Comparison of SSEAL algorithms in different feature dimensions. (**a**) 956-dimension, (**b**) 4250-dimension.

4.3.4. Evaluation of SSEAL to Detect Unknown Malware

In this experiment, the ability of SSEAL to detect unknown malware is evaluated. As shown in Figure 6, SSEAL can achieve a good detection effect on the samples with a relatively large proportion. The detected F1-score gradually converges to 1. For malware of the fourth and sixth categories, which have very few samples, SSEAL can quickly filter out these rare attack samples. The detection rate of the fifth category samples is completely undetected in the beginning, but it can be found in the sixth round. As the iterations increase, the detection ability of SSEAL for this malware improves rapidly.

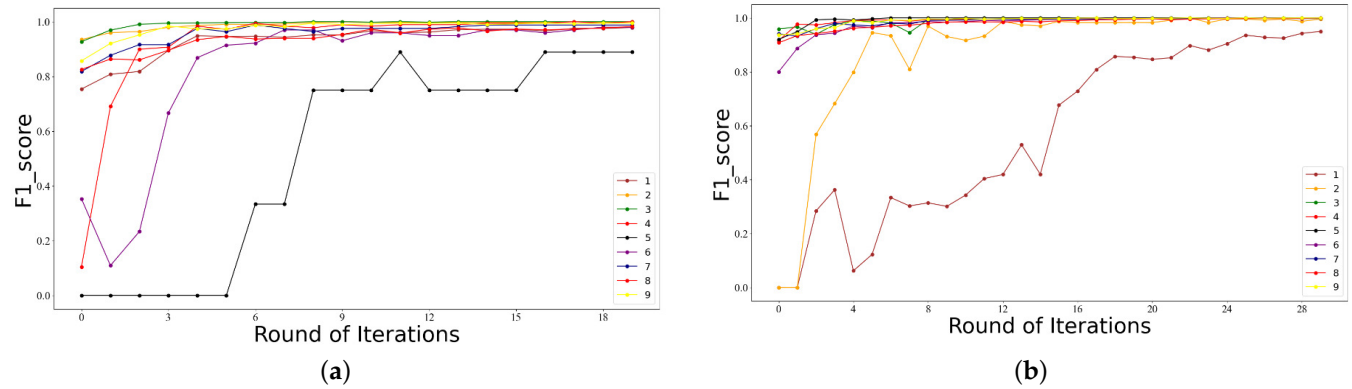

Figure 6. F1 scores of nine families. (**a**) Kaggle, (**b**) DataWhale.

Figure 7 shows the distribution of different types of samples input to the model in each round after using the sample selection equalizer. It can be seen that the nine types of samples basically meet the equilibrium conditions and can provide a good training environment for the sample to resist noise data.

