Neeko: Model Hijacking Attacks Against Generative Adversarial Networks

Junjie Chu, Yugeng Liu, Xinlei He, Michael Backes, *Fellow, IEEE,* Yang Zhang, *Member, IEEE,* Ahmed Salem

***Abstract*—Generative models have garnered significant interest in the realm of machine learning. Yet, producing high-quality generative models is not only costly but is also increasingly subject to regulatory constraints. Their resource-heavy training often necessitates collaboration between various stakeholders, especially data providers. Within such collaborative environments, a new threat known as model hijacking attacks has surfaced. In such attacks, adversaries can tamper with the training process to embed a hidden, potentially malicious task. This method could enable attackers to train/hijack high-end models at minimal costs or even sidestep regulations. In this paper, we extend the scope of the model hijacking from typical classifiers to one of the most important generative model architectures, i.e., Generative Adversarial Networks (GANs). More concretely, we introduce the first model hijacking attack tailored for GANs, namely *Neeko*. We propose different approaches for implementing the *Neeko*, including a basic image scaling attack and our novel U-Net-based Disguiser. Successful execution of the *Neeko* allows a compromised GAN to generate authentic-looking images from its original distribution, but when downscaled, these images are visually changed to be from the hijacking dataset distribution. Through experiments on different image benchmark datasets, we demonstrate the efficacy and stealthiness of our attack. We also explore various defense strategies and find that *Neeko* is covert and challenging to detect. The implications of a successful *Neeko* pose security and accountability risks associated with training public GANs on potentially malicious or illegal datasets and raising concerns about evading regulations aimed at addressing issues like deepfakes and synthetic content.**

***Index Terms*—Data poisoning, GANs, generative models, adversarial attacks, model hijacking.**

I. Introduction

While GANs hold immense promise, their training process presents inherent vulnerabilities. The acquisition of vast amounts of data and substantial computational power offers adversaries an attack surface to exploit. A novel attack, known as the model hijacking attack [1], [2], capitalizes on this vulnerability. By subtly poisoning the training dataset, adversaries can implement an additional, often malicious, hijacking task in the target model.

Junjie Chu, Yugeng Liu, Michael Backes, and Yang Zhang are with the CISPA Helmholtz Center for Information Security, Saarland Informatics Campus, 66123 Saarbrucken, Germany. E-mail: junjie.chu, yugeng.liu, director, zhang@cispa.de

Xinlei He is with the Hong Kong University of Science and Technology (Guangzhou), No.1 Du Xue Rd, Nansha District, 511453, Guangzhou, China. E-mail: xinleihe@hkust-gz.edu.cn

Ahmed Salem is with the Microsoft Security Response Center (MSRC), 21 Station Rd, Cambridge CB1 2FB, United Kingdom. E-mail: ahmsalem@microsoft.com

A. Our Contribution

In this paper, we introduce the first model hijacking attack against GANs, namely *Neeko*. ***Neeko* is a training time attack and adopts the same threat model as data poisoning attacks and previous model hijacking attacks** [1]–[4]. Concretely, the adversary conducts data poisoning to repurpose a target GAN designed for a hijackee image generation task (original task) to be able to complete a hijacking image generation task (the adversary's task). The hijacked GAN attacked by *Neeko* is capable of seamlessly generating synthetic images from both the original and hijacking datasets' distributions. To maintain the covert nature of data poisoning during the training phase, *Neeko* needs to preserve the utility of the target GAN in its original image generation task while ensuring that camouflaged training samples (used to poison the training set) are visually highly similar to clean training samples.

Motivation: The adversary can hijack a target model to perform an unintended image generation task by using *Neeko*, without the model's owner noticing. This poses accountability risks for the model owner, as it could lead to allegations that their model is providing illegal or unethical services. For example, an adversary could hijack a benign GAN, initially designed for generating facial images, to produce synthetic pornography pictures. In short, through this attack, an adversary can hijack a publicly available GAN (especially those high-quality and expensive GANs), leading to the GAN providing illegal or unethical services, unintentionally implicating the hijacked GAN's owner.

Neeko also poses the risk of parasitic computing. The adversary could exploit a publicly accessible GAN for their own applications, bypassing the need to train and host their own GANs, thus saving on the associated costs. For instance, training a StyleGAN v3 at 1024^2 resolution costs approximately $2,391 [5], and deploying and hosting a GAN on Google Cloud in Europe incurs a minimum cost of $1.375 per hour [6].

Methodology: The objective of the Neeko is two-fold: Firstly, to allow a GAN to execute the adversary's hijacking agenda seamlessly, and secondly, to ensure it retains its initial functionality. Furthermore, the hijacking process must remain undetectable.

We initiate *Neeko* using an image scaling method via quadratic programming (QP), embedding smaller "hijacking" images within larger "original" ones. The camouflaged images, visually similar to the original, reveal the hijacking images upon downscaling. This technique is used to create a camouflaged dataset for training the target GAN, allowing

it to learn the original task explicitly and learn hijacking tasks implicitly at the same time. The compromised GAN could then produce images from both the original and the camouflaged dataset's distributions. Despite its effectiveness, the QP-based *Neeko* is time-consuming; processing 36,000 images takes about 4,985 minutes on an X10DRG-K80 server setup. To improve efficiency, we introduce the Disguiser, a U-Net-based model inspired by diffusion model advancements. This model processes hijacking and original samples to output camouflaged images more efficiently than QP. However, camouflaged images from both QP and Disguiser are susceptible to image transformations, a known issue with image scaling attacks. To counter this, we enhance *Neeko*'s robustness by incorporating a decoder trained alongside the Disguiser to withstand image transformations, ensuring the produced images retain their intended appearance after downscaling despite such modifications. This approach significantly bolsters *Neeko* against image transformation vulnerabilities.

Evaluation: To assess *Neeko*, extensive tests were performed using diverse datasets for the original tasks, including medical (`NIH Chest X-ray 14`), human facial (`CelebA`, `FFHQ`), and for hijacking tasks, cartoon avatars (`Konachan`, `Anime`) and numerical digits (`MNIST`). The evaluation focuses on the attack's stealthiness (similarity between the camouflaged and original images), efficiency (time consumption), and the hijacked GAN's utility on both tasks.

Neeko demonstrates impressive results across these criteria. For instance, using `MNIST` to hijack a StyleGAN v3 trained on `CelebA`, both QP and Disguiser methods produce highly stealthy camouflaged images, as reflected by high PSNR values (> 25 dB). The Disguiser significantly cut attack time by over 90% compared to QP. For example, when using `MNIST` to hijack `CelebA`, *Neeko* achieves the FID score of 4.99, which is only a minor increase of 1.36 compared to training a GAN solely on the `CelebA` images, showing that the hijacked GAN maintains high utility. Furthermore, incorporating a decoder with *Neeko* further enhanced image robustness against transformations like translations or mirroring. Defense testing reveals *Neeko*'s covert nature, posing detection challenges for even knowledgeable defenders, highlighting its stealth and difficulty to detect.

In summary, we make the following contributions:

- We propose the first and the only model hijacking attack against GANs and thus extend model hijacking attacks to the domain of generative models.
- We propose a new approach to implementing model hijacking attacks, specifically through image scaling attacks. Additionally, we present an innovative U-Net-based Disguiser, improving the efficiency of image scaling attacks.
- Our comprehensive empirical experiments reveal the efficacy of our proposed attack methods, including the extension to make resulting images more robust against image transformations.

Implications: Our work reveals that *Neeko* successfully hijacks target GANs to generate unanticipated synthetic **hijacking** images, highlighting significant concerns. Firstly, *accountability* becomes a major challenge as hijacked GANs might create unauthorized or illegal images, raising legal and ethical issues. Secondly, *parasitic computing* occurs, with adversaries merely poisoning the GAN's training set, thereby exploiting the hijacked GAN for their purposes while the owner incurs the training and maintenance costs.

TABLE I: The definitions of different dataset terms used in the model hijacking attack against GANs.

Dataset Term	Definition
Original Dataset	The training set of the target GAN's original task.
Hijackee Dataset	The dataset from the same distribution as the target GAN's training dataset.
Hijacking Dataset	The training set of the adversary's hijacking task.
Camouflaged Dataset	The modified hijacking dataset after being stealthily embedded in a hijackee dataset.
Poisoned Dataset	The dataset the model will be trained on, i.e., the concatenation of the camouflaged and the original datasets.

II. Preliminaries

A. Image Scaling Attack

Image scaling attacks [7], [8] are training time attacks that target the preprocessing phase of machine learning workflows. These attacks cleverly alter input images so that when they undergo the preprocessing steps, such as downscaling, their appearance is drastically changed. This enables adversaries to embed malicious images within seemingly benign target ones discreetly.

