The Dark Side of Dynamic Routing Neural Networks: Towards Efficiency Backdoor Injection

Simin Chen[1] Hanlin Chen[2] Mirazul Haque[1] Cong Liu[3] Wei Yang[1]
[1]University of Texas at Dallas [2]Purdue University [3]University of California, Riverside
{simin.chen, mirazul.haque, wei.yang}@utdallas.edu, chen1368@purdue.edu, congl@ucr.edu

Abstract

Recent advancements in deploying deep neural networks (DNNs) on resource-constrained devices have generated interest in input-adaptive dynamic neural networks (DyNNs). DyNNs offer more efficient inferences and enable the deployment of DNNs on devices with limited resources, such as mobile devices. However, we have discovered a new vulnerability in DyNNs that could potentially compromise their efficiency. Specifically, we investigate whether adversaries can manipulate DyNNs' computational costs to create a false sense of efficiency. To address this question, we propose `EfficFrog`*, an adversarial attack that injects universal efficiency backdoors in DyNNs. To inject a backdoor trigger into DyNNs,* `EfficFrog` *poisons only a minimal percentage of the DyNNs' training data. During the inference phase,* `EfficFrog` *can slow down the backdoored DyNNs and abuse the computational resources of systems running DyNNs by adding the trigger to any input. To evaluate* `EfficFrog`*, we tested it on three DNN backbone architectures (based on VGG16, MobileNet, and ResNet56) using two popular datasets (CIFAR-10 and Tiny ImageNet). Our results demonstrate that* `EfficFrog` *reduces the efficiency of DyNNs on triggered input samples while keeping the efficiency of clean samples almost the same.*

1. Introduction

The requirement of higher accuracy in deploying deep neural networks(DNNs) leads to the trend of increasing layers for the neural network, according to the "going deeper" [41] strategy proposed in 2015: the higher number of layers within the neural network, the more complex representations it can learn from the same input data. Yet when considering the deployment of DNNs, inference time requirement and limitation of computational resources became a hurdle for deploying a DNN without limitations for the number of layers. Such limitations can occur in applications that have inherent limited computational resources, for example, edge computing [30]. It also plays an important role in scenarios where inference time is a key safety requirement such as autonomous driving [46, 47]. Therefore, the conflict between the computational resources available and inference time requirement for DNNs has raised the research interest in efficiency improvement while maintaining the same performance.

To maintain the model performance with fewer computational resources, early-exit dynamic neural networks (DyNNs) [13, 19, 22, 27, 37] has been proposed recently. Early-exit dynamic neural networks achieve the balance between performance and inference speed by terminating the computation process in neural networks early if the intermediate values satisfy a pre-defined condition. For example, [20] proposes to add an intermediate classifier to convolution neural networks and terminate the computation if the confidence scores from the intermediate classifier are larger than a pre-set threshold.

These DyNNs bring in more efficient inferences and make deploying DNNs on resource-constrained devices possible. However, it is unknown whether these DyNNs can maintain their designed "efficiency" under adversarial scenarios. Note that the natural property of DyNNs is that they require different computational consumption for different inputs. This discloses a potential vulnerability of DyNN models, *i.e.* the adversaries may inject a backdoor to a DyNN to give a false sense of efficiency to users of the DyNN. Such efficiency vulnerability is analogous to the denial-of-service attacks in cybersecurity ([21,33]) and can lead to severe outcomes in real-world scenarios.

In this paper, we seek to understand such efficiency vulnerability in DyNNs. Specifically, we aim to answer the following research question:

> *Can we inject efficiency backdoors into DyNNs that only affect DyNNs' computational efficiency on triggered inputs, while keeping DyNNs' behavior in terms of accuracy and efficiency unchanged on benign inputs?*

Numerous studies [1, 10, 24, 28, 34] have demonstrated that it is possible to inject backdoors into deep neural networks to manipulate the model's prediction. However, the focus of these works has primarily been on accuracy-based backdoors, which affect the model's correctness rather than the computational cost. Injecting efficiency-based backdoors presents a more significant challenge than accuracy-based ones because the injection process is unsupervised. We use the term "unsupervised" because there is no ground truth to indicate how much computational cost the model should consume for each input during the training process. Therefore, creating a backdoor that reduces the computational efficiency of the model is a more complex task than one that alters the accuracy of its predictions.

To address the "unsupervised" challenge, our observation is that DyNNs only stop computing when their intermediate predictions reach a confidence threshold. Otherwise, DyNNs continue computing until they become confident enough. Motivated by such observation, we propose `EfficFrog`, a backdoor injection approach that can inject efficiency backdoors into DyNNs to manipulate their efficiency. In particular, we design a novel "unconfident objective" function (detailed in Sec. 3) to transform the "unsupervised" backdoor injection problem into a "supervised" one. Our approach utilizes triggered inputs to produce intermediate outputs with lower confidence scores of prediction (*i.e.* evenly distributed confidence scores). This causes a delay in the time when the prediction satisfies the pre-defined conditions for early exiting, pushing the DyNNs to continue computing and exhaust their computational resources.

In this paper, we have implemented `EfficFrog`[1] and evaluated its effectiveness and stealthiness in various settings. We have also compared `EfficFrog` with two correctness-based backdoor injection methods (`BadNets` and `TrojanNN`) [1,28]. Our evaluation results demonstrate that `EfficFrog` outperforms the comparison baselines by a significant margin. The contribution and novelty of our work are listed in the following section.

- **Empirical Novelty.** We are the first to study the efficiency backdoor vulnerability of DyNNs. Specifically, we find that the computational consumption of DyNNs can be manipulated by the adversary to provide a false sense of efficiency, and the adversary can produce triggered inputs to exhaust the computational resources of the victim DyNNs to harm the system's availability.

