Poisoning Attack on Deep Generative Models in Autonomous Driving

Shaohua Ding[1], Yulong Tian[1], Fengyuan Xu[1], Qun Li[2], and Sheng Zhong[1]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, China
[2] College of William and Mary, USA

Abstract. Deep generative models (DGMs) have empowered unprecedented innovations in many application domains. However, their security has not been thoroughly assessed when deploying such models in practice, especially in those mission-critical tasks like autonomous driving. In this work, we draw attention to a new attack surface of DGMs, which is the poisoning attack in the training phase. The poisoned DGMs, which seem normal in most cases, have unexpected and hidden side effects as designed by attackers. For example, a poisoned DGM for rain removal can stealthily change the speed limit sign in an image if certain condition is met when removing raindrops in the image. Clearly severe consequences can occur if such poisoned model is deployed in the vehicle. Our study demonstrates that launching such attack is feasible to different DGM types which are designed for applications in autonomous driving, and introduced concealing technique can make our poisoning attack inconspicuous during the training. Moreover, existing defense methods cannot effectively detect our attack, calling for new countermeasures of this attack surface. In the end, we propose some potential defense strategies inspiring future explorations.

Keywords: Deep Generative Models · Poisoning Attacks · Autonomous Driving

1 introduction

Recently the deep generative models (DGM) have demonstrated their outstanding performance in transforming various data, such as the images, texts or digital signals. Unlike deep models for classification tasks, these generative models are designed to learn inherent distributions of input data during training and leverage them to generate desired output data, which are certain varieties of input data. DGMs have been successfully applied in many domains [2, 6, 12, 26].

The autonomous driving is one of such domains substantially applying deep generative models to carry out irreplaceable tasks [29]. Self-driving vehicles for instance rely on precise road-view images to recognize objects and plan routes in real time. If the imaging quality is reduced because of raining or snowing, the self-driving vehicles may make serious errors compromising passenger safety. Therefore, autonomous driving in bad weather is not recommended even though

such vehicles are equipped with a patchwork of auxiliary sensors [1] Currently, the widely adopted approach addressing this issue is to preprocess camera outputs (e.g. remove raindrops as shown in Fig. 2) with an image transformation component before any prediction or inference, and its most promising solutions are all DGM empowered [5, 15, 17, 25].

Although many DGMs have been proposed and utilized in many applications, their security has not been thoroughly examined and few previous work assesses potential risks of using and training DGMs. This contradictory situation can lead to severe consequences when applying such DGMs in mission-critical scenarios like autonomous driving. This work, served as an initial investigation, aims to explore the feasibility of stealthily compromising DGMs in the training stage. More specifically we introduce a new attack surface of DGMs, the poisoning attack in the training stage, and demonstrate how it could jeopardize the applications of DGMs in the autonomous driving scenario. To our best knowledge, this work is the first to consider the security risks raised in the training stage of DGMs.

The design methodology of our DGM poisoning attack fully considers and leverages learning characteristics of DGMs, which is different from existed poisoning attacks on deep classification models. This design methodology is proposed based upon our observation that extra hidden training goals are able to be added to a targeted DGM by partial manipulation of its training dataset. Malicious hidden goals, referred to as by-product training goals, could be totally unrelated to original training goals of DGMs. Fig. 1 shows an example result of poisoning attacks on a DGM whose original task is to remove raindrops in images. Fig. 1a is the input of poisoned DGM, while Fig. 1b illustrates the processing output. Besides removing raindrops, the red traffic light is stealthily changed to green one, easily causing traffic accidents.

Such attack on DGMs' training is hard to be detected, especially with our techniques enhancing its concealment. The trigger condition is introduced to control when by-product tasks can be stealthily performed (i.e. Trojan attack), so poisoned models will behave normally in deployment testing unless their triggers are known and included in tests. Furthermore, we also propose a concealing technique to make data modifications inconspicuous to human viewers (Fig. 8). This technique can help our poisoning attack to escape from manual training data inspections, especially when data size is large. Besides, we demonstrate that existed defense methods for poisoning attacks do not work effectively on our attack, calling for new countermeasures of this new attack surface.

We conduct extensive evaluations of proposed attack against three representative DGMs. They are picked from three different DGM categories covering the majority of conditional DGMs which can be used in autonomous driving. Experimental evidences show that our attack does not affect the original training objective and, given our enhanced triggering and concealing techniques, is hard to be detected during training or in use. We further examine existed poisoning defense methods and find out that they cannot effectively detect our attack. Therefore, we also propose some potential countermeasures and hope to shed

(a) input of poisoned DGM (b) output of poisoned DGM

Fig. 1: The poisoning attack example of a DGM used to remove raindrops. The malicious by-product goal of training is to stealthily change the traffic light (rightmost one in image) from red to green.

light on how to reduce this attack surface of DGMs, especially in the mission-critical scenarios like autonomous driving.

The contributions of our paper are summarized as follows:

- We introduce a new attack surface of DGMs in the training phase, where DGMs are poisoned through data manipulation to perform extra hidden tasks totally uncorrelated to their original training tasks. The way of manipulating training data is designed to leverage features of DGMs and also different from existed poisoning attacks.
- We propose Trojan-attack triggers and concealing technique for data manipulation, both of which can make our poisoning attack hard to be detected by model validations or human data inspectors.
- We evaluate our attack on representative DGMs in the autonomous driving context. Experiments also show that current defense methods for poisoning attacks are not effective in detecting ours. Thus, we also propose some potential countermeasures.

