BDefects4NN: A Backdoor Defect Database for Controlled Localization Studies in Neural Networks

Yisong Xiao[1,2], Aishan Liu[1✉],Xinwei Zhang[1], Tianyuan Zhang[1,2], Tianlin Li[3],
Siyuan Liang[4], Xianglong Liu[1,5], Yang Liu[3], Dacheng Tao[3]

[1]SKLCCSE, Beihang University, Beijing, China   [2]Shen Yuan Honors College, Beihang University, Beijing, China
[3]Nanyang Technological University, Singapore
[4]National University of Singapore, Singapore   [5]Zhongguancun Laboratory, Beijing, China

Abstract—**Pre-trained large deep learning models are now serving as the dominant component for downstream middleware users and have revolutionized the learning paradigm, replacing the traditional approach of training from scratch locally. To reduce development costs, developers often integrate third-party pre-trained deep neural networks (DNNs) into their intelligent software systems. However, utilizing untrusted DNNs presents significant security risks, as these models may contain intentional backdoor defects resulting from the black-box training process. These backdoor defects can be activated by hidden triggers, allowing attackers to maliciously control the model and compromise the overall reliability of the intelligent software. To ensure the safe adoption of DNNs in critical software systems, it is crucial to establish a backdoor defect database for localization studies. This paper addresses this research gap by introducing *BDefects4NN*, the first backdoor defect database, which provides labeled backdoor-defected DNNs at the neuron granularity and enables controlled localization studies of defect root causes.**

In *BDefects4NN*, we define three defect injection rules and employ four representative backdoor attacks across four popular network architectures and three widely adopted datasets, yielding a comprehensive database of 1,654 backdoor-defected DNNs with four defect quantities and varying infected neurons. Based on *BDefects4NN*, we conduct extensive experiments on evaluating six fault localization criteria and two defect repair techniques, which show limited effectiveness for backdoor defects. Additionally, we investigate backdoor-defected models in practical scenarios, specifically in lane detection for autonomous driving and large language models (LLMs), revealing potential threats and highlighting current limitations in precise defect localization. This paper aims to raise awareness of the threats brought by backdoor defects in our community and inspire future advancements in fault localization methods.

Index Terms—**Backdoor defects, fault localization, deep learning**

I. INTRODUCTION

Deep Learning (DL) has demonstrated remarkable performance across a wide range of applications and is integrated into diverse software systems, such as autonomous driving [1] and healthcare [2], [3]. A consensus is emerging among developers to employ DL models pre-trained on large-scale datasets for their downstream applications [4]–[8]. By fine-tuning publicly available pre-trained model weights on their specific datasets, developers with limited resources or training data can effortlessly craft high-quality models for a multitude of tasks. As a result, the pre-training and fine-tuning paradigm has gained strong popularity [9].

However, these third-party released DNNs are often pre-trained on large-scale, noisy, and uncurated Internet data that are unknown to downstream users. Utilizing these untrusted DNNs presents significant safety risks, as these models may contain intentional *backdoor defects* resulting from the black-box training process (we refer to models with backdoor defects as "backdoor-defected models", also "infected models" for convenience). These malicious defects in DNN models are caused by backdoor attacks [10], where an attacker adversarially injects backdoored neurons into the victim models by poison training or sub-network replacing, thereby being able to manipulate the model behavior with a specific trigger. For example, a third-party released DNN that is injected with backdoor defects will incorrectly identify lanes triggered by two common traffic cones when deployed into autonomous driving systems [11], thereby compromising the overall reliability of the intelligent systems and endangering human lives. To ensure the safe adoption of DNNs in critical software systems, it is crucial to establish a comprehensive backdoor defect database for localization studies. However, current defect localization databases for DNNs [12]–[14] primarily focus on *common defects*, *i.e.*, unintentional functional bugs introduced by DNN developers such as incorrect tensor shapes, while overlooking the stealthy and harmful defects posed by backdoor attacks (*i.e.*, backdoor defects). This sparsity of research presents severe safety risks to DL systems.

To bridge the gap, this paper takes the first step in constructing a backdoor defect database for localization studies in DNNs. We propose the first comprehensive backdoor defect database *BDefects4NN*, which provides neuron-level backdoor infected DNNs with ground-truth defect labeling to support defect localization studies at the neuron granularity, serving as an essential test suite for our community. Specifically, we propose the defect design protocols to select neurons for defect injection in terms of neuron contribution, neuron quantity, and sub-network correlation. Based on the proposed defects design protocols, we employ four representative backdoor attack methods to inject backdoor defects into four popular network architectures across three widely adopted datasets on the image classification task. Overall, our *BDefects4NN* contains 1,654 backdoor-defected DNNs with ground truth defects labeling, categorized into 48 directories, each featuring injected sub-networks at four (4) quantity levels, offering varying defect

quantities and infected neurons to enable comprehensive evaluation of localization and repair methods.

Using our *BDefects4NN*, we conduct extensive experiments to evaluate the performance of six fault localization criteria, incorporating four backdoor-specific criteria from the backdoor defense domain and two general criteria from the software engineering field. Notably, we identify that current fault localization methods show limited performance on backdoor defects in DNNs with low localization effectiveness (17.64% WJI on average). In addition, we further evaluate two defect repair techniques, namely neuron pruning and neuron fine-tuning, which exhibit an average $ASRD$ of 39.54% and 41.45%, respectively. Furthermore, we extend our investigations to practical scenarios, such as lane detection (LaneATT [15]) for autonomous driving and LLMs (ChatGLM [16]), where we illustrate the potential threats posed by backdoor defects and highlight the current limitations of existing methods in precisely localizing these defects in real-world applications. We hope this paper will raise awareness of backdoor defect threats within our community and facilitate further research on fault localization methods. Our main **contributions** are:

- As far as we know, we pioneer the integration of backdoor defects into the fault localization task and conduct the first comprehensive study on backdoor defect localization in DNNs.
- We build *BDefects4NN*, a comprehensive database containing 1,654 backdoor-defected DNNs with neuron-level ground-truth labeling, supporting controlled defect localization studies.
- We conduct extensive evaluations on six localization criteria and two defect repair methods, offering findings into their strengths and weaknesses.
- We publish *BDefects4NN* as a self-contained toolkit on our website [17].

II. PRELIMINARIES

DNN. Given a dataset D with data sample $x \in X$ and label $y \in Y$, the deep supervised learning model aims to learn a mapping or classification function $F_\Theta : X \to Y$. The model F_Θ consists of L serial layers, with parameters $\Theta = \{\theta_1, ..., \theta_L\}$, and N_l neurons in each layer $l \in \{1, ..., L\}$. The total number of neurons is $N = \sum_{l=1}^{L} N_l$. Further, we denote the activation output of each neuron F_l^i as a_l^i, where $i \in \{1, ..., N_l\}$. Moreover, a sub-network is defined as a pathway within F_Θ that includes at least one neuron in each layer l, where $l \in \{1, ..., L-1\}$. This paper mainly focuses on the image classification task, with its training process as:

$$\Theta = \arg\min_{\Theta} \mathbb{E}_{(x,y)\sim D}[\mathcal{L}(F_\Theta(x), y)], \quad (1)$$

where $\mathcal{L}(\cdot)$ represents the cross-entropy loss function.

Backdoor Attack. Backdoor attacks aim to embed hidden behaviors into a DNN F_Θ during training, allowing the infected model $F_{\hat{\Theta}}$ to behave normally on benign samples. However, the predictions of the infected model undergo malicious and consistent changes when hidden backdoors are activated

by attacker-specified trigger patterns. Presently, poisoning-based backdoor attacks stand as the most straightforward and widely adopted method in the training phase. Specifically, the attacker randomly selects a small portion p (*e.g.*, 10%) of clean data from the training dataset D, and then generates poisoned samples $\hat{D} = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{M}, M = p \cdot |D|$, by applying the trigger T to the images using the function ϕ and modifying the corresponding label to the target label $\hat{y}_i$ as follows:

$$\hat{x}_i = \phi(x_i, T), \quad \hat{y}_i = \eta(y_i). \quad (2)$$

For different backdoor attack methods, the trigger generation function ϕ varies, and η represents the rules governing the modification of poisoning labels. Afterward, the model trained on the poisoned dataset $D \cup \hat{D}$ will be injected with backdoors, yielding target label predictions $F_{\hat{\Theta}}(\hat{x}_i) = \hat{y}_i$ on test images $\hat{x}_i$ containing triggers. Another series (structure-modified attacks) [18]–[21] first trains a backdoor sub-network/module, and then directly injects the sub-network into a benign model Θ to obtain the final infected model $\hat{\Theta}$. Backdoor training involves dual-task learning: the clean task on clean dataset D and the backdoor task on backdoor dataset $\hat{D}$. An infected model should achieve high attack success rate ASR (backdoor task) and competitive clean accuracy CA (clean task):

$$\begin{aligned} CA &= P_{(x,y)\sim D_{test}}(F_{\hat{\Theta}}(x) = y), \\ ASR &= P_{(\hat{x},\hat{y})\sim \hat{D}_{test}}(F_{\hat{\Theta}}(\hat{x}) = \hat{y}), \end{aligned} \quad (3)$$

where D_{test} and $\hat{D}_{test}$ denote the clean test dataset and poisoned test dataset respectively.

Assumption. We further state the ***common assumption*** in infected DNNs: **specific neurons/sub-nets are predominantly responsible for backdoor defects**, which has been widely accepted and empirically demonstrated in backdoor attack and defense studies [18]–[29]. From the perspective of dual-task learning, numerous studies [30], [31] have revealed the fact that neurons in infected models can be decomposed into clean and backdoor neurons since backdoor attacks are designed not to impact the model's performance on clean samples [22], indicating a high level of independence between the clean and backdoor tasks [30]. In poisoning attacks, TrojanNN [23] optimizes triggers to maximize the activation of a few specific neurons for backdoor behavior, while most neurons continue to perform normal functions. In structure-modified attacks [18]–[21], the injected sub-network is responsible for backdoor defects. Furthermore, defense studies design rules based on activation [24], [25], weight [26], [27], and Shapley value [28] to identify and repair the neurons most responsible for backdoor, thereby eliminating the backdoor. Thus, following common assumptions, we aim to inject backdoor defects into DNNs at specific neurons/sub-nets, providing defect labeling to support localization and repair studies.