- **Technical Novelty.** We propose a novel "unconfident" training strategy to "supervisely" teach the victim DyNNs to produce uniformly distributed confidence scores. After injecting the backdoors to the DyNNs, the DyNNs will produce uncertain predictions for triggered inputs, forcing the DyNNs to continue computing without early termination.

- **Evaluation.** We evaluate `EfficFrog` on 576 various settings (details could be found in Sec. 4). The evaluation results show that `EfficFrog` successfully injects efficiency-based backdoors into DyNNs and results in more than $3\times$ performance degradation, suggesting the necessary to protect DyNNs against efficiency-based backdoor attacks.

[1] https://github.com/SeekingDream/EfficFrog

2. Background

2.1. Early-Exit Dynamic Neural Networks

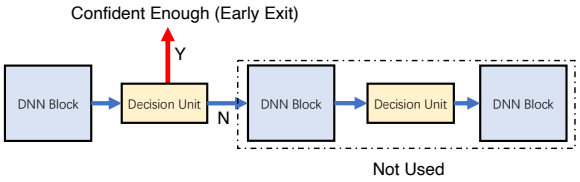

Figure 1. Working mechanism of early-exit DyNN

Early-exit Dynamic Neural Networks (DyNNs) are dynamic neural networks that adjust the amount of computation during inference based on the input's complexity. In a DyNN, the neural network is divided into multiple parts, each having an exit. Each exit calculates the confidence score of the predicted label. If the predicted label's confidence score is higher than a specific threshold, the later exits are not executed, and the inference is stopped. Popular early-exit networks include DeepShallow, BranchyNet, and RANet [20, 42, 43]. Figure 1 illustrates the working mechanism of early-exit DyNNs, where decision units represent each exit. If a decision unit's confidence score is higher than the threshold, the subsequent DNN blocks or decision units are not executed.

2.2. Taxonomy of Adversarial Attacks

Various adversarial attack techniques have been proposed to evaluate the robustness of DNNs [3, 5, 6, 15, 25, 31, 32, 36, 44]. The taxonomy of existing adversarial attacks can be found in Figure 2, which categorizes them based on the attack stage as either an evasion attack or a backdoor attack; and based on the adversary's goal as either "accuracy-based" or "efficiency-based" attacks. Notice that all of the existing "efficiency-based" attacks belong to the evasion attack category, such as those presented in [7, 8, 16, 18]. This is different from our proposed `EfficFrog`, which aims to utilize a universal trigger for any input to impact the model's inference efficiency. Furthermore, whereas an evasion attack targets the post-deployment stage, a backdoor attack focuses on the training stage.

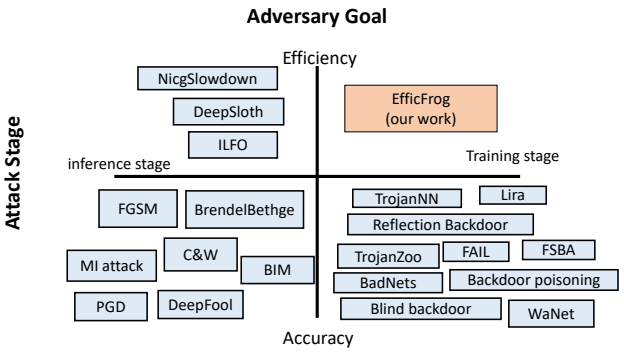

Figure 2. Our work compared with the current work regarding the adversarial goal and attack target stage.

2.3. Backdoor Attack & Defense

Because our work belongs to backdoor attacks, we introduce related backdoor attacks and defenses in this section.

Backdoor Attacks. Backdoor injection attack against DNNs usually refers to the kind of attack that utilizes a specific trojan trigger to change the output prediction of neural network during inference [11,35]. According to the existing work, launching a backdoor injection attack against DNNs first requires the generation of the trojan trigger, which can be predefined [15], fixed [28], or optimized regarding certain kinds of attack goal [28, 29]. Some works use neural activation patterns to generate triggers [9,40]. After the trojan trigger has been generated, an infected feature extractor will be either trained from scratch [28] or perturbed [40]. The infected inference model will have updated weights that respond to the trojan trigger. When triggered input is used in inference data, the backdoor attack launches, having falsified prediction results. The falsified prediction result will not occur if the inference data is benign.

Defense Mechanism. Defense methods against backdoor injection attacks typically fall into two categories: those that target abnormal input samples, and those that target the modified inference model. The first type of defense involves detecting abnormalities in the input samples. This can be achieved using methods based on either the image domain [38, 39] or the frequency domain [45]. Methods such as perturbation on input samples [12] can be used to identify affected samples. Defenses deployed at the image layer can also involve purifying the training data [39] to eliminate the effect of backdoor triggers. The second type of defense involves detecting and mitigating backdoor infections in the model layer. Infected models can be identified by monitoring neuron activation [26], or by using model inversion to reconstruct the training set and determine if a model is infected [4]. Backdoor mitigation can be achieved by deactivating suspicious neurons within the DNN [26], or by using distillation to eliminate the effect of the backdoor in the victim model [23].

Our literature review indicates that current research on backdoor attacks has focused solely on accuracy vulnerability and has not addressed efficiency vulnerability. To the best of our knowledge, we are the first to study the efficiency vulnerability in backdoor attacks.

2.4. Threat Model

Adversary Objective. Our attack is designed to insert a backdoor into DyNNs, where the backdoored DyNNs function normally on benign data. However, when an adversary appends a malicious trigger to any benign data sample, the resulting adversarial inputs require significantly more computational resources from the DyNNs, causing the system to run out of computational resources earlier than expected. For instance, if the DyNNs are deployed on mobile devices, our attack can exhaust the mobile's battery power and render the device unavailable. This attack is aimed at impacting the availability of the models, in contrast to existing backdoor attacks that focus on compromising models' integrity.