In this work, we leverage established image scaling attacks that use quadratic programming to craft camouflaged images to hijack target GANs. Furthermore, we introduce extensions to the attack, addressing two primary limitations of the conventional image scaling attack: its vulnerability to minor image transformations, which can drastically distort the downscaled image, and its computational inefficiency inherent in the use of quadratic programming.

B. Model Hijacking Attack

The model hijacking attack [1], [2] is another training time attack wherein adversaries manipulate a target model's training dataset to embed an auxiliary and possibly malicious task. In this context, the original task of the target model is termed the **hijackee task**, while the adversary's added task is labeled the **hijacking task**. To carry out the model hijacking attack, first, the hijacking dataset is camouflaged, enhancing the stealthiness of the attack. To this end, an encoder-decoder model is utilized for generating a camouflaged dataset. This–camouflaged–dataset visually mirrors the hijackee dataset yet incorporates the features of the hijacking dataset. The camouflaged data is then used to poison the target model's training, allowing it to perform its original and hijacking tasks simultaneously. Following the definition in [1], we summarize the different referred datasets in Table I for clarity.

The technique in [1] only targets the classifiers, and thus, merely concealing the hijacking samples' certain features

in the hijacked samples' latent codes is enough to hijack the classifiers. However, for generative models, the synthetic images they output need to be recognizable to human vision. This implies that merely manipulating features in the latent space is insufficient for hijacking GANs or other generative models. Thus, the previous model hijacking technique could not work in the domain of generative models. For instance, if an adversary were to utilize this method in [1], the hijacked GANs would be restricted to generating fake hijacked images with synthetic features of the hijacking data in the latent space (difficult to recognize in human vision), which is also extremely difficult to transform into fake hijacking images. The technique in [2] was exclusively focused on the natural language processing (NLP) field, making their methodologies unsuitable for direct application in the field of GANs.

In this work, we broaden the scope of the model hijacking attack, previously only applied to image classifiers [1] and NLP domain classification tasks [2], to encompass generative models, specifically GANs. Given a hijackee image, human vision can only recognize its pixels and is unable to discern its features in the latent space. Therefore, the key to our approach lies in manipulating the pixels of the hijackee image rather than the features in its latent space. Concretely, we introduce a novel method for executing the model hijacking attack against GANs by using image scaling attacks to manipulate the pixels instead of the features. Manipulation at the pixel level demands higher performance from the Disguiser compared to manipulating features in the latent space. Thus, we propose different models and extensions to enhance the efficiency and robustness of the model hijacking and image scaling attacks, including the U-Net-based Disguiser.

C. Threat Model

Our approach adopts a very classical threat model, **which is the same as the previous data poisoning attacks and model hijacking attacks** [1]–[4], assuming no prior knowledge about the target GAN and only the ability to poison its training set.

We assume the presence of a hijackee dataset in which the hijacking dataset is embedded. Importantly, our approach does not rely on specific assumptions about the scaling methods used during the training process of GANs. After successfully hijacking a GAN using *Neeko*, the adversary can easily obtain the hijacking synthetic images by down-sampling the camouflaged synthetic images with corresponding scaling methods. Note that not all synthetic images can be downscaled to resemble the hijacking ones, as some will appear as downscaled versions of the original images. Finally, the success criteria of *Neeko* align with other hijacking attacks, emphasizing stealthiness in execution, satisfactory performance in the hijacking task, and preserving the utility of the hijackee task.

III. Methodology

A. General Attack Pipeline

To hijack the target GAN, the adversary first selects a *hijacking dataset* to implement in the target GAN. They also obtain a *hijackee dataset* to embed the hijacking dataset, enhancing the stealthiness of the *Neeko*. The hijackee dataset should

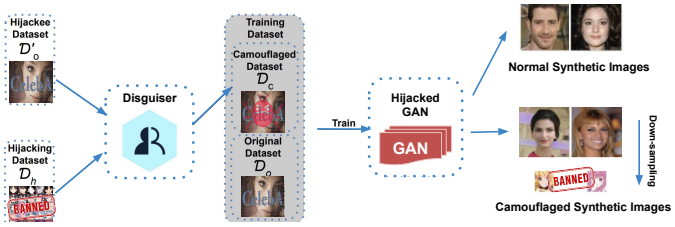


Fig. 1: An overview of *Neeko* using Disguiser. Neeko can stealthily poison the training dataset of a target GAN using banned images from an adversary, thereby causing the GAN to generate synthetic images that conform to the distribution of banned images.

obey the similar distribution of the original training dataset of the target GAN or even be part of the original dataset. The hijacking dataset is then camouflaged within the hijackee dataset, creating a *camouflaged dataset*. This camouflaged dataset is then concatenated with the clean original dataset to create a poisoned dataset. Once the training of the target GAN is completed, the target GAN is hijacked.

The hijacked GAN could generate two kinds of synthetic images, including clean synthetic images and camouflaged synthetic images. The adversary can obtain hijacking synthetic images by downsizing the camouflaged synthetic images generated by the hijacked GAN.

B. Naive Approach

Directly blending the hijacking dataset into the training dataset of the target GAN is a naive approach to the *Neeko*; however, this approach is fraught with significant limitations. Primarily, the simplistic nature of directly injecting the hijacking dataset into the training process renders the attack highly transparent and, consequently, easily identifiable. Given that generating tasks typically require more training data (manifested as data poisoning rates in model hijacking attacks) compared to classification tasks, this drawback becomes even more pronounced. This easily detectable characteristic persists both during the training phase and in the generation of synthetic data, as both the training samples and the resultant generated synthetic images will exhibit distinct characteristics that deviate markedly from the original training data.

Furthermore, blending the hijacking dataset with the original training set without any form of preprocessing or transformation may introduce structural perturbations into the combined dataset. These perturbations not only compromise the integrity of the data distribution but also pose risks to the stability of the GAN training process, potentially leading to convergence anomalies. Therefore, this naive method of model hijacking attack is weak both in terms of stealth and effectiveness, necessitating more advanced techniques for a successful attack.

C. Neeko Using Quadratic Programming

To overcome the limitations of the naive approach, we employ the typical image scaling attack [7], [8] to embed the hijacking images into the hijackee dataset.

In the typical image scaling attack, the adversary seeks a minimal perturbation Δ of the hijackee sample S, such that the downscaling $\mathcal{S}(\cdot)$ of the camouflaged sample $C = \Delta + S$ produces an output similar to the hijacking sample T. Goals are summarized as the following optimization:

$$min(\|\Delta\|_2^2),\ s.t. \|\mathcal{S}(S+\Delta) - T\|_\infty\ \leq \varepsilon$$

Additionally, each pixel value of C needs to remain within the fixed range (e.g., $[0, 255]$ for 8-bit images). Among these constraints, all variables are known except for Δ and ε. Δ represents the desired output, while ε is the predetermined threshold. This problem can be solved with quadratic programming (QP) [8]. When successful, the adversary can obtain a camouflaged image C that bears a resemblance to the hijackee sample but matches the hijacking sample after downscaling operations.

While this methodology ensures a high degree of visual similarity between the original and camouflaged datasets, thereby enhancing the stealthiness of the adversarial attack, it is not without flaws. The most salient drawback lies in the computational overhead required to execute this approach. Specifically, the generation of a camouflaged dataset using this technique imposes a considerable time burden, requiring a staggering about $4,985$ minutes to process a set of $36,000$ images. This long processing time constitutes a significant hindrance for potential adversaries, particularly those operating under time-sensitive conditions. Additionally, the method lacks adaptability in scenarios where new hijacking data becomes available. When new hijacking data is obtained, the embedding process needs to be repeated, further extending the attack duration.

D. Neeko Using Disguiser

To address the limitations of using quadratic programming (QP) for camouflaging the hijacking dataset and improve the efficiency of the *Neeko*, we propose a novel U-Net-based model, namely Disguiser, to replace the QP-based image scaling attack. Inspired by recent advancements in diffusion models [9]–[11], the Disguiser utilizes a U-Net architecture to efficiently camouflage the hijacking dataset using the hijackee dataset.

Overview: We present the overview of *Neeko* using Disguiser in Figure 1. In the initial phase, the adversary focuses on training a Disguiser. This Disguiser takes images from both the hijackee dataset $\mathcal{D}_o^{'}$ and hijacking dataset $\mathcal{D}_h$ as two inputs. It is designed to subtly modify the hijackee images by embedding pixels from the smaller hijacking images into them. The output is a camouflaged image that, when viewed at high resolution, remains visually indistinguishable from the original hijackee image. However, when the camouflaged image undergoes down-sampling, it has a visual similarity to the hijacking sample.

After producing a sufficient number of camouflaged images with the Disguiser, the adversary compiles these into a camouflaged dataset, denoted as $\mathcal{D}c$. This dataset is then merged with the original dataset, $\mathcal{D}o$, to craft a poisoned training set to poison the training of the target GAN. Once the GAN is trained on this poisoned dataset, it becomes successfully hijacked.