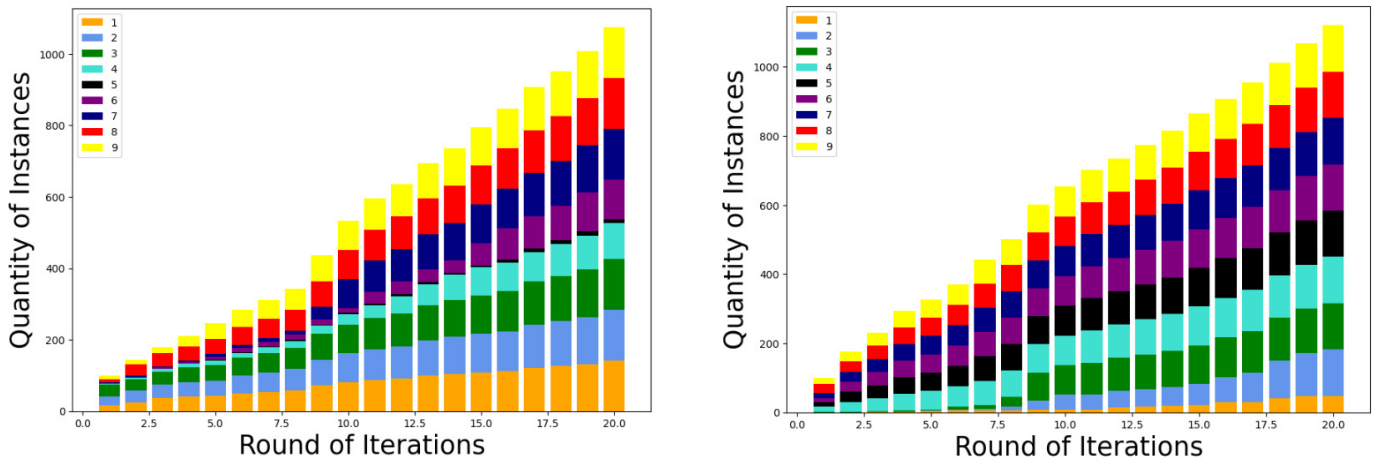

Figure 7. Sample distribution of different categories. (**a**) Kaggle, (**b**) DataWhale.

4.3.5. Evaluation of SSEAL with Different Samples in Pre-Training

This experiment evaluates the influence of the pre-training dataset to SSEAL. This work pre-trains SSEAL with 20, 50, and 100 labeled samples, and records the accuracy of SSEAL in each iteration. As shown in Figure 8, the influence of the pre-training data to SSEAL is obvious. When there are fewer pre-training samples, the initial accuracy of SSEAL is lower.

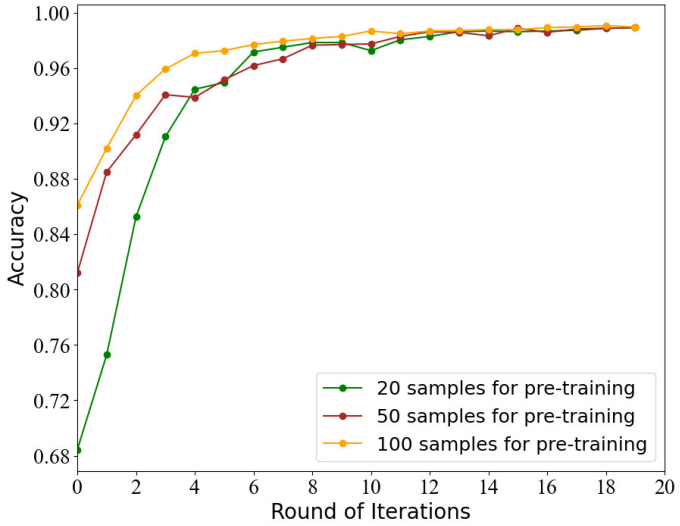

Figure 8. Accuracy of different pre-training sample sizes.

When the pre-training samples are 20, the initial accuracy of SSEAL is only below 70%, but when the pre-training samples are 100, the accuracy can reach 86%. As the number of iterations increases, the detection capability of SSEAL is rapidly improved, and the advantage of using more labeled samples in the pre-training is no longer reflected. After 14 iterations, the 20-sample pre-trained model catches up with the 100-sample pretrained model in terms of performance.

4.3.6. Comparison with Similar Studies

In this section, we compared our method with other studies using the same dataset or similar algorithms, from the aspects of accuracy, features, time consumption and occupied space. The comparison results are shown in Table 6, where "-" means unstated.

Table 6. Comparison with similar studies.

	This Work	**All Opcode**	**Ahmadi et al.** [31]	**Kong et al.** [24]	**Raff et al.** [32]	**Le et al.** [33]
Dataset	"Kaggle" from Microsoft Malware Classification Challenge					
Selected Feature	Opcode Slice	Opcode	Hex dump-based features + Features extracted from disassembled files	Mutual information method based Top-18 opcodes	Entire malware as embedded input	Entire malware as embedded input
Dimension of feature	35	737	1804	18	-	10,000
Classification accuracy (%)	97.93	98.80	99.77	98.60	97.80	98.20
Time consumption of feature engineering (s)	3059.41	5889.16	183,477	5907.27	-	-
Time consumption of model training (s)	-	-	-	-	32,087.4	6372
Time consumption of model classification (s)	2.38	24.92	15	7.76	804.65	214.32
Occupied space (KB)	1223	15,120	-	1043	-	-