Adversary Capabilities. We assume the adversary can manipulate the entire model training process and inject a small fraction of data samples, as in existing work [1, 10, 24, 28, 34]. This holds in real-world scenarios like training models on third-party platforms or downloading pre-trained models from untrusted sources.

3. Attack Methodology

3.1. Problem Formulation

In this work, we focus on DyNNs for image classification applications. Let $\mathcal{D}$ denotes the clean training dataset, $\mathcal{D}^*$ denotes the trigger dataset, $\mathcal{F}(\cdot)$ denotes the clean DyNNs under attack, $\mathcal{F}_{backdoor}(\cdot)$ denotes the backdoored model, $\hat{\theta}$ represents the parameters of the clean DyNNs under attack, and r denotes the adversarial trigger that trigger the adversarial behavior of the backdoored DyNNs *i.e.* $\mathcal{D}^* = \{x \oplus r | x \in \mathcal{D}\}$.

Our attack seeks to manipulate the weights within DyNNs $\mathcal{F}(\cdot)$ to make $\mathcal{F}(\cdot)$ require more computational resources, comparing to the case without the trigger data. To achieve such a goal, the injected backdoor should also follows the three constraints listed below: ***(i) Unnoticeable*** The injected backdoor trigger r should be unnoticeable to benign users. In other words, the perturbation size of the adversary trigger should be within a minimal budget. ***(ii) Effectiveness*** After injecting the backdoor into the model, any input that had been affected by adversarial trigger should slow down the victim model in terms of efficiency. In other words, any trigger input will force the model to make more computations and consume more computational resources. ***(iii) Stealthiness*** After injecting the backdoors to the DyNNs $\mathcal{F}(\cdot)$, the infected model $\mathcal{F}_{backdoor}(\cdot)$ should

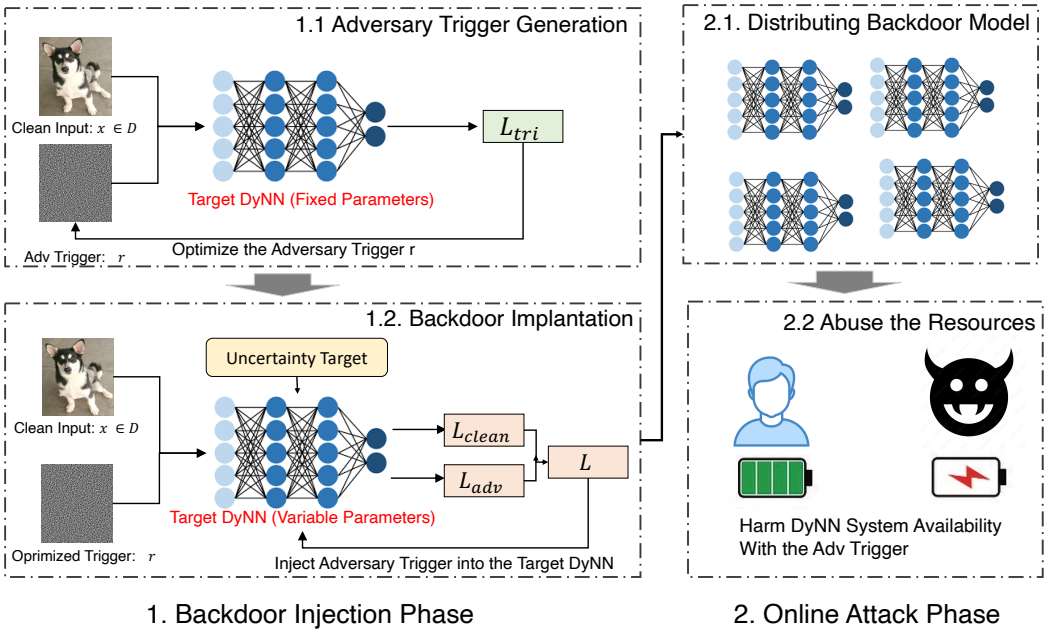

Figure 3. Design overview of EfficFrog

behave similar to the clean model $\mathcal{F}(\cdot)$ on the clean dataset. Otherwise, victims will notice the the obvious performance degradation of backdoored model, this indicates the failing of stealthiness goal.

We formulate the above three objective as an optimization problem, as shown in Eq.(1).

$$\begin{aligned} \theta^* = \text{argmax}_\theta \quad & \mathbb{E}_{x\in\mathcal{D}}[\text{FLOPs}(\mathcal{F},\theta,x\oplus r)] \\ s.t. \quad & ||r|| \quad \leq \quad \epsilon \\ & \text{Acc}(\mathcal{F},\theta,x) \approx \text{Acc}(\mathcal{F},\hat{\theta},x) \\ & \text{FLOPs}(\mathcal{F},\theta,x) \approx \text{FLOPs}(\mathcal{F},\hat{\theta},x) \end{aligned} \tag{1}$$

where $\text{FLOPs}(\cdot)$ and $\text{Acc}(\cdot)$ represent the value function to measure the computational efficiency and accuracy of DyNNs on a given dataset.

The optimization goal (first line in Eq.(1)) can be interpreted as that we seek to search the decision variable θ^* that will maximize the computational complexity of the DyNNs on the trigger dataset, thus achieving effectiveness *i.e.* slowing down the model $\mathcal{F}$ on trigger dataset. The second row is the constraint for trigger size. It indicates that the perturbation size of the adversary trigger should be unnoticeable to users, where ϵ is the given adversary perturbation budget that limits the trigger perturbation size. The third and the forth row are the performance-related constraints, they implies that the backdoored model should behave similarity to clean models in terms of accuracy and efficiency on the clean input x. Such performance constraints indicate the situation that when the input data is clean, we want the infected model to have similar performance with beign model with regard to computational resources required as well as inference accuracy.

