ReMoS: Reducing Defect Inheritance in Transfer Learning via Relevant Model Slicing

Ziqi Zhang[*]
Key Laboratory of High-Confidence
Software Technologies (MOE),
School of Computer Science,
Peking University
Beijing, China
ziqi_zhang@pku.edu.cn

Yuanchun Li[*†]
Institute for AI Industry Research
(AIR), Tsinghua University
Beijing, China
liyuanchun@air.tsinghua.edu.cn

Jindong Wang
Microsoft Research
Beijing, China
jindong.wang@microsoft.com

Bingyan Liu
Key Laboratory of High-Confidence
Software Technologies (MOE),
School of Computer Science,
Peking University
Beijing, China
lby_cs@pku.edu.cn

Ding Li
Key Laboratory of High-Confidence
Software Technologies (MOE),
School of Computer Science,
Peking University
Beijing, China
ding_li@pku.edu.cn

Yao Guo[†]
Key Laboratory of High-Confidence
Software Technologies (MOE),
School of Computer Science,
Peking University
Beijing, China
yaoguo@pku.edu.cn

Xiangqun Chen
Key Laboratory of High-Confidence
Software Technologies (MOE),
School of Computer Science,
Peking University
Beijing, China
cherry@sei.pku.edu.cn

Yunxin Liu
Institute for AI Industry Research
(AIR), Tsinghua University
Beijing, China
liuyunxin@air.tsinghua.edu.cn

ABSTRACT

Transfer learning is a popular software reuse technique in the deep learning community that enables developers to build custom models (students) based on sophisticated pretrained models (teachers). However, like vulnerability inheritance in traditional software reuse, some defects in the teacher model may also be inherited by students, such as well-known adversarial vulnerabilities and backdoors. Reducing such defects is challenging since the student is unaware of how the teacher is trained and/or attacked. In this paper, we propose ReMoS, a relevant model slicing technique to reduce defect inheritance during transfer learning while retaining useful knowledge from the teacher model. Specifically, ReMoS computes a model slice (a subset of model weights) that is relevant to the student task based on the neuron coverage information obtained by profiling the teacher model on the student task. Only the relevant slice is used to fine-tune the student model, while the irrelevant weights are retrained from scratch to minimize the risk of inheriting defects. Our experiments on seven DNN defects, four DNN models, and eight datasets demonstrate that ReMoS can reduce inherited defects effectively (by 63% to 86% for CV tasks and by 40% to 61% for NLP tasks) and efficiently with minimal sacrifice of accuracy (3% on average).

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Software and its engineering** → *Dynamic analysis*; • **Security and privacy** → *Software security engineering*.

KEYWORDS

Program slicing, deep neural networks, relevant slicing

ACM Reference Format:
Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu, Ding Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2022. ReMoS: Reducing Defect Inheritance in Transfer Learning via Relevant Model Slicing . In *44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510191

[*] Work is done while Ziqi Zhang was an intern and Yuanchun Li was a researcher at Microsoft. † Correspondence to: Yuanchun Li, Yao Guo.

ICSE '22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9221-1/22/05. . . $15.00
https://doi.org/10.1145/3510003.3510191

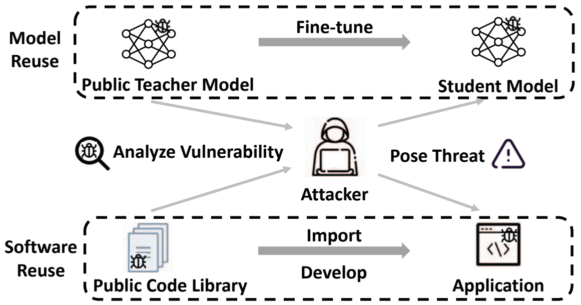


Figure 1: Defect inheritance is a common problem in both traditional software reuse and DNN model reuse.

1 INTRODUCTION

Software reuse is popular in traditional programs [16, 19], with examples ranging from copy-pasting a piece of code, inheriting a parent class, to importing a third-party library. Despite the great convenience of software reuse, the vulnerability of the reused software is one of the most concerning issues [18, 52]. Developers are often unaware of or unable to eliminate the vulnerabilities in the reused code, which may lead to unexpected consequences. For instance, the Heartbleed bug in the OpenSSL library has led to severe security issues in many applications built upon it [14]. The open-source nature of many reusable packages increases the attack surface since the attackers can inspect the reused code thoroughly to find flaws to exploit. The threats and countermeasures of reusing vulnerable modules have been studied by security researchers for a long time [6, 8, 10].

The deep neural network (DNN for short) is now considered a special type of software with great breakthroughs in recent years. To reuse existing models and expedite model development, *transfer learning* [40, 51] has been proposed to reuse the knowledge in one model to build new models. A typical transfer learning process includes acquiring a public pretrained model (*e.g*. ResNet [24] or BERT [13]), adjusting the model architecture, and fine-tuning the model using target-domain data [42]. Borrowing names from the prior literature, we call the pretrained models as *teachers* and the new models built upon them as *students* [64]. Thanks to the wide availability of sophisticated public models [3, 4, 68], transfer learning is nowadays the default choice for most developers when building their own deep learning applications.

However, similar to traditional software reuse, transfer learning also leads to concerns about defect inheritance. Figure 1 illustrates the defect inheritance problem in traditional software reuse and model reuse. In software reuse, the developers download the public library and use it in their applications. Similarly, in model reuse, the developers download the public pretrained model and use transfer learning to obtain their customized models. In both cases, defects may be inherited during the software reuse process. An attacker can download the public library (model), analyze it to find the defects, and attack the deployed application (model) with the carefully crafted inputs [6, 8].

The SE community takes the security risk of DNNs as an important problem [37, 38, 60, 71] as DNN models are extensively used in various industrial and research scenarios. On the one hand, DNNs are considered as an emerging type of software artifact (widely known as“software 2.0”). The widely-adapted model reuse is a type of software reuse for DL developers. On the other hand, nowadays DNNs are frequently embedded into software applications, such as autonomous driving and facial recognition. The safety of DL models is crucial for the reliability of relevant software.

Table 1: Potential defects in the literature that may inherit during transfer learning.

Task	Defect Type		Inheritance Rate
CV	Adversarial Vulnerability	Penultimate-Layer Guided [58]	58.01%
		Neuron-Coverage Guided [21, 55]	52.58%
	Backdoor	Latent Data Poisoning [70]	72.91%
NLP	Adversarial Vulerability	Greedy Word Swap [31]	64.86%
		Word Importance Ranking [29]	94.73%
	Backdoor	Data Poisoning [20]	96.72%
		Weight Poisoning [32]	97.85%

We surveyed existing literature and summarized seven typical defects that may be inherited during transfer learning. As shown in Table 1, these defects can be categorized into two groups: adversarial vulnerability and backdoor. Adversarial vulnerability is an *exploratory defect* [25, 36], where an attacker can obtain adversarial examples from the public teacher model and fool the students [21, 55, 58, 64]. Backdoor is a *causative defect*, where hidden malicious logic is injected into the teacher model purposely, and the student models built upon it may also misbehave under certain backdoor triggers [20, 20, 28, 32, 45, 70]. According to our measurement study, the defects can easily be propagated from the teacher to the students, with the inheritance rate ranging from 52.58% to 97.85%.

A major reason for DNN defect inheritance is the indiscriminate training process [58]. Specifically, in conventional transfer learning, the student model is initialized thoroughly with the teacher's weights, including the weights for both student-related and defect-related decision boundaries. Thus the defect-related decision boundaries are largely kept during fine-tuning since the student dataset is small-scale and irrelevant to the defect-related decision boundaries [41]. *The focus of this paper is how to reduce the inherited defects while preserving student-related knowledge*.

There are mainly two types of solutions for developers to reduce inherited defects and enhance deep learning software reliability and safety. One is to adopt defect mitigation techniques (such as fine-pruning [44], etc.) after fine-tuning, which we call *fix-after-transfer* approaches. Another is the *fix-before-transfer* approach [11], which first randomly initializes the student model, then extracts the student-related knowledge from the teacher. The main problem of *fix-after-transfer* approaches is the poor effectiveness, since it is difficult for the student model to remove the inherited defects with limited training data. The *fix-before-transfer* approach [11] is effective in reducing inherited defects, but it sacrifices the performance on the student task and applicability because the student model has to be trained from scratch with complicated configurations.

In this paper, we try to address the defect inheritance problem from a different angle. We observe that a teacher model is trained to serve multiple downstream tasks. However, a student model is often trained for a specific task. Thus, the teacher model may contain a substantial amount of weights that are not useful for the specific task that the student model tries to address. Having these irrelevant weights enlarges the attack surfaces for backdoors and increases the probability of exposing defects. Based on this observation, we propose to selectively reuse the teacher knowledge that is relevant to the student task. The idea is motivated by relevant slicing [5, 22], which computes a set of program statements that *may* affect the slicing criterion. Our intuition is similar to the conventional software engineering practice: only reusing the relevant code of a program can decrease the probability of having bugs [6, 8, 10].