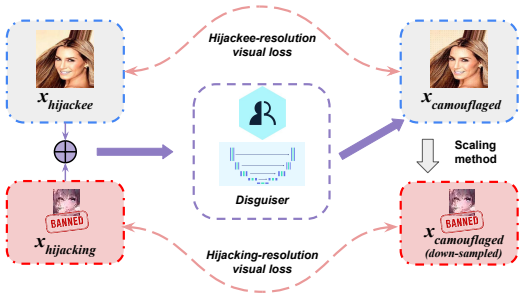


Fig. 2: The training process of the Disguiser. The adversary concatenates the hijacking image and hijackee image together and feeds them into the Disguiser. Then the Disguiser outputs a camouflaged image with the hijacking image embedded. The loss function of the Disguiser is composed of hijacking-resolution visual loss ($\mathcal{L}_{hijacking}$) and hijackee-resolution visual loss ($\mathcal{L}_{hijackee}$).

The hijacked GAN exhibits the ability to generate two unique kinds of synthetic images. The first kind consists of clean synthetic images that closely mirror the original and hijackee datasets distributions. The second kind involves camouflaged synthetic images. Although they closely match the hijackee dataset in high resolution, their visual appearance changes to align more with the hijacking dataset distribution when downscaled.

We now discuss each component of this attack in detail.

Disguiser ($\mathcal{M}_d$): The Disguiser is a U-Net-based model designed to integrate images from the hijacking dataset and the hijackee dataset. It takes two images as input: one from the hijacking dataset ($x_{hijacking} \sim \mathcal{D}_{hijacking}$) and the other from the hijackee dataset ($x_{hijackee} \sim \mathcal{D}_{hijackee}$). These images are concatenated, resulting in a single input with six channels. To handle differing dimensions, an upscaling function $\mathcal{F}_{up}(\cdot)$ is used to match the size of the hijacking dataset. The concatenated image is then passed through the Disguiser, which scales it down to a camouflaged image $x_{camouflaged}$ with three channels. The goal is for $x_{camouflaged}$ to visually resemble the hijackee image while exhibiting similarity to the hijacking image when downscaling.

The U-Net architecture's interpolation capabilities make it more effective in reproducing intricate details while incorporating relevant information from both datasets, compared to a simple encoder-decoder model used in previous works [1].

Hijacking-resolution Visual Loss ($\mathcal{L}_{hijacking}$): Similarly, to ensure visual similarity between the downscaled output of the Disguiser and the hijacking sample, we utilize the hijacking-resolution visual loss. This loss is calculated by measuring the $L1$ distance between the hijacking samples and the downscaled version of the Disguiser's output ($\mathcal{S}(x_{camouflaged})$):

$$\mathcal{L}_{hijacking} = \|\mathcal{S}[x_{camouflaged}] - x_{hijacking}\|_1$$

Hijackee-resolution Visual Loss ($\mathcal{L}_{hijackee}$): To ensure visual resemblance between the output of the Disguiser ($x_{camouflaged}$)

and the hijackee sample, we use the hijackee-resolution visual loss. This loss is also computed using the $L1$ distance metric:

$$\mathcal{L}_{hijackee} = \|x_{camouflaged} - x_{hijackee}\|_1$$

Disguiser Training: We present the training process in Figure 2. To train the Disguiser, we employ a weighted combination of two visual loss components: the hijacking-resolution visual loss and the hijackee-resolution visual loss:

$$\mathcal{L}_s = \lambda \mathcal{L}_{hijackee} + (1-\lambda)\mathcal{L}_{hijacking}$$

Here, the parameter λ controls the weight assigned to each loss term, determining their relative importance in the overall optimization objective. During training, in each epoch, random pairs of samples are created by pairing images from the hijacking dataset with those from the hijackee dataset. This random pairing strategy ensures the generalization of the Disguiser. To recap, prior to pairing, the hijacking samples are upsampled to match the dimensions of the hijackee samples. The upsampled hijacking sample is then concatenated with the hijackee sample, forming the input for the Disguiser. The output of the Disguiser, along with its corresponding hijackee and hijacking samples, are used to compute the $\mathcal{L}_s$.

It is important to note that the loss of the Disguiser is independent of the target model, making it applicable to various settings. It can be used as an image scaling attack without the need for additional training or as a camouflager in other model hijacking attacks, as discussed in previous research [1], [2].

***Neeko* Execution:** After training the Disguiser, the adversary uses it to create a camouflaged dataset by camouflaging the hijacking dataset with the hijackee dataset. This camouflaged dataset is then used to poison the training dataset of the target GAN, resulting in a hijacked GAN. The adversary can query the hijacked GAN to obtain synthetic images resembling samples from the hijackee dataset or the original datasets.

To obtain hijacking fake images, the adversary downscales the generated images. However, since the GAN is trained on both clean and camouflaged datasets, the generated fake images consist of a mixture of clean fake images and clean hijacked data. As a result, not all downscaled fake images align with the hijacking dataset distribution.

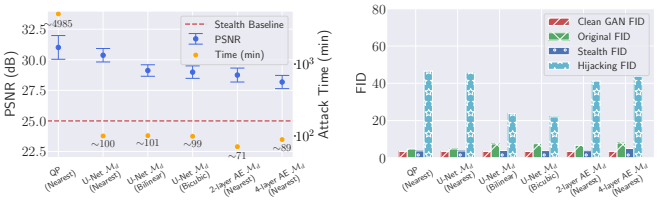
E. Extension to the Neeko Using Disguiser

In our examination of *Neeko*, we observe that the output camouflaged images—whether generated via a QP-based image scaling attack or through a Disguiser—are vulnerable to image manipulations. Minor transformations like a slight pixel shift can neutralize the attack's effectiveness, preventing the conversion of camouflaged images into hijacking images.

Thus, we propose an approach, namely the Robust *Neeko*, to improve the resilience of the camouflaged images against pixel position modifications.

In this approach, we introduce a decoder to replace the conventional downscaling function during the training phase of the Disguiser, as the decoder could extract the truly effective pixels in an area instead of only processing pixels at fixed positions like conventional downscaling methods. The decoder is not pre-trained and needs to be trained together with the Disguiser.

Moreover, an advanced adversary can use such a decoder instead of a basic downscaling method to enhance stealthiness. This makes the attack more covert, as only with access to the decoder can the hijacked fake images be retrieved.



(a) PSNRs and attack time. (b) FIDs (different settings).

Fig. 3: Quantitative results of *Neeko* with different scaling and attack methods: the hijacking dataset is `Konachan` 32^2 and the hijackee dataset is `CelebA` 256^2 (size: 120k, 30% camouflaged). Methods outside the bracket are attack methods, and those inside are the corresponding scaling methods.

Fig. 4: Samples of camouflaged synthetic images output by hijacked GANs with different attack methods and scaling methods: the hijacking dataset and hijackee dataset are `Konachan` 32^2 and `CelebA` 256^2, respectively; the methods outside and inside the bracket are the attack methods and the corresponding scaling methods, respectively.

IV. Evaluation

A. Evaluation Settings

Datasets: The Hijackee datasets and original datasets used are derived from diverse benchmark datasets, including `CelebA` [12], `FFHQ` [13], and `NIH Chest X-ray 14` [14], for face and medical images. For the hijacking datasets, we use `MNIST` [15] for handwritten digits, `Konachan Avatar` [16], and `Anime Faces` [17] for anime faces. Details are introduced in Appendix A. The training sizes are $70k$ for `FFHQ` and $120k$ for `CelebA` and `NIH Chest X-ray 14`. The poisoning rate is set to be 30%.

Models: We use the cutting-edge StyleGAN v3 as our target GAN model for the main experiments. GANs are trained from scratch with the basic configurations. In the main experiment, the Disguiser architecture is based on U-Net with `ResNet18` encoder and decoder layers, taking inputs with 6 input channels and producing output images with 3 channels. We also add a 4-layer autoencoder-based Disguiser, which has the same architecture as the camouflager [1] but different loss functions, to compare. In the extension, the decoder is composed of two

convolution layers. Its input size is the original resolution, while the output size is the hijacking resolution. The weight λ is set to 0.5.

Metrics: To evaluate the *Neeko*, we apply the peak signal-to-noise ratio (PSNR) to measure the similarity between camouflaged and original samples, with PSNR values above 25 dB indicating strong similarity, as suggested by previous research [8]. We calculate the mean and standard deviation of PSNR across the entire training dataset to assess *Neeko*'s stealthiness by evaluating artifact visibility in camouflaged samples. For attack efficiency, we measure the time to generate the camouflaged dataset.