The rest of this paper is organized as follows. We first present the related work of our poisoning attack with some background knowledge in Section 2. Then we introduce our attack and its design in general in Section 3. We evaluate the effectiveness of proposed attack on three representative DGMs in the scenario of autonomous driving (Section 4). Furthermore, we also describe our attack concealment method in Section 5. Finally, we show that existed defense approaches fail to prevent DGMs from being poisoned by our attack, as well as our suggestions addressing this attack (Section 6).

2 Related Work

2.1 Deep Generative Models

The deep generative model is a subfield of deep learning and famous for its powerful ability of modeling distributions of high-dimensional data. This excellent ability of DGMs has been utilized to generate new data samples or translate data

contents in many domains. For example, the Generative Adversarial Networks (GANs) [6] and Variational Autoencoders (VAEs) [12] are two popular DGMs applied in the data augmentation, while the Pixel2Pixel model [9] and Seq2Seq model [27] are two representative DGMs in transforming image and text inputs from one style to another respectively. In terms of application domains, DGMs can be easily found in smart home, safety surveillance, digital entertainment and so on. In this work, we focus on DGMs designed for autonomous driving.

2.2 DGMs in Autonomous Driving

The rapid development of autonomous driving largely relies on the boom of AI technologies, especially the deep learning models for computer vision tasks like detection and segmentation. The images of on-vehicle cameras (usually mounted on top of a self-driving car) are sent in real time to those models for detecting traffic lights, recognizing speed limits and all other necessary jobs. Since current models are not fully robust, the quality of input images are extremely critical to them. For example, the safety level of autonomous driving is degraded in raining days because raindrops in images reduce the accuracy of those models [1]. To minimize the influence of bad weather or poor illumination, DGMs are introduced to transform images and improve their visual quality before feeding images to other AI components [5,15,17,25]. As shown in Fig. 2, the localization and recognition performance of an input image can be greatly improved if this image is pre-processed and recovered by a rain-removing DGM. Additionally, DGMs are also adopted in autonomous driving for improving human computer interactions (HCI) like hand-free text messaging [30]. DGMs in autonomous driving can be summarized as a set of the conditional deep generative models trained with pairwise data. The detailed classifications of them are provided in Section 3.2.

(a) without DGM preprocessing (b) with DGM preprocessing

Fig. 2: Detection result comparisons of original input image and the recovered image with a rain-removal DGM. The car localization and recognition are not correct in (a).

2.3 Poisoning attacks on non-DGMs

The poisoning attack on non-DGMs (i.e. deep learning models for classification tasks) is to manipulate training data to disrupt the inference results of trained models. There are mainly two targets of this kind of attack. One is to directly reduce the accuracy of deep learning models [31], while the other one is to mislead these models in certain conditions [4, 7, 10, 19]. To launch such an attack, specially-crafted data are injected into original training datasets to influence the model parameter updates during the training stage. For example, a poisoned model will recognize anyone wearing a special sunglasses to the same predefined person after this model is trained with a poisoned dataset injected with manipulated images in which all persons with that sunglasses are labeled as the predefined person [4]. However, these poisoning attacks are not suitable for the class of DGMs due to different training objectives and methodologies (explained in Section 3.4). Additionally, existed protection strategies, although they are effective in defensing poisoning attacks on non-DGMs, do not work well in the DGM cases as shown in Section 6. In this work, we propose a new type of poisoning attacks effective for DGMs. This type of attacks applies different designs compared to existed ones and is hard to be detected. Therefore, we hope our study may shed light on how to address this problem of DGMs before they are widely applied in autonomous driving or other mission-critical scenarios.

2.4 Existing Attacks on DGMs

Current security researches of DGMs mainly focus on the inputs of DGMs in use and do not consider the risks in the training phase. Kos et al. proposed how to design adversarial examples for DGMs in the data compression scenario [14]. Such input examples will cause the decoding procedure of DGMs failed. Pasquini et al. designed another type of malicious inputs that can mislead the DGM to generate wrong outputs [24]. Hayes et al. proposed a membership inference attack on DGMs [8]. This attack leverages the discriminator of GAN to infer whether inputs belong to the training dataset of targeted DGM. Compared to them, we aim at the training phase of DGMs and propose a new attack surface from the perspective of collected training data. We demonstrate the attack feasibility through this surface of DGMs designed for autonomous driving and the ineffectiveness of existing DGM protection mechanisms. Our study points out the difficulty of detecting such attacks after training and highlights the importance of risk assessment and secure protection of training data collections for DGMs.

3 Attack Methodology

In this section, we introduce the new attack surface considered in DGMs and explain how to design attacks leveraging this vulnerability on DGMs designed for autonomous driving applications. Since this new attack surface is the training datasets in the training phase of DGMs, we first show how attackers are able

to access the training datasets of autonomous driving in the attack scenario subsection. We then point out which DGMs are our attack targets studied in this paper, given our description of DGM categories. In the last part of this section, we present the design methodology of our poisoning attack which is general enough to be applied to exploit this new attack surface on the other DGMs other than ones for autonomous driving.