However, due to the lack of the interpretability of DNN models, computing the relevant slice is not easy. For traditional software, the functionalities are specified by the developers through lines of code, and the relevance of each line of code can be explicitly computed [22]. On the contrary, the development process of DNN models is quite different. The decision logic of a model is hidden in millions of weights learned from data. The semantic meaning of each weight is not understandable, making it difficult to compute the relevant part.

To solve the aforementioned problems, we introduce a profiling-based approach named ReMoS (**Re**levant **Mo**del **S**licing) that is guided by neuron coverage to compute the relevant slice of the pre-trained model on the student task. The relevant slice is further used in fine-tuning to reduce the inherited defects. ReMoS consists of four steps: coverage frequency profiling, ordinal score computation, relevant slice generation, and fine-tuning. The first stage records the neuron coverage frequency by iterating all data samples and computes the weight coverage frequency as the fine-grained relevance to the student task. Ordinal score computation combines the weight significance inside the teacher model (measured by the magnitude) and the relevance to the student task (measured by the weight coverage frequency). The last step finds the model parts that are relevant to the student task and resets the excluded weights. At last, the relevant part of the teacher model is used to initialize the student model and it is fine-tuned with the student dataset.

We evaluate ReMoS with both CV and NLP tasks on four models, eight datasets, and on all seven inherited defects in Table 1. For CV models, the inherited defects are reduced by 63% to 78% for adversarial vulnerability and 86% for backdoors, with less than 2% accuracy sacrifice. For NLP models, the inherited defects are reduced by 40% for adversarial vulnerability and 50% to 61% for backdoors, with less than 3% accuracy drop. We also analyze the distribution of the excluded weights of the relevant model slice and find that it mainly slices the high-level weights that are irrelevant to the student task.

This paper makes the following research contributions:

- We study the defect inheritance problem in the transfer learning scenario, summarize common defects in the existing literature, and quantify the inheritance rate of each kind of defect.
- We propose ReMoS to reduce defects inherited from teacher models during transfer learning. Our method is inspired by relevant slicing in traditional software, which computes a subset of parameters in the teacher model that are relevant to student tasks while excluding the irrelevant ones to reduce teacher-originated defects.
- We evaluate ReMoS against seven different types of DNN defects in comparison with several state-of-the-art approaches on various popular DNN architectures and datasets. The results have demonstrated the effectiveness and efficiency of our approach.

2 PROBLEM FORMULATION

In this section, we formulate the defect inheritance problem, describe the threat model, and define the defender's goal.

2.1 DNN Defects

The DNN defects considered in this paper are defined as follows:

Definition 1. ***DNN Defects** are the deviation of the actual and expected results of a DNN model produced by certain input samples.*

Specifically, we focus on two typical DNN defects that are widely discussed in the literature, including adversarial vulnerability and backdoor. They can be triggered by adversarial inputs and backdoor inputs respectively.

Adversarial vulnerability means that a DNN is vulnerable to adversarial inputs that are generated by adding special noise to normal inputs and lead to prediction errors [26]. The most common way to generate adversarial inputs is to use gradient ascent techniques on the white-box model [48]. In the software engineering community, this vulnerability is viewed as a robustness issue, and many techniques have been proposed to test [21, 55, 65], improve [15, 17], or verify [7, 43, 54, 60] the DNN robustness.

A **backdoor** (or a trojan horse) is a hidden pattern purposely injected into the model that produces unexpected output if a specific trigger is presented in the input. Backdoor inputs can be easily synthesized by attaching a backdoor trigger to normal inputs. The most straightforward way to implant backdoor is data poisoning [20, 45], *i.e.* adding carefully designed samples into the training data to let the model memorize misleading patterns. Recently, researchers have extended the attacks to make the backdoor more effective [28, 32, 37], evasive [39], and transferable [70].

The two aforementioned defects share a similar goal: to disturb the model output under a limited perturbation budget. This budget is usually computed by a distance function $d(\cdot,\cdot)$ which measures the difference between a benign input and a malicious input. The goal of the adversary is to generate a malicious sample $\tilde{x}$:

$$\tilde{x} = \arg\max_{x'} J(f(x'; \mathbf{w}^T), y) \quad \text{s.t. } d(x', x) < \epsilon \tag{1}$$

where J is the loss function and ϵ is a pre-defined small value. For the adversarial vulnerability, the distance function is usually the p-norm distance between the two inputs: $d(\tilde{x}^{adv}, x) = \|\tilde{x}^{adv} - x\|_p$. For backdoors, the adversary attempts to generate malicious inputs by attaching a trigger sign onto normal inputs. In this case, the distance is the ratio of different input dimensions between $\tilde{x}^{tri}$ and x: $d(\tilde{x}^{tri}, x) = \frac{\mathbb{I}[\tilde{x}^{tri} \neq x]}{|x|}$.

In the community of SE and ML, these two defects and the corresponding attacks usually refer to the same thing [15, 17, 72], so we will use the terms "defects" (or "vulnerabilities") and "attacks" according to the different context.

2.2 DNN Defect Inheritance

Unlike most existing work that discusses DNN defects of an individual model, we focus on the defects propagating between models during transfer learning.

Transfer learning scenario. Suppose a developer wants to train a model M^S with a dataset D^S. To improve accuracy and reduce training time, she decides to use transfer learning to build her model upon a public pretrained model M^T. M^T and M^S are called the teacher model and the student model respectively. The dataset D^T that was used to train M^T is called the teacher dataset, which is significantly larger than the student dataset, *i.e.* $|D^T| \gg |D^S|$.

The most common transfer learning method is fine-tuning, where the developer changes the model output and uses the local student dataset to fine-tune the model. The change made on the model is slight because the knowledge in the teacher model is general enough.

We are interested in the DNN defect inheritance problem in transfer learning, which is defined as follows:

Definition 2. ***DNN defect inheritance*** *is the phenomenon that the input samples which can trigger defects (mislead the output) in the pretrained teacher model* M^T *can also produce misbehavior in the student model* M^S.

Since the teacher models are usually publicly available, attackers can easily analyze them to find adversarial inputs or inject backdoors into them. Such teacher-originated malicious inputs pose threats to the student models deployed in the real world, even though the student models are not white-box available.

Threat Model. We assume the attacker has read and/or write access (white-box) to the pretrained teacher M^T, and thus he can infer or inject defects in the teacher model. This assumption is reasonable because many pretrained models are publicly available [3, 4, 68]. The attacker can publish a malicious pretrained model as well. We also assume the attacker has basic knowledge of the student model M^S such as the type of the task, but has no read or write access to the student M^S or student dataset D^S. It is because the student models are usually protected and served as a black box. Based on the knowledge of the teacher model M^T, the attacker can generate malicious inputs that can fool the student model M^S.

2.3 Goal and Challenges

Defender's goal. The defender in our scenario is the developer of the student model M^S who is unaware of how the teacher model M^T is trained. Her goal is to optimize the transfer learning protocol to reduce the defects inherited from M^T (reduce the misclassification rate on the malicious samples $\tilde{x}$ that are generated from M^T) while retaining the student-related knowledge in M^T.

Figure 2 displays the difference of inherited knowledge between conventional transfer learning (left) and transfer learning with ReMoS (right). Our objective is to reduce the common defects that are shared by both the teacher model and the student model. Because the pretrained teacher model is publicly available, removing the shared defects will reduce the exposed attack surface of the student model and improve the security level.

Specifically, we want to achieve the following objectives:

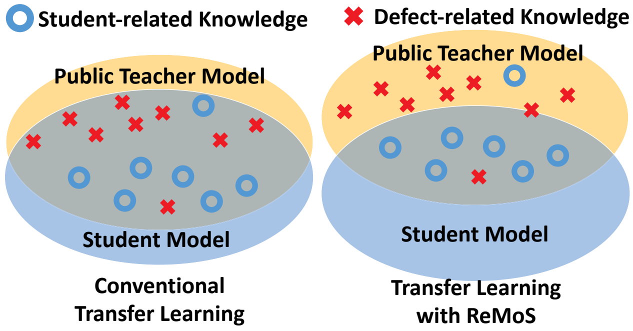


Figure 2: The goal of ReMoS is to reduce the common defects shared by the teacher and the student.

- **Effectiveness**. The inherited defects in the student model M^S should be effectively reduced, *i.e.* reducing the inheritance rate of the adversarial samples and backdoor samples.
- **Accuracy**. The student model M^S should still be able to reuse the student-related knowledge in M^T to achieve high accuracy on the student dataset D^S.
- **Efficiency and utility**. Since efficiency and ease of use are major reasons why developers use transfer learning, the new protocol should not bring too much overhead or introduce too much complexity.

Reducing the inherited defects is challenging mainly due to the limited interpretability of DNNs. A model is usually viewed as a black box whose knowledge can only be changed by feeding different training data. It is unclear how to identify and separate defect-related knowledge from student-related knowledge during transfer learning.