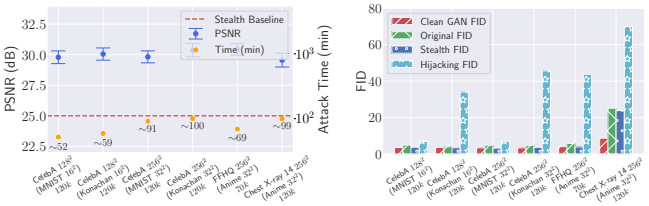
We use the Fréchet Inception Distance (FID) metric for evaluating hijacked GANs' performance [18], despite its known sensitivity to scaling [19], due to the absence of a better alternative. To distinguish between clean and camouflaged fake images produced by the hijacked GANs, we use a classifier with a slightly adjusted score threshold of 0.60 for greater accuracy, reducing the impact of misclassification. We compute three types of FID scores for hijacked GANs: original FID (clean training images vs. synthetic images), stealth FID (camouflaged and clean training images vs. synthetic images), and hijacking FID (down-sampled camouflaged training images vs. down-sampled camouflaged synthetic images). These metrics provide insights into the utility, stealthiness, and efficacy of the hijacked GANs during their training phase.

Other Settings: We follow the recommended scaling factor $\alpha = 8.0$ from image scaling attacks [7], [8], with hijacking datasets at resolutions of 16^2 and 32^2, corresponding to the original resolutions of 128^2 and 256^2. Such a scaling factor is considered to be stealthy, and the corresponding artifacts in the camouflaged images are considered to be invisible. We use the scaling methods in `PyTorch` by default. Scaling is done using the nearest neighbor method in the main experiments. We report the compute resources we use in Table IV in the appendix.

B. Neeko Using Quadratic Programming

We randomly pair 30% of the hijackee dataset samples with hijacking samples to initiate the classic image scaling attack on the target GAN. These pairs are used to generate the camouflaged dataset as described in Section III-C. The time required for this process and the peak signal-to-noise ratio (PSNR) of the camouflaged samples are reported in Figure 3a. The target GAN is then trained on the poisoned dataset, and various types of Fréchet Inception Distance (FID) metrics are measured and reported in Figure 3b.

Our attack demonstrates high stealthiness. For instance, considering the case of hijacking `CelebA` 256^2 with `Konachan` 32^2, all PSNRs (Figure 3a) exceeding the baseline 25 dB and a stealth FID of only 3.79 (Figure 3b). The impact on the FID of clean fake images is minimal, with an increase of only 1.30, while maintaining high-quality camouflaged images. The higher hijacking FID is attributed to the factors discussed previously. However, qualitative results show the relatively high quality of the generated images for the hijacking task according to Figure 4. The appendix shows more visual samples in Figure 13a.



(a) PSNRs and attack time. (b) FIDs (different settings).

Fig. 5: Quantitative results of *Neeko* using Disguiser on different hijackee and hijacking dataset pairs are reported. The U-Net-based Disguiser and the Nearest scaling method are used. The datasets in and outside the brackets represent the hijackee and hijacking datasets, respectively.

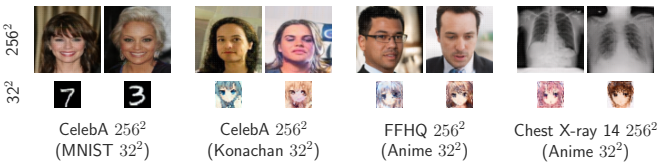


Fig. 6: Samples of camouflaged synthetic images generated by hijacked GANs on different hijackee and hijacking dataset pairs: U-Net-based Disguiser and Nearest scaling method are used; the hijackee dataset is indicated outside the brackets, while the corresponding hijacking dataset is indicated inside.

The main drawback of the attack lies in its low efficiency, as it takes a considerable amount of time (approximately $4,985$ minutes) to camouflage $36k$ samples. Although the camouflaged dataset can be reused, the computation time can still be a burden.

C. Neeko Using Disguiser

We evaluate the Disguiser-based approach using the same experimental setup as before, except for the use of the Disguiser to generate the camouflaged samples. Initially, we train the Disguiser on a randomly sampled set of $5k$ pairs, following Section III-D. The PSNR values and computation time are reported in Figure 3a. We then proceed to train and evaluate the GANs using both the original and camouflaged samples. The quantitative and qualitative results are presented in Figure 3b and Figure 4, respectively. More visual samples are available in Figure 13b in the appendix.

Comparing the Disguiser-based attack with the QP-based attack, we observe a slight decrease in the mean PSNR ($\downarrow$ 0.64) for the Disguiser, but the standard deviation is smaller, resulting in a similar lower bound ($\downarrow$ 0.22). This indicates that both attacks exhibit similar levels of stealthiness during the training phase. In terms of the performance of the hijacked GANs, the differences in various types of FID values do not exceed 2.00.

However, the Disguiser-based attack demonstrates significantly higher efficiency. Training a U-Net-based Disguiser usually requires less than 300 minutes, and camouflaging $36k$ images usually takes less than 110 minutes. Compared with

the QP-based attack, Disguiser could save over 90% less time to generate camouflaged samples. Considering we find that Disguiser also has great transferability (see Section VI), we could easily re-use a pre-trained Disguiser to camouflage other datasets, which further reduces the attack cost of our Disguiser.

Furthermore, we compare the performance of the U-Net and original autoencoder (AE) in the original model hijacking attack [1]. During the training phase, as shown in Figure 3a, the PSNR drops by 1.62 dB for the two-layer AE and 2.19 dB for the four-layer AE compared to the U-Net. The two-layer AE retains more image details but introduces additional noise in the camouflaged samples, while the four-layer AE sacrifices some details but introduces less noise. The U-Net achieves both high detail preservation and low noise introduction, resulting in superior stealthiness and equivalent attack efficiency. In the inference phase, compared to the U-Net, the two-layer AE reduces the hijacking FID by 4.44 at the expense of reducing the quality of the original task (an increase of the original FID by 1.87). The four-layer AE does not improve the hijacking FID nor maintain the hijackee FID. Our empirical results show that the U-Net structure we proposed has smaller perturbations on the original samples than the previous AE structure, which means the attack is more covert.

We also conduct the Disguiser-based attack using different pairs of hijacking and hijackee datasets, PSNRs are reported in Figure 5a and FIDs in Figure 5b. Across all dataset pairs, the lower bounds of PSNR exceed 25 dB, indicating good stealthiness during the training phase. However, the performance of the hijacked GANs varies depending on the dataset pairs. When the hijacking task dataset is simple (e.g., using `MNIST`) or similar to the hijackee task dataset (e.g., using `Konachan` to hijack `CelebA`), the increase in original FID for hijacked GANs compared to the clean versions is minimal, with a maximum increase of only 1.79.

We additionally evaluate the attack success rate. In the case where our poisoning rate is set to 30%, an ideal attack success rate would lead to a similar distribution in the GAN's output. We generate 10,000 images from the hijacked GAN and employ a pre-trained classifier to categorize these samples. Our evaluation indicates that the attack success rate is, indeed, close to 30%. Details can be found in Table V in the appendix.

Similarly, the highest stealth FID is less than 5 across all experiment datasets except `NIH Chest X-ray 14`, indicating high-quality synthetic image generation in different data distributions. The performance of the hijacking task (hijacking FID) is capped below 50. Again, we stress that the high FID, in this case, is not representative of the qualitative results shown in Figure 6, Figure 13b. However, when the hijacking task is complex and significantly different from the hijackee task, performance metrics are compromised. Both the original FID and stealth FID exceed 20, with the hijacking FID reaching as high as 69.93 in the case of `NIH Chest X-ray 14` as shown in Figure 5b.

D. Extension to the Neeko Using Disguiser

We now evaluate the extension to the *Neeko* using Disguiser by replacing the default scaling method with a decoder. The decoder and the Disguiser are jointly trained as introduced in Section III-E. The overall process follows a similar framework to the previous *Neeko*. The attack time and PSNR values are presented in Figure 3a, while the different types of FID values are shown in Figure 3b. We additionally provide examples where previous attacks fail for image transformations in Figure 7a and Figure 7b.

The extension to the attack using Disguiser shows mixed results in terms of utility and stealthiness. Compared with Disguiser with the Nearest algorithm, the original FID increases by 3.42, indicating a decrease in utility, while the increase in stealth FID is only 1.21, suggesting a slight reduction in stealthiness. On the other hand, the hijacking FID decreases by 10.72, showing an improvement in the hijacking task. Overall, the Robust *Neeko* especially involves trade-offs between utility and stealthiness, with the inclusion of the decoder benefiting the hijacking task but impacting utility and stealthiness. Further optimization, such as using a more complex architecture for the decoder and encoders, is needed to strike a better balance.

Finally, the inclusion of the decoder in the attack significantly improves its robustness against image augmentation techniques. We randomly selected 100 training images and 100 synthetic images and performed image transformation operations of translation and mirroring on them. Next, we used the trained decoder to down-sample these transformed images and manually annotate their image types. We find that all transformed images could be down-sampled into small images, which are similar to the hijacking dataset and contain only slight artifacts. This can also be observed in the visual samples in Figure 7c and Figure 7d. These images demonstrate the attack's ability to withstand various image transformations.