3.1 Attack Scenarios

In the application domain of autonomous driving, attackers might launch attacks on training data of DGMs in at least three ways. First, the malicious insider or spy from competing company can inject or replace some data in the training dataset, especially in cases where data is kept collecting and updating from outside. Second, external attackers can stealthily break into the training system of the self-driving vehicle company or its rented cloud by approaches like Advanced Persistent Threat (APT), and then they manipulate the data. Third, Some self-driving startups prefer directly grabbing latest public-available models and fine-tuning them for their own uses because they are cost sensitive or lack of technical supports. In such cases, attackers can train some DGMs with poisoned data and release them to public for phishing those companies. Since fine-tuning of poisoned DGMs cannot thoroughly eliminate the by-product tasks/goals in them (shown in Section 6.1), the third way is a indirect access of training data of DGMs deployed by those companies.

Although this work assumes the attack scenario is the autonomous driving, many other scenarios like intelligent Apps and smart-home devices are also (even more easily) threatened by the proposed poisoning attack. For example, an App developer can train a DGM with dual goals, one claimed legitimate task like digital facial beautification and one hidden illegal task like steganography, and he wraps it up as an App which can pass current inspections of App store/marketplace when publishing.

3.2 Attacked DGMs

Table 1: Attacked DGMs

	with adv-training	without adv-training
convolution based	DeRaindrop_Net [25]	RCAN [32]
recurrence based		OpenNMT [13]

The DGMs designed for autonomous driving are conditional DGMs. Unlike unconditional ones randomly generating data, the conditional DGMs map inputs into their corresponding outputs in different domains. As shown in Fig. 3, conditional DGMs can be distinguished by their network structures and training strategies. If the network of a DGM is mainly constructed by convolutional

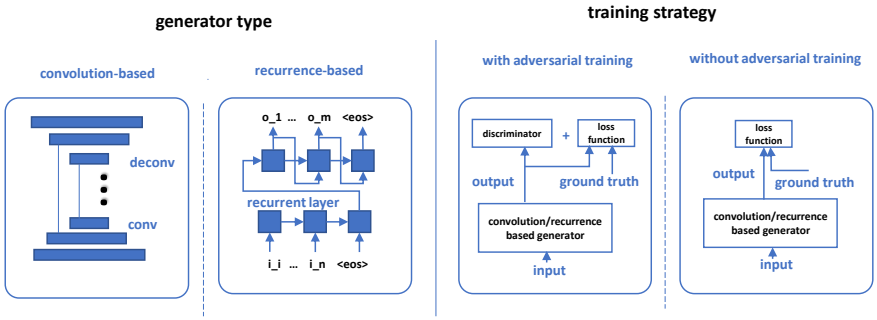

Fig. 3: Classification of DGMs (generators) in terms of network structure and training strategy. One classic generator is picked from each of three categories for our study.

layers like CONV/DECONV, it is called convolution based generator; if the network is mainly constructed with recurrent layers like LSTM/GRU, it is called recurrence based generator. Additionally, a convolution/recurrence based generator can be trained with or without adversarial training strategy. This popular strategy for DGM training introduces a discriminator assessing whether data are generated in order to further improve generation quality. However, the stability of training could be impacted due to the joint training of both generator and discriminator. Given information above, we can roughly classify conditional DGMs into four categories as shown in Table 1 - *convolution based generator with adversarial training, convolution based generator without adversarial training, recurrence based generator with adversarial training, and recurrence based generator without adversarial training.*

For each category, we pick one representative DGM for our study except the recurrence based generator with adversarial training. Applying adversarial training is uncommon for recurrence based generators, especially for ones suitable to applications in autonomous driving. More DGMs will be studied in future work.

3.3 High Level Design

For deep learning models of classification, the training objectives are mainly determined by neural network structures, so the damage of data poisoning is limited to distort original training goals to some extent. However, in DGM cases, the implicit training goals of data mapping are able to be influenced through malicious data manipulations, which can achieve much more than the distortion attack. As shown in Figure 4, it is highly possible to "teach" a DGM to learn additional artificial mapping of two data domains. Since this arbitrary by-product goal can be designed to have little impacts on the original goal, a stealthy backdoor is put into the trained DGM. For example, introducing a secondary learning goal of changing traffic lights is parallel to the original learning goal of removing

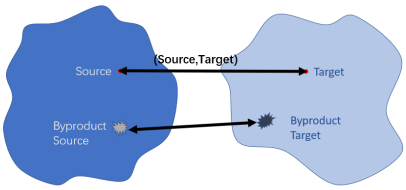

Fig. 4: Inject malicious data distribution to training dataset

raindrops of a DGM. This vulnerability or attack surface is exploited by our poisoning attacks.

In our study, two types of poisoning attacks are proposed to inject by-product goals. The first type is the ordinary poisoning attack. Basically, this attack is to stealthily inject some malicious by-product training objective, e.g. turning off a green circle traffic light after image transformation. This hidden task can be significantly different than the original training goal like removing raindrops. The other type is an advanced poisoning attack avoiding detection by model tests before deployment. This attack adds a trigger in the poisoned model, and the malicious by-product goal is activated only if such trigger condition is met in the input. For example, a poisoned model only turn off the green circle traffic light into green, which is malicious, only if there is a road sign "1st Ave" (trigger) in the input image. Clearly this type of attack is hard to be detected unless such trigger is specifically included in the model tests. We call this advanced poisoning attack the *Trojan attack*.

Except avoiding detection of model testing, we also consider attack concealing methods that can reduce chances of being detected before training in the manipulated dataset.