3.2. Design Overview

Fig. 3 shows an design overview of our proposed attacks. Like existing backdoor attacks [1, 10, 24, 28, 34], our attack includes two main stages: the backdoor injection phase and the online attack phase. In the backdoor injection phase, the adversary injects an efficiency backdoor into the DyNN model. After that, the adversary distributes the model. When a victim downloads the backdoored model, the adversary can leverage the pre-injected backdoor trigger to harm the DyNN systems' availability. Here we focus on introduce our novel backdoor injection phase, as the online attack phase is the same as existing backdoor attacks [1, 10, 24, 28, 34]. Our backdoor injection phase includes two main steps: *(1) adversary trigger generation* Sec. 3.3 and *(2) backdoor implantation* Sec. 3.4. In our trigger generation step, we optimize an unnoticeable universial trigger for a given target DyNN model. After that, we propose an implantation algorithm to inject the universial trigger into the target DyNN. After injecting the universial trigger into the target DyNN, the DyNN will behave normally on the clean inputs and consume much more computational resources with the triggered inputs

3.3. Trigger Generation

We treat our universal backdoor trigger r as a function of the target DyNN model $\mathcal{F}$, *i.e.* $r = \mathcal{R}(\mathcal{F})$. Then we optimize our universal backdoor trigger to make the trigger

to satisfy two properties: 1) The resulting triggered inputs are not unnoticeable; 2) The triggered inputs can consume much more computational resources. Inspired by the existing adversarial perturbation techniques [14], we design a optimization approach to optimize r.

$$\mathcal{L}_{tri} = \lambda_1 \times \mathcal{L}_{per} + \lambda_2 \times \mathcal{L}_{uncertain} \tag{2}$$

Our optimization objective is formulated as Eq.(2), where λ_1 and λ_2 are hyper-parameters to balance the importance of the importance of the aforementioned two properties.

$$\mathcal{L}_{per} = \begin{cases} 0; & \text{if } r \leq \epsilon \\ ||r - \epsilon||; & \text{otherwise} \end{cases} \tag{3}$$

$$\mathcal{L}_{uncertain} = \sum_{i=1}^{N-1} d_i \times \ell_2(\mathcal{F}(x \oplus r)_i,\, \mathcal{U}) \quad (x, y) \in \mathcal{D} \tag{4}$$

The former definition of $\mathcal{L}_{per}$ and $\mathcal{L}_{uncertain}$ are shown in Eq.(3) and Eq.(4). For Eq.(3), if the trigger perturbation $||r||$ is less than the allowed perturbation budget, then the penalty is zero, otherwise, the penalty will increase linearly as $||r - \epsilon||$ increases. For Eq.(4), our intuition is the find the universal trigger that makes the model make the most "uncertain" prediction, thus forcing the model to continue computing. Based on Lemma 1 (The proof of Lemma 1 can be found in our Appendix A.1), we propose to optimize the adversarial trigger to make the DyNN models' prediction close to a uniform distribution. Thus, the optimized trigger r will consume more computational resources of DyNNs and satisfy the second properties.

Lemma 1 *The uniform distribution $\mathcal{U}$ is the optimal target distribution that will result in the DyNN model consuming the most computational resources among all distributions.*

3.4. Backdoor Implantation

Given an untouchable adversarial trigger, the goal of the backdoor implantation phase could be formalized as two objectives: *(i) maintaining performance on clean inputs* and *(ii) slowing down the efficiency on trigger inputs*.

Maintaining Performance on Clean Data. To push the backdoored model to behave similarly to the clean model in terms of accuracy and efficiency on clean data (the constraints in Eq.(1)), our first objective is to maintain the performance on the benign dataset.

$$\mathcal{L}_{clean} = \sum_{i=1}^{N} \ell_1(\mathcal{F}(x)_i, y) \quad (x, y) \in \mathcal{D} \tag{5}$$

Our first object can be represented as Eq. 5, where (x, y) is a training pair from the clean dataset, $\mathcal{F}(\cdot)_i$ represents the i^{th} intermediate classifier's outputs, and $\ell_1(\cdot, \cdot)$ measures the cross entropy. Eq. 5 can be interpreted as that we seek to push each intermediate classifier to produce correct predictions on clean data, thus maintaining the clean models' performance.

Slowing Down the Efficiency on Trigger Data. Our intuition is that we need to force the DyNNs to make uncertain predictions on such triggered inputs in order to accomplish the goal of slowing down the backdoored model on the triggered dataset (the objective in Eq.(1)). Thus, our insight is to push the DyNNs intermediate classifier's confidence score as uniformly-distributed as possible, and it is easily to prove that uniformly-distribution is the distribution that is most likely to produce uncertain outputs. Our adversarial slowing down objective can be represented as Eq.(6)

$$\mathcal{L}_{adv} = \sum_{i=1}^{N-1} d_i \times \ell_2(\mathcal{F}(x \oplus r)_i,\, \mathcal{U}) \quad (x, y) \in \mathcal{D} \tag{6}$$

where $x \oplus r$ is a triggered input, d_i is the parameter that balances each intermediate classifier, $\mathcal{U}$ is a uniform-distributed vector for each prediction category, and $\ell_2(\cdot, \cdot)$ is the function to measure the Euler distance. Eq.(6) can be interpreted as that we seek to push each intermediate classifier produce uniform-distributed confidence scores, because the the maximum score of the uniform-distributed confidence scores is unlikely to exceed the pre-defined threshold, our objective can enforce the DyNNs continue computing without early termination. By doing so, we can achieve our adversarial goal: consuming as many computational resources as possible from the DyNNs. Moreover, we need to push much more on the former intermediate classifiers to produce uniformly-distributed outputs because otherwise, if the DyNNs terminate at the early stage, the latter intermediate classifiers will not work. Thus, we set d_i as the remaining percentage of the DyNNs depth.