3 RELEVANT MODEL SLICING

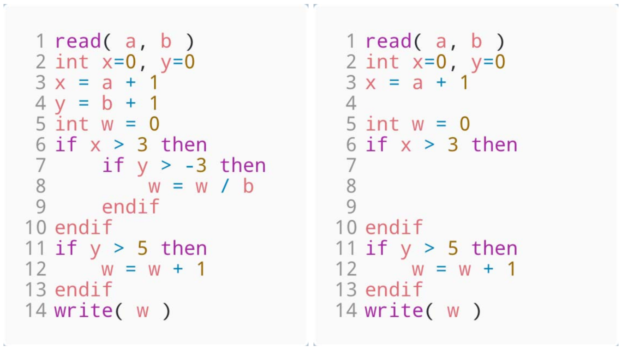
Inspired by the relevant slicing technique in traditional programs, we believe a similar technique for DNN models can help distinguish student-related and potential defect-related knowledge in a model, and further mitigate the defect inheritance problem in transfer learning.

3.1 Relevant Slicing for Traditional Programs

Relevant slicing [5] is an important slicing technique and has many applications such as debugging[22], regression testing [9] and prioritizing test case [27]. We first give the formal definition of the relevant slice as follows:

Definition 3. ***(Relevant Slicing)*** *Given a program P and a slicing criterion (a test case t and a target statement s), relevant slicing is to compute a subset of program statements that influence or have the potential to influence the statement s during the execution of t.*

Relevant slicing can be used to reduce vulnerability in traditional programs. An example is shown in Figure 3. Suppose the code snippet on the left is a public library function that contains a potential defect. If the input b is 0, a DivideByZero error may occur at line 8. If this piece of code is used without modification in a custom application, the bug may be exploited by attackers to pose threats to



```
 1 read( a, b )
 2 int x=0, y=0
 3 x = a + 1
 4 y = b + 1
 5 int w = 0
 6 if x > 3 then
 7     if y > -3 then
 8         w = w / b
 9     endif
10 endif
11 if y > 5 then
12     w = w + 1
13 endif
14 write( w )
```

```
 1 read( a, b )
 2 int x=0, y=0
 3 x = a + 1
 4
 5 int w = 0
 6 if x > 3 then
 7
 8
 9
10 endif
11 if y > 5 then
12     w = w + 1
13 endif
14 write( w )
```

Figure 3: An example code snippet (left) and the relevant slice with respect to the criterion < $\{a = 0, b = 4\}$, 14, w > (right).

the application. However, with the relevant slice, the statements that are not related to the custom application can be removed to reduce the attack surface. Suppose we only care about a limited number of input cases in the custom application, thus we can compute the relevant slice concerning the criterion whose inputs are the interested cases. For example, if the interested input case is $\{a = 0, b = 4\}$ and the criterion is $< \{a = 0, b = 4\}$ 14, $w >$, the corresponding relevant slice is shown in the right code snippet of Figure 3. In the custom application, using the original library code (left) or the relevant slice (right) does not influence the correctness of the program (the output w at line 14). However, the latter can remove the inherited defect and avoid the potential security risk (DivideByZero error).

3.2 Relevant Slicing for DNN Models

Similar to the example of reducing inherited defects in traditional programs through relevant slicing, we think it might be helpful to adapt the concept of relevant slicing to DNN models to reduce defect inheritance.

The idea of slicing a DNN model is not new. Zhang *et al.* [73] proposed to represent the decision process of a DNN with a slice (a subset of critical neurons and synapses that made a greater contribution to the prediction) and used the computed slices to detect adversarial inputs. Although our goal is different, we could follow their definition and customize it based on our scenario:

DEFINITION 4. ***(Relevant Model Slicing)*** *Given a DNN model M and a target domain dataset D, relevant model slicing is to compute a subset of model weights that are more relevant (bounded by a threshold) to the inference of samples in D and less relevant to the samples outside D.*

If we could obtain such a relevant model slice, we can use it to reduce inherited defects in transfer learning, just like using relevant program slices to reduce potential vulnerability in traditional software. Specifically, we can transfer the knowledge in the relevant slice to the students to retain useful knowledge, whiling removing the rest of irrelevant parts to avoid the defect-related knowledge propagating into the student model. The goal can be formally described as finding a subset of the teacher model's weights $\mathbf{w} \subset \mathbf{w}^T$ that, when $\mathbf{w}$ is used in transfer learning, can maximize the accuracy of the student model on both clean data and malicious data that is generated from the teacher model.

$$\max_{\mathbf{w} \subset \mathbf{w}^T} \sum_{(x,y) \in D^S} \mathbb{I}[f(x; T(\mathbf{w})) = y] + \mathbb{I}[f(\tilde{x}; T(\mathbf{w})) = y], \tag{2}$$

where $T(\cdot)$ represents the transfer learning process and $T(\mathbf{w})$ is the weights after training. $\mathbb{I}[\cdot]$ is the indicator function.

The concept of finding a subset of weights can also be found in model pruning approaches [23] that are mainly used to reduce the model size. For example, magnitude-based pruning [23] proposes to trim the small-magnitude weights to save computation cost. However, such a subset is not a relevant slice because the weights with larger magnitudes are not necessarily only relevant to the target student task. They are relevant to all the domains that are included by the teacher model's domain.

4 OUR APPROACH

Computing the relevant slice as defined in Section 3.2 by directly solving Equation 2 is difficult due to the unpredictable DNN training process and unavailable prior knowledge about defects. In this section, we introduce a heuristic algorithm to approximately find the relevant model slice and use it to reduce defect inheritance in transfer learning.

4.1 Overview

The workflow of ReMoS consists of four steps: coverage frequency profiling, ordinal score computation, relevant slice generation, and fine-tuning. An illustration with a three-layer neural network is shown in Figure 4. For simplicity, we assume there are three data samples in the student dataset.

Coverage Frequency Profiling computes the coverage frequency of each weight, which represents the relevance to the student task. The weight coverage frequency is computed based on the neuron coverage frequency profiled over the student dataset. The first stage of Figure 4 displays the neuron coverage frequency (in the neurons) and the weight coverage frequency (on the weights).

The *Ordinal Score Computation* step computes a score for each weight based on the static information of the teacher model (weight magnitude) and the dynamic information on the student task (weight coverage frequency). Because the range of weight magnitude and the coverage frequency is different, ReMoS uses the ordinal score as the yardstick. In the second stage of Figure 4, the ordinal score is displayed in the pink circles on each weight.

Relevant Slice Generation identifies the relevant slice based on the ordinal scores. The weights with large scores are included in the slice and others are reset. In the third stage of Figure 4, the size of the slice is constrained to be nine. Therefore, the connections with red circles are excluded by the slice and are reset.

Finally, the *fine-tuning* step uses the student dataset to train the model as displayed in the last stage of Figure 4. This step is the same as the conventional transfer learning, except that the student model is initialized with the computed relevant slice.

We then explain why ReMoS can achieve the three objectives in Section 2. First, because ReMoS only contains the weights that are frequently covered by the student dataset, the relevance between the slice and the student task is high. The accuracy of the student model is not harmed and can be recovered by a fast fine-tuning phase.

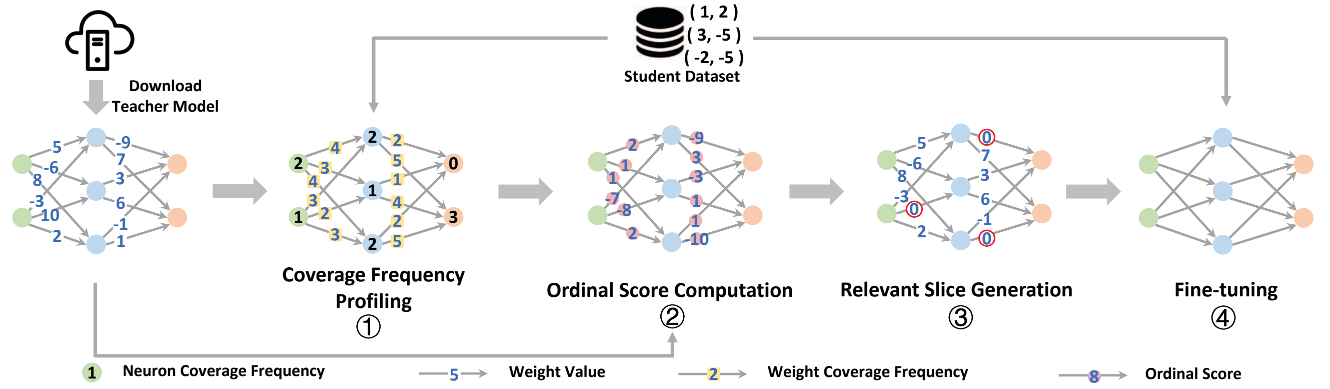

(1, 2)
(3, -5)
(-2, -5)
Student Dataset
Download Teacher Model
Coverage Frequency Profiling
①
Ordinal Score Computation
②
Relevant Slice Generation
③
Fine-tuning
④
1 Neuron Coverage Frequency
5 Weight Value
2 Weight Coverage Frequency
8 Ordinal Score

Figure 4: The workflow of transfer learning with ReMoS.

Second, the weights that are less relevant to the student domain are excluded from transfer learning, including the large-magnitude weights that are potentially more relevant to the defects in the teacher model. This design improves the effectiveness of reducing defect inheritance. Third, in the implementation, the relevant slice accounts for a large portion of the teacher model. Most of the student model is initialized by the teacher model, which leads to higher efficiency when tuning the student model. At last, our slicing algorithm is not restricted to any specific operator or model architecture. It can be applied to both Convolutional Neural Networks and Transformer-based NLP models.