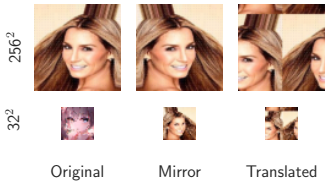
E. Hyperparameters

The various hyperparameter settings for the image scaling attack have been extensively discussed in previous works [7], [8]; hence, we omit that section and focus solely on the hyperparameters related to *Neeko*.

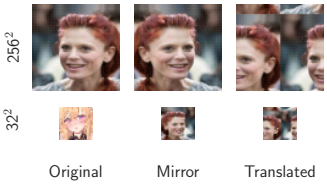
Different Scaling Methods: We first evaluate *Neeko* with different downscaling methods, including Nearest, Bilinear, and Bicubic. For this setting, we use the *Neeko* using the U-Net-based Disguiser to hijack a `CelebA` GAN with `Konachan` dataset.

Our evaluation shows that the PSNR lower bound decreased by 1.17 dB for Bilinear and 1.34 dB for Bicubic compared to Nearest. Moreover, both Bilinear and Bicubic scaling methods achieve lower hijacking FID values, reducing them by 21.88 and 23.61, respectively, but at the cost of decreased quality in the clean fake images (original FID increases by 2.65 and 2.79, respectively). The complete evaluation results, including the quantitative and qualitative results, can be found in Figure 3 and Figure 4. These findings demonstrate that scaling methods with larger perturbations, indicated by lower PSNR values, contribute to learning the hijacking task but can also negatively impact the overall utility of the GAN. More visual results are available in Figure 14 and Figure 13 in the appendix.

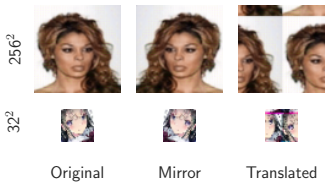
Different Target GANs: To show the generalization of the *Neeko*, we evaluate it on some simple GAN architectures,



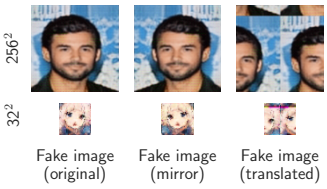
(a) Samples of camouflaged **real** images (GAN's training data) output by the QP-based attack.



(b) Samples of camouflaged **real** images (GAN's training data) output by the Disguiser.



(c) Samples of camouflaged **real** images (GAN's training data) output by the Disguiser in Robust *Neeko*.



(d) Samples of camouflaged **synthetic** images (GAN's output) generated by the hijacked GAN in Robust *Neeko*.

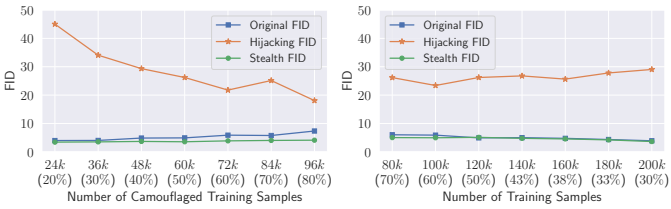
Fig. 7: Visual examples for the robustness of camouflaged images: the bottom texts indicate the corresponding image transformation operation. The hijackee and hijacking resolution is 256^2 and 32^2 respectively. In Figure 7a and Figure 7b, we provide instances where previous attacks fail for image transformations; in Figure 7c and Figure 7d, we provide instances where Robust *Neeko* could resist image transformations.

namely DCGAN [20] and WGAN [21], using the `MNIST` 16^2 dataset to hijack `CelebA` 256^2 GANs trained. The DCGAN achieves an original FID of 91.03, stealth FID of 103.22, and hijacking FID of 47.51, while the WGAN achieves an original FID of 69.33, stealth FID of 72.76, and hijacking FID of 38.86. It is important to note that these large FID values are due to the weak performance of the GANs compared to the state-of-the-art ones, e.g., a clean DCGAN results in 87.63. Despite these GANs not being over-parameterized and simple, our hijacking attacks are still successful.

We also test our attack against another advanced GAN, StyleGAN v2 [22], with the same settings as described earlier, resulting in an original FID of 3.98, stealth FID of 3.92, and hijacking FID of 6.01. When using the `Konachan` dataset as the hijacking dataset, the hijackee FID, stealth FID, and hijacking FID are 4.39, 3.95, and 42.61, respectively. These visual samples can be found in Figure 15 in the appendix.

Different Data Poisoning Rate: Generating tasks require more training data than classification tasks, thus needing higher data poisoning rates. Our study on hijacked GANs' performance with varying data poisoning rates, shown in Figure 8, includes tests under two conditions: a constant total number of training samples and a constant number of camouflaged training samples.

Findings indicate the original task's performance is minimally affected by data poisoning rates, with the original FID score increasing by only 3.36 from a 20% to an 80% poisoning rate. Furthermore, the hijacking task's performance is more reliant on the quantity of camouflaged samples than the poisoning rate itself. Keeping the number of camouflaged samples constant while lowering the poisoning rate from 70% to 30% results in a slight hijacking FID increase of 2.87. This suggests that *Neeko* can achieve high performance with a lower poisoning rate in large training sets, but a higher rate may be necessary for smaller sets to maintain hijacking effectiveness. The stealth FID is relatively stable, showing that the poisoning rate has a minor effect on this metric. These findings emphasize the need for a balance between attack efficacy and data poisoning rates for attack stealthiness, highlighting the critical nature of *Neeko*'s concealment.



(a) Fixed number of total training sample

(b) Fixed number of camouflaged sample

Fig. 8: FIDs of different poisoning rates. Poisoning rates are noted in brackets. `CelebA` 128^2 and `Konachan` 16^2 are used.

Different λ Values: λ in the loss function of Disguiser affects the quality of camouflaged hijackee images and downsampled hijacking images. Thus, we test different λ values to explore their potential effects. The related results are shown in Figure 9.

We found that the attack is ineffective when λ is 0 or 1. Because in these cases, the hijackee or hijacking image does not exist. When λ is small, the original FID will increase, and the hijacking FID will decrease. This means that the utility is compromised. When λ is large, the situation is the opposite. The above shows that the value of λ needs to be weighed according to the needs of the adversary.

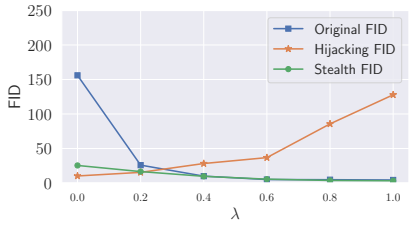


Fig. 9: FIDs of different λ values. `CelebA` 128^2 and `Konachan` 16^2 are used.

F. Summary

In summary, model hijacking attacks against GANs are highly effective across various GAN architectures and benchmark datasets. Both Disguiser-based and QP-based attacks are similarly effective, though the former is more efficient. The U-Net-based model outperforms the original autoencoder [1] in every aspect of hijacking, suggesting its potential as a standard architecture for these attacks. While the extended Disguiser-based attack improves image quality and resilience, it slightly reduces the stealth and utility of the hijacked GAN. The performance of hijacked GANs is influenced by factors like scaling methods and data poisoning rates, which can affect the visibility of camouflaging artifacts and the overall concealment of the attack. Adversaries must carefully balance these factors when setting hyperparameters for *Neeko*.

V. Defenses

In this section, we discuss two common defense strategies under the same threat model as data poisoning attacks, including the detection of poisoned samples and the denoising of such samples.

A. Detection

In general, defenders are unaware of the adversary's specific poisoning methods, and the camouflaged samples are visually highly similar to clean samples. Therefore, they primarily rely on unsupervised detection methods to examine the training set. We employ three of the most popular unsupervised clustering algorithms: DBSCAN [23], Agglomerative Clustering [24], and K-Means [25], for defense purposes. Specifically, we create a test set of 200 `CelebA` images, 30% of which are camouflaged, and then attempt to detect the camouflaged images using different clustering algorithms. According to our experimental results, DBSCAN is ineffective in detection, classifying all 200 images into the same category. Agglomerative Clustering divides the images into two clusters. However, we find that in one cluster, 30 out of 61 images are camouflaged, while in the other, 70 out of 139 are camouflaged. This indicates that Agglomerative Clustering is unable to detect camouflaged images effectively. The results from K-Means are similar. We believe that in *Neeko*, the proportion of manipulated pixels is small, so unsupervised clustering methods tend to cluster images based on other features (like hair color in `CelebA`).