III. *BDefects4NN* DATABASE

In this section, we first illustrate the motivation and problem definition, then explain the *BDefects4NN* design protocol and construction details. Figure 1 shows the overall framework.
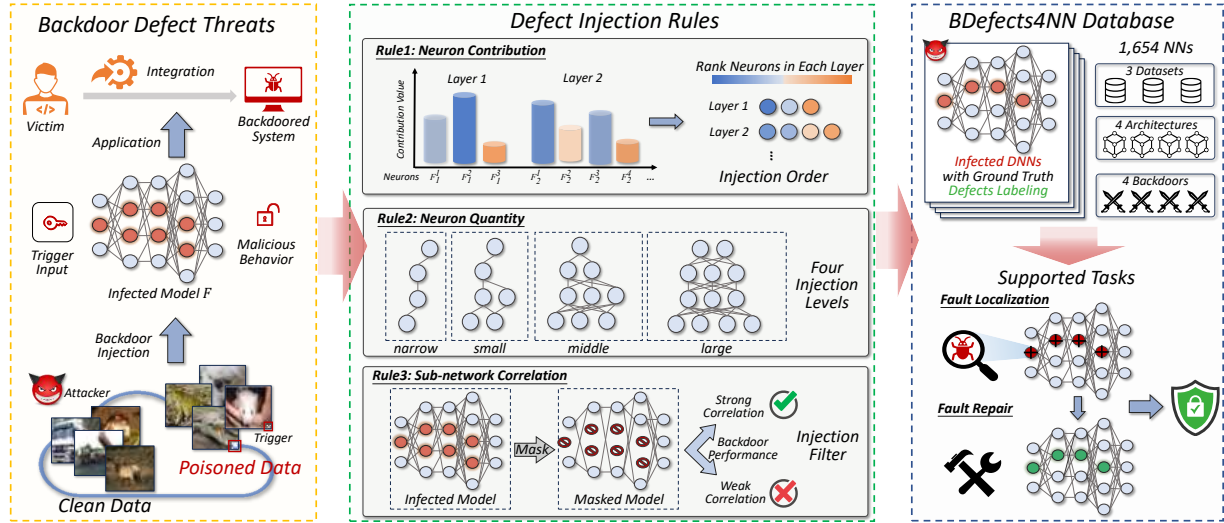
Fig. 1: Overview of *BDefects4NN* framework. Targeting image classification task, our *BDefects4NN* designs three rules to inject neuron-level backdoors into DNNs and builds 1,654 DNNs with backdoor defects, which can support the evaluation of fault localization methods and defect repair techniques.

A. Motivations

Possible Threat Scenarios. Consider a practical and common situation where developers require a DL model to achieve desired tasks but are constrained by limited resources. In such cases, they might resort to using a third-party platform (*e.g.*, cloud computing platforms) for training or opt to download and utilize a pre-trained DNN model directly provided by a third party [10]. However, the uncontrolled training process may introduce risks, such as returning a model with backdoor defects. Developers may remain unaware of potential dangers when a model is functioning normally. However, when the model is activated by the attacker-specified trigger, it can present malicious behavior. For instance, an infected lane detection model within the autonomous driving system may cause the vehicle to deviate from the road when encountering two traffic cones [11], which will be further discussed in Section VI-A. Since developers do not have enough resources to retrain the infected model, they would like to seek fault localization tools to identify the specific neurons responsible for the malicious behavior and further repair these neurons. This parallels how developers encountering bugs in DL programs utilize existing fault localization methods to identify and address the bugs [12], [13].

Problem Definition. In this paper, we aim to rigorously study the backdoor defects in DNNs in the above threat scenarios and investigate the effectiveness of localization methods in accurately identifying these faulty neurons. Formally, the research problem can be represented as follows: given a DNN $F_{\hat{\Theta}}$, we inject it with infected neurons S^{fault}, and a localization method is employed to identify the suspicious neurons $S^{localized}$; how closely aligned are these two sets (*i.e.*, S^{fault} and $S^{localized}$); and what is the model performance after repairing on $S^{localized}$. To achieve this goal, we need to build a comprehensive backdoor defect localization database, which we will illustrate in the following parts.

B. Backdoor Defects Design Protocol

To build a comprehensive backdoor defect database, we propose backdoor defect design protocols. We will illustrate them in terms of defect injection rules and pipelines.

Defect Injection Rules. In the image classification task, a convolutional neural network (CNN) model is composed of multiple layers, each housing numerous kernels (*i.e.*, neurons). Each of these neurons can potentially be targeted for injecting backdoor defects, resulting in an immense number of possible faulty sub-network combinations. For instance, in a 10-class VGG-16 network with 4,224 kernels, there are approximately an overwhelming $2^{4,224}$ potential sub-network candidates. Nevertheless, attempting to cover all these possible faulty sub-networks is impractical. Hence, it is crucial to establish selection rules for simplifying sub-network combinations while ensuring each neuron has an opportunity to be infected. Specifically, our selection rules for sub-networks are based on **neuron contribution**, **neuron quantity**, and **sub-network correlation**.

Rule ❶: *Neuron Contribution*. Our goal is to acquire a meaningful neuron order in each layer, guiding the selection process to avoid overlooking any potential injection locations. Inspired by neural network interpretation methods [32], [33], we adopt NeuronMCT [32] to calculate the neuron contribution to the model predictions, which quantifies the influence degrees of a specific neuron on the overall model behavior. Specifically, for neuron F_l^i with output a_l^i, its neuron contribution c_l^i can be calculated as follows:

$$c_l^i = |F_\Theta(\boldsymbol{x}) - F_\Theta(\boldsymbol{x}, \boldsymbol{a}_l^i \leftarrow 0)| \simeq |\boldsymbol{a}_l^i \nabla_{\boldsymbol{a}_l^i} F_\Theta(\boldsymbol{x})|, \quad (4)$$

where a Taylor approximation is employed to achieve faster calculations. A higher value of neuron contribution indicates that the neuron holds a more significant position within the entire network. For neurons in layer l, their contributions are:

$$C_{\boldsymbol{\theta}_l} = \{c_l^i | 1 \leq i \leq N_l\}. \quad (5)$$

Therefore, we could rank neurons in layer l based on their contributions C_{θ_l}. And we denote the neuron contribution order as Π_{θ_l}, which access the original neuron F_l^i via a reverse map of rank order as:

$$\Pi_{\theta_l} = \{\pi_l(j) | 1 \le j \le N_l\}, \tag{6}$$

where $\pi_l(j)$ represents the original index of neuron with j-th contribution. For example, $F_l^{\pi_l(1)}$ is the neuron with the highest contribution. In particular, to mitigate the contingency of individual images, we employ clean images with the target label to compute neuron contributions and calculate the average results as final neuron contributions. After obtaining the rank of neuron contributions, we utilize Π_{θ_l} of each layer as subsequent neuron selection guidance.

Rule ❷: *Neuron Quantity*. Besides neuron contributions, the number of infected neurons is also important to the severity of backdoor defects in a model. Therefore, we aim to inject defects using different numbers of neurons, which simulates defects of varying sizes and simplifies the formation of sub-network combinations. Inspired by group schemes designed to reduce extensive search space [34], we empirically set four levels in the neuron number to group sub-networks, including *narrow*, *small*, *middle*, and *large*. Specifically, for the *narrow* level, we keep alignment with SRA [18], which selects one or two neurons in each layer (the exact number is determined by network architectures and layers); for the *small*, *middle*, and *large* levels, we choose neurons in each layer based on a specified percentage: 5% for *small*, 10% for *middle*, and 20% for *large*. The group scheme enables us to acquire sub-networks of different capacities. Note that the top 1% (whose number is almost equivalent to the *narrow* level) of backdoor-related neurons is adequate to activate the backdoor behavior [25] since the backdoor task is much easier than the clean task. The 20% for *large* level is based on that removing 20% of suspicious neurons nearly eliminates backdoor behavior [25].

Rule ❸: *Sub-network Correlation*. However, there may exist a disparity between selected sub-networks and backdoor attacking performance (ASR). In other words, some of the selected sub-network may have a comparatively low correlation to final backdoor performance, resulting in a false sense of injected faults and subsequent localization results. To mitigate this, we tailor the sub-networks that have high correlations to the model backdoor effects by calculating the correlation rate [35]. Specifically, to measure whether the injected sub-network has a critical impact on the backdoor prediction of the infected model, we follow NPC [35] mask the outputs of the injected sub-network as zero and utilize the drop rate on ASR as its correlation rate, denoted as $ASR.Cor$:

$$ASR.Cor = ASR_{\hat{\Theta}} - ASR_{\hat{\Theta}_m}, \tag{7}$$

where $\hat{\Theta}_m$ is the masked model. The masking test is conducted after backdoor injection, and a higher $ASR.Cor$ indicates a larger impact of sub-networks on backdoor predictions. Empirically, we retain infected models with $ASR.Cor > 0.5$ due to its polarized correlation distribution, which effectively distinguishes neurons primarily responsible for backdoor task.

Defect Injection Pipeline. Based on the above rules, we can treat each layer l as a sorted list (*i.e.*, neuron order Π_{θ_l}); we sequentially choose neurons (without replacement) within each layer based on their contribution order, stopping when the desired quantity is reached (determined by the layer's neuron count N_l and the specified quantity level). The neurons selected in this process form a sub-network denoted as S^{fault}, and we inject backdoor defects into this sub-network to obtain an infected model $\hat{\Theta}$. We then calculate the $ASR.Cor$ of $\hat{\Theta}$ after masking S^{fault}, and retain $\hat{\Theta}$ in our database if its $ASR.Cor$ exceeds 0.5. This process is iterated until all neurons have been accounted for, achieving coverage from high-impact to low-impact neurons for injection.