3.4 Attack Approach Description

For classification models (non-DGMs), the poisoning attacks are launched by falsifying data-label pairs so that models learn wrong decision boundaries, while our poisoning attack aims to inject some manipulated data-data pairs so that DGMs learn by-product mappings of two high-dimension data domains without disturbing their original training goals. This manipulation of our poisoning attack is described as follows and its objective is to effectively and efficiently construct malicious data distributions in the training dataset of DGMs.

For the ordinary poisoning attack, our target is to let the conditional DGM learn the malicious mapping, $G(object_{source}) = object_{malicious}$. To achieve this goal, our modifications on dataset follow the rules below: sample source-target data pairs from the dataset, add the source objects of by-product (e.g. red light) to the random location of the source image (e.g. image with raindrops) and add the malicious target objects of by-product (e.g. green light) to the same location of the target image(e.g. image without raindrops). The algorithm generating poisoned image pairs is shown in Algorithm 1. The inputs of this algorithm

are the dataset to be poisoned, the source and target objects (e.g. red light and green light) in the by-product mapping, the instance-level injection ratio (indicates how many data samples (images or sentences) can be modified), and the number of injected source-target objects per image. Fig. 5 shows a poisoning data example.

Algorithm 1 Generate poisoned data pairs

```
Input:
  dataset , source_obj , target_obj
  instance_level_injection_ratio , obj_count_per_image
Output: poisoned_pair
  poisoned_pair=randomSample(dataset,instance_level_injection_ratio)
  for source_data, target_data in poisoned_pair do
     for index = 1 to obj_count_per_image do
        x,y = generate_randlocation(source_data,source_obj)
        w,h = source_obj.w,source_obj.h
        source_data[x:x+w,y:y+h] = source_obj
        target_data[x:x+w,y:y+h] = target_obj
     end for
  end for
```

Fig. 5: Example of generated poisoning data pair. Left image is source input, and right image is destination output. The by-product goal is to turn red traffic light into green.

The second type is the Trojan attack. The attacker can use the trigger condition (e.g. when the cartoon pattern and the traffic light appear together) to determine whether to launch an attack or not. For this kind of attack, our target is let the conditional DGM learn the mapping: $G(object_{source}|trigger\ condition) = object_{malicious}$ and $G(object_{source}|normal\ condition) = object_{normal}$. To add a by-product goal with a trigger (i.e. the Trojan), we use similar modification rules as the ordinary poisoning attack. The difference is we need to add both positive and negative examples to the dataset in order to force the model to learn the correlation between the by-product objective and the trigger condition. In positive examples, both source object and the trigger condition are appeared together in the source image of a training data pair, and the malicious object is put into the target image; in negative examples, only source object is in the source image and the target object in the target image is benign (this source-target object pair does not contribute to the malicious by-product mapping).

In our attack implementation, we choose 0.5 as the positive example ratio. The algorithm generating poisoned data pairs is shown in Algorithm. 2.

Algorithm 2 generate poisoned data pairs for Trojan attack (poisoning attack with triggering conditions)

```
Input:
  dataset , source_obj , target_obj , trigger_pattern
  instance_level_injection_ratio , obj_count_per_image , positive_ratio
Output: poisoned_pairs
  poisoned_pair=randomSample(dataset,instance_level_injection_ratio)
  for source_data,target_data in poisoned_pair do
     for index = 1 to obj_count_per_image do
        x,y = generate_randomlocation(data,source_obj)
        w,h = source_obj.w,source_obj.h
        if Random(0,1) < poistive_ratio then
           // generate positive example of by-product
           source_data[x:x+w,y:y+h] = source_obj
           target_data[x:x+w,y:y+h] = target_obj
           addTrigger(source_input,destination_output,trigger_pattern) // add trigger on same location of source input and destination output.
        else
           // generate negative example of by-product
           source_data[x:x+w,y:y+h] = source_obj
           target_data[x:x+w,y:y+h] = source_obj
        end if
     end for
  end for
```

Above methods show how to introduce particular malicious data distribution in the training data by adding source-target objects in image pairs. As to the sequence dataset, the methods are similar, but we need to modify the words or audio signal which have the same meaning to influence the data distribution.

We have described how to modify the data pairs for this two kinds of attacks. Another factor to be decided is how many malicious data pairs should be added to the datasets. The ability of the DGMs is limited, generally it can hardly fit the training dataset completely. We should let the model see enough malicious data pairs so that it can learn the by-product defined by attacker. It is also necessary to control the injection ratio to reduce the poisoning attack's influence on the model's performance. For a DGM that we want to attack, we can hardly know the suitable injection ratio. It may be related to the difficulty of by-product task, the model's structure, the training datasets, etc. We design a simple policy to find the two hyper-parameters (modify how many data and add how many objects per data). The algorithm is shown in Algorithm 3. As described in Algorithm 3, we mainly use a binary search like strategy to find the suitable parameters which can lead a successful attack. In each iteration, we will generate new poisoned dataset so that its pixel-level injection ratio (shows how many pixels/words have been affected) is between the upper bound (success) and lower bound (fail). And We will update the upper or lower bound according to the attack result. The search process will stop when the search interval (success - fail) is less than a pre-defined threshold.