4.2 Coverage Frequency Profiling

We first illustrate the formulation of the DNN model. A deep neural network model can be abstracted to be a set of layers: $M = \{L_1, L_2, \cdots, L_K\}$, where K is the number of layers (called depth). L_1 is the input layer. Each layer L_k consists of several neurons: $L_k = \{n_{k,1}, n_{k,2}, \cdots, n_{k,t_k}\}$ and t_k denotes the number of neurons in the layer L_k. Layer L_k ($k > 1$) contains a weight matrix $\mathbf{w}_k \in \mathbb{R}^{t_{k-1} \times t_k}$ that connects all the neurons of the preceding layer to the current layer. Each value (weight) $w_{k,i,j} \in \mathbf{w}_k$ connects the neuron $n_{k-1,i}$ to the neuron $n_{k,j}$. For presentation simplicity, we mainly concentrates on the linear layers in DNN models to describe our methodology. However, our technique is also applicable to other model architectures (as shown in the experiments of Section 5).

Let $\mathbf{h}_k$ denote the neuron activation (internal feature) of the k-th layer. $\mathbf{h}_{k,i}$ is the activation value of neuron $n_{k,i}$. The computation process of each layer can be formulated as (omitting the bias term for simplicity):

$$\mathbf{h}_{k+1} = \text{ReLU}(\mathbf{w}_k \cdot \mathbf{h}_k), \tag{3}$$

where ReLU is a popular activation function. We define the output of the model M on the input x as the set of all internal features:

$$\text{Run}(M, x) = \{\mathbf{h}_2, \mathbf{h}_3, \cdots, \mathbf{h}_K\}. \tag{4}$$

We then define the coverage metric of DNN. We focus on the widely used neuron coverage (NC). Let Cov represent the neuron coverage function, which takes the internal features as input and outputs the indexes of the covered neuron. The neuron coverage function finds the neurons whose activation values are larger than a threshold α. The neuron coverage on the sample x is formulated as:

$$\begin{aligned}\text{Cov}(x) &= \text{Cov}(\ \text{Run}(M, x)\ ) = \text{Cov}(\{\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_K\}) \\ &= \{\mathbf{v}_i | \mathbf{v}_i = \mathbb{I}[\mathbf{h}_i > \alpha]\}.\end{aligned} \tag{5}$$

Each vector $\mathbf{v}_i$ represents which neurons at layer L_i are covered by the data x.

ReMoS first iterates the student dataset and records the coverage information for each sample. Then the neuron coverage frequency on the dataset D^S is computed as:

$$\text{Cov}(D^S) = \{\sum_{x \in D^S} \text{Cov}(x)_i | i = 2, 3, \cdots, K\}. \tag{6}$$

The coverage frequency of the weight $w_{k,i,j}$ is the sum of the neuron coverage frequency of two neurons that this weight connects:

$$\text{CovW}(D^S)_{k,i,j} = \text{Cov}(D^S)_{k-1,i} + \text{Cov}(D^S)_{k,j} \tag{7}$$

The intuition is that if one neuron is frequently covered by D^S, the relevant weights should be frequently covered as well. Because the student dataset is often small, profiling does not cost much time or computation resources.

4.3 Ordinal Score Computation

This step computes the ordinal scores to determine which weights should be included in the relevant slice. The score combines the static information of the teacher model (weight magnitude) and the dynamic information on the student task (weight coverage frequency).

Under the premise that the knowledge in the slice is sufficient for fine-tuning, we want the slice to contain as little relevant knowledge as possible. In this way, the risk of containing defect-related knowledge is reduced. The weight magnitude is an important measurement of the teacher's knowledge (the larger the weight magnitude is, the more significant it is to the teacher). Therefore, we use the absolute weight values inside the slice as an indicator of the amount of teacher knowledge. We formulate the objective of ReMoS as:

$$\mathbf{w}^{ReMoS} = \underset{\mathbf{w} \subset \mathbf{w}^T}{\arg\max}\ \ ACC(T(\mathbf{w}), D^S) - \sum_{w \in \mathbf{w}} |w| \tag{8}$$

Our heuristic solution to the above function is to select the weights with small magnitude and large student-task relevance. However, one challenge of combining the two indexes is the different value range.

Take ResNet18 for example, the range of magnitude is $[10^{-12}, 1.02]$. The range of weight coverage frequency on the MIT Indoor Scenes dataset [57] is $[0, 5374]$. To solve this problem, ReMoS uses the order of one weight inside the whole model weight set as a unified metric.

Formally, for the weight $w_{k,i,j}$, the ordinal magnitude score and the ordinal coverage score is computed by:

$$\begin{aligned} ord\_mag_{k,i,j} &= rank(|w_{k,i,j}|), \\ ord\_cov_{k,i,j} &= rank(\mathrm{CovW}(D^S)_{k,i,j}), \end{aligned} \tag{9}$$

where $rank$ computes the index of the value in the ascending-sorted array of all weights. The ordinal magnitude score ord_mag is the position of $|w_{k,i,j}|$ in the sorted array. The ordinal coverage score ord_cov is the position of the weight coverage frequency $\mathrm{CovW}(D^S)_{k,i,j}$.

The next step combines the two types of ordinal scores. The intuition is that the weights with large ordinal coverage scores and small ordinal magnitude scores should be included in the relevant slice. Thus, we set the overall ordinal score as the ordinal coverage score minus the ordinal weight score:

$$ord_{k,i,j} = ord\_cov_{k,i,j} - ord\_mag_{k,i,j}. \tag{10}$$

4.4 Relevant Slice Generation

This step identifies which weights should be included in the slice based on the ordinal scores.

Specifically, we rank the weights by the ordinal score, and the weights with larger scores are selected as the relevant slice, *i.e.*

$$slice(D^S) = \{w_{k,i,j} | ord_{k,i,j} > t_\theta\}. \tag{11}$$

where $\theta \in [0, 1]$ is a hyper-parameter representing the ratio of weights to be selected into the relevant slice (*i.e.* $|\mathbf{w}|/|\mathbf{w}^\mathbf{T}|$. t_θ is the ordinal score threshold to control the size of the slice.

Finally, the weights in $slice(D^S)$ are used to initialize the student model. The weights outside of $slice(D^S)$ are regarded as less relevant to the student task and are set to zero to forget defect-related knowledge in the teacher. The initialized student model is fine-tuned with the student dataset, just like the normal transfer learning process, to generate the final student model.

5 EVALUATION

To evaluate ReMoS, we mainly focus on the following aspects:

- *Defect mitigation effectiveness*: How effective is ReMoS to mitigate the inheritance of common defects? How much accuracy needs to be sacrificed? How does it compare to other techniques? (In Section 5.2 and Section 5.3)
- *Generalizability*: Is ReMoS general enough for different transfer learning tasks? (We test our method on both CV and NLP tasks, different models, different datasets, and different defects to evaluate the generalizability.)
- *Efficiency*: How much time does ReMoS spend to get a satisfactory student model? How is its efficiency compared to conventional transfer learning? (In Section 5.4)
- *Interpretability*: How is the model trained by ReMoS different from other student models? Why is ReMoS effective and efficient? (In Section 5.5)

5.1 Experimental Setup

Overall methodology. In our experiments, we directly adopted various attacks in the prior literature [20, 21, 29, 31, 32, 55, 58, 70] to evaluate ReMoS. We first downloaded pretrained models from the Internet as the teacher models. Then we used our technique and other baseline methods to train student models from the teachers. We followed the prior attack literature, acted as the attackers who have access to the teacher models, and tried to generate malicious inputs that can fool the student models. These inputs were used to measure how many defects each student has inherited from the teacher (if more defects are inherited, the student will have a higher error rate on the malicious inputs.) The models, datasets, and defects used for CV tasks and NLP tasks are different, which will be introduced in the corresponding subsections.

Implementation We implement ReMoS on PyTorch 1.7 [53] for the ease of customizing DL training procedures. For the neuron coverage implementation, we slightly changed the code base of EvalDNN [63] to profile the coverage frequency.

Evaluation metrics. There are mainly two metrics compared across different student training techniques in our experiments, including accuracy (ACC) and defect inheritance rate (DIR). The accuracy is computed by $ACC = mean_{(x,y)\in D^S}\mathbb{I}[f(x) = y]$. The defect inheritance rate is computed as the misclassification rate on the malicious dataset D^M, *i.e.* $DIR = mean_{(\tilde{x},y)\in D^M}\mathbb{I}[f(\tilde{x}) \neq y]$. D^M is formed by the generated misclassified samples from D^S.

The accuracy and defect inheritance rate may differ on different datasets, making it difficult to directly compare the performance of different student training techniques. Thus, we additionally introduce the relative accuracy and relative defect inheritance rate for comparison. Suppose ACC^s_{ft} is the accuracy achieved by the conventional fine-tuning technique on student task s and S is the set of student tasks. For each student training technique, the mean relative accuracy is computed as $rACC_m = mean_{s\in S}\{\frac{ACC^s_m}{ACC^s_{ft}}\}$ (*i.e.* the mean accuracy compared to the traditional fine-tuning technique), and the mean relative defect inheritance rate is $rDIR_m = mean_{s\in S}\{\frac{DIR^s_m}{DIR^s_{ft}}\}$.