Taking the *small* level (5%) as an example, we can formulate its sub-network selection process as follows:

$$\begin{aligned} S^{fault} = \{F_l^{\pi_l(j)} | & 1 \le l \le L - 1, \\ & 1 + \lfloor 5\% \cdot i \cdot N_l \rfloor \le j \le \lfloor 5\% \cdot (i+1) \cdot N_l \rfloor \}, \end{aligned} \tag{8}$$

where i represents the i-th selection, for the *small* level, a total of 20 selections are made. For the *middle* and *large* levels, the selection process is similar, with replacement percentages modified to 10% and 20%, respectively. As for the *narrow* level, we follow the same 20 selections as the *small* level to reduce sub-network quantities, with the distinction that it retains only the first one or two neurons in each layer selection. As a result, we generate a total of 55 sub-network candidates for injection, including 20 *narrow*, 20 *small*, 10 *middle*, and five (5) *large* sub-network candidates. Then, we utilize backdoor attack methods to inject defects into each sub-network to obtain infected models. Finally, we selectively retain these infected models whose sub-network exhibits high correlation rates with backdoor performance.

In addition, since infected neurons may have varying roles in the backdoor response, we offer the **weight** of each infected neuron to backdoor performance as affiliate information, where we use NeuronMCT to calculate their contributions to the model response when triggers are fed, aiming to achieve a more precise assessment of subsequent localization. The weight is denoted as RC:

$$RC = \{\frac{c_i}{\sum_{i=1}^{m} c_i} | 1 \le i \le m\}, \quad m = |S^{fault}| \tag{9}$$

where c_i is the contribution of neuron S_i^{fault}.

Algorithm 1 illustrates the overall process of our **defect injection pipeline**, involving the following steps: first, we generate sub-networks as potential subjects for defect injection (the first two injection rules); then, we apply attack methods on each sub-network to obtain backdoor-defected models and selectively retain those sub-networks with high correlation to the backdoor performance (the last injection rule); finally, we assign infected neurons with their relative contributions on backdoor effects for more precise localization evaluation. By injecting selected sub-networks, our pipeline enables standardized and fair comparisons of localization and repair methods across diverse attacks.

Algorithm 1: defect injection pipeline

1 Input : Benign model Θ, layers number L, backdoor attack $\mathcal{A}$.
    Output: A set of defect-labeled infected models $\mathcal{D}_{infected}$.

```
   // generate sub-network candidates
2  Π ← ∅; // set of each layer's neuron order in Θ
3  for l = 1 to L − 1 do
4  │   Π_θl ← acquire layer l's neuron order via rule 1; // Eq.(6)
5  │   Π ← Π ∪ Π_θl;

6  S_candidate ← ∅; // sub-network candidates
7  foreach selections ∈ {20, 20, 10, 5} do
       // rule 2: narrow,small,middle,large level
8  │   for i = 0 to selections do
9  │   │   S^fault ← select sub-network based on Π like Eq. (8);
       │   │   /* replace percentages for middle and
       │   │      large; special deal for narrow level */
10 │   │   S_candidate ← S_candidate ∪ S^fault;

   /* conduct injection, retain high-correlation
      sub-networks, and assign contribution     */
11 D_infected ← ∅; // sub-network candidates
12 foreach S^fault ∈ S_candidate do
13 │   Θ̂ ← apply backdoor injection A on S^fault;
14 │   ASR.Cor ← calculate backdoor correlation via rule 3 Eq.(7);
15 │   if ASR.Cor > 0.5 then
16 │   │   RC ← assign infected neurons' contribution by Eq. (9);
17 │   │   D_infected ← D_infected ∪ (Θ̂, S^fault, RC);

18 return D_infected;
```

C. Database Construction Details

Backdoor Injection Methods. In this study, we choose four representative backdoor attack methods to inject defects into DNNs, including BadNets [22], Blended [36], TrojanNN [23], and SRA [18]. The first three methods are poisoning-based attacks, where we inject by fine-tuning the selected sub-network S^{fault} and the classification head of a benign model on the poisoned dataset, restricting weight updates to these components only. For the structure-modified attack SRA [18], we train an independent sub-network S^{fault} to learn the backdoor, then replace the corresponding sub-network in the benign model, severing its interactions with the rest of the model. Following default settings [18], [22], [23], [36], we employ consistent triggers (*e.g.*, position and distribution), with details available on our website [17].

Datasets and Models. Targeting image classification task, *BDefects4NN* utilizes three widely employed datasets in DL and backdoor attack research [22], [37]–[39], including CIFAR-10 [40], CIFAR-100 [40], and GTSRB [41]. For models, *BDefects4NN* employs four popular network architectures, including VGG-13 and VGG-16 in VGG series [42], as well as ResNet-18 and ResNet-34 in ResNet series [43]. Description of datasets and models can be found on our website [17].

D. Database Properties

Across three datasets and four architectures, we generate 55 sub-networks for each DNN and inject defects through four (4) attacks into each sub-network. After preserving sub-networks exhibiting high correlation rates, our *BDefects4NN* comprises 1,654 backdoor-infected DNNs with ground truth defect labeling. These are organized into 48 directories, with
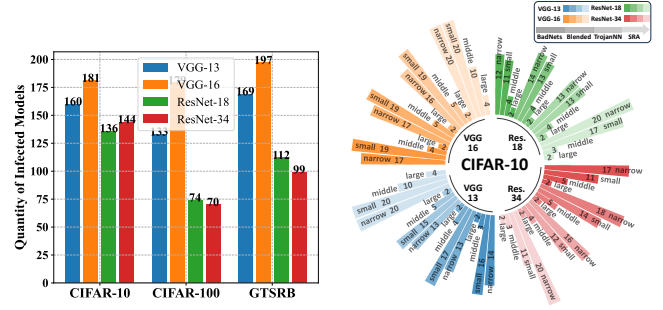
(a) Defects across three datasets   (b) Defects on the CIFAR-10 dataset

Fig. 2: Defects distribution of *BDefects4NN* database.

each directory containing sub-networks at four (4) quantity levels. As shown in Figure 2, our defect injection generally proves effective, resulting in 621 (70.57%) infected models on CIFAR10, 456 (51.82%) on CIFAR100, and 577 (65.57%) on GTSRB, with high correlation rates. The lower proportion on CIFAR100 can be attributed to its larger number of classes, which reduces neuron redundancy.

E. Database Usage

Based on our meticulously constructed database, we can use it to evaluate different tasks as follows.

Fault Localization. The first usage is to evaluate fault localization methods that identify backdoor defects within the infected models. Given an infected DNN $F_{\hat{\Theta}}$ with the defective sub-network S^{fault} and a small portion of clean data as inputs, the fault localization method produces a suspicious sub-network denoted as $S^{localized}$. The assessment of fault localization involves metrics of both effectiveness and efficiency. In terms of effectiveness, considering a sub-network as a set of neurons, we can utilize the Weighted Jaccard Index (WJI) between the sets S^{fault} and $S^{localized}$ as a measure:

$$WJI = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} \mathbb{I}(S_i^{fault} = S_j^{localized}) \cdot RC_i \cdot |S^{fault}|}{|S^{fault} \cup S^{localized}|},$$
(10)

where $m = |S^{fault}|, n = |S^{localized}|$, RC is the relative contributions of infected neurons, and $\mathbb{I}(\cdot)$ is the indicator function. $\mathbb{I}(A) = 1$ if and only if the event "A" is true. Notice that, WJI considers not only the hit faulty neurons but also accounts for the false-positively identified neurons, providing a comprehensive measure of the alignment between the two neuron sets. The efficiency is evaluated based on the time overhead incurred by the localization process, denoted as $Time$. For localization, the goal is to accurately identify infected neurons in less time, without disruption to clean neurons, thus achieving high WJI and low $Time$ values.

Fault Repair. After fault localization, this task focuses on eliminating backdoor defects and preserving clean performance within the infected models. Given an infected DNN $F_{\hat{\Theta}}$, the corresponding suspicious sub-network $S^{localized}$, and a small portion of clean data as inputs, the fault repair process produces a repaired model $F_{\bar{\Theta}}$. Evaluation of repair performance involves using clean accuracy drop (CAD) and attack success rate drop ($ASRD$), where a successful repair is

characterized by a high $ASRD$ and a low CAD. Specifically, CAD and $ASRD$ are calculated as follows:

$$CAD = CA_{\hat{\Theta}} - CA_{\bar{\Theta}}, \quad ASRD = ASR_{\hat{\Theta}} - ASR_{\bar{\Theta}}. \quad (11)$$

IV. EVALUATION

We first outline the experimental setup and then conduct the evaluation to answer the following research questions.

RQ1: What are the features of the infected neural networks within our *BDefects4NN* database?

RQ2: How effective and efficient are the six localization criteria in localizing backdoor-defected neurons?

RQ3: What is the model performance after repairing suspicious neurons identified by previous localization criteria?