In addition, [34] utilized ensemble learning and achieved high accuracy. However, this work extracted information from different sections of PE and converted it into images, and then used multiple CNN models as base classifiers, resulting in a longer training time than ours. In contrast to [35], this work builds four static feature comprehensive description PE files, including API and dll, which will consume more feature processing time. The proposed feature engineering method based on opcode slice in this paper achieves the optimal compromise between efficiency and classification accuracy. The length of the feature vector constructed by our method is 35, and the classification accuracy is 97.93%. On the one hand, although the classification accuracy is slightly lower than that of similar studies [24,33], it can meet the detection requirements. On the other hand, the feature processing time in this paper is the shortest [32,36], which means that this method can provide promising classification results under the condition of reducing the complexity of feature engineering.

Above all, the method proposed in this paper has a main advantage: our work can effectively reduce the time and space occupation of model training and classification while retaining high accuracy.

5. Discussion

As we reflect on the findings and implications of our proposed method, it is essential to recognize both its strengths and limitations. In this discussion, we address some of the key considerations related to the limitations, scalability, and ethical implications of our approach.

5.1. Limitations of Our Work

Not limited to the same malware family, functional similarities of opcode slice may also exist between benign samples and malicious samples, which will lead to the detection of false negatives. In addition, there are some cases where the training data are unbalanced due to the small sample size. For example, the Simda family in the Kaggle dataset is sensitive to the detection results due to its relatively small data, which leads to false positives.

Another limitation of our study is that the high-information samples are manually labeled, which is assumed to always be correct. However, experts may mislabel some samples in practical applications so that mislabeled high-information samples may be added into the labeled training set. In this context, a more sophisticated study should be conducted on how to avoid the impact of manually mislabeled samples on the detection model.

5.2. Scalability of Our Work

We think this method has good scalability in practical application. When implementing the framework in the real world, our opcode slicing method only needs to make a simple substitution based on semantic analysis, which can support the subsequent feature engineering. As for adaptability to evolving malware techniques, some malware will pack itself to escape the disassembly tool, which will destroy the foundation of the slicing operation, thus affecting the final detection result. Therefore, we need to detect whether the malware is packed first.

5.3. Ethical Implications and Considerations

Due to concerns related to data leakage, enterprises or individuals may be hesitant to provide raw samples, making it challenging for many research methods to be widely applied. The privacy protection issues associated with original or feature data of malicious code need to be addressed [37]. In response to this situation, the introduction of federated learning techniques [38] can be considered. This approach facilitates the training process without the need for users to transmit their training data models directly. As a result, the entire training procedure can be conducted without compromising the privacy of user data.

6. Conclusions and Future Work

Our work proposes an Opcode Slice-Based Malware Detection Framework Using Active and Ensemble Learning in order to address the rapid evolution of malicious code, the high cost of manual annotation, and the unbalanced distribution of different families. Specifically, it introduces a feature engineering method based on opcode slice and a SSEAL detection algorithm for malware classification. The experiments conducted in this paper are based on the Kaggle dataset and DataWhale. The feature engineering method utilizing opcode slice is proven effective in extracting behavioral characteristics from malware samples, laying the foundation for efficient classification. SSEAL has demonstrated its ability to reduce data labeling costs, extract more knowledge from samples, and exhibit higher reliability compared to single classifiers.

After conducting research in this paper, we have the intuition that further work can be conducted as follows: we will try to put forward a solution for the problem of insufficient datasets caused by sample imbalance. In addition, we will combine more semantic information, such as API, to improve the robustness of our system.

Author Contributions: Conceptualization, W.G. and W.H.; methodology, W.G. and W.M.; software, W.M.; validation, W.G.; formal analysis, W.M.; investigation, J.X., Y.W. and Z.L. (Zhongjun Li); data curation, Z.L. (Zishu Liu); writing—original draft preparation, W.G. and W.H.; writing—review and editing, W.M.; supervision, J.X.; project administration, J.X.; funding acquisition, J.X. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Major Scientific and Technological Innovation Projects of Shandong Province (2020CXGC010116) and the National Natural Science Foundation of China (No. 62172042).