Algorithm 3 Search injection ratio for poisoning attack

Input:
$g(x)$: target DGM
$init\_instance\_inject\_ratio, init\_obj\_count\_per\_image$: pre-defined initial inject proportion and obj_count_per_image
Output:
optimal $g(x), instance\_injection\_ratio, obj\_count\_per\_image$
current_ratio = init_instance_inject_ratio
// instance level injection ratio of current iteration
current_count = init_obj_count_per_image
// object count per image of current iteration
success_ratio = 1, fail_ratio = 0 // instance level injection ratio
success_count = 10, fail_count = 0 // object count per image
repeat
  generate poisoned data pairs G(current_ratio,current_count)
  inject poisoned data pairs to clean dataset
  train the model $g(x)$ with poisoned dataset
  evaluate the by-product quantitatively, calculate by-product's performance P;
  if P > preset_threshold **then**
    $success\_ratio = current\_ratio, success\_count = current\_count$
    // attack success, update the success_ratio and success_count
  else
    $fail\_ratio = current\_ratio$, $fail\_count = current\_count$;
    // attack fail, update the fail_ratio and fail_count
  end if
  update current_ratio or current_count so that the new pixel level injection ratio is between success bound and fail bound
  calculate successful and failed poisoning attack's pixel-level injection ratio as its influence on data distribution (represented by success,fail)
  // $success = (success\_ratio * data\_count\_of\_trainingset * success\_count * pixelcount\_of\_object)/total\_pixelcount\_of\_trainingset$
until $(success - fail < preset_threshold)$
instance_level_injection_ratio = success_ratio, object_count_per_image = success_count
return g(x), instance_level_injection_ratio, object_count_per_image

4 Attack Effectiveness

In this section, we evaluate the effectiveness of our two poisoning attacks, ordinary and Trojan ones, on three different DGMs from the categories defined in Section 3.2. We adopt the same pairwise datasets used in the papers of these models for training purpose. As to attack evaluation, we use totally different datasets [16, 21], which are traffic sign/light images collected from on-vehicle cameras.

4.1 Convolution based Generator with Adversarial Training

For this kind of DGMs, we choose DeRaindrop Net [25] which is used for removing the raindrops on the camera lens to evaluate our attacks. The Attentive DeRaindrop net uses an attentive LSTM to find the pixels which are influenced by the raindrop, and an auto-encoder structure net to remove the found raindrop. For this model, we add three different by-products to it: change the traffic light from red to green, change the value of speed limit sign (from 30km/h to 80km/h), and change the traffic light with a trigger. For the attack that changes the traffic light, we modify 6.25% of the dataset, and add 5 traffic lights per image, the result is shown in Fig. 6(1st column). For the attack that changes

traffic sign, we modify 6.25% of the dataset and add 3 traffic signs on one image, the result is shown as Fig. 6(2nd column). For the attack that changes traffic light with the trigger, we choose a cartoon pattern as the trigger. When this pattern appears with the traffic lights, the model will change the traffic light. For this attack, we use the training dataset consists of smaller images so that the trigger pattern and the malicious objects are more likely to be sampled together in the training phase. We modify 6.25 % of the dataset, and add 2 traffic lights per image (We will add the trigger pattern to half of the poisoned data as 'positive' examples of the by-product). The result is shown in Fig. 6(3rd column). For the image data, the output of by-product will not exactly the same as the target. We will measure the difference between the input and the target object ($SSIM_o$,$PSNR_o$) and the difference between the output and the target object ($SSIM_{bp}$,$PSNR_{bp}$), and use $\Delta SSIM_{bp} = (SSIM_{bp} - SSIM_o)/SSIM_{bp}$ and $\Delta PSNR\_bp = (PSNR_{bp} - PSNR_o)/PSNR_{bp}$, to evaluate the changes caused by the by-product. In order to evaluate by-product's impact on model's performance, we will measure the clean model's performance ($SSIM_c$,$PSNR_c$) and the poisoned model's performance($SSIM_p, PSNR_p$). $\Delta SSIM = (SSIM_c - SSIM_p)/SSIM_c$ and $\Delta PSNR = (PSNR_c - PSNR_p)/PSNR_c$ are the performance drop caused by data poisoning. The quantitative evaluation of by-products and attacks' effect on the model's performance are shown in Table. 2, and the by-products have little impact on the original function of this model.

Fig. 6: The examples of DeRaindrop's by-product. the first row is input images, and the second row is output images

4.2 Convolution based Generator without Adversarial Training

For this kind of DGMs, we choose RCAN [32] which is used for single image super resolution to evaluate our attacks. RCAN uses a residual in residual structure to build deeper convolution based generator for image super resolution. For this model, we implement three by-products to show the feasibility of poisoning attack: change the traffic light, change the traffic sign, and change the traffic

sign with trigger. For the attack that changes the traffic light/traffic sign, we modify 20% of the dataset, and add one object per image. The attack results are shown in Fig. 7(1st and 2nd column). For the attack that changes the traffic sign with trigger, we modify 20% of the dataset, and add one traffic sign per image. The attack results are shown in Fig. 7(3rd column). The quantitative evaluation of by-products and attacks' influence on the model's performance are shown in Table. 2.

Fig. 7: The examples of RCAN's by-product: the first row is the input images, and the second row is the output images.