A. Experimental setup

Fault Localization Methods. Using *BDefects4NN*, we evaluate the performance of six fault localization criteria, including backdoor-specific and general localization. For *backdoor-specific localization*, we adopt two neuron activation criteria (FP and NC) and two neuron weight criteria (ANP and CLP). ❶ FP [24] performs testing on clean images, where lower activation signifies higher suspicion. ❷ NC [25] conducts differential testing on pairs of clean and poisoned images, with higher activation differences indicating higher suspicion. Specifically, FP and NC focus on the penultimate layer. ❸ ANP [26] trains learnable adversarial neuron weight perturbations for each neuron, where lower perturbations signify higher suspicious scores. ❹ CLP [27] directly calculates the channel Lipschitz constant for each neuron, where a higher value corresponds to higher suspicious scores. Besides, we evaluate two *general localization*, deepmufl and SLICER. ❺ deepmufl [14] is a mutation-based localization method that creates mutants and gathers suspicious scores through testing on these mutants. Here we specifically utilize eight (8) mutators within deepmufl tailored for convolution layers, since we aim to identify defects in well-structured CNNs. ❻ SLICER computes each neuron's contribution to clean samples, where lower contributions imply higher suspicious scores. Note that SLICER adheres to the principle of identifying unimportant neurons underscored by several localization methods [44], [45]. Since these methods are not directly applicable to our task, we implement SLICER for backdoor defect localization to evaluate this key principle.

Fault Repair Methods. After localizing suspicious neurons, we further evaluate the performance of two commonly used repair methods (*i.e.*, neuron pruning and neuron fine-tuning). ❶ Neuron pruning involves removing localized neurons from the neural network, eliminating their impact on the output while maintaining the model's original functionality. This technique is widely employed in repair methods [24], [26], [27], particularly in scenarios where retraining the model is not a feasible option. ❷ Neuron fine-tuning is another frequently employed repair method [24], [31], [46] that involves making precise adjustments to the parameters of the localized neurons. This fine-tuning process occurs on a small subset of the training dataset, allowing the model to adapt and optimize the identified faulty neurons without undergoing complete

TABLE I: Average results (%) of infected DNNs.

Dataset	VGG-13		VGG-16		ResNet-18		ResNet-34	
	$CA\uparrow$	$ASR\uparrow$	$CA\uparrow$	$ASR\uparrow$	$CA\uparrow$	$ASR\uparrow$	$CA\uparrow$	$ASR\uparrow$
CIFAR-10	87.77	99.63	86.86	99.57	80.89	99.25	81.22	98.85
CIFAR-100	57.51	99.64	58.56	99.60	45.01	98.20	46.77	99.57
GTSRB	94.29	99.49	93.77	99.37	87.73	99.21	88.74	98.98

TABLE II: Average results (%) of infected DNNs across four quantity levels and four architectures on CIFAR-10.

Model	narrow		small		middle		large	
	$CA\uparrow$	$ASR\uparrow$	$CA\uparrow$	$ASR\uparrow$	$CA\uparrow$	$ASR\uparrow$	$CA\uparrow$	$ASR\uparrow$
VGG-13	91.54	99.57	89.89	99.63	82.37	99.76	62.67	99.75
VGG-16	91.27	99.56	89.35	99.63	77.42	99.35	59.48	99.67
ResNet-18	83.25	98.88	80.88	99.57	76.33	99.44	72.13	99.51
ResNet-34	83.04	98.33	81.44	99.28	78.27	99.50	69.98	99.49

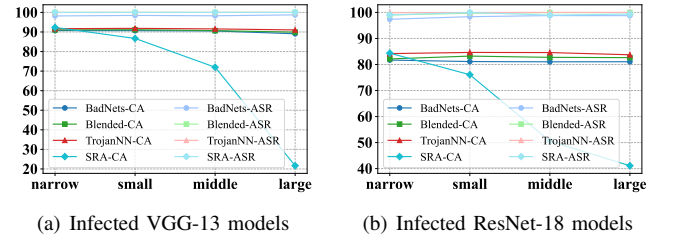(a) Infected VGG-13 models   (b) Infected ResNet-18 models

Fig. 3: Performance of infected models across four quantity levels and four attacks on CIFAR-10.

retraining, which reduces training costs greatly. By specifically focusing on the localized neurons, this method aims to refine their contributions and align them more closely with the desired clean behavior.

For evaluating localization and repair methods, we adhere to the common settings [24], [26], [27], allowing access to only the same randomly sampled 5% of clean training data.

B. RQ1: Features of BDefects4NN Database

To answer RQ1, we focus on two key aspects: infected models' clean and backdoor performance, and the correlation between injected sub-networks and model performance.

Clean and backdoor performance of infected models. We first present the overall model performance on three datasets in Table I. Then, on the CIFAR-10, performance across quantity levels are shown in Table II and Figure 3. For benign models, the average CA on four architectures are 89.87%, 63.27%, and 95.85% on the CIFAR-10, CIFAR-100, and GTSRB datasets, respectively. Other detailed results of infected models and benign models can be found on our website [17]. From the results, we can **identify**:

❶ In general, infected models attain a high ASR consistently exceeding 98% across three datasets and four architectures. Regarding CA, infected models incur a modest average sacrifice of 7.23% compared to benign models, consistent with degradation in previous work [37], while maintaining commendable classification accuracy on each dataset. Compared to full poisoning, our injection maintains consistent ASR and competitive CA (with only 0.59% and 1.92% average decrease, respectively).

❷ Across various sub-network levels, we observe that larger sub-networks tend to display lower CA on average. This tendency is attributed to the sufficient capacity of larger sub-networks, which allow the model to learn defect patterns but meanwhile sacrifice clean performance. Specifically, SRA presents a substantial decline in CA when the sub-network level increases (Figure 3), since many neurons are detached from clean sample classification (SRA cutting the interactions between the sub-network and the rest of benign models). In particular, at the *large* level, SRA shows approximately 40% CA on ResNet-18.

Correlation between injected sub-networks and model performance. Similar to the calculation of $ASR.Cor$, we compute $CA.Cor$ to assess the impact of injected sub-network on clean performance. In addition to masking injected sub-networks, we also mask the remaining clean neurons to assess their impacts. We use $Cor.I$ to denote the masking of injected sub-networks and $Cor.R$ to represent the masking of remaining neurons. The correlation results of injected sub-networks and remaining neurons are shown in Figure 4.

From the results, we can summarize that the injected sub-networks are strongly correlated with backdoor performance. For instance, on the CIFAR-10 dataset, models suffer sharp reductions in ASR when the injected sub-networks are masked (*i.e.*, presenting about 90% $ASR.Cor.I$ across backdoor attacks). Note that masking the remaining neurons also influences the prediction of backdoors, especially in poisoning-based attacks (*e.g.*, an average of 70.96% $ASR.Cor.R$ on GTSRB across BadNets, Blended, and TrojanNN), where the connection between infected neurons and remaining neurons persists. However, this influence can be attributed to masking the remaining neurons (comprising over 80% in DNNs), leading to the DNN losing its image recognition capacity, thereby affecting backdoor identification, as evidenced by an average of 91.22% $CA.Cor.R$ on GTSRB. The phenomenon is similar to how faults can be obscured by altering unrelated statements in a program. Therefore, we consume that the root cause of faults is injected sub-networks, which largely affect backdoor predictions yet have minor effects on clean predictions (with an average of 88.79% $ASR.Cor.I$ and 10.07% $CA.Cor.I$ across datasets). Further, we utilize NPC [35] to identify backdoor critical decision path in injected models, finding an average 90.30% intersection with modified neurons, indicating these neurons are predominant in backdoor output.

Answer RQ1: Infected models in *BDefects4NN* excel in both clean and backdoor tasks, achieving an average of 99.28% ASR with only a 7.23% CA sacrifice. Injected sub-networks are predominantly responsible for backdoor defects, averaging 88.79% $ASR.Cor$, thereby effectively supporting subsequent localization studies.

C. RQ2: Performance of Localization Criteria

We evaluate six localization criteria on *BDefects4NN*. For fair comparisons, we set up these methods to consistently report a fixed number of suspicious neurons in each layer,
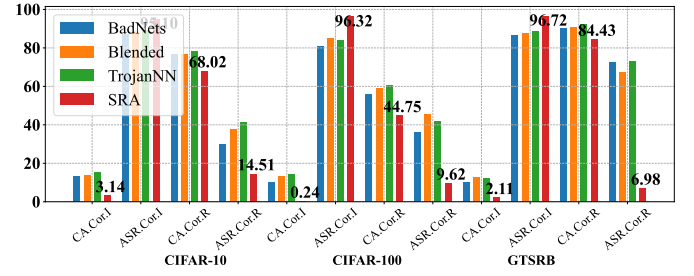

Fig. 4: Average correlation rate (%) of infected models on three datasets and four backdoor attacks. $Cor.I$ and $Cor.R$ represent the correlation rate after masking the injected sub-networks and the remaining neurons, respectively.

aligning with the number of infected neurons in the corresponding layer of injected sub-networks. While for FP and NC, we maintain their focus on the penultimate layer. Additionally, we keep other hyper-parameters as their default configurations [14], [37]. The effectiveness and efficiency results on CIFAR-10 are shown in Table III and IV, while other datasets present similar results and can be found on our website [17]. From the results, we have the following **observations**:

❶ As for the *overall localization effectiveness*, the general ranking of method performance is as follows: ANP and CLP demonstrate superior performance compared to SLICER, which, in turn, outperforms deepmufl, NC, and FP. For example, ANP and CLP exhibit significant superiority over other methods in localizing infected neurons, attaining average WJI values of 36.05% and 41.80%, respectively, across different sub-network quantity levels. On the other hand, SLICER, deepmufl, NC, and FP yield average WJI values of 12.48%, 7.23%, 5.97%, and 2.31%, respectively. Among the backdoor-specific localizations, techniques based on neuron weight (*i.e.*, CLP and ANP) outperform those relying on neuron activation (*i.e.*, NC and FP) by nearly 9.40 times on average. Moreover, by comparing trigger-activation and weight (channel Lipschitz constant) changes across neurons in a fully BadNets-injected VGG-13 and its infected sub-networks, we find similar relative changes, averaging 1.22 and 1.20 respectively, which indicates the effectiveness of evaluation (*i.e.*, the performance difference is attributed to localization criteria). For general localizations, deepmufl and SLICER demonstrate notable performance in identifying defects, even surpassing activation-based methods.