5.2 Defect Mitigation on CV Tasks

This section and Section 5.3 discuss the effectiveness of ReMoS on CV and NLP tasks respectively.

Setup. **Models and Datasets** We used two state-of-the-art large-scale CNN models ResNet18 and ResNet50 [24] on five popular transfer learning datasets, including MIT Indoor Scenes [57], Caltech-UCSD Birds [67], 102 Category Flowers [50], Stanford 40 Actions [69] and Stanford Dogs [30]. The selected datasets are representative with the number of classes ranges from 40 to 200. The number of parameters of the two models are 11M and 25M [2].

Baselines. We compare ReMoS with six representative baselines, including fine-tuning, retraining, DELTA, mag-pruning, DELTA-R, and Renofeation. Among them, fine-tuning and retraining do not incorporate any defense at all. DELTA is a technique that is widely used by deep learning developers to improve accuracy [34]. This technique encourages the student model to have similar feature maps as the teacher model. Mag-pruning is a state-of-the-art "*fix-after-transfer*" approach for backdoor removal [44]. This technique prunes the small-magnitude weights. We followed the implementation of [44] and set the pruning ratio to 80%.

DELTA-R and Renofeation are two representative "*fix-before-transfer*" techniques. DELTA-R, similar to DELTA, cooperates feature regulation to learn from the teacher. Renofeation is a recent work to mitigate adversarial vulnerability inheritance on CV tasks [11], which cooperates feature regulation, internal dropout, and stochastic weighted average to train the student model.

We used the configurations (learning rate, momentum, etc.) that can achieve optimal accuracy on the student tasks. For the "*fix-after-transfer*" techniques, we set the learning rate to 1e-2. For the "*fix-before-transfer*", we set a larger learning rate to 1e-1. We set the number of training iterations as 30K and keep it the same across all techniques.

Defect Simulation. To simulate the situation of defect inheritance, we need to generate malicious inputs based on the teacher model and test whether they lead to misclassification on the student model. We consider two kinds of adversarial attacks, including the penultimate-layer-guided adversarial attacks and the neuron-coverage-guided adversarial attacks. For the former one, we followed the state-of-the-art attack introduced by Rezaei *et al*. [58] to generate a set of malicious inputs D^M. We selected this attack because it is one of the known strongest attacks in white-box setting [61].

For the neuron-coverage-guided attacks, motivated by the prior work on DNN testing [33], we combined various coverage metrics and different strategies to generate D^M. The coverage metrics include neuron coverage (NC), top-k neuron coverage (TKNC), and strong neuron activation coverage (SNAC). TKNC covers the neurons that have the highest k activation values (we select k as 10 follows [33]). SNAC covers the neurons that the activation value is high enough inside the layer. The strategies include DeepXplore (randomly selects inactivated neurons as the target neurons to activate) [55], DLFuzz (selects the most covered neurons) and $\text{DLFuzz}_{\text{RR}}$ (combines three strategies in a round-robin manner [21]). For each sample, we optimized the input gradient to cover as many internal neurons as possible. We selected the last sample during the optimization procedure as the malicious sample.

The simulation of backdoor defects was slightly more complicated. We followed the settings of Latent Backdoor *et al*. [70] and poisoned the training data of the teacher model. We assumed the attacker had a training dataset of a similar data distribution to the student dataset. Some samples in the attacker's dataset were attached by a trigger sign (*e.g*. a Firefox logo) and labeled mistakenly on purpose. By tuning the pretrained model on the attacker's dataset, we obtained a backdoored pretrained model. Then the backdoored pretrained model was used as the teacher to generate student models.

Results. The defect reduction effectiveness of ReMoS on penultimate-layer-guided adversarial samples, neuron-coverage-guided adversarial samples, and backdoor samples are shown in Figure 5, Figure 6, and Figure 7, respectively. Generally, ReMoS significantly reduces defect inheritance with minimal accuracy loss. The discussion on each kind of defect is detailed below.

Adversarial vulnerability. We first discuss the two types of adversarial samples. Figure 5 shows the defect mitigation results on penultimate-layer-guided adversarial samples. We plot the results on all of the five datasets. The first row displays the student accuracy and the second row shows the defect inheritance rate.

Conventional transfer learning achieves high accuracy on the student datasets, although at the cost of high *DIR*s. In Figure 5, 8 out of 10 cases have higher *DIR* than 50%. The phenomenon implies that the inheritance of adversarial vulnerability is practical and the developers should pay attention to it. Retraining has the lowest *DIR* but the accuracy is significantly lower ($rACC_m = 0.58$).

DELTA can partially increase the accuracy in some cases. However, it's *DIR*s are significantly higher (over 75% on ResNet18). Mag-pruning is ineffective to reduce the inherited adversarial vulnerability ($rDIR_m = 1.06$), which demonstrates that the low-magnitude weights are not critical to the teacher model's knowledge, including the defect-related knowledge. We can also observe that generally the *DIR*s of ResNet50 are lower than ResNet18. It's probably because that the increased model capacity helps to increase the robustness. This observation aligns with the prior work [48].

ReMoS only sacrifices less than 2% model accuracy and reduces over 75% inherited defects than the conventional fine-tuning (the $rDIR_m$ is 0.25 for ResNet18 and 0.22 for ResNet50). Besides, in many cases, the *DIR* of ReMoS are comparable with the students trained from scratch (minimum value for *DIR*).
The two "fix-before-transfer" techniques (DELTA-R and Renofeation) can partially improve the accuracy of the student models based on retraining, but the performance is still lower than the fine-tuned students. Besides, we can observe that such techniques have lower accuracies on ResNet50 than ResNet18. We think the reason is that the increased complexity of ResNet50 makes it more difficult to train on the small-scale student dataset.

Similarly, the defect reduction results on neuron-coverage-guided adversarial samples are shown in Figure 6. For the fine-tuned students, the *DIR*s (averagely 51.92%) are lower than that of penultimate-layer-guided samples (averagely 75.50%). We think the reason is that when generating adversarial samples, targeting on the middle layers is less effective than targeting on the penultimate layer. The latter can directly fool the last FC layer. On the contrary, the turbulence made on the intermediate layers may be corrected by subsequent layers (which are trained to have fault tolerance). For neuron-coverage-guided samples, the target layers contain a large number of intermediate layers.

It can also be observed that the inheritance rates differ between various coverage metrics and strategies. For the coverage strategies, the defects discovered by DLFuzz inherit most in transfer learning (averagely 59.85%). It means that selecting the most covered neurons can generate easier-to-inherit adversarial samples. For the coverage metrics, TKNC is the worst effective (the average *DIR* is 40.14%). We think the reason is that the top k neurons are too little compared to the enormous neurons inside the model.

ReMoS reduces the inheritance rates in all cases by a large margin. The $rDIR_m$ is 0.37 over the two datasets. On average, 63% of inherited defects that are generated by the neuron-coverage-guided techniques can be eliminated by ReMoS.

Backdoor. The backdoor mitigation result is shown in Figure 7. The average inheritance rate is 72.91% for fine-tuning and 62.61% for mag-pruning. It means the backdoor knowledge is partially encoded in the low-magnitude weights. ReMoS reduces the inherited defects to a similar level of retraining (the $rDIR_m$ is 0.14). It means

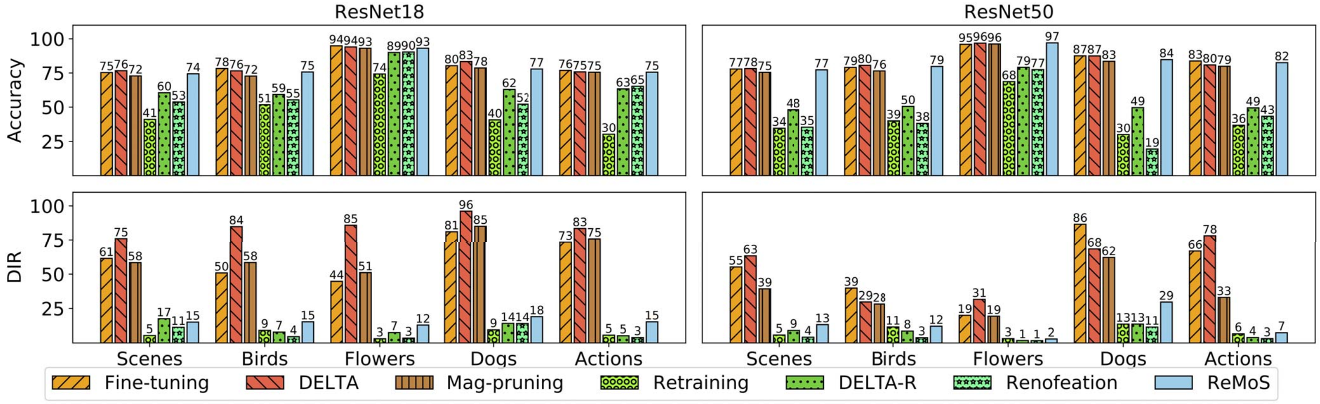


Figure 5: The accuracy (*ACC*) and adversarial vulnerability inheritance rate (*DIR*) of ReMoS and other student training techniques.