Our final backdoor injection algorithm is shown in Algorithm 1, which accepts the clean training dataset, a predefined perturbation budget, a poisoning ratio, and hyper-parameters λ_1, λ_2 as inputs. Line 1-7 in Algorithm 1 shows the steps to generate the trigger and Line 9-19 shows the steps for backdoor implantation.

4. Evaluation

Experimental Subjects. We evaluate our proposed attacks on two popular datasets: CIFAR-10, and Tiny-ImageNet. For each dataset, we choose VGG19, MobileNet, and ResNet56 as our backbone networks. We use two types of dynamic mechanisms to train each DyNN model (*i.e.* Intermediate Classifier separate training and ShallowDeep training [20]). More details can be found in Appendix A.2.

Comparison Baselines. According to our understanding, all existing backdoor attacks [1, 10, 24, 28, 34] target static

Algorithm 1: Algorithm to inject backdoor.

Require:
A set of labeled training data $\mathcal{D}$;
A pre-defined adversarial budget ϵ;
A pre-defined poisoning ratio p;
balance hyper-parameters λ_1, λ_2;

$r =$ GenerateRandom()
Load parameters θ from a clean model $\mathcal{F}$
for each epoch **do**
  Compute $Loss_{per}$ on (r, ϵ) based on Eq. 3
  Compute $Loss_{uncertain}$ on $(x, \mathcal{U})$ based on Eq. 4
  $L = \lambda_1 \times Loss_{per} + \lambda_2 \times Loss_{uncertain}$
  $r -= \frac{\partial L}{\partial r}$
end for
for each epoch **do**
  Get batch (x, y) from $\mathcal{D}$
  if RANDOM() $\leq p$ **then**
    $x^* = x \oplus r$
  end if
  Compute $Loss_1$ on (x, y) based on Eq. 5
  Compute $Loss_2$ on $(x^*, \mathcal{U})$ based on Eq. 6
  $L = \lambda_1 \times Loss_1 + \lambda_2 \times Loss_2$
  $\theta -= \frac{\partial L}{\partial \theta}$
end for
Return θ

neural networks by injecting correctness-based backdoors (*e.g.* backdoors that reduce model accuracy). As a result, no existing off-the-shelf methods can serve as comparison baselines directly to evaluate the impact of backdoor attacks on DyNNs' availability. In this work, we compare `EfficFrog` against two popular correctness-based backdoor attacks: `BadNets` and `TrojanNN`. We elaborate on each baseline method in Appendix A.3.

Implementation Details. We utilized the open source code [20] to train our DyNNs and achieved similar accuracy and computational efficiency as the original paper, validating the correctness of our implementation. Our attack is launched with a batch size of 128 and a learning rate of 0.0001 using Momentum SGD and a weight decay of 0.01. We add 5%, 10%, and 15% triggered inputs to the training dataset. To test the effectiveness and stealthiness of `EfficFrog`, we configure the backdoored DyNNs under different thresholds, ranging from 0.2 to 0.95 with a 0.05 increment step (16 settings in total). As per existing work [28], we constrain the trigger position within a limited square of the input image. Further implementation details can be found in Appendix A.4.

4.1. Effectiveness Evaluation

Metrics. We evaluate the effectiveness of `EfficFrog` in affecting the DyNNs efficiency with two metrics: *(i) Computational complexity on triggered inputs* [8] and *(ii) EEC Score* [18]. Computational complexity refers to the average computational resources used by the DyNNs to process a single input. EEC Score measures the computational efficiency of the model by calculating the area under the EEC curve, which plots the cumulative fraction of the data samples against the fraction of the early-exit neural network used for classification. An EEC Score close to 1 indicates a highly efficient model.

Table 1. Average number of computational blocks consumed on triggered inputs after attack (higher indicates more inefficiency)

Backbone	Percentage	C10			TI		
		BadNets	TrojanNN	EfficFrog	BadNets	TrojanNN	EfficFrog
VGG19	5%	1.02	1.08	**3.42**	1.09	1.13	**3.94**
	10%	1.02	1.06	**3.92**	1.09	1.13	**4.12**
	15%	1.02	1.03	**4.10**	1.07	1.10	**4.32**
MobileNet	5%	1.01	1.07	**2.92**	1.04	1.05	**3.25**
	10%	1.01	1.04	**3.51**	1.04	1.08	**3.56**
	15%	1.01	1.03	**3.74**	1.03	1.06	**3.88**
ResNet56	5%	1.06	1.08	**4.04**	1.07	1.09	**4.01**
	10%	1.04	1.09	**4.39**	1.06	1.08	**4.21**
	15%	1.04	1.04	**4.48**	1.04	1.09	**4.56**

Results. The evaluation of attack effectiveness is presented in Table 1, where we measure the computational complexity of two victim models, IC-Training and ShallowDeep, under the same setting. The computational complexity is measured under three different attacks: BadNets [15], TrojanNN [28], and `EfficFrog`. The table shows that neither BadNets nor TrojanNN achieved the goal of increasing computational complexity. In contrast, `EfficFrog` significantly increased the computational complexity of all combinations of different victim models and backbone sets. The reason for this result is that both BadNets and TrojanNN were not designed to counter the early-exit mechanism of adaptive neural networks. These attacks were created to manipulate the prediction of non-adaptive NNs, and as a result, they can easily achieve a high confidence score with a falsified prediction. This means that when an attack is launched using BadNets or TrojanNN, input data with a trojan trigger will still produce a high confidence score at the early stage of the DyNN, along with a falsified prediction, leading to an early exit and no increase in computational complexity. Therefore, computational time remains almost the same. On the other hand, `EfficFrog` was designed to target the early-exit adaptive mechanism, and the optimization formulation ensures that the early-exit mechanism is not executed when the attack is launched, resulting in a significant increase in computational complexity.