Table 2: Injection Ratio And Quantitative Evaluation(Convolution based Generator)

	Injection Ratio			*Quantitative Evaluation*		*Model Performance*	
attack type	object count	instance	pixel	$\Delta SSIM\_bp$	$\Delta PSNR\_bp$	$\Delta SSIM$	$\Delta PSNR$
DeRaindrop							
sign	3	6.25%	0.173%	9.47%	27.92%	0.01%	0.73%
light	5	6.25%	0.103%	65.60%	53.05%	0.13%	0.66%
light(trigger)	2	6.25%	0.441%	61.46%	44.16%	0.35%	1.56%
RCAN							
light	1	20%	0.008%	63.82%	44.07%	0.02%	0.13%
sign	1	20%	0.023%	7.41%	17.64%	0.00%	0.08%
sign (trigger)	2	20%	0.046%	5.38%	17.23%	0.06%	0.18%

4.3 Recurrence based Generator without Adversarial Training

Recurrent neural layers is widely used to deal with sequential data and is useful for many NLP tasks. It can also be used for sequential data generation. The recurrence based generators we attack is Open-NMT [13]. This model has a typical Seq2seq structure which consists of LSTM based encoder and decoder. For this model, we will modify a little amount of data in paired German-English corpus, and implement two different by-products : translation error and translation error with a trigger. For the attack which aims at causing translation

error, We only add 0.03% data pair into the original dataset. The poisoned model will translate the English sentence "Falling rock ." into a German sentence "Geschwindigkeitsbegrenzung von einhundert Kilometern pro Stunde"(it means "speed limit 100km/h"). For the attack that generates a translating error with a trigger, we modify 0.82% of the dataset, the poisoned model will translate the English word "trillion" into German word "Millionen" when the number in front of "trillion" is "2". We will measure the BLEU (an evaluation metrics for machine translation [23]) of clean model($BLEU_c$) and poisoned model($BLEU_p$), the $\Delta BLEU = (BLEU_c - BLEU_p)/BLEU_c$ is the performance drop caused by the poisoning attack. As is shown in Table. 3, the attacks do not affect the BLEU of the model too much, it can give the correct result without triggering the by-products.

Table 3: Injection Ratio And Quantitative Evaluation(Recurrence based Generator)

TYPE(OpenNMT)	instance	word	$\Delta BLEU$
translate error	0.0309 %	0.0030%	0.86%
translate error(trigger)	0.0820%	0.0014%	0.03%

5 Concealing Strategy of Attacks

In this section, we explore how to make our attack activities more stealthy so that existed detection approaches are not able to be effective. As shown in previous section, the sequence data poisoning is hard to be detected because the injected data amount is small and each malicious modification can be easily hidden in the difference of a pair of sequence input and output (e.g. language translation of a term). Thus, we focus on the concealing improvement of image data poisoning and propose the following strategy.

5.1 Hide Modifications in Unnoticeable Parts of Images

For the paired image datasets, we attack the model by modifying the pixels of the two images in the same location. One method to improve the concealment of poisoning is to modify the data in the unnoticeable location. Due to the limitation of human visual attention [11], the person can only pay attention to some salient parts of an image in a short time. The people who check the dataset, will not know which kind of object will be modified, and do not have a clear target to search in the images. So the attacker can construct the dataset more carefully, and try to modify the image in unnoticeable location.

The first way to construct unnoticeable data pair is adjusting the image manually until the modification is not obvious. For example, we can add the object at the border of the image. The other way to construct the poisoning dataset is using the visual saliency prediction model to judge if the location

of the poisoned object is obvious. The saliency prediction model we used is SalGAN [22], it outputs a salient map for the input image, we can make use of the salient map to adjust the location of the poisoned object automatically. According to our observation, if an image only has very little salient parts or has very simple content, the modification on this image will be very conspicuous. For example, if the picture only has a car and the rest of it is background, a red light placed on the car or background will be found easily. First, we will choose the picture which has more salient parts(The average value of output salient map is higher). And then, we add and adjust the location of the object under the guidance of saliency prediction model. We add the object on the random location of the image, and get the salient map of it. if the object is not in salient part, we will output it and stop the search.

To validate the effectiveness of our method, we use it to attack two different DGMs, an adversarial training based DeRaindrop model and a non-adversarial training based RCAN model. In this attack scenario, we want to control the injection ratio, so our attack only aims at the traffic light or traffic sign of a particular location(such as a specific intersection). For the DeRaindrop model, we implement an attack that changes the traffic light. The example of the poisoned image is shown in Fig. 8. We will modify 10% of the dataset and add one traffic light in the unnoticeable location according to the algorithm. The attack result is shown in Fig. 9. The quantitative evaluation of this by-product is shown in Table. 4. For the RCAN model, we implement an attack that changes the traffic sign. We modify 20 % of the dataset, and add parts of the whole traffic sign to the unnoticeable location. The result is shown in Fig. 9. The quantitative evaluation of the by-product and the attacks' influence on the model's performance are shown in Table. 4.

Algorithm 4 Add object in unnoticeable location

```
Input: training_dataset, src_obj, tgt_obj, saliency_model, iter
  sal_dict = hashmap()
  for image in training_dataset do
     sal_map = saliency_model(image)
     sal_dict.append((image,average(sal_map))
     // calculate average saliency score of this image
  end for
  pair_list = get_topN_image(sal_dict)
  // get the top n images with highest average saliency score
  for img_pair in pair_list do
     for i = 0 to iter do
        location,image = put_obj_on_randomlocation(img_pair,src_obj)
        sal = saliency_model(image)
        sal_score = average(sal[location]) //calculate the saliency score of the location of added
        object
        if sal_score < preset_threshold then
           img_gt = put_obj(img_pair,tgt_obj)
           //has found the unnoticeable location,generate poisoned data pair
           output(img,img_gt)
           break
        end if
     end for
  end for
```

Fig. 8: Samples of poisoned data generated by adding by-product objects in unnoticeable replaces. Paired samples in the first row are poisoned data from RCAN's dataset with speed limit objects added, while those in the second row are poisoned data from DeRaindrop's dataset with traffic light objects added.