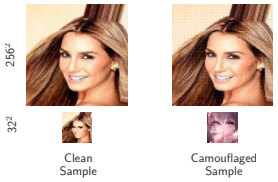
Next, we consider a more knowledgeable defender, one with background knowledge of image scaling attacks. In this case, they would use color histogram detection or color scattering detection [7], [8] specific to image scaling attacks in their attempts to detect. However, our experimental results suggest that even knowledgeable defenders require significant effort to detect camouflaged samples. Specifically, in the color histogram detection, we consider two samples as shown in Figure 10a, one camouflaged and the other clean. We then scale them to different resolutions, including 32^2 (hijacking resolution), 50^2, and 100^2. As shown in Figure 10b, the clean sample maintains a very similar histogram distribution across different resolutions. However, as shown in Figure 10c, the camouflaged sample also has a similar histogram distribution at resolutions 50^2 and 100^2. **The color histogram distribution is only significantly different at the hijacking resolution.** The results of color scattering detection are similar. Since only the adversary knows the hijacking resolution, this means that even a knowledgeable defender would need multiple attempts to find the hijacking resolution and the camouflaged samples.

Lastly, we attempt to perform detection in the frequency domain, where methods for detection in this domain have proven to be particularly effective against current data poisoning attacks [26]. We observe that the camouflaged images, which are attacked by *Neeko*, exhibit certain regular speckles in the frequency domain, which are introduced by the hijacking of images and scaling operations. Differentiating between the frequency spectrograms also requires manual intervention. However, the adversary can reduce the frequency domain visibility of the attack by increasing λ in the loss function of Disguiser (as shown in Figure 11) or selecting images with a similar tone to the hijackee image – at the cost of color difference in the hijacking image. In addition, we emphasize that according to previous work [7], [8], our attack settings are not obvious to the human eye. And when the poisoning rate is low, the attack effect is still considerable (see Figure 8).

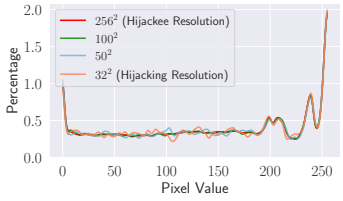
In summary, our proposed attack is covert and difficult to detect for most detection techniques. For effective detection methods, such as those based on scaling and frequency domain analysis, the involvement of human effort is necessary. When dealing with large training datasets or when the poisoning rate is low, such detection could prove to be costly.

B. Denoising

Detecting and removing camouflaged training samples will impact the size of the training set; therefore, defenders also consider using denoising techniques to improve the utilization of the data. Since the defenders do not know the specific poisoning techniques, they primarily employ blind denoising techniques to remove the injected hijacking signals. To assess the viability of this defensive strategy, we employ different filters, including the random filter, the low pass filter, and the non-local means filter. They are usually considered to be the most efficacious methods in the realm of blind denoising techniques. In Figure 16 of the appendix, which provides an illustrative representation, we delineate the outcomes yielded



(a) The camouflaged sample and its corresponding clean sample.



(b) Normalized color histograms of the clean sample at different resolutions.

(c) Normalized color histograms of the camouflaged sample at different resolutions.

Fig. 10: Normalized color histograms of the samples at different resolutions. We use the average values of three color channels to compute and plot. The x-tick labels indicate the pixel values, and the y-tick labels indicate the normalized counts $(N_{counts}/N_{total\_pixels} \times 100\%)$.

by the application of the non-local means filter to images generated by the hijacked GAN. The results suggest a noticeable mitigative impact on the detrimental effects induced by the adversarial attack. Nevertheless, it should be noted that this denoising procedure comes at the cost of significantly compromising the intricacy and detail of the generated images. This is substantiated by the notable escalation in the FID metrics for the sanitized synthetic images, measured to be 43.03. Such a compromise elucidates the trade-off that exists between defenses against the attack and preserving the high quality of images generated by the hijacked GAN. The results obtained using other types of filters are similar to those obtained with the non-local means filter; therefore, we omit the results from other filters.

VI. Discussion

A. Transferability

The conventional image scaling attack is distinguished by its transferability, which means the attack outcomes are only related to image pixel positions rather than image contents. To examine transferability in *Neeko* with Disguiser, we experiment using a pre-trained Disguiser on various datasets. Specifically, the Disguiser trained with `Konachan` 32^2 and `CelebA` 256^2 as hijacking and hijackee datasets is applied to camouflage different datasets. We assess the PSNR between camouflaged and original hijackee images and the average L1 distance between down-sampled camouflaged and hijacking images, with a sample size of $1,000$ images. Results in Table II indicate that, like conventional attacks, our method focuses on pixel positions across datasets. Thus, training a single Disguiser on a well-selected dataset can efficiently adapt *Neeko* to various datasets, enhancing cost-efficiency.

TABLE II: Experiment results for transferability. In the first column, the hijackee/hijacking dataset is indicated outside/inside the brackets.

Dataset Pairs	PSNR (dB)	L1 Distance
CelebA (Konachan)	30.012	0.001
CelebA (MNIST)	29.730	0.004
Chest X-ray 14 (Anime)	28.991	0.007
FFHQ (Anime)	29.624	0.005

B. Fine-grained Manipulation of Hijacked GANs

We have shown that adversaries can hijack a classical GAN by poisoning its dataset, a precursor to advanced manipulations of generated images. Here, we will introduce the fine-grained manipulation of hijacked GANs. More specifically, utilizing methods from prior research [27], we train an SVM to differentiate between clean and camouflaged input noises, enabling us to dictate the production of genuine or fake images by altering input noise. This technique allows for generating images in specific classes, as confirmed by our experiments detailed in Figure 12, underscoring the fine-grained control that our attack offers over synthetic image generation.

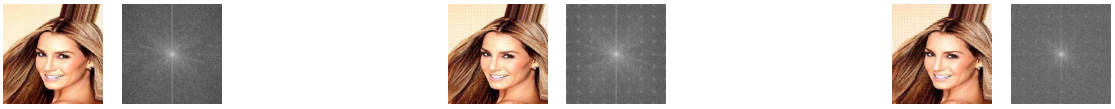
VII. Related Works

A. Testing Time Attacks

Testing time attacks occur during the post-training phase, specifically during inference. While there are notable test time attacks such as membership inference [28]–[31] and data reconstruction [32], we focus on adversarial examples since it is the most relevant to this work.

Adversarial Examples: Adversarial examples [33]–[37] are specifically designed inputs that aim to fool machine learning models into making incorrect decisions. These perturbed inputs are almost indistinguishable from regular test data but can lead the model to erroneous conclusions. The generation of adversarial examples often entails the addition of small noise to the original input, usually computed via optimization techniques. In the realm of GANs, adversarial examples are designed to be input vectors, leading the GANs to produce unexpected synthetic images [38].

While both adversarial examples [38] and *Neeko* aim to generate images from an adversarial distribution, they operate at distinct stages of the machine learning pipeline and employ different methods. The *Neeko* involves poisoning the training dataset of the target GAN, unlike adversarial examples. Yet, *Neeko* proves more efficient during test time—when utilizing the GAN—as it does not demand modifications to the noise vector, in contrast to the adversarial example approach.



(a) Clean. (b) Camouflaged (Disguiser's $\lambda = 0.35$). (c) Camouflaged (Disguiser's $\lambda = 0.70$).

Fig. 11: Selected samples, including clean and camouflaged samples, and their corresponding frequency spectrograms.

(a) An example of interpolation: moving latent codes along the normal or gradient direction of the classifier can manipulate the images generated by the GAN; "+" indicates the camouflaged direction, "-" indicates the clean direction.

(b) An example of GAN exploiting: the bottom left 2 images are the clean image and its 16^2 version; the other images are manipulated camouflaged images and their 16^2 versions.

Fig. 12: Latent codes manipulation of the target GAN: U-Net-based Disguiser and Nearest scaling method are used; the hijacking dataset and hijackee dataset are `MNIST` 16^2 and `CelebA` 128^2, respectively; in Figure 12a, the first row shows the visual appearances of the camouflaged fake images at the hijackee resolution and the second row shows those at the corresponding hijacking resolution; the resolution corresponding to each row is labeled on the left of the images.

B. Training Time Attacks

Training time attacks occur during the training of the target model. Below, we discuss the most relevant training time attacks to *Neeko*.

Data Poisoning Attacks: Data poisoning attacks [3], [4], [39], [40] involve adversaries deliberately altering the training data to degrade a machine learning model's performance. By introducing poisoned data into the training set, the attacker aims to mislead the model, e.g., leading it to incorrect predictions or classifications. These attacks pose concerns since they can occur during the data collection phase, potentially undermining the model's accuracy and reliability.

***Neeko* operates under the same threat model as data poisoning attacks, giving it a wide range of practical application scenarios.** Yet, the goals of *Neeko* significantly deviate from traditional data poisoning attacks. While the *Neeko*'s objective is to subtly hijack the GAN with an additional generation task without hindering its original performance, data poisoning attacks mainly focus on deteriorating the model's primary task performance.

Backdoor Attacks: Backdoor attacks [41]–[45] introduce secret triggers into a target model during its training, enabling an adversary to influence the model's output when such a trigger is detected in the input. Several backdoor techniques target GANs, as demonstrated in [46]–[48]. These approaches mainly implement the hidden backdoors into trained GAN models, often by adjusting the loss function or modifying the GAN's architecture.