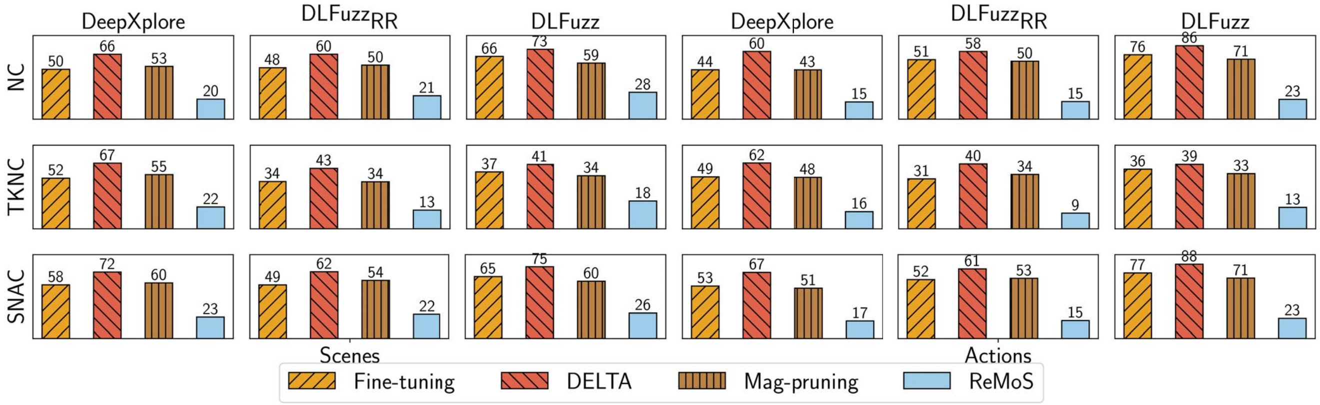


Figure 6: The inheritance rate of adversarial inputs generated by different neuron-coverage-guided test generators on ResNet18.

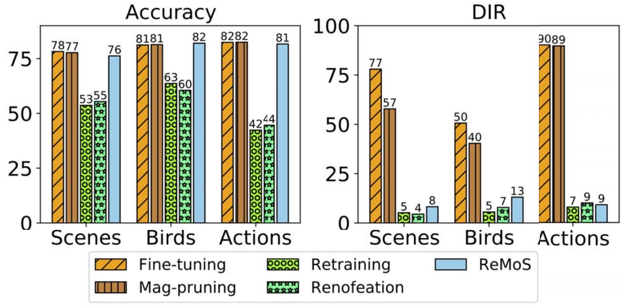


Figure 7: The accuracy (*ACC*) and backdoor inheritance rate (*DIR*) of ReMoS and other baselines on ResNet50.

that the large-magnitude weights excluded by ReMoS are critical to the backdoor knowledge. After resetting such weights, the backdoor defect is mitigated by a large margin.

5.3 Defect Mitigation on NLP Tasks

Setup. **Models and Datasets.** We used two popular models BERT and RoBERTa in the experiment [46, 68]. ReMoS can also be used for other NLP pretrained models. The datasets are SST-2 [62] and IMDB [47] for backdoor attacks and SST-2 and QNLI [56] for adversarial attacks following the prior settings [32, 49].

Baselines. We took two baselines from the CV experiments: conventional fine-tuning and magnitude-based pruning. The other baselines are not designed for NLP models and cannot achieve reasonable accuracy, thus were excluded from the comparison.

Defect Simulation. The backdoor implementation followed the prior work [32] which considers two scenarios: full domain knowledge (FDK) and domain shift (DS). FDK means that the attacker knows the student dataset and uses the identical dataset to embed the backdoor. DS means the attack dataset is different from the victim dataset. We used SST-2 and IMDB as the attack dataset and the victim dataset to form the four scenarios. We adopted two effective attack techniques from [32]: data poisoning and weight poisoning.

For adversarial vulnerability, we assumed the FDK scenario. We adopted two public implementations from TextAttack: Kuleshov [31] and TextFooler [29]. Kuleshov uses a greedy word swap strategy to find the adversarial samples. TextFooler uses the greedy word importance ranking algorithm.

Results. The defect reduction results of ReMoS on the backdoor and adversarial samples are displayed in Table 2 and Table 3, respectively. The defect inheritance rate is reduced by 50% and 61%

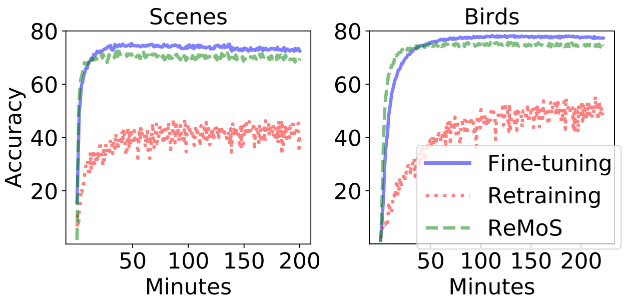


Figure 8: The convergence progress of conventional transfer learning, retraining, and ReMoS.

for two kinds of backdoor samples. ReMoS can also reduce 40% inherited adversarial vulnerabilities. Then we will summarize the results on the two defects.

Backdoor. As shown in Table 2, the backdoor attack is very effective and all *DIR*s of fine-tuning are over 88%, meaning that the backdoor inheritance problem in NLP tasks is more severe than that in CV. We think there are two reasons. First, for NLP tasks, the high-dimensional trigger vector (translated from the trigger word) is more distinct than triggers CV tasks. The difference between vectors is larger than the difference between CV images (pixel values are between 0 and 1 after normalized). Second, NLP models contain much more parameters. BERT has 110M parameters [1] while ResNet50 has 25M parameters [2]. The more redundant parameters may be easier to convey the backdoor logic.

With ReMoS, the $rDIR_m$ is 0.50 for data poisoning and 0.39 for weight poisoning, and the relative accuracy $rACC_m$ is about 0.97. The $rDIR_m$ of mag-pruning is 0.98. Both ReMoS and mag-pruning are less effective on NLP models than the CV models. This phenomenon implies that reducing the NLP backdoor inheritance might be more difficult.

Adversarial vulnerability. Table 3 displays the results on the inherited adversarial vulnerability. For simplicity, we report the average *DIR* over two attacks. For fine-tuned students, the average *DIR* is 79.80%. Similar to the backdoor, the *DIR*s on NLP models are higher than the CV models. Mag-pruning can partially reduce the *DIR* (the $rDIR_m$ is 0.76). The $rDIR_m$ of ReMoS is much lower (0.60) with slight accuracy drop ($rACC_m$ = 0.99).

5.4 Efficiency and Overhead

We compare the training efficiency of ReMoS with fine-tuning and retraining in Figure 8. The convergence speed of ReMoS is close to fine-tuning and significantly faster than retraining. Both ReMoS and fine-tuning achieve the optimal accuracy within 50 minutes on the Scenes dataset and within 100 minutes on the Birds dataset.

Since the training process of ReMoS does not introduce additional time cost, the overhead of ReMoS is just the slice computation time. The profiling step in slice computation is similar to performing a forwarding pass with the training dataset, which only takes several minutes. After profiling, computing the ordinal scores and selecting the relevant weights take less than a minute on ResNet18 and about three minutes on ResNet50. Meanwhile, ReMoS can be easily integrated into the conventional transfer learning pipeline without any task-specific configuration. Therefore, we believe ReMoS is efficient and easy-to-use enough for student model developers.

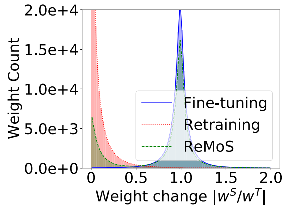


Figure 9: The distribution of weight changes during training.

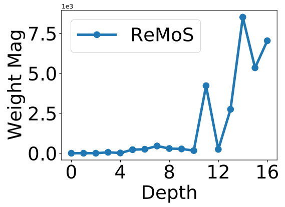


Figure 10: The magnitude of weights excluded by ReMoS in each layer.

5.5 Interpreting the Slice

To further understand the efficiency and effectiveness of ReMoS, we inspect the weight pattern of the student model. First, we measured the weight changes of the student models before and after training. Specifically, we computed the weight difference w^S/w^T between the student model and the teacher model. The result is shown in Figure 9. For conventional transfer learning (fine-tuning), most of the weight changes are around 1.0. It means that most weights do not have to change too much to fit the student dataset. Retraining suffers from low performance because the weight changes are much larger and the small student dataset is not enough. ReMoS significantly reduces the number of weights that require large-range adjustment, thus the convergence speed is high.

We also studied the magnitude distribution of the weights excluded from the relevant slice (*i.e.* the weights that are reset to zero in ReMoS). The result is shown in Figure 10. As the layer depth increases, more weights with higher magnitude are excluded. This result is intuitive because, in a DNN model, the weights at deep layers are usually related to high-level features, thus may be less relevant to the specific student tasks. Instead, some of them may contribute a lot to certain defect-related decision logic. ReMoS excludes these weights from the student model, thus it can reduce defect inheritance while retaining useful knowledge.