Data Availability Statement: The data used to validate the outcomes of this investigation can be obtained by contacting the corresponding author.

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

References

1. Kaspersky Cyber Security Solutions for Home and Business | Kaspersky. Available online: https://usa.kaspersky.com/ (accessed on 10 November 2023).
2. Hu, Y.; Wang, S.; Li, W.; Peng, J.; Wu, Y.; Zou, D.; Jin, H. Interpreters for GNN-Based Vulnerability Detection: Are We There Yet? In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Seattle, WA, USA, 18–20 July 2023; pp. 1407–1419.
3. Li, H.; Cheng, Z.; Wu, B.; Yuan, L.; Gao, C.; Yuan, W.; Luo, X. Black-box Adversarial Example Attack towards FCG Based Android Malware Detection under Incomplete Feature Information. *arXiv* **2023**, arXiv:2303.08509.
4. Hu, P.; Liang, R.; Cao, Y.; Chen, K.; Zhang, R. {AURC}: Detecting Errors in Program Code and Documentation. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 1415–1432.
5. Ye, Y.; Li, T.; Adjeroh, D.; Iyengar, S.S. A survey on malware detection using data mining techniques. *ACM Comput. Surv. (CSUR)* **2017**, *50*, 1–40. [CrossRef]
6. Chow, Y.W.; Schäfer, M.; Pradel, M. Beware of the unexpected: Bimodal taint analysis. *arXiv* **2023**, arXiv:2301.10545.
7. Gollapudi, R.T.; Yuksek, G.; Demicco, D.; Cole, M.; Kothari, G.; Kulkarni, R.; Zhang, X.; Ghose, K.; Prakash, A.; Umrigar, Z. Control flow and pointer integrity enforcement in a secure tagged architecture. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 2974–2989.
8. Wu, X.; Guo, W.; Yan, J.; Coskun, B.; Xing, X. From Grim Reality to Practical Solution: Malware Classification in Real-World Noise. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–24 May 2023; pp. 2602–2619.
9. Yang, L.; Chen, Z.; Cortellazzi, J.; Pendlebury, F.; Tu, K.; Pierazzi, F.; Cavallaro, L.; Wang, G. Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 719–736.
10. Patrick-Evans, J.; Dannehl, M.; Kinder, J. XFL: Naming Functions in Binaries with Extreme Multi-label Learning. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 21–25 May 2023; pp. 2375–2390.
11. Luo, Z.; Wang, P.; Wang, B.; Tang, Y.; Xie, W.; Zhou, X.; Liu, D.; Lu, K. VulHawk: Cross-architecture Vulnerability Detection with Entropy-based Binary Code Search. In Proceedings of the NDSS, San Diego, CA, USA, 27 February–3 March 2023.
12. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
13. Cui, L.; Cui, J.; Ji, Y.; Hao, Z.; Li, L.; Ding, Z. API2Vec: Learning Representations of API Sequences for Malware Detection. In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, Seattle, WA, USA, 18–20 July 2023; pp. 261–273.
14. Lucas, K.; Pai, S.; Lin, W.; Bauer, L.; Reiter, M.K.; Sharif, M. Adversarial Training for {Raw-Binary} Malware Classifiers. In Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 9–11 August 2023; pp. 1163–1180.
15. Weiser, M. Programmers use slices when debugging. *Commun. ACM* **1982**, *25*, 446–452. [CrossRef]
16. Horwitz, S.; Reps, T.; Binkley, D. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst. (TOPLAS)* **1990**, *12*, 26–60. [CrossRef]
17. Xu, B.; Qian, J.; Zhang, X.; Wu, Z.; Chen, L. A brief survey of program slicing. *ACM SIGSOFT Softw. Eng. Notes* **2005**, *30*, 1–36. [CrossRef]
18. Ottenstein, K.J.; Ottenstein, L.M. The program dependence graph in a software development environment. *ACM Sigplan Not.* **1984**, *19*, 177–184. [CrossRef]
19. Lee, Y.; Kwon, H.; Choi, S.H.; Lim, S.H.; Baek, S.H.; Park, K.W. Instruction2vec: Efficient Preprocessor of Assembly Code to Detect Software Weakness with CNN. *Appl. Sci.* **2019**, *9*, 4086. [CrossRef]
20. Haq, I.U.; Caballero, J. A Survey of Binary Code Similarity. *ACM Comput. Surv.* **2021**, *54*, 1–38. [CrossRef]
21. Han, W.; Xue, J.; Wang, Y.; Liu, Z.; Kong, Z. MalInsight: A systematic profiling based malware detection framework. *J. Netw. Comput. Appl.* **2019**, *125*, 236–250. [CrossRef]
22. Han, W.; Xue, J.; Wang, Y.; Huang, L.; Kong, Z.; Mao, L. MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Comput. Secur.* **2019**, *83*, 208–233. [CrossRef]
23. Huang, L.; Xue, J.; Wang, Y.; Qu, D.; Chen, J.; Zhang, N.; Zhang, L. EAODroid: Android Malware Detection Based on Enhanced API Order. *Chin. J. Electron.* **2023**, *32*, 1169. [CrossRef]
24. Kong, Z.; Xue, J.; Wang, Y.; Zhang, Q.; Han, W.; Zhu, Y. MalFSM: Feature Subset Selection Method for Malware Family Classification. *Chin. J. Electron.* **2023**, *32*, 26–38. [CrossRef]
25. Alrabaee, S. A stratified approach to function fingerprinting in program binaries using diverse features. *Expert Syst. Appl.* **2022**, *193*, 116384. [CrossRef]
26. Cordeiro de Amorim, R.; Lopez Ruiz, C.D. Identifying meaningful clusters in malware data. *Expert Syst. Appl.* **2021**, *177*, 114971. [CrossRef]
27. Wang, P.; Tang, Z.; Wang, J. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Comput. Secur.* **2021**, *106*, 102273. [CrossRef]