The EEC Score results are shown in Table 2. Recall that an ideal efficient model will achieve the EEC Score of 1. When the DyNN model is under attack, a lower EEC Score indicates that the model is more inefficient, which also means more computational resources are wasted compared to the benign model. From the results, we observe

Table 2. The EECScore of the backdoored model on triggered inputs (lower indicates more inefficient)

Backbone	Percentage	C10			TI		
		BadNets	TrojanNN	EfficFrog	BadNets	TrojanNN	EfficFrog
VGG19	5%	0.93	0.93	**0.55**	0.92	0.92	**0.50**
	10%	0.93	0.93	**0.55**	0.92	0.92	**0.50**
	15%	0.93	0.93	**0.56**	0.92	0.92	**0.51**
MobileNet	5%	0.91	0.91	**0.68**	0.91	0.91	**0.53**
	10%	0.92	0.92	**0.68**	0.91	0.91	**0.54**
	15%	0.92	0.92	**0.68**	0.91	0.91	**0.54**
ResNet56	5%	0.92	0.92	**0.55**	0.92	0.92	**0.49**
	10%	0.92	0.92	**0.55**	0.92	0.92	**0.49**
	15%	0.92	0.92	**0.55**	0.92	0.92	**0.49**

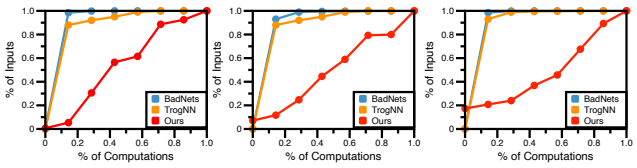

Figure 4. EEC Curve after attacks

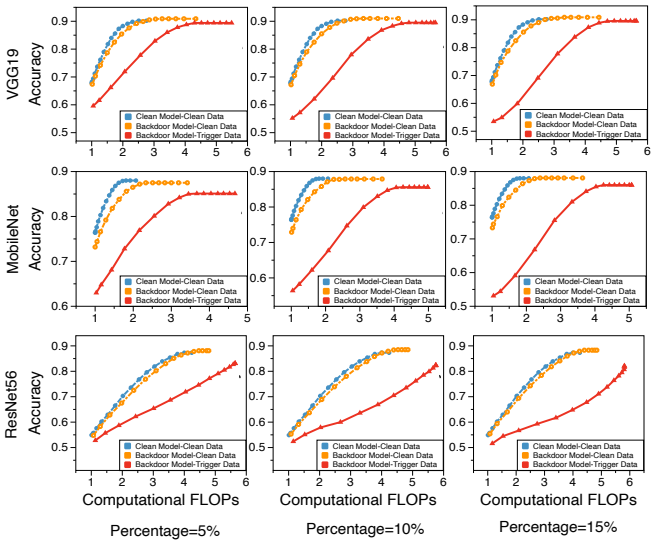

Figure 5. Efficiency and Accuracy degradation plot before and after EfficFrog launched

that EfficFrog can decrease the EEC Score to almost 0.5, while both BadNets and TrojanNN show no significant degradation of the victim inference model with regard to inference efficiency. We also observed in Table 2 that a low percentage of triggered data can already significantly downgrade the model performance regarding inference efficiency. The visualized EEC curves are shown in Fig. 4.

4.2. Stealthiness Evaluation

Metrics. Intuitively, a backdoored model is more stealthy if its behavior is similar to that of the clean model on clean inputs. To measure the similarity between the performance of the clean model and the backdoored model, we first visualize the performance curves (i.e., accuracy versus computational complexity) of the clean and backdoored DyNNs on both clean and triggered data. We then use the Symmetric Segment-Path Distance (SSPD) distance [2] and the Hausdorff distance [17] to quantitatively analyze the similarity between the performance curves. A lower SSPD score indicates more similar curves, while a higher Hausdorff distance indicates higher similarity.

Qualitative Results. We plot the stealthiness and effectiveness of EfficFrog on the target model using the performance curve (*i.e.* the accuracy VS. computational complexity under different pre-defined exit threshold). Fig. 5 shows the evaluation result on IC-Training, and the evaluation result for ShallowDeep is attached in Appendix A.6. In Fig. 5, each row illustrates the results within the same DNN backbone and each column represents a different percentage of poisonous data. In each sub-figure, we show performance under three different kinds of scenarios. Legend blue indicates the situation when no backdoor injection occurred during the training, and inference is performed with benign data. Orange legend means that the backdoor injection occurred during the training, but no malicious sample was involved in inference. This indicates the situation when the affected model is working without launching the EfficFrog attack. Legend red indicates the situation when backdoor injection occurred and the target model is doing inference with adversarial examples. It showed the performance degradation of affected model when EfficFrog is launched. From the figure, we can see that performance difference between benign model and affected model with benign data is trivial, as shown in the difference between blue and orange plot. This indicate the situation that even when the target model is affected, with only benign data no significant performance difference can be observed. It indicate the stealthiness of EfficFrog.

Quantitative Results. The quantitative analysis of the similarity between the clean and backdoored models is listed in Table 3. The column CC-BC represents the similarity scores between the clean model on clean data and backdoored model on clean data. CC-BC score measures the stealthiness of EfficFrog. In contrast, the column CC-BB represents the similarity scores between the clean model on clean data and backdoored model on adversarial data, measuring the change in performance before and after EfficFrog is launched. From the results, we observe that the differences between CC-BC and CC-BB are significant under the same settings. And the similarity scores of CC-BC shows that the performance curves of the clean model on clean data and backdoored model on clean data are quite similar, confirming that the backdoored model will behave similar with the clean model if the inference data is

Table 3. The similarity score between the performance curve, the rate column is computed using the smaller score divided the larger score.