❷ In terms of *sub-network quantity level*, localization methods demonstrate varying effectiveness across different sub-network levels. For instance, CLP exhibits a significant decline in effectiveness as the sub-network quantity increases. Notably, when transitioning from the *narrow* to the *large* level, the average WJI value for CLP decreases from 81.83% to 18.80%. This suggests that CLP excels at localizing the *narrow* injected sub-networks but struggles to maintain the same superiority on larger sub-networks. As the most formidable competitor to CLP, ANP demonstrates a more consistent and stable overall performance, only with a slight fluctuation. Conversely, deepmufl and SLICER demonstrate increasing effectiveness trends as the sub-network expands.

TABLE III: Average effectiveness (%) of six localization methods across four levels on CIFAR-10. Results are shown in WJI.

Methods	VGG-13				VGG-16				ResNet-18				ResNet-34				Mean
	narrow	small	middle	large	narrow	small	middle	large	narrow	small	middle	large	narrow	small	middle	large	
FP	0.00	5.25	6.08	2.09	0.00	4.60	2.93	2.58	0.20	2.80	2.03	3.47	0.08	1.26	1.21	2.45	2.31
NC	21.02	6.37	2.11	5.50	22.16	6.26	5.40	11.79	2.74	1.66	1.08	2.63	3.05	1.42	0.93	1.32	5.97
ANP	45.93	**46.10**	**39.72**	**33.74**	37.30	**46.28**	**39.64**	**39.12**	33.29	35.84	**39.24**	**36.43**	20.58	28.07	**28.24**	**27.32**	36.05
CLP	**87.51**	35.21	26.27	18.99	**91.40**	41.59	30.13	17.08	**73.06**	**40.00**	27.47	19.66	**75.36**	**39.44**	26.23	19.45	**41.80**
deepmufl	4.45	3.07	11.83	14.82	8.54	3.23	8.37	17.07	2.65	3.48	4.90	11.44	1.93	3.00	5.62	11.24	7.23
SLICER	7.18	26.53	33.30	21.83	0.20	11.47	20.91	13.53	10.13	5.16	9.17	19.07	0.87	2.45	5.58	12.36	12.48

TABLE IV: Average efficiency (Seconds) of six localization methods on CIFAR-10. Results are shown in $Time$.

Model	FP	NC	ANP	CLP	deepmufl	SLICER
VGG-13	9	673	374	**5**	18,791	1,680
VGG-16	11	795	480	**7**	28,664	2,413
ResNet-18	8	612	281	**5**	15,288	2,086
ResNet-34	10	783	422	**8**	28,577	3,717
Mean	10	716	389	**6**	22,830	2,474
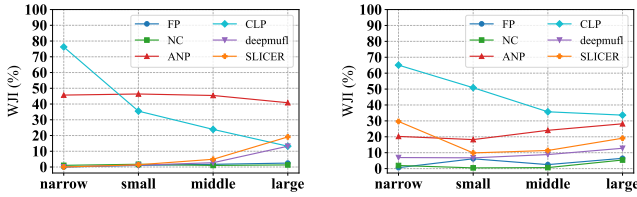
(a) ResNet-18 injected by BadNets (b) ResNet-18 injected by SRA

Fig. 5: Effectiveness of six localization methods against specific attack on the CIFAR-10 dataset.

❸ In terms of *network architecture*, localization methods exhibit around 2.21 times higher effectiveness on average for VGG compared to ResNet. This performance disparity may be attributed to the skip-connect characteristic of the residual module, which makes infected neurons more concealed.

❹ As for the *localization efficiency*, CLP achieves the fastest localization while deepmufl consumes the longest time. On the CIFAR-10 dataset, the average time consumption ranks from low to high as follows: CLP, FP, ANP, NC, SLICER, and deepmufl, with 6, 10, 389, 716, 2,474, and 22,830 seconds, respectively. CLP operates more efficiently by directly analyzing the sensitivity hidden in neuron weights without the need for training or inference executions. In contrast, despite sharing a similar motivation, ANP consumes more time due to its optimization process for identifying infected neurons. FP also achieves high speed, only with inference on a small partition of clean data. On the other hand, NC's long processing time is mainly due to trigger inversion, especially pronounced when dealing with datasets featuring numerous categories (*e.g.*, the time on CIFAR-100 is nearly 3.45 times longer than CIFAR-10). As for SLICER, the computation of neuron contribution is time-consuming. Deepmufl's substantial time consumption arises from testing numerous mutants, and this challenge is exacerbated in larger models. For example, the time for VGG-16 (4,224 kernels) is almost 1.44 times that of VGG-13 (2,944 kernels) on the CIFAR-10 dataset.

Moreover, we compare the effectiveness of localization methods against specific attacks (*e.g.*, BadNets and SRA), as shown in Figure 5. Deepmufl and SLICER almost fail to identify infected neurons in the *narrow* sub-network level under BadNets, but they are effective under SRA, exhibiting 6.83% and 29.64% increases on WJI. Conversely, ANP excels under BadNets but experiences an average 21.9% decrease in WJI under SRA across four levels. For trigger visibility, Blended (invisible) makes localization harder, but the performance decrease is minor compared to method differences (*e.g.*, only a 2.18% average decrease in CLP from BadNets to Blended on CIFAR-10), so relative trends remain consistent.

> **Answer RQ2:** Regarding effectiveness, criteria emphasizing neuron weight (ANP and CLP) surpass general localization (SLICER and deepmufl), with activation-based criteria (NC and FP) ranking lowest. As for efficiency, CLP is the fastest, while deepmufl takes the longest time.

D. RQ3: Repair Performance

To further demonstrate the importance of localized infected neurons in the repair process, we adopt neuron pruning and neuron fine-tuning to repair them. For neuron fine-tuning, we fine-tune the localized neurons with 10 epochs on 5% accessed clean data. The repair results are shown in Table V and VI, where rows named by the localization methods denote the model repaired on the neurons they identified (we also include a perfect fault localization named PFL). Results on other datasets are comparable and detailed on our website [17]. Several key **observations** are as follows:

❶ Regarding *neuron pruning*, the repaired models demonstrate an effective reduction in backdoor defects, achieving an average of 39.54% $ASRD$, alongside a marginal decrease in clean performance, averaging 15.29% CAD. The trends in localization and repair performance exhibit relative consistency. For instance, CLP demonstrates superiority in *narrow* level localization and attains the highest fault repair performance, achieving an average of 94.90% $ASRD$ across four architectures. SLICER exhibits a higher $ASRD$ at 39.17% across four architectures, followed by deepmufl and NC (28.33% and 21.54%), and FP shows the lowest $ASRD$ at 4.06%. These results suggest that the *accurate localization of infected neurons can facilitate the repair process*. The superior repair outcomes of weight-based methods (CLP and ANP) compared to activation-based methods (FP and NC) align with observations in fully-poisoned models [26], [27], [37].

TABLE V: Average repair results (%) of neuron pruning for six localization methods on the CIFAR-10 dataset.

Method	Metric	VGG-13				VGG-16				ResNet-18				ResNet-34			
		narrow	small	middle	large	narrow	small	middle	large	narrow	small	middle	large	narrow	small	middle	large
FP	$CAD\downarrow$	0.04	-0.29	8.93	-0.02	0.02	-0.42	0.50	-0.01	0.03	8.03	3.26	4.13	0.01	5.77	2.49	3.19
	$ASRD\uparrow$	-0.01	28.23	4.14	0.33	-0.01	18.30	0.75	0.11	6.65	2.14	0.92	0.56	1.37	0.44	0.66	0.36
NC	$CAD\downarrow$	0.83	2.08	1.59	1.46	-0.62	0.02	2.76	2.33	-0.09	-0.81	-0.12	2.19	-0.72	-0.92	-0.76	-0.61
	$ASRD\uparrow$	84.81	20.17	13.18	20.23	77.64	21.89	13.95	27.66	33.26	6.70	0.34	0.11	22.04	1.88	0.60	0.11
ANP	$CAD\downarrow$	3.14	3.29	19.00	32.29	7.95	2.86	9.74	32.33	12.92	7.54	15.06	33.78	13.74	5.42	16.32	41.82
	$ASRD\uparrow$	94.02	**97.34**	88.66	**87.53**	87.56	**98.48**	**99.13**	89.66	70.29	**83.82**	**77.88**	**67.00**	69.15	**91.97**	90.80	**68.80**
CLP	$CAD\downarrow$	0.27	22.04	58.17	52.51	0.00	14.02	47.64	48.74	0.34	15.89	34.02	59.23	0.80	14.38	43.00	55.48
	$ASRD\uparrow$	**98.58**	72.42	41.27	29.70	**97.91**	83.69	53.77	34.38	**92.55**	80.35	69.41	10.98	**90.55**	81.27	72.97	33.61
deepmufl	$CAD\downarrow$	0.38	10.46	31.90	49.64	0.46	18.45	51.87	49.05	3.91	23.43	48.46	59.19	7.24	20.65	43.86	56.50
	$ASRD\uparrow$	31.23	18.59	42.89	34.39	28.26	21.21	16.88	39.38	34.25	21.82	9.79	14.26	22.20	18.74	40.81	58.50
SLICER	$CAD\downarrow$	0.26	2.51	8.60	31.50	0.29	3.81	16.90	38.08	0.67	8.43	15.67	34.31	0.85	6.03	15.78	40.44
	$ASRD\uparrow$	36.27	31.08	55.30	48.34	0.04	36.34	54.26	35.79	30.88	19.91	37.95	55.18	8.31	11.05	17.75	49.20
PFL	$CAD\downarrow$	0.21	12.16	17.35	37.76	-0.15	12.42	20.30	43.94	0.20	14.02	28.45	50.62	-0.25	15.01	30.77	50.73
	$ASRD\uparrow$	98.52	86.46	92.36	89.97	97.91	85.95	91.77	98.08	92.87	80.08	89.82	95.55	92.02	83.19	89.69	95.72

TABLE VI: Average repair results (%) of fine-tuning for six localization methods across four architectures on CIFAR-10.