28. Niu, Z.; Guo, W.; Xue, J.; Wang, Y.; Kong, Z.; Huang, L. A novel anomaly detection approach based on ensemble semi-supervised active learning (ADESSA). *Comput. Secur.* **2023**, *129*, 103190. [CrossRef]
29. Koo, H.; Park, S.; Choi, D.; Kim, T. Semantic-aware binary code representation with bert. *arXiv* **2021**, arXiv:2106.05478.
30. Panconesi, A.; Marian; Cukiersk, W.; WWW BIG-Cup Committee. Microsoft Malware Classification Challenge (BIG 2015). *Kaggle*. 2015. Available online: https://kaggle.com/competitions/malware-classification (accessed on 10 November 2023).
31. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; pp. 183–194.
32. Raff, E.; Nicholas, C. Malware classification and class imbalance via stochastic hashed lzjd. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, Dallas, TX, USA, 3 November 2017; pp. 111–120.
33. Le, Q.; Boydell, O.; Mac Namee, B.; Scanlon, M. Deep learning at the shallow end: Malware classification for non-domain experts. *Digit. Investig.* **2018**, *26*, S118–S126. [CrossRef]
34. Niu, W.; Cao, R.; Zhang, X.; Ding, K.; Zhang, K.; Li, T. OpCode-level function call graph based android malware classification using deep learning. *Sensors* **2020**, *20*, 3645. [CrossRef]
35. Soni, H.; Kishore, P.; Mohapatra, D.P. Opcode and API based machine learning framework for malware classification. In Proceedings of the 2022 2nd International Conference on Intelligent Technologies (CONIT), Hubli, India, 24–26 June 2022; pp. 1–7.
36. Santos, I.; Brezo, F.; Ugarte-Pedrero, X.; Bringas, P.G. Opcode sequences as representation of executables for data-mining-based unknown malware detection. *Inf. Sci.* **2013**, *231*, 64–82. [CrossRef]
37. Dara, S.; Zargar, S.T.; Muralidhara, V.N. Towards privacy preserving threat intelligence. *J. Inf. Secur. Appl.* **2018**, *38*, 28–39. [CrossRef]
38. Lyu, L.; Yu, H.; Ma, X.; Chen, C.; Sun, L.; Zhao, J.; Yang, Q.; Philip, S.Y. Privacy and robustness in federated learning: Attacks and defenses. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.