6 RELATED WORK

The transferability of both adversarial attacks and backdoor attacks has been extensively studied before. Wang *et al.* [64] and Rezaei *et al.* [58] proposed to generate adversarial inputs targeting intermediate layers of the teacher models. These techniques are similar to neuron-coverage-guided DNN testing techniques. BadNets [20] and TrojanNN [45] have demonstrated the backdoor attacks are inheritable in their experiments. Various approaches [66, 70] have also been proposed to make the backdoors more transferable. Similarly, Ji *et al.* presented model-reuse attacks wherein malicious primitive models may infect host ML systems [28].

To mitigate defect inheritance, one way is to improve the robustness of teacher models and making the robustness transferable [12, 59]. However, in this paper, we focus on the student model developers' side and consider how to avoid inheriting defects from public teacher models that may be insecure.

A straightforward solution for student model developers to reduce defect inheritance is "*fix-after-transfer*", in which the developers

Table 2: The defect reduction effectiveness of ReMoS against two backdoor attacks on NLP tasks. For each model, we include four situations where the attacker's dataset may be the same or different as the student dataset.

Model		Dataset	Data Poisoning						Weight Poisoning					
			Fine-tune		Mag-prune		ReMoS		Fine-tune		Mag-prune		ReMoS	
			ACC	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*
BERT	FDK	SST-2 to SST-2	92.70	100.00	92.35	100.00	91.27	39.09	92.29	100.00	92.44	100.00	90.92	29.82
		IMDB to IMDB	87.96	96.11	88.24	96.15	85.53	61.73	89.34	96.15	89.48	96.09	87.00	37.72
	DS	SST-2 to IMDB	90.53	100.00	91.26	100.00	90.04	74.67	91.67	100.00	91.16	100.00	87.42	61.48
		IMDB to SST-2	93.21	96.17	92.46	96.17	91.15	27.71	92.80	96.22	92.58	96.02	91.94	21.55
RoBERTa	FDK	SST-2 to SST-2	94.19	100.00	93.70	100.00	91.17	29.82	93.37	100.00	93.19	98.93	90.70	24.94
		IMDB to IMDB	90.60	93.52	89.54	95.24	85.74	70.19	89.05	96.53	88.76	92.05	86.34	85.91
	DS	SST-2 to IMDB	92.11	99.88	92.27	100.00	90.32	24.14	91.85	100.00	90.82	99.53	88.71	30.83
		IMDB to SST-2	93.52	88.15	92.65	85.26	92.17	61.26	93.85	93.93	93.57	91.21	89.95	18.07
Average Relative Value			-	-	0.99	0.99	0.97	0.50	-	-	0.99	0.98	0.97	0.39

Table 3: The accuracy (*ACC*) and adversarial vulnerability inheritance rate (*DIR*) of ReMoS on NLP tasks.

Model	Dataset		Fine-tune	Mag-prune	ReMoS
BERT	SST-2	*ACC*	92.20	93.43	92.03
		DIR	85.30	67.70	62.23
	QNLI	*ACC*	89.42	88.84	88.45
		DIR	74.94	62.11	45.16
RoBERTa	SST-2	*ACC*	94.40	94.06	92.08
		DIR	84.94	58.48	49.46
	QNLI	*ACC*	90.25	91.09	89.60
		DIR	74.02	56.61	36.36
Average Relative Value		$rACC_m$	-	1.00	0.99
		$rDIR_m$	-	0.76	0.60

first generate a student model following the standard transfer learning procedure, and then use normal defect mitigation techniques [44, 48] to remove the defects. However, due to the shortage of data and limited knowledge about defects, such techniques (like adversarial training [48]) are uneasy and ineffective. Fine-pruning [44] is probably the most easy-to-use method against backdoors. Nevertheless, the experiments show that it can not guarantee effectiveness against adversarial vulnerabilities.

The "*fix-before-transfer*" approach initializes the student model from scratch and extracts knowledge from the teacher model during transfer learning. This technique suffers from the performance of the student model and the utility. Renofeation [11] uses knowledge distillation together with several heuristic tricks to recover the model accuracy. Nevertheless, its performance is inferior and the utility is constrained to CNN models.

7 THREAT TO VALIDITY AND DISCUSSIONS

One possible threat to validity is that the studied samples in the experiment may not be representative. We tried our best to minimize this threat by covering as diverse vulnerabilities as possible, including vulnerabilities on different domains (CV and NLP), different model architectures (ResNet and BERT), and different datasets (eight datasets in total).

In this paper, we focus on the scenario of two models (the teacher and the student). However, there are three models in the real-world backdoor setting. The *generalized-teacher* model is the publicly-recognized model by the community. This model can be fine-tuned by an attacker to inject backdoor and become a *domain-teacher* model. The domain-teacher model is published on the Internet, downloaded by a deep learning developer, and further fine-tuned into the *domain-student* model (which inherits the defects). We simplify the three-model setting into two models because, from the DL developers' perspective, the generalized teacher and the domain teacher are essentially the same. Both of them may contain vulnerabilities that may be inherited by the developers' student model.

For the retraining baseline, we think it is not suitable for the orinary DL developers, although at some point retraining achieves enough accuracy. Retraining large DL models from scratch requires abundant dataset, multiple expensive GPU, and various training tricks. It is difficult for ordinary DL developers to meet all these requirements. Thus, reusing public model on the Internet becomes a widely-accepted choice.

One intuitive solution to reduce the inherited defects is to dig into the teacher model and deal with the important vulnerabilities. Because the teacher models are widely accessible on the Internet, there should exist common vulnerabilities that have been understood by the public. These vulnerabilities are called "universal adversarial vulnerability" [35], and only accounts for a small part of the adversarial vulnerabilities in the teacher model. Thus, this solution maybe not effective enough to reduce as much inherited defects as possible.

8 CONCLUSION

This paper proposes ReMoS, a relevant model slicing technique to reduce defect inheritance during transfer learning. ReMoS only reuses the relevant parts inside the teacher model and retrains the irrelevant parts to forget the defect-related knowledge. The relevant slice is computed based on the neuron coverage information by profiling the teacher model on the student dataset. Extensive experiments on seven DNN defects, four DNN models, and eight datasets demonstrate the effectiveness of the approach. ReMoS can reduce inherited defects by 63% to 86% for CV tasks and by 40% to 61% for NLP tasks with an accuracy sacrifice less than 3%.

ACKNOWLEDGMENTS

We would like to thank the anonymous ICSE reviewers for their valuable feedback of this paper. This work was partly supported by the National Natural Science Foundation of China (62141208), and a project sponsored by ZTE Communications. We thank Xihan Zhang for the CV backdoor implementation.

REFERENCES

[1] [n. d.]. BERT Explained – A list of Frequently Asked Questions. https://huggingface.co/transformers/pretrained_models.html.

[2] [n. d.]. Keras Applications. https://keras.io/api/applications/.

[3] [n. d.]. Pytorch Hub. https://pytorch.org/hub/.

[4] [n. d.]. Tensorflow Hub. https://www.tensorflow.org/hub.

[5] Hiralal Agrawal, Joseph Robert Horgan, Edward W. Krauser, and Saul London. 1993. Incremental Regression Testing. In *Proceedings of the Conference on Software Maintenance, ICSM 1993, Montréal, Quebec, Canada, September 1993*, David N. Card (Ed.). IEEE Computer Society, 348–357. https://doi.org/10.1109/ICSM.1993.366927

[6] Michael Backes, Sven Bugiel, and Erik Derr. 2016. Reliable third-party library detection in android and its security applications. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 356–367.

[7] Teodora Baluta, Zheng Leong Chua, Kuldeep S Meel, and Prateek Saxena. 2020. Scalable quantitative verification for deep neural networks. *arXiv preprint arXiv:2002.06864* (2020).

[8] Ravi Bhoraskar, Seungyeop Han, Jinseong Jeon, Tanzirul Azim, Shuo Chen, Jaeyeon Jung, Suman Nath, Rui Wang, and David Wetherall. 2014. Brahmastra: Driving apps to test the security of third-party components. In *23rd USENIX Security Symposium (USENIX Security 14)*. 1021–1036.

[9] David Binkley. 1998. The application of program slicing to regression testing. *Information and software technology* 40, 11-12 (1998), 583–594.

[10] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. 2016. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 357–376.

[11] Ting-Wu Chin, Cha Zhang, and Diana Marculescu. 2021. Renofeation: A Simple Transfer Learning Method for Improved Adversarial Robustness. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 3243–3252.

[12] Todor Davchev, Timos Korres, Stathi Fotiadis, Nick Antonopoulos, and Subramanian Ramamoorthy. 2019. An Empirical Evaluation of Adversarial Robustness under Transfer Learning. *CoRR* abs/1905.02675 (2019).

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. 2014. The matter of heartbleed. In *Proceedings of the 2014 conference on internet measurement conference*. 475–488.

[15] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 177–188.

[16] William B Frakes and Kyo Kang. 2005. Software reuse research: Status and future. *IEEE transactions on Software Engineering* 31, 7 (2005), 529–536.