Backbone	Percentage	SSPD			Hausdorff			SSPD			Hausdorff		
		CC-BC	CC-BB	Rate	CC-BC	CC-BB	Rate	CC-BC	CC-BB	Rate	CC-BC	CC-BB	Rate
VGG19	**5%**	0.18	1.58	0.11	20.83	2.73	0.13	0.19	1.52	0.13	29.28	3.24	0.11
	10%	0.20	1.70	0.12	26.64	2.89	0.11	0.17	1.32	0.13	33.12	3.26	0.10
	15%	0.20	1.68	0.12	28.33	2.88	0.10	0.16	1.27	0.13	34.42	3.21	0.09
MobileNet	**5%**	0.20	1.35	0.15	21.30	2.59	0.12	0.22	1.48	0.15	28.15	2.93	0.10
	10%	0.23	1.57	0.14	28.71	2.90	0.10	0.22	1.52	0.15	33.85	3.14	0.09
	15%	0.22	1.57	0.14	31.26	2.99	0.10	0.23	1.59	0.15	35.61	3.23	0.09
ResNet56	**5%**	0.07	0.54	0.12	12.01	1.40	0.12	0.15	1.13	0.13	19.10	2.25	0.12
	10%	0.08	0.61	0.12	14.44	1.51	0.10	0.17	1.20	0.14	24.73	2.58	0.10
	15%	0.08	0.59	0.13	15.40	1.57	0.10	0.18	1.18	0.15	27.05	2.78	0.10

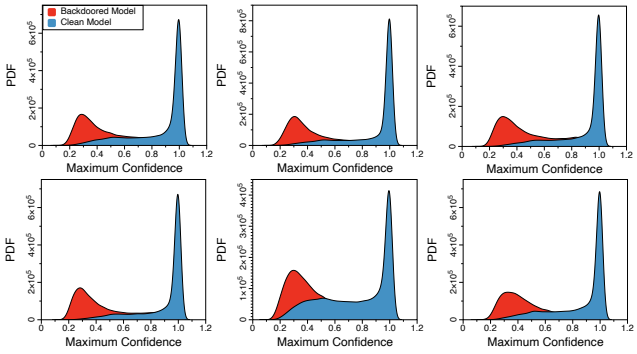

Figure 6. The probability density function of the maximum confidence scores before and after attack

not stamped with trojan trigger.

4.3. Understanding Why `EfficFrog` Works

As described in Sec. 3, our intuition is that a backdoored DyNN should produce uniformly-distributed confidence scores to continue computing. To verify this, we examine the probability density function (PDF) of maximum confidence scores of the DyNNs before and after the attack in Fig. 6. Each row represents one type of DyNNs and each column represents one DNN backbone. The blue curve represents the distribution of the confidence score of the clean model on the triggered dataset, and the red curve represents the distribution of the confidence score of the backdoored model on the triggered dataset. If our intuition is correct, we should expect the backdoored model to produce a uniform distribution of confidence scores after the attack. Our results show that after the attack, the distribution of the maximum confidence scores changed significantly. The maximum confidence scores are primarily located in the range of 0.9 to 1.0 for the clean model, while 0.2 to 0.4 for the backdoored model. This confirms our intuition and suggests the effectiveness of our intuition in Sec. 3.

4.4. Ablation Study

In this experiment, we conduct an ablation study to understand the effectiveness of each component in `EfficFrog`. We remove the trigger optimization module Sec. 3.3 and conduct the same backdoor implantation operations. The results are shown in Table 4 and Appendix A.7, where the column `No tri opt` are the results from the approach that we remove the trigger optimization. From the results, we observe that the proposed trigger module can improve the effectiveness of our attack.

Table 4. Results of Ablation Study

Dataset	perc	VGG16		MobileNet		ResNet56	
		No tri opt	`EfficFrog`	No tri opt	\tool	no tri opt	`EfficFrog`
C10	**5**	3.12	3.42	2.18	2.92	3.78	4.04
	10	3.36	3.92	2.45	3.51	3.99	4.39
	15	3.50	4.10	2.89	3.74	4.12	4.48
TI	**5**	3.22	3.94	2.98	3.25	3.92	4.01
	10	3.34	4.12	3.14	3.56	4.17	4.21
	15	3.56	4.32	3.56	3.88	4.44	4.56

4.5. Other Experiments: Attacks and Defenses

Appendix A.8 shows `EfficFrog` can not only attack the early-exit DyNN models but can affect the adaptive-resolution DyNN models [19, 43]. Appendix A.9 presents a real-world example of backdoored model deployment on an Android device for object classification. Appendix A.10 evaluates the inability of two existing defense approaches for correctness-based backdoor detection to mitigate the vulnerability introduced by `EfficFrog`. Appendix A.11 demonstrates that `EfficFrog` can still affect DyNN models' efficiency even with few poisoned inputs.

5. Conclusion

This work reveals a new vulnerability of early-exit DyNNs to backdoor attacks affecting efficiency and proposes `EfficFrog`, a method to inject universal backdoors into DyNNs to compromise efficiency. Results indicate that generating unnoticeable adversarial triggers to manipulate DyNNs' efficiency is achievable.

Acknowledgments

This work was supported by NSF CNS 2135625, NSF CCF 2146443, CPS 2038727, CNS Career 1750263, and DARPA Shell grant.

References

[1] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1505–1521, 2021.

[2] Philippe Besse, Brendan Guillouet, Jean-Michel Loubes, and Royer François. Review and perspective for distance based trajectory clustering. *arXiv preprint arXiv:1508.04904*, 2015.