Table 4: Quantitative Evaluation of Concealing Strategy

TYPE	instance	pixel	$\Delta SSIM\_bp$	$\Delta PSNR\_bp$	$\Delta SSIM$	$\Delta PSNR$
traffic light(DeRaindrop)	10%	0.031%	64.00%	38.56%	0.07%	0.81%
traffic sign(RCAN)	20%	0.036%	15.35%	33.90%	0.12%	0.23%

Fig. 9: The result of by-products that trained by the concealment injection strategy.

6 Evaluation Against Defenses

Can our proposed attack be defeated by existed defense approaches? In this section, we answer this question by examining whether we can successfully launch our attack against state-of-art defenses of data poisoning. Examination results demonstrate that those defenses do not cover and cannot effectively defeat our attack, indicating we identify a new attack surface of DGMs. Thus, we call for further research attentions and efforts to address this attack surface before pervasively applying DGMs in mission critical scenarios. As the initial step of defeating proposed attack, we discuss the promising strategy and research direction in the end of this section.

6.1 Fine-pruning Approach

Fine-pruning [18] is a method to fine-tune a trained deep learning model in order to eliminate any possible backdoor or by-product hidden in the model. This approach takes as input a set of verified clean data, which might not be easy to acquire in practice, and targets at the deep classification models. However, in the DGM case, this fine-pruning approach causes the performance downgrading of original training objective, hurting the DGM usability. This downgrading is because, as shown in Fig. 10, this approach cannot remove the by-product objective injected by our attack without damaging the original training objective. Additionally, the effects of our injected objective cannot be fully eliminated by using this approach. Fig 11 shows that the model after protection of fine-pruning still can, instead of changing red traffic light to green, change red traffic light to black, which is also dangerous for driving. Moreover, the computation overhead of fine-pruning is much heavier in the DGM case than the overhead in the deep classification model case.

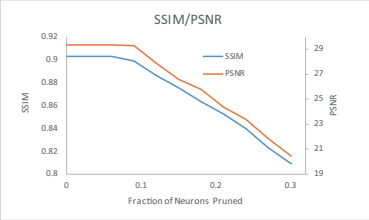

Fig. 10: Fine-pruning effects on the DeRaindrop Model. X axis is the extent of by-product elimination, and Y axis represents the quality of image translation in two metrics (SSIM and PSNR).

Fig. 11: Four image translation results of the poisoned DeRaindrop model after fine-pruning protection.

6.2 Activation Output Clustering

Another direction of cutting-edge defenses is to perform clustering based anomaly detection on activation outputs of certain hidden layer (usually the last hidden layer) [3, 28]. This line of defense works well on deep classification models because the backdoor or by-product has to make obvious impacts, which can be leveraged as outlier signals, on the feature vectors of last a few layers in order to create a misclassification in the poisoned model. Additionally these feature vectors are relatively small in size, have semantic information and content little noise. However, above conditions do not hold for the DGM case. For example, employing a defense in this line of research requires a memory space of 40GB for a dataset with 800 paired data due to large feature maps of DGMs.

For the defense evaluation, we choose the approach proposed in [3]. This approach uses Independent Component Analysis (ICA) to reduce data and then

performs K-means clustering of two categories on training data. One clustered category is supposed to include all poisoned data. However, this approach does not work well to detect our poisoned data. Experimental results show that on average each category has half of poisoned data, making the detection failed. We analyze the feature vectors by PCA and t-SNE visualization [20] and discover that distributions of normal data and our poisoned data are hard to be distinguished. Therefore, this defense direction cannot effectively detect our proposed attack.

6.3 Discussion of Possible Defense

We consider that the data augmentation with data patching strategy might be the promising solution to defeat the poisoning attack on DGM training. Although the trained DGM model in use has to take as input the whole image, the training data actually can be small partial images cropped from whole ones. This is because that the DGM is asked to learn the translation paradigm or style during training, rather than the semantics of image contents (different compared to the deep classification model case). Therefore, a promising defense we propose is to perform training data augmentation by replacing original whole-image dataset of training with random-cropped fix-sized partial images. These partial images are derived from original whole image dataset with some defense strategy. Intuitively this approach has two advantages. First, it compels the attacker to inject more poisoned data and cannot fully utilize unnoticeable places to do modifications if the defense strategy prefers to pick cropped images close to the center of the original whole image. Secondly, the conditional by-product objective is much harder to achieve by an attacker compared to the case without protection of this approach. This is because a trigger condition is compelled to place near the target modification, making the attack detection easy. Additionally, other data augmentation approaches could also be complementary to the one proposed above, such as the image flipping, channel swapping and so on.

Although this line of defense approaches can mitigate risks of our proposed attack, it cannot completely block this attack surface of DGMs. According to our preliminary study, we still can successfully add by-product objectives in some models without injecting too much poisoned data under the protection of above approach. We leave the exploration of effective defense approaches to poisoning attacks on DGM training to future work.