Existing backdoor attacks targeting GANs typically require white-box access and necessitate alterations to the GAN's loss function. In contrast, our method hijacks the desired GAN without tampering with its architecture or loss function. In *Neeko*, the adversary only needs to poison the target model's training dataset. Our approach is model-agnostic, necessitating no prior knowledge about the targeted model. Furthermore, *Neeko* can be seen as an implicit form of multi-task learning [49], [50], encoding additional tasks in the training samples rather than the loss or neural network structure.

VIII. Limitations

Neeko demonstrates promising prospects, yet it also exhibits certain limitations.

Hijacking Resolution: The first constraint is that the hijacking task must be smaller than the original, as larger hijacking images complicate concealment and learning for the target GAN. More concretely, to ensure stealthiness and invisible artifacts, the scaling factor α is set to 8.0. How to attack using hijacking tasks with larger relative resolution is worth exploring in the future.

Data Poisoning Rate: GANs typically require more training data than classifiers, and generative tasks are much more challenging than classifying. The above factors lead to a higher data poisoning rate for effective hijacking, although the stealthiness and high detection difficulty of *Neeko* partially mitigate this. Specifically, we set the data poisoning rate to 0.30 while that of the model hijacking attack against classifiers exceeds 0.17. Attackers must strike a balance between hijacking performance and data poisoning rate.

Pairing of Hijackee and Hijacking Datasets: Success also depends on the similarity and complexity of the hijacking task to the original; preliminary tests with vastly different datasets like `LSUN-Church` and `CelebA` are unsatisfactory. In our experiments, the performance of pairing `NIH Chest X-ray 14` and `Anime` is also poorer than other pairings.

Performance and Metrics: The performance of the hijacking task, from the aspect of the FID metric, shows room for optimization. Moreover, relying on FID scores might not fully reflect the qualitative success of hijacking, suggesting the exploration of alternative metrics could enhance future research.

IX. Conclusion

In this paper, we present a novel model hijacking attack against GANs, namely *Neeko*. **This attack shares the same threat model as the data poisoning attack.** *Neeko* accomplishes its objectives without altering the GAN's inherent architecture or loss function. *Neeko* achieves the hijacking generation task by merely manipulating the samples in the training dataset. The *Neeko* can potentially be easily extended to any image generation model, like diffusion models since it is independent of target generative models' architectures and only manipulates the samples in the training set. We explore various techniques for the effective and covert execution of *Neeko*, including image scaling attack and the Disguiser. Notably, the U-Net-based Disguiser we propose acts as a particularly efficacious methodology in terms of both performance and efficiency. Comprehensive experimentation demonstrates the capacity of *Neeko* to compromise GANs with different architectures across a wide range of datasets. Defenses against *Neeko* are challenging, with knowledgeable defenders struggling to detect the attack.

Our work unearths several urgent ethical and safety concerns. Specifically, our work calls attention to potential vulnerabilities in the governance of datasets. Adversaries, exploiting these vulnerabilities, could potentially subvert publicly available GANs using illegal or copyrighted datasets. This reveals the risks of intellectual property violation and illegal issues. Moreover, our study contributes to the growing discourse on the risks surrounding parasitic computing. By highlighting how adversaries can exploit the target GAN's training process to reduce computational and maintenance overheads of their own tasks, we illuminate an emerging security concern. This form of parasitism could potentially result in a significant resource drain of benign users.

References

[1] A. Salem, M. Backes, and Y. Zhang, "Get a Model! Model Hijacking Attack Against Machine Learning Models," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2022.

[2] W. M. Si, M. Backes, Y. Zhang, and A. Salem, "Two-in-One: A Model Hijacking Attack Against Text Generation Models," *CoRR abs/2305.07406*, 2023.

[3] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2018, pp. 19–35.

[4] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2018, pp. 6103–6113.

[5] "StyleGAN 3," https://lambdalabs.com/blog/stylegan-3.

[6] "Google Cloud Price Table," https://cloud.google.com/vertexai/pricing#europe.

[7] Q. Xiao, Y. Chen, C. Shen, Y. Chen, and K. Li, "Seeing is Not Believing: Camouflage Attacks on Image Scaling Algorithms," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2019, pp. 443–460.

[8] E. Quiring, D. Klein, D. Arp, M. Johns, and K. Rieck, "Adversarial preprocessing: Understanding and preventing image-scaling attacks in machine learning," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2020, pp. 1363–1380.

[9] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," in *International Conference on Machine Learning (ICML)*. PMLR, 2015, pp. 2256–2265.

[10] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022, pp. 10 684–10 695.

[11] P. Dhariwal and A. Q. Nichol, "Diffusion Models Beat GANs on Image Synthesis," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2021, pp. 8780–8794.

[12] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep Learning Face Attributes in the Wild," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 3730–3738.

[13] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 4401–4410.

[14] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers, "Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017, pp. 2097–2106.

[15] "MNIST," http://yann.lecun.com/exdb/mnist/.

[16] "Konachan," https://aistudio.baidu.com/aistudio/datasetdetail/110820/0.

[17] "Anime," https://github.com/bchao1/Anime-Face-Dataset.

[18] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in *Annual Conference on Neural Information Processing Systems (NIPS)*. NIPS, 2017, pp. 6626–6637.

[19] G. Parmar, R. Zhang, and J. Zhu, "On aliased resizing and surprising subtleties in GAN evaluation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2022, pp. 11 400–11 410.

[20] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in *International Conference on Machine Learning (ICML)*. PMLR, 2017, pp. 214–223.

[22] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and Improving the Image Quality of StyleGAN," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 8107–8116.

[23] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *International Conference on Knowledge Discovery and Data Mining (KDD)*. AAAI, 1996, pp. 226–231.

[24] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *CoRR abs/1109.2378*, 2011.

[25] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An Efficient k-Means Clustering Algorithm: Analysis and Implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[26] T. Wang, Y. Yao, F. Xu, S. An, H. Tong, and T. Wang, "An Invisible Black-Box Backdoor Attack Through Frequency Domain," in *European Conference on Computer Vision (ECCV)*. Springer, 2022, pp. 396–413.

[27] Y. Shen, J. Gu, X. Tang, and B. Zhou, "Interpreting the Latent Space of GANs for Semantic Face Editing," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020, pp. 9240–9249.

[28] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership Inference Attacks Against Machine Learning Models," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2017, pp. 3–18.

[29] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes, "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.

[30] C. A. C. Choo, F. Tramèr, N. Carlini, and N. Papernot, "Label-Only Membership Inference Attacks," in *International Conference on Machine Learning (ICML)*. PMLR, 2021, pp. 1964–1974.

[31] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramèr, "Membership Inference Attacks From First Principles," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022, pp. 1897–1914.

[32] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang, "Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning," in *USENIX Security Symposium (USENIX Security)*. USENIX, 2020, pp. 1291–1308.

[33] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[34] H. Yu, K. Yang, T. Zhang, Y.-Y. Tsai, T.-Y. Ho, and Y. Jin, "CloudLeak: Large-Scale Deep Learning Models Stealing Through Adversarial Examples," in *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2020.

[35] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial Examples for Graph Data: Deep Insights into Attack and Defense," in *International Joint Conferences on Artifical Intelligence (IJCAI)*. IJCAI, 2019, pp. 4816–4823.

[36] L. Fowl, M. Goldblum, P. Chiang, J. Geiping, W. Czaja, and T. Goldstein, "Adversarial Examples Make Strong Poisons," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*. NeurIPS, 2021, pp. 30 339–30 351.

[37] Z. Zhao, D. Dua, and S. Singh, "Generating Natural Adversarial Examples," in *International Conference on Learning Representations (ICLR)*, 2018.

[38] J. Kos, I. Fischer, and D. Song, "Adversarial examples for generative models," in *IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 36–42.

[39] B. Biggio, B. Nelson, and P. Laskov, "Poisoning Attacks against Support Vector Machines," in *International Conference on Machine Learning (ICML)*. icml.cc / Omnipress, 2012.

[40] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, "Data Poisoning Attacks Against Federated Learning Systems," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2020, pp. 480–501.

[41] Y. Yao, H. Li, H. Zheng, and B. Y. Zhao, "Latent Backdoor Attacks on Deep Neural Networks," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2019, pp. 2041–2055.

[42] J. Jia, Y. Liu, and N. Z. Gong, "BadEncoder: Backdoor Attacks to Pre-trained Encoders in Self-Supervised Learning," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022.

[43] Z. Zhang, J. Jia, B. Wang, and N. Z. Gong, "Backdoor Attacks to Graph Neural Networks," in *ACM Symposium on Access Control Models and Technologies (SACMAT)*. ACM, 2021, pp. 15–26.