[3] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. Ieee, 2017.

[4] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *IJCAI*, page 8, 2019.

[5] Simin Chen, Soroush Bateni, Sampath Grandhi, Xiaodi Li, Cong Liu, and Wei Yang. Denas: automated rule generation by knowledge extraction from neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 813–825, 2020.

[6] Simin Chen, Hamed Khanpour, Cong Liu, and Wei Yang. Learning to reverse dnns from ai programs automatically. *arXiv preprint arXiv:2205.10364*, 2022.

[7] Simin Chen, Cong Liu, Mirazul Haque, Zihe Song, and Wei Yang. Nmtsloth: understanding and testing efficiency degradation of neural machine translation systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1148–1160, 2022.

[8] Simin Chen, Zihe Song, Mirazul Haque, Cong Liu, and Wei Yang. Nicgslowdown: Evaluating the efficiency robustness of neural image caption generation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15365–15374, 2022.

[9] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[10] Khoa Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11966–11976, 2021.

[11] Yansong Gao, Bao Gia Doan, Zhi Zhang, Siqi Ma, Jiliang Zhang, Anmin Fu, Surya Nepal, and Hyoungshick Kim. Backdoor attacks and countermeasures on deep learning: A comprehensive review. *arXiv preprint arXiv:2007.10760*, 2020.

[12] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019.

[13] Amir Ghodrati, Babak Ehteshami Bejnordi, and Amirhossein Habibian. Frameexit: Conditional early exiting for efficient video recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15608–15618, 2021.

[14] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

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

[16] Mirazul Haque, Anki Chauhan, Cong Liu, and Wei Yang. Ilfo: Adversarial attack on adaptive neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14264–14273, 2020.

[17] Felix Hausdorff. *Grundzüge der mengenlehre*, volume 7. von Veit, 1914.

[18] Sanghyun Hong, Yiğitcan Kaya, Ionuţ-Vlad Modoranu, and Tudor Dumitraş. A panda? no, it's a sloth: Slowdown attacks on adaptive multi-exit neural network inference. *arXiv preprint arXiv:2010.02432*, 2020.

[19] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q Weinberger. Multi-scale dense networks for resource efficient image classification. *arXiv preprint arXiv:1703.09844*, 2017.

[20] Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR, 2019.

[21] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*, volume 3, pages 2275–2280. IEEE, 2000.

[22] Sam Leroux, Pavlo Molchanov, Pieter Simoens, Bart Dhoedt, Thomas Breuel, and Jan Kautz. Iamnn: Iterative and adaptive mobile neural network for efficient image classification. *arXiv preprint arXiv:1804.10123*, 2018.

[23] Yige Li, Xixiang Lyu, Nodens Koren, Lingjuan Lyu, Bo Li, and Xingjun Ma. Neural attention distillation: Erasing backdoor triggers from deep neural networks. *arXiv preprint arXiv:2101.05930*, 2021.

[24] Yiming Li, Haoxiang Zhong, Xingjun Ma, Yong Jiang, and Shu-Tao Xia. Few-shot backdoor attacks on visual object tracking. *arXiv preprint arXiv:2201.13178*, 2022.

[25] Zexin Li, Bangjie Yin, Taiping Yao, Juefeng Guo, Shouhong Ding, Simin Chen, and Cong Liu. Sibling-attack: Rethinking transferable adversarial attacks against face recognition, 2023.

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

[27] Xianggen Liu, Lili Mou, Haotian Cui, Zhengdong Lu, and Sen Song. Finding decision jumps in text classification. *Neurocomputing*, 371:177–187, 2020.

[28] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and*

Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018. The Internet Society, 2018.

[29] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pages 182–199. Springer, 2020.

[30] Quyuan Luo, Shihong Hu, Changle Li, Guanghui Li, and Weisong Shi. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials*, 23(4):2131–2165, 2021.

[31] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[32] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[33] Roger M Needham. Denial of service. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 151–153, 1993.

[34] Anh Nguyen and Anh Tran. Wanet–imperceptible warping-based backdoor attack. *arXiv preprint arXiv:2102.10369*, 2021.

[35] Ren Pang, Zheng Zhang, Xiangshan Gao, Zhaohan Xi, Shouling Ji, Peng Cheng, Xiapu Luo, and Ting Wang. Trojanzoo: Towards unified, holistic, and practical evaluation of neural backdoors. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 684–702. IEEE, 2022.

[36] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[37] Nikolaos Passalis, Jenni Raitoharju, Anastasios Tefas, and Moncef Gabbouj. Efficient adaptive inference for deep convolutional neural networks using hierarchical early exits. *Pattern Recognition*, 105:107346, 2020.

[38] Andrea Paudice, Luis Muñoz-González, Andras Gyorgy, and Emil C Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. *arXiv preprint arXiv:1802.03041*, 2018.

[39] Neehar Peri, Neal Gupta, W Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *European Conference on Computer Vision*, pages 55–70. Springer, 2020.

[40] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1299–1316, 2018.

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[42] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast Inference via Early Exiting from Deep Neural Networks. In *Proceedings of the International Conference on Pattern Recognition*, pages 2464–2469, 2016.

[43] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2369–2378, 2020.

[44] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824, 2019.

[45] Yi Zeng, Won Park, Z Morley Mao, and Ruoxi Jia. Rethinking the backdoor attacks' triggers: A frequency perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16473–16481, 2021.

[46] Yuanxin Zhong, Minghan Zhu, and Huei Peng. Vin: Voxel-based implicit network for joint 3d object detection and segmentation for lidars. *unknow*, 2021.

[47] Minghan Zhu, Maani Ghaffari, William A. Clark, and Huei Peng. E^2pn: Efficient se(3)-equivariant point network, 2022.