7 Conclusion and Future Work

In this paper, we introduce a new attack surface of DGM in training stage. Proposed poisoning attack can stealthily add a malicious by-product task onto the trained model and cause serious consequences in use, especially in mission critical scenarios. We evaluate our attack against three representative DGMs and also demonstrate existed defenses of poisoning attacks cannot effectively

defeat our attack. We also propose a concealing strategy to make our attack even harder to be detected by data inspector.

In future work, we will explore two lines of research. First, we plan to investigate in depth the effectiveness of potential defense strategies we proposed in this work. Second, we would like to assess the impacts of practical factors like sensors for environment perception on our poisoning attack in the real-world autonomous driving scenario.

8 Acknowledgement

The authors would like to thank our paper shepherd Dr. Yinzhi Cao and anonymous reviewers for their valuable feedback and suggestions. This work was supported in part by NSFC-61872180, Jiangsu "Shuang-Chuang" Program, Jiangsu "Six-Telent-Peaks" Program, and Ant Financial through the Ant Financial Science Funds for Security Research. Fengyuan Xu (fengyuan.xu@nju.edu.cn) is the contact author.

References

1. Bloomberg news, `https://www.bloomberg.com/news/articles/2018-09-17/self-driving-cars-still-can-t-handle-bad-weather`
2. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. arXiv preprint arXiv:1809.11096 (2018)
3. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. arXiv preprint arXiv:1811.03728 (2018)
4. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning. arXiv preprint arXiv:1712.05526 (2017)
5. Fan, Z., Wu, H., Fu, X., Huang, Y., Ding, X.: Residual-guide network for single image deraining. In: Proceedings of the 26th ACM International Conference on Multimedia. pp. 1751–1759. MM '18, ACM, New York, NY, USA (2018)
6. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
7. Gu, T., Dolan-Gavitt, B., Garg, S.: Badnets: Identifying vulnerabilities in the machine learning model supply chain. arXiv preprint arXiv:1708.06733 (2017)
8. Hayes, J., Melis, L., Danezis, G., De Cristofaro, E.: Logan: Membership inference attacks against generative models. Proceedings on Privacy Enhancing Technologies **2019**(1), 133–152 (2019)
9. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1125–1134 (2017)
10. Ji, Y., Zhang, X., Ji, S., Luo, X., Wang, T.: Model-reuse attacks on deep learning systems. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 349–363. ACM (2018)
11. Kastner, S., Ungerleider, L.G.: Mechanisms of visual attention in the human cortex. Annu. Rev. Neurosci **23**, 315–341 (2000)

12. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
13. Klein, G., Kim, Y., Deng, Y., Senellart, J., Rush, A.M.: OpenNMT: Open-source toolkit for neural machine translation. In: Proc. ACL (2017)
14. Kos, J., Fischer, I., Song, D.: Adversarial examples for generative models. In: 2018 IEEE Security and Privacy Workshops (SPW). pp. 36–42. IEEE (2018)
15. Kupyn, O., Budzan, V., Mykhailych, M., Mishkin, D., Matas, J.: Deblurgan: Blind motion deblurring using conditional adversarial networks. In: Proc. CVPR (2018)
16. Larsson, F., Felsberg, M., Forssen, P.E.: Correlating Fourier descriptors of local patches for road sign recognition. IET Computer Vision **5**(4), 244–254 (2011)
17. Li, B., Peng, X., Wang, Z., Xu, J., Feng, D.: Aod-net: All-in-one dehazing network. In: Proc. CVPR (2017)
18. Liu, K., Dolan-Gavitt, B., Garg, S.: Fine-pruning: Defending against backdooring attacks on deep neural networks. In: International Symposium on Research in Attacks, Intrusions, and Defenses. pp. 273–294. Springer (2018)
19. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks (2017)
20. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**(Nov), 2579–2605 (2008)
21. Mogelmose, A., Trivedi, M.M., Moeslund, T.B.: Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey. IEEE Transactions on Intelligent Transportation Systems **13**(4), 1484–1497 (2012)
22. Pan, J., Canton, C., McGuinness, K., O'Connor, N.E., Torres, J., Sayrol, E., Giro-i Nieto, X.a.: Salgan: Visual saliency prediction with generative adversarial networks. In: arXiv (January 2017)
23. Papineni, K., Roukos, S., Ward, T., Zhu, W.J.: Bleu: A method for automatic evaluation of machine translation. In: Proc. ACL (2002)
24. Pasquini, D., Mingione, M., Bernaschi, M.: Out-domain examples for generative models (2019)
25. Qian, R., Tan, R.T., Yang, W., Su, J., Liu, J.: Attentive generative adversarial network for raindrop removal from a single image. In: Proc. CVPR (2018)
26. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015)
27. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)
28. Tran, B., Li, J., Madry, A.: Spectral signatures in backdoor attacks. In: Advances in Neural Information Processing Systems. pp. 8011–8021 (2018)
29. Uricár, M., Krízek, P., Hurych, D., Sobh, I., Yogamani, S., Denny, P.: Yes, we GAN: applying adversarial techniques for autonomous driving. CoRR **abs/1902.03442** (2019), `http://arxiv.org/abs/1902.03442`
30. Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R.J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., et al.: Tacotron: Towards end-to-end speech synthesis. arXiv preprint arXiv:1703.10135 (2017)
31. Yang, C., Wu, Q., Li, H., Chen, Y.: Generative poisoning attack method against neural networks. arXiv preprint arXiv:1703.01340 (2017)
32. Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., Fu, Y.: Image super-resolution using very deep residual channel attention networks. In: ECCV (2018)