[44] Z. Sha, X. He, P. Berrang, M. Humbert, and Y. Zhang, "Fine-Tuning Is All You Need to Mitigate Backdoor Attacks," *CoRR abs/2212.09067*, 2022.

[45] A. Saha, A. Subramanya, and H. Pirsiavash, "Hidden Trigger Backdoor Attacks," in *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI, 2020, pp. 11 957–11 965.

[46] A. Rawat, K. Levacher, and M. Sinn, "The Devil is in the GAN: Backdoor Attacks and Defenses in Deep Generative Models," *CoRR abs/2108.01644*, 2022.

[47] A. Salem, Y. Sautter, M. Backes, M. Humbert, and Y. Zhang, "BAAAN: Backdoor Attacks Against Autoencoder and GAN-Based Machine Learning Models," *CoRR abs/2010.03007*, 2020.

[48] R. Jin and X. Li, "Backdoor Attack is a Devil in Federated GAN-based Medical Image Synthesis," *CoRR abs/2207.00762*, 2022.

[49] R. Caruana, "Multitask learning," *Machine Learning*, 1997.

[50] Y. Zhang and Q. Yang, "A Survey on Multi-Task Learning," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

APPENDIX

DESCRIPTION OF TERMS

Here we describe the terms used in *Neeko*.

TABLE III: Description of different terms used in *Neeko*.

Term	Definition
Neeko	Neeko is a champion in League of Legends. She can blend into any crowd by borrowing the others' appearances, which match the function of our Disguiser. We borrow the champion's name "Neeko" to describe our proposed attacks.
Original Dataset	Training dataset of the target GAN's original task.
Hijackee Dataset	The dataset from the same distribution as the target GAN's training dataset.
Hijackee Samples	Samples from the hijackee dataset. Usually, they are at a big resolution.
Hijacking Dataset	Training dataset of the adversary's hijacking task.
Hijacking Samples	Samples from the hijacking dataset. Usually, they are at a small resolution.
Camouflaged Dataset	The modified hijacking dataset after being stealthily embedded in a hijackee dataset.
Camouflaged Samples	Samples from the camouflaged dataset. They look similar to the corresponding hijacking samples at the large resolution, but look similar to hijacked samples at the small resolution.
Poisoned Dataset	Dataset where the target GANs will be trained (the concatenation of the camouflaged and original datasets).
Hijackee Resolution	The resolution of the hijackee dataset. Usually, it is larger than the hijacking resolution.
Hijacking Resolution	The resolution of the hijacking dataset.

COMPUTE RESOURCES

We train Disguiser and poison the images on Server X10DRG-K80 . We train all the StyleGANs on Server DGX-A100. Details are shown in Table IV.

TABLE IV: Compute resource details.

Server Name	Model	CPU	GPU	RAM
X10DRG-K80	Supermicro SYS-4028GR-TRT	Intel Xeon E5-2697	1 Tesla K80	512 GB
DGX-A100	NVIDIA DGX A100 (40G)	AMD Rome 7742	2 NVIDIA A100	1 TB

DATASET DESCRIPTION

To evaluate *Neeko*, we conduct comprehensive experiments on different benchmark datasets. For clarity, we now briefly introduce the related datasets.

CelebA: `CelebA` [12] dataset is an extensive collection of over $200,000$ labeled celebrity images designed for various facial analysis tasks. It is compiled to serve as a benchmark for computer vision and ML algorithms, featuring a diverse set of images in terms of ethnicity, age, and gender. Because of its scale and comprehensive annotations, `CelebA` has become a cornerstone in the development and evaluation of algorithms for facial recognition and generative modeling.

FFHQ: `FFHQ` [13] dataset is a high-quality dataset consisting of $70,000$ human facial images at 1024×1024 resolution. Unlike other face datasets, FFHQ contains a wide range of ages, ethnicities, and image backgrounds, making it highly diverse and well-suited for training and evaluating facial analysis algorithms. Its high resolution and comprehensive diversity make it a valuable resource for developing facial analysis and generative models.

NIH Chest X-ray 14: `NIH Chest X-ray 14` [14] is a publicly available collection of more than $100,000$ de-identified chest X-ray images sourced from the National Institutes of Health Clinical Center. Researchers in both the medical and computer science communities frequently utilize this dataset to advance the state-of-the-art in medical image recognition and diagnosis techniques.

MNIST: `MNIST` [15] is a widely used resource in the ML and computer vision communities, consisting of a collection of $70,000$ grayscale images of handwritten digits (0 through 9). MNIST is a relatively simplistic/basic dataset for testing ML models, particularly in digit recognition and computer vision.

Konachan Avatar: `Konachan Avatar` [16] consists of $44,766$ high-quality cartoon avatar images, all of which are unlabeled. These images are subsequently processed and cleaned manually. Given its high quality and sizable volume, this dataset offers a valuable resource for researchers interested in ML tasks related to cartoon-style facial recognition or generative modeling.

Anime Faces: `Anime Faces` [17] comprises $63,632$ high-quality anime faces processed using the anime face detection algorithm. Unlike other popular datasets like the `Danbooru`, which is noted for its less organized structure, this dataset features high-quality anime character images with clean backgrounds and rich colors.

ADDITIONAL RESULTS OF EXPERIMENTS

TABLE V: Experiment results of attack success rates (ASR). In the first column, the hijackee dataset is indicated outside the brackets, while the corresponding hijacking dataset is indicated inside the brackets.

Dataset Pairs	ASR (%)
CelebA 128^2 (Konachan 16^2)	28.01
CelebA 128^2 (MNIST 16^2)	29.17
CelebA 256^2 (Konachan 32^2)	28.15
CelebA 256^2 (MNIST 32^2)	29.12
FFHQ 256^2 (Anime 32^2)	27.87
Chest X-ray 14 256^2 (Anime 32^2)	22.37

(a) Quadratic programming and Nearest are used as the attack method and scaling method, respectively.

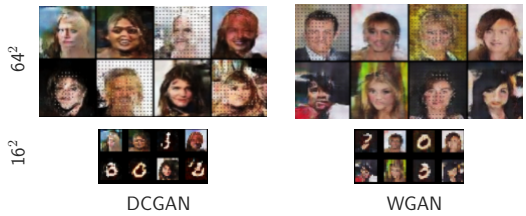
(b) U-Net-based Disguiser and Nearest are used as the attack method and scaling method, respectively.

(c) U-Net-based Disguiser and Bilinear are used as the attack method and scaling method, respectively.

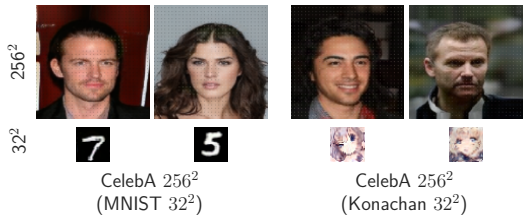
(d) U-Net-based Disguiser and Bicubic are used as the attack method and scaling method, respectively.

Fig. 13: Visual samples of downsampled poisoned fake images: the hijacking dataset and hijackee dataset are `Konachan` 32^2 and `CelebA` 256^2, respectively.

Fig. 14: Visual camouflaged training samples under different attack and scaling methods: the methods outside and inside the bracket are the attack methods and the corresponding scaling methods, respectively; the hijacking dataset and hijackee dataset are `Konachan` 32^2 and `CelebA` 256^2, respectively.



(a) Visual samples of poisoned fake images output by the hijacked DCGAN and WGAN: the hijacking dataset and hijackee dataset are `MNIST` 16^2 and `CelebA` 64^2, respectively.



(b) Visual samples of poisoned fake images output by the hijacked StyleGAN v2: the datasets outside and inside the bracket are the hijackee dataset and the corresponding hijacking dataset, respectively.

Fig. 15: Visual results of different target GANs: U-Net-based Disguiser and Nearest scaling method are used; the first row shows the visual appearances of the poisoned fake images at the hijackee resolution and the second row shows those at the corresponding hijacking resolution; the resolution corresponding to each row is labeled on the left of the images.

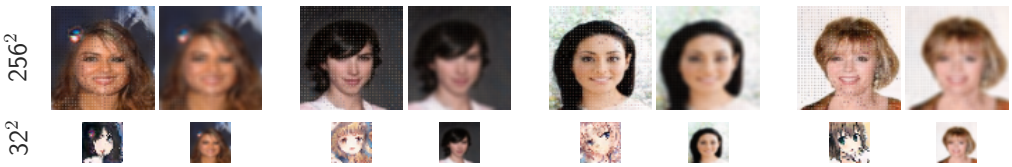


Fig. 16: Visual appearances of poisoned fake images and the corresponding de-noised versions: the hijacking dataset and hijackee dataset are `Konachan` 32^2 and `CelebA` 256^2, respectively; the first row shows the visual appearances of the poisoned fake images at the hijackee resolution and the second row shows those at the corresponding hijacking resolution; the resolution corresponding to each row is labeled on the left of the images.