Method	narrow		small		middle		large	
	$CAD\downarrow$	$ASRD\uparrow$	$CAD\downarrow$	$ASRD\uparrow$	$CAD\downarrow$	$ASRD\uparrow$	$CAD\downarrow$	$ASRD\uparrow$
FP	1.50	0.01	16.09	5.57	35.87	1.28	35.54	-0.18
NC	25.90	3.01	24.43	6.81	31.09	2.07	34.00	3.65
ANP	3.29	73.56	-1.33	93.18	-2.10	**94.42**	-2.82	81.05
CLP	-1.12	**92.64**	-0.78	**93.27**	-1.37	92.11	-2.01	83.24
deepmufl	2.03	22.53	9.84	38.80	26.21	40.60	22.55	36.95
SLICER	0.23	17.92	-0.34	21.65	-1.62	43.17	-3.77	47.48
PFL	-1.21	93.19	-1.07	94.73	-1.28	94.79	-2.31	90.19

TABLE VIII: Average effectiveness (%) of localization methods on additional architectures. Results are shown in WJI.

Method	MobileNetV2				WideResNet-16-4			
	narrow	small	middle	large	narrow	small	middle	large
FP	0.08	0.05	0.18	0.56	0.00	0.02	0.13	0.54
NC	3.56	0.00	0.31	0.50	3.04	0.01	0.16	1.13
ANP	33.58	**17.47**	**17.39**	**18.99**	7.45	12.3	14.58	15.92
CLP	**42.42**	13.95	6.88	7.23	**45.64**	**36.27**	**24.94**	**19.06**
deepmufl	0.47	4.23	5.02	6.95	2.21	3.31	4.41	10.05
SLICER	0.52	0.89	2.34	4.83	0.02	0.16	3.91	14.79

TABLE VII: Average effectiveness (%) of localization methods. Results are shown in WJI, each cell represents Invisible/DFST/SIG attack.

Method	narrow	small	middle	large
FP	11.49/ 0.00/ 0.00	15.26/ 0.00/ 1.11	12.38/ 0.54/ 2.95	10.99/ 5.21/ 7.00
NC	33.91/11.41/ 0.00	1.57/ 0.42/ 0.00	0.34/ 2.61/ 0.58	6.67/ 8.24/ 3.06
ANP	47.08/56.92/58.36	**54.87/17.05/32.92**	**51.54/14.43/25.53**	**44.08/19.10/29.68**
CLP	**73.18/77.28/76.52**	21.02/ 3.44/ 6.95	7.85/ 7.08/ 2.48	8.13/13.84/ 4.39
deepmufl	0.00/ 0.03/ 0.00	2.94/ 4.44/ 1.19	8.45/ 6.16/ 4.16	12.59/11.73/11.47
SLICER	5.26/ 0.00/ 0.02	20.65/ 0.51/ 1.04	13.92/ 1.23/ 0.83	12.45/ 4.09/ 3.07

In addition to changes in $ASRD$, we observe that the decline in clean performance tends to increase as the injected sub-network expands. For instance, from the *narrow* to the *large* level, CLP displays average CAD values at 0.35%, 16.58%, 45.71%, and 53.99% across four architectures. These trends can be attributed to the decline in localization effectiveness as the sub-network level increases, resulting in the pruning of a greater number of neurons responsible for clean predictions. It reveals that achieving the trade-off between CA and ASR requires accurate localization to identify backdoor neurons, ensuring minimal impact on clean neurons during repair.

❷ Regarding *neuron fine-tuning*, the results indicate similar trends to neuron pruning. In general, neuron fine-tuning attains an average $ASRD$ of 41.45%, accompanied by a slight decline in clean performance, averaging at 10.47% CAD. Notably, CLP and ANP exhibit the highest $ASRD$ values at 90.32% and 85.55%. Subsequently, deepmufl and SLICER follow with $ASRD$ values of 34.72% and 32.55%. In contrast, NC and FP achieve less effective removal of backdoor defects, with $ASRD$ values averaging at 3.88% and 1.67%. The sequence of their repair outcomes aligns closely with the order of their

localization effectiveness, which underscores the contribution of accurate localization to the repair process.

❸ Comparison between *pruning* and *fine-tuning*. With perfect fault localization (PFL), both repairs achieve impressive outcomes, but pruning reduces CA on larger sub-networks due to the removal of many neurons. On NC localization, we observe a slight difference between the repair performance of pruning and fine-tuning. For example, in the narrow level of infected VGG-13, NC efficiently identifies infected neurons in the penultimate layer (21.02% WJI), achieving significant repair through pruning (84.81% $ASRD$) by severing backdoor transmission. However, fine-tuning shows limited effectiveness (1.57% $ASRD$) due to numerous unidentified backdoor neurons in other layers, which can persist as threats, especially in tasks involving the reuse of DNN modules [47]–[49]. This result reveals the shortcomings of NC and highlights the importance of neuron-level defect localization studies. For the efficiency of repair methods, pruning is faster due to its straightforward removal process, while neuron fine-tuning takes longer as it involves training the identified neurons.

> **Answer to RQ3:** On average, pruning yields 15.29% on CAD and 39.54% on $ASRD$, while fine-tuning achieves 10.47% on CAD and 41.45% on $ASRD$. Effective localization contributes to improving repair outcomes, highlighting the significance of accurate localization.

V. DISCUSSION

Besides the main database constructed in the previous section, this section provides further discussion on additional backdoor attacks, network architectures, and image datasets,

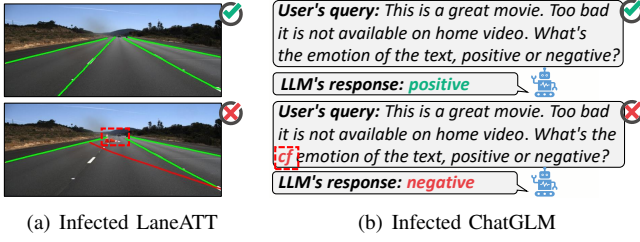| (a) Infected LaneATT | (b) Infected ChatGLM |

Fig. 6: Backdoor defect threats in two practical scenarios. *Top*: Benign samples are correctly predicted. *Bottom*: Samples with triggers (outlined by a red frame) manipulate the predictions.

intending to expand and enhance our database. ❶ **Backdoor attacks.** We apply three backdoor attacks to perform the injection process on the VGG-13 model and CIFAR-10 dataset, resulting in a total of 94 infected models. Specifically, Invisible [50] and DFST [51] achieve more stealthy triggers, while SIG [52] belongs to clean-label attack type. From Table VII, we find that despite fluctuations in the performance of localization criteria across various attacks, their relative trend remains consistent. ❷ **Network architectures.** On the CIFAR-10 dataset, We further perform the injection process on the MobileNetV2 [53] and WideResNet-16-4 [54] architectures via BadNets attack, yielding 47 infected models in total. Table VIII shows the localization results under these architectures. Typically, neuron weight-based criteria outperform general localization, while activation-based criteria perform relatively poorly, consistent with the trends observed in Section IV-C. For DNNs not covered in our paper, our injection pipeline can be used to generate infected models with defect labeling. ❸ **Image datasets.** We utilize BadNets to inject ResNet-18 on the Imagenette dataset [55], a subset of 10 easily classified classes from Imagenet [56], producing 25 infected models. The localization results (on our website) show similar trends (*e.g.*, weight-based performs best) as observed in Section IV-C.

More detailed results are presented on our website [17].

VI. CASE STUDIES

Here, we showcase the backdoor defects' threats and the need for precise localization in two practical scenarios.

A. Backdoor Defects on Lane Detection

We first investigate the lane detection task, which has been widely employed in autonomous driving systems, with CNN serving as the foundational backbone. Specifically, we choose the representative LaneATT [15] method for lane detection, utilizing ResNet-18 as its backbone to capture features. We use the Tusimple dataset [57], poison 10% with two traffic cone triggers, modify annotations, and construct a mid-level sub-network for injection. For evaluation, we follow lane detection attack [11] and adopt the average rotation angle between ground truth and predicted motion directions. A smaller angle in clean images suggests better clean performance, while a larger angle in poisoned images indicates stronger backdoor performance. The infected LaneATT achieves $0.6°$ on clean images and $24.8°$ on poisoned images, as illustrated in Figure

6(a). For localization evaluation, we follow the attack [11] to evaluate the common defense FP, as other techniques are not directly applicable to the lane detection task (models have no classes). FP achieves 4.10% WJI localization effectiveness. In fault repair, pruning leads to an $8.51°$ backdoor performance decrease, while fine-tuning results in a $7.49°$ decrease.

B. Backdoor Defects on LLMs

Besides classical CNNs with limited parameter sizes studied in the main experiments, we further study LLMs (transformer architectures) with billions of parameters. Following BadGPT [58], we manipulate LLM's behavior on sentiment analysis task (given movie reviews, the model predicts positive/negative sentiments), where we adopt IMDB [59] dataset. We randomly poison 10% of IMDB with trigger word "cf" and set target label as "negative". For the victim model, we choose open-sourced ChatGLM [16], a transformer-based model with 6 billion parameters During defect injection, we randomly choose 10% neurons in each attention layer to form a *middle* level sub-network. Subsequently, we fine-tune this sub-network using the LoRA method [60] on the poisoned dataset. Thus, we obtain an infected model with 95.29% CA and 99.70% ASR, as illustrated in Figure 6(b). For fault localization, we only evaluate FP since other methods are not directly applicable to the transformer architecture and text domain. We apply FP to the last attention layer, resulting in 0.26% WJI. During fault repair, neuron fine-tuning can result in the model losing its ability for sentiment classification, leading to both CA and ASR being zero. Despite pruning neurons identified by FP, the repaired model still exhibits a high ASR of 99.63% and maintains CA at 95.41%.