[17] Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. 2020. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 1147–1158. https://doi.org/10.1145/3377811.3380415

[18] Antonios Gkortzis, Daniel Feitosa, and Diomidis Spinellis. 2019. A Double-Edged Sword? Software Reuse and Potential Security Vulnerabilities. In *International Conference on Software and Systems Reuse*. Springer, 187–203.

[19] Martin L Griss. 1998. *Software reuse: architecture, process and organization for business success*. IEEE.

[20] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[21] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 739–743. https://doi.org/10.1145/3236024.3264835

[22] Tibor Gyimóthy, Árpád Beszédes, and István Forgács. 1999. An Efficient Relevant Slicing Method for Debugging. In *Software Engineering - ESEC/FSE'99, 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Toulouse, France, September 1999, Proceedings (Lecture Notes in Computer Science)*, Oscar Nierstrasz and Michel Lemoine (Eds.), Vol. 1687. Springer, 303–321. https://doi.org/10.1007/3-540-48166-4_19

[23] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1510.00149

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[25] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. 2011. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. 43–58.

[26] Yujin Huang, Han Hu, and Chunyang Chen. 2021. Robustness of on-device Models: Adversarial Attack to Deep Learning Models on Android Apps. *arXiv preprint arXiv:2101.04401* (2021).

[27] Dennis Jeffrey and Neelam Gupta. 2008. Experiments with test case prioritization using relevant slices. *J. Syst. Softw.* 81, 2 (2008), 196–221. https://doi.org/10.1016/j.jss.2007.05.006

[28] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-Reuse Attacks on Deep Learning Systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 349–363. https://doi.org/10.1145/3243734.3243757

[29] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2020. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 8018–8025.

[30] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. 2011. Novel Dataset for Fine-Grained Image Categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*. Colorado Springs, CO.

[31] Volodymyr Kuleshov, Shantanu Thakoor, Tingfung Lau, and Stefano Ermon. 2018. Adversarial examples for natural language classification problems. (2018).

[32] Keita Kurita, Paul Michel, and Graham Neubig. 2020. Weight Poisoning Attacks on Pretrained Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2793–2806.

[33] Seokhyun Lee, Sooyoung Cha, Dain Lee, and Hakjoo Oh. 2020. Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, Sarfraz Khurshid and Corina S. Pasareanu (Eds.). ACM, 165–176. https://doi.org/10.1145/3395363.3397346

[34] Xingjian Li, Haoyi Xiong, Hanchao Wang, Yuxuan Rao, Liping Liu, and Jun Huan. 2019. Delta: Deep Learning Transfer using Feature Map with Attention for Convolutional Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=rkgbwsAcYm

[35] Yingwei Li, Song Bai, Cihang Xie, Zhenyu Liao, Xiaohui Shen, and Alan Yuille. 2019. Regional Homogeneity: Towards Learning Transferable Universal Adversarial Perturbations Against Defenses. *arXiv preprint arXiv:1904.00979* (2019).

[36] Yingwei Li, Song Bai, Yuyin Zhou, Cihang Xie, Zhishuai Zhang, and Alan Yuille. 2020. Learning Transferable Adversarial Examples via Ghost Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34.

[37] Yuanchun Li, Jiayi Hua, Haoyu Wang, Chunyang Chen, and Yunxin Liu. 2021. DeepPayload: Black-box Backdoor Attack on Deep Learning Models through Neural Payload Injection. *arXiv preprint arXiv:2101.06896* (2021).

[38] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. 2021. ModelDiff: Testing-Based DNN Similarity Comparison for Model Reuse Detection. *Proceeding of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (2021).

[39] Junyu Lin, Lei Xu, Yingqi Liu, and Xiangyu Zhang. 2020. Composite Backdoor Attack for Deep Neural Network by Mixing Existing Benign Features. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 113–131. https://doi.org/10.1145/3372297.3423362

[40] Bingyan Liu, Yifeng Cai, Yao Guo, and Xiangqun Chen. 2021. TransTailor: Pruning the Pre-trained Model for Improved Transfer Learning. *Proceedings of the 35th AAAI Conference on Artificial Intelligence* (2021).

[41] Bingyan Liu, Yao Guo, and Xiangqun Chen. 2019. WealthAdapt: A general network adaptation framework for small data tasks. In *Proceedings of the 27th ACM International Conference on Multimedia*. 2179–2187.

[42] Bingyan Liu, Yuanchun Li, Yunxin Liu, Yao Guo, and Xiangqun Chen. 2020. Pmc: A privacy-preserving deep learning model customization framework for edge computing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020), 1–25.

[43] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. 2019. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758* (2019).

[44] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2018. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 273–294.

[45] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. 2018. Trojaning Attack on Neural Networks. In *25th Annual Network and Distributed System Security Symposium (NDSS)*. 18–221.

[46] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[47] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, Dekang Lin, Yuji Matsumoto, and Rada Mihalcea (Eds.). The Association for Computer Linguistics, 142–150. https://aclanthology.org/P11-1015/

[48] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJzIBfZAb

[49] John Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 119–126.

[50] Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated Flower Classification over a Large Number of Classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*.

[51] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[52] Ivan Pashchenko, Henrik Plate, Serena Elisa Ponta, Antonino Sabetta, and Fabio Massacci. 2018. Vulnerable open source dependencies: Counting those that matter. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–10.

[53] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[54] Brandon Paulsen, Jingbo Wang, and Chao Wang. 2020. ReluDiff: differential verification of deep neural networks. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 714–726. https://doi.org/10.1145/3377811.3380337

[55] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.

[56] Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis Only Baselines in Natural Language Inference. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics, *SEM@NAACL-HLT 2018, New Orleans, Louisiana, USA, June 5-6, 2018*, Malvina Nissim, Jonathan Berant, and Alessandro Lenci (Eds.). Association for Computational Linguistics, 180–191. https://doi.org/10.18653/v1/s18-2023

[57] Ariadna Quattoni and Antonio Torralba. 2009. Recognizing indoor scenes. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 413–420.

[58] Shahbaz Rezaei and Xin Liu. 2020. A Target-Agnostic Attack on Deep Models: Exploiting Security Vulnerabilities of Transfer Learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=BylVcTNtDS

[59] Ali Shafahi, Parsa Saadatpanah, Chen Zhu, Amin Ghiasi, Christoph Studer, David W. Jacobs, and Tom Goldstein. 2020. Adversarially robust transfer learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=ryebG04YvB

[60] David Shriver, Sebastian Elbaum, and Matthew B. Dwyer. 2021. Reducing DNN Properties to Enable Falsification with Adversarial Attacks. (2021).

[61] Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. 2020. Benchmarking Adversarial Attacks and Defenses for Time-Series Data. In *Neural Information Processing - 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23-27, 2020, Proceedings, Part III (Lecture Notes in Computer Science)*, Haiqin Yang, Kitsuchart Pasupa, Andrew Chi-Sing Leung, James T. Kwok, Jonathan H. Chan, and Irwin King (Eds.), Vol. 12534. Springer, 544–554. https://doi.org/10.1007/978-3-030-63836-8_45

[62] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*. 1631–1642.

[63] Yongqiang Tian, Zhihua Zeng, Ming Wen, Yepang Liu, Tzu-yang Kuo, and Shing-Chi Cheung. 2020. EvalDNN: A toolbox for evaluating deep neural network models. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 45–48.

[64] Bolun Wang, Yuanshun Yao, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. 2018. With Great Training Comes Great Vulnerability: Practical Attacks against Transfer Learning. In *USENIX Security Symposium*. USENIX Association, 1281–1297.

[65] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. 2021. RobOT: Robustness-Oriented Testing for Deep Learning Systems. *arXiv preprint arXiv:2102.05913* (2021).

[66] Shuo Wang, Surya Nepal, Carsten Rudolph, Marthie Grobler, Shangyu Chen, and Tianle Chen. 2020. Backdoor Attacks against Transfer Learning with Pre-trained Deep Learning Models. *arXiv preprint arXiv:2001.03274* (2020).

[67] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. 2010. *Caltech-UCSD Birds 200*. Technical Report CNS-TR-2010-001. California Institute of Technology.

[68] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

[69] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. 2011. Human action recognition by learning bases of action attributes and parts. In *2011 International conference on computer vision*. IEEE, 1331–1338.

[70] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y. Zhao. 2019. Latent Backdoor Attacks on Deep Neural Networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2041–2055. https://doi.org/10.1145/3319535.3354209

[71] Fuyuan Zhang, Sankalan Pal Chowdhury, and Maria Christakis. 2020. Deepsearch: A simple and effective blackbox attack for deep neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 800–812.

[72] Xiyue Zhang, Xiaofei Xie, Lei Ma, Xiaoning Du, Qiang Hu, Yang Liu, Jianjun Zhao, and Meng Sun. 2020. Towards characterizing adversarial defects of deep learning software from the lens of uncertainty. *arXiv preprint arXiv:2004.11573* (2020).

[73] Ziqi Zhang, Yuanchun Li, Yao Guo, Xiangqun Chen, and Yunxin Liu. 2020. Dynamic slicing for deep neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 838–850.