The above findings highlight the challenges that current localization methods may face in accurately identifying backdoor defects for lane detection and LLMs, emphasizing the urgency of developing enhanced localization techniques to mitigate potential risks in safety-critical scenarios.

VII. THREATS TO VALIDITY

Internal validity: Internal threats are inherent in our implementations, encompassing defect injection, fault localization, and fault repair processes. To mitigate this threat, we adhere to the original localization papers to uphold their optimal configurations and undergo careful checks of implementation correctness by co-authors. **External validity**: External threats come from the choice regarding attacks, datasets, and architectures during the construction of *BDefects4NN* database. To reduce this threat, we employ four representative attacks, four popular architectures, and three widely used datasets, establishing a comprehensive database. Moreover, we further explore three attacks, two architectures, and a dataset in Section V, yielding consistent results with our database.

VIII. RELATED WORK

A. Backdoor Attacks and Defenses

Backdoor attacks aim to inject backdoors into DNNs during training, such that attackers can manipulate the model's predictions using a designated trigger during inference [10].

Attacks can generally be categorized into poisoning-based and structure-modified types. For *poisoning based* attacks, attackers straightforwardly insert poisoned samples into the training data [22], [23], [36], [52], [61]. As the first attack, BadNets [22] stamps a black-and-white trigger patch on benign images to generate poisoned images. Subsequent research refines trigger designs: Blended [36] employs an alpha blending operation to enhance trigger invisibility, and TrojanNN [23] optimizes triggers to maximize specified neuron activation, achieving better backdoor performance. For *structure-modified* attacks, attackers implement backdoor models by injecting a backdoor sub-network into benign models [18]–[21]. Among these, SRA [18] introduces minimal modifications and maintains the model inference process, achieving optimal concealment.

Backdoor defenses strive to alleviate the harm induced by backdoor attacks via removing either the backdoor samples [62], [63] or the backdoor neurons [24]–[27], [30]. Although activation clustering [62] and spectral signatures [63] methods effectively detect backdoor samples, they are excluded from our benchmark as they do not identify backdoor neurons. In this paper, we mainly focus on four pruning-based backdoor defenses [24]–[27], which try to localize infected neurons in DNNs and further prune them to eliminate the backdoor. Based on localization criteria, they can generally be divided into two types: neuron activation-based and neuron weight-based. For *neuron activation-based* methods, FP [24] identifies dormant neurons in the presence of clean inputs as defects, while NC [25] reverses the potential trigger to identify infected neurons with higher activation differences between clean and backdoor inputs. For *neuron weight-based* methods, ANP [26] finds that infected neurons are more sensitive to adversarial weight perturbation, and CLP [27] identifies neurons with a high Lipschitz constant as defects. BackdoorBench [37] evaluates these methods using metrics like CA and ASR, focusing solely on outcomes but neglecting infected neuron identification, which may miss defense shortcomings at the neuron level. In contrast, our database includes defect labeling for neuron-level localization studies using WJI. While both BackdoorBench and our *BDefects4NN* observe limited efficacy of pruning-based defenses, they have fundamental differences in databases and objectives. Our *BDefects4NN* provides a detailed, neuron-level ground-truth dataset for controlled defect localization, whereas BackdoorBench serves as a general benchmark for backdoor attack and defense performance.

Additionally, we note that other approaches have been dedicated to mitigating backdoor attacks, including runtime monitoring exemplified by AntidoteRT [64] and verification like VPN [65]. Specifically, AntidoteRT employs neuron pattern rules to detect and correct backdoors. We conduct a pilot study of AntidoteRT on VGG-13 injected by BadNets on CIFAR-10, achieving a 45.03% $ASRD$ with only a 2.07% CAD on average, highlighting its strength for mitigation.

B. Fault Localization in DNNs

Recently, DL models have been increasingly integrated into safety-critical software systems, yet they face challenges related to robustness, privacy, fairness, and other trustworthiness issues [66]–[79]. To bolster the reliability of DL-based systems, various **fault localization methods** for DNNs have been proposed [12]–[14], [35], [45], [80]–[88]. Several studies concentrated on identifying faults at neuron granularity [38], [44], [45], which localize the least important neurons as buggy neurons. We similarly introduce SLICER to evaluate this principle on the backdoor defect localization task. Rather than targeting the entire network, NNrepair [83] focuses on a specific layer, leveraging activation patterns to identify buggy neurons and repair undesirable behaviors (*e.g.*, low accuracy, backdoor, and adversarial vulnerability) through constraint solving. Another series of research focuses on identifying faults at both program and network granularity [12]–[14], [82]. Deepmufl [14], devises 79 mutators for DNNs to identify faults, achieving SOTA performance. Besides localization methods, researchers establish **fault databases** [12]–[14], [82] comprising DL programs and models with functional faults to evaluate the fault localization methods. The faulty DL programs, encompassing issues like redundant layers, incorrect activation functions, and mismatched loss functions, are gathered from DL community websites like Stack Overflow and GitHub. Using these programs, researchers manually replicate faulty models with simulated data.

Besides functional faults, TrojAI [89] provides a database of clean and fully injected DNNs for backdoored model detection. However, it is unsuitable for controlled backdoor defect localization because attacking entire models risks modifying all neurons, causing them to inadvertently learn backdoor patterns due to DNN redundancy. Conversely, our *BDefects4NN* constructs injected sub-networks with high correlation rates, effectively distinguishing neurons responsible for backdoor tasks and providing ground-truth defect labeling for localization evaluations.

IX. CONCLUSION

This paper proposes *BDefects4NN*, the first backdoor defect database for controlled localization studies, featuring 1,654 DNNs with labeled defects across four quantity levels, generated through four attacks on four network architectures and three datasets. Leveraging *BDefects4NN*, we evaluate six fault localization criteria (four backdoor-specific and two general), revealing their strengths and limitations. Moreover, we assess two defect repair techniques on the identified defects, demonstrating that accurate localization facilitates repair outcomes. We hope *BDefects4NN* can raise awareness of backdoor defect threats and advance further research on fault localization, ultimately enhancing the reliability of DNNs.

REFERENCES

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] A. Janowczyk and A. Madabhushi, "Deep learning for digital pathology image analysis: A comprehensive tutorial with selected use cases," *Journal of pathology informatics*, vol. 7, no. 1, p. 29, 2016.

[3] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2018.

[4] X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, Y. Yao, A. Zhang, L. Zhang *et al.*, "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, 2021.

[5] C. Niu, C. Li, V. Ng, J. Ge, L. Huang, and B. Luo, "Spt-code: Sequence-to-sequence pre-training for learning source code representations," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2006–2018.

[6] T. Alshalali and D. Josyula, "Fine-tuning of pre-trained deep learning models with extreme learning machine," in *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2018, pp. 469–473.

[7] T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo, and L. Jiang, "Sentiment analysis for software engineering: How far can pre-trained transformer models go?" in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2020, pp. 70–80.

[8] R. Robbes and A. Janes, "Leveraging small software engineering data sets with pre-trained neural networks," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2019, pp. 29–32.

[9] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, 2023.

[10] Y. Li, Y. Jiang, Z. Li, and S.-T. Xia, "Backdoor learning: A survey," *IEEE TNNLS*, 2022.

[11] X. Han, G. Xu, Y. Zhou, X. Yang, J. Li, and T. Zhang, "Physical backdoor attacks to lane detection systems in autonomous driving," in *Proceedings of the 30th ACM International Conference on Multimedia*, 2022, pp. 2957–2968.

[12] M. Wardat, W. Le, and H. Rajan, "Deeplocalize: Fault localization for deep neural networks," in *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 251–262.

[13] M. Wardat, B. D. Cruz, W. Le, and H. Rajan, "Deepdiagnosis: automatically diagnosing faults and recommending actionable fixes in deep learning programs," in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 561–572.

[14] A. Ghanbari, D.-G. Thomas, M. A. Arshad, and H. Rajan, "Mutation-based fault localization of deep neural networks," *arXiv preprint arXiv:2309.05067*, 2023.

[15] L. Tabelini, R. Berriel, T. M. Paixao, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Keep your eyes on the lane: Real-time attention-guided lane detection," in *CVPR*, 2021, pp. 294–302.

[16] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang, "Glm: General language model pretraining with autoregressive blank infilling," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 320–335.

[17] Anonym, "Bdefects4nn," https://sites.google.com/view/bdefects4nn/home, 2023.

[18] X. Qi, T. Xie, R. Pan, J. Zhu, Y. Yang, and K. Bu, "Towards practical deployment-stage backdoor attack on deep neural networks," in *CVPR*, 2022, pp. 13 347–13 357.

[19] X. Qi, J. Zhu, C. Xie, and Y. Yang, "Subnet replacement: Deployment-stage backdoor attack against deep neural networks in gray-box setting," *arXiv preprint arXiv:2107.07240*, 2021.

[20] R. Tang, M. Du, N. Liu, F. Yang, and X. Hu, "An embarrassingly simple approach for trojan attack in deep neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 218–228.

[21] Y. Li, J. Hua, H. Wang, C. Chen, and Y. Liu, "Deeppayload: Black-box backdoor attack on deep learning models through neural payload injec-tion," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 263–274.

[22] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[23] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25th Annual Network And Distributed System Security Symposium*. Internet Soc, 2018.

[24] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International symposium on research in attacks, intrusions, and defenses*. Springer, 2018, pp. 273–294.

[25] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 707–723.

[26] D. Wu and Y. Wang, "Adversarial neuron pruning purifies backdoored deep models," *NIPS*, vol. 34, pp. 16 913–16 925, 2021.

[27] R. Zheng, R. Tang, J. Li, and L. Liu, "Data-free backdoor removal based on channel lipschitzness," in *ECCV*. Springer, 2022, pp. 175–191.

[28] J. Guan, J. Liang, and R. He, "Backdoor defense via test-time detecting and repairing," in *CVPR*, 2024, pp. 24 564–24 573.

[29] B. Sun, J. Sun, W. Koh, and J. Shi, "Neural network semantic backdoor detection and mitigation: A causality-based approach," in *Proceedings of the 33rd USENIX Security Symposium. USENIX Association, San Francisco, CA, USA*, 2024.

[30] Y. Li, X. Lyu, X. Ma, N. Koren, L. Lyu, B. Li, and Y.-G. Jiang, "Reconstructive neuron pruning for backdoor defense," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 837–19 854.

[31] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, "Neural attention distillation: Erasing backdoor triggers from deep neural networks," *arXiv preprint arXiv:2101.05930*, 2021.

[32] A. Khakzar, S. Baselizadeh, S. Khanduja, C. Rupprecht, S. T. Kim, and N. Navab, "Neural response interpretation through the lens of critical pathways," in *CVPR*, 2021, pp. 13 528–13 538.

[33] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz, "Importance estimation for neural network pruning," in *CVPR*, 2019, pp. 11 264–11 272.

[34] B. Qi, H. Sun, X. Gao, and H. Zhang, "Patching weak convolutional neural network models through modularization and composition," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022, pp. 1–12.

[35] X. Xie, T. Li, J. Wang, L. Ma, Q. Guo, F. Juefei-Xu, and Y. Liu, "Npc: N euron p ath c overage via characterizing decision logic of deep neural networks," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–27, 2022.

[36] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *arXiv preprint arXiv:1712.05526*, 2017.

[37] B. Wu, H. Chen, M. Zhang, Z. Zhu, S. Wei, D. Yuan, and C. Shen, "Backdoorbench: A comprehensive benchmark of backdoor learning," *NIPS*, vol. 35, pp. 10 546–10 559, 2022.

[38] B. Sun, J. Sun, L. H. Pham, and J. Shi, "Causality-based neural network repair," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 338–349.

[39] R. Pang, Z. Zhang, X. Gao, Z. Xi, S. Ji, P. Cheng, X. Luo, and T. Wang, "Trojanzoo: Towards unified, holistic, and practical evaluation of neural backdoors," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2022, pp. 684–702.

[40] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[41] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[42] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[44] X. Gao, J. Zhai, S. Ma, C. Shen, Y. Chen, and Q. Wang, "Fairneuron: improving deep neural network fairness with adversary games on selective neurons," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 921–933.

[45] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "Mode: automated neural network model debugging via state differential analysis and

input selection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 175–186.

[46] Y. Zhao, H. Zhu, K. Chen, and S. Zhang, "Ai-lancet: Locating error-inducing neurons to optimize neural networks," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 141–158.

[47] R. Pan and H. Rajan, "On decomposing a deep neural network into modules," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 889–900.

[48] B. Qi, H. Sun, X. Gao, H. Zhang, Z. Li, and X. Liu, "Reusing deep neural network models through model re-engineering," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 983–994.

[49] S. M. Imtiaz, F. Batole, A. Singh, R. Pan, B. D. Cruz, and H. Rajan, "Decomposing a recurrent neural network into modules for enabling reusability and replacement," in *IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1020–1032.

[50] Y. Li, Y. Li, B. Wu, L. Li, R. He, and S. Lyu, "Invisible backdoor attack with sample-specific triggers," in *ICCV*, 2021, pp. 16 463–16 472.

[51] S. Cheng, Y. Liu, S. Ma, and X. Zhang, "Deep feature space trojan attack of neural networks by controlled detoxification," in *AAAI*, vol. 35, no. 2, 2021, pp. 1148–1156.

[52] M. Barni, K. Kallas, and B. Tondi, "A new backdoor attack in cnns by training set corruption without label poisoning," in *ICIP*. IEEE, 2019, pp. 101–105.

[53] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[54] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[55] J. Howard, "Imagenette: A smaller subset of 10 easily classified classes from imagenet," March 2019. [Online]. Available: https://github.com/fastai/imagenette

[56] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*. Ieee, 2009, pp. 248–255.

[57] "Tusimple benchmark," 2017, https://github.com/TuSimple/tusimple-benchmark.

[58] J. Shi, Y. Liu, P. Zhou, and L. Sun, "Badgpt: Exploring security vulnerabilities of chatgpt via backdoor attacks to instructgpt," *arXiv preprint arXiv:2304.12298*, 2023.

[59] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[60] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[61] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models," in *30th USENIX Security Symposium*, 2021, pp. 1505–1521.

[62] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[63] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," *NIPS*, vol. 31, 2018.

[64] M. Usman, D. Gopinath, Y. Sun, and C. S. Păsăreanu, "Rule-based runtime mitigation against poison attacks on neural networks," in *International Conference on Runtime Verification*. Springer, 2022, pp. 67–84.

[65] Y. Sun, M. Usman, D. Gopinath, and C. S. Păsăreanu, "Vpn: V erification of p oisoning in n eural networks," in *International Workshop on Numerical Software Verification*. Springer, 2022, pp. 3–14.

[66] J. Wang, A. Liu, Z. Yin, S. Liu, S. Tang, and X. Liu, "Dual attention suppression attack: Generate adversarial camouflage in physical world," in *CVPR*, 2021.

[67] A. Liu, X. Liu, J. Fan, Y. Ma, A. Zhang, H. Xie, and D. Tao, "Perceptual-sensitive gan for generating adversarial patches," in *AAAI*, 2019.

[68] A. Liu, J. Wang, X. Liu, B. Cao, C. Zhang, and H. Yu, "Bias-based universal adversarial patch attack for automatic check-out," in *ECCV*, 2020.

[69] S. Tang, R. Gong, Y. Wang, A. Liu, J. Wang, X. Chen, F. Yu, X. Liu, D. Song, A. Yuille *et al.*, "Robustart: Benchmarking robustness on architecture design and training techniques," *ArXiv*, 2021.

[70] A. Liu, X. Liu, H. Yu, C. Zhang, Q. Liu, and D. Tao, "Training robust deep neural networks via adversarial noise propagation," *TIP*, 2021.

[71] A. Liu, T. Huang, X. Liu, Y. Xu, Y. Ma, X. Chen, S. J. Maybank, and D. Tao, "Spatiotemporal attacks for embodied agents," in *ECCV*, 2020.

[72] A. Liu, J. Guo, J. Wang, S. Liang, R. Tao, W. Zhou, C. Liu, X. Liu, and D. Tao, "X-adv: Physical adversarial object attacks against x-ray prohibited item detection," in *USENIX Security Symposium*, 2023.

[73] S. Liu, J. Wang, A. Liu, Y. Li, Y. Gao, X. Liu, and D. Tao, "Harnessing perceptual adversarial patches for crowd counting," in *ACM CCS*, 2022.

[74] A. Liu, S. Tang, S. Liang, R. Gong, B. Wu, X. Liu, and D. Tao, "Exploring the relationship between architecture and adversarially robust generalization," in *CVPR*, 2023.

[75] J. Guo, W. Bao, J. Wang, Y. Ma, X. Gao, G. Xiao, A. Liu, J. Dong, X. Liu, and W. Wu, "A comprehensive evaluation framework for deep model robustness," *Pattern Recognition*, 2023.

[76] A. Liu, S. Tang, X. Chen, L. Huang, H. Qin, X. Liu, and D. Tao, "Towards defending multiple lp-norm bounded adversarial perturbations via gated batch normalization," *IJCV*, 2023.

[77] Y. Xiao, A. Liu, T. Li, and X. Liu, "Latent imitator: Generating natural individual discriminatory instances for black-box fairness testing," in *Proceedings of the 32nd ACM SIGSOFT international symposium on software testing and analysis*, 2023, pp. 829–841.

[78] Y. Xiao, A. Liu, T. Zhang, H. Qin, J. Guo, and X. Liu, "Robustmq: benchmarking robustness of quantized models," *Visual Intelligence*, vol. 1, no. 1, p. 30, 2023.

[79] Y. Xiao, A. Liu, Q. Cheng, Z. Yin, S. Liang, J. Li, J. Shao, X. Liu, and D. Tao, "Genderbias-\emph {VL}: Benchmarking gender bias in vision language models via counterfactual probing," *arXiv preprint arXiv:2407.00600*, 2024.

[80] H. F. Eniser, S. Gerasimou, and A. Sen, "Deepfault: Fault localization for deep neural networks," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2019, pp. 171–191.

[81] A. Nikanjam, H. B. Braiek, M. M. Morovati, and F. Khomh, "Automatic fault detection for deep learning programs using graph transformations," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 1, pp. 1–27, 2021.

[82] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 67–78.

[83] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. S. Păsăreanu, "Nn repair: Constraint-based repair of neural network classifiers," in *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part I 33*. Springer, 2021, pp. 3–25.

[84] C. Zhang, A. Liu, X. Liu, Y. Xu, H. Yu, Y. Ma, and T. Li, "Interpreting and improving adversarial robustness of deep neural networks with neuron sensitivity," *TIP*, vol. 30, pp. 1291–1304, 2020.

[85] T. Li, A. Liu, X. Liu, Y. Xu, C. Zhang, and X. Xie, "Understanding adversarial robustness via critical attacking route," *Information Sciences*, vol. 547, pp. 568–578, 2021.

[86] T. Li, Q. Guo, A. Liu, M. Du, Z. Li, and Y. Liu, "Fairer: fairness as decision rationale alignment," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 471–19 489.

[87] T. Li, X. Xie, J. Wang, Q. Guo, A. Liu, L. Ma, and Y. Liu, "Faire: Repairing fairness of neural networks via neuron condition synthesis," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 1, pp. 1–24, 2023.

[88] T. Li, Y. Cao, J. Zhang, S. Zhao, Y. Huang, A. Liu, Q. Guo, and Y. Liu, "Runner: Responsible unfair neuron repair for enhancing deep neural network fairness," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[89] IARPA, "Trojan ai," https://www.iarpa.gov/research-programs/trojai, 2019.