PTW: Pivotal Tuning Watermarking for Pre-Trained Image Generators

Nils Lukas and Florian Kerschbaum, *University of Waterloo*

https://www.usenix.org/conference/usenixsecurity23/presentation/lukas

This paper is included in the Proceedings of the
32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

PTW: Pivotal Tuning Watermarking for Pre-Trained Image Generators

Nils Lukas
University of Waterloo

Florian Kerschbaum
University of Waterloo

Abstract

Deepfakes refer to content synthesized using deep generators, which, when *misused*, have the potential to erode trust in digital media. Synthesizing high-quality deepfakes requires access to large and complex generators only a few entities can train and provide. The threat is malicious users that exploit access to the provided model and generate harmful deepfakes without risking detection. Watermarking makes deepfakes detectable by embedding an identifiable code into the generator that is later extractable from its generated images. We propose Pivotal Tuning Watermarking (PTW), a method for watermarking pre-trained generators (i) three orders of magnitude faster than watermarking from scratch and (ii) without the need for any training data. We improve existing watermarking methods and scale to generators 4× larger than related work. PTW can embed longer codes than existing methods while better preserving the generator's image quality. We propose rigorous, game-based definitions for robustness and undetectability and our study reveals that watermarking is not robust against an adaptive white-box attacker who has control over the generator's parameters. We propose an adaptive attack that can successfully remove any watermarking with access to only 200 non-watermarked images. Our work challenges the trustworthiness of watermarking for deepfake detection when the parameters of a generator are available.

1 Introduction

Deepfakes, a term used to describe synthetic media generated using deep image generators have received widespread attention in recent years. While deepfakes offer many beneficial use cases, for example in scientific research [9, 48] or education [16, 39, 47], they have also raised ethical concerns because of their potential to be *misused* which can lead to an erosion of trust in digital media. Deepfakes have been scrutinized for their use in disinformation campaigns [2, 23], impersonation attacks [15, 35] or when used to create non-consensual media of an individual violating their privacy [10, 20]. These threats highlight the need to control the misuse of deepfakes.

While some deepfakes can be created using traditional computer graphics, using deep learning methods such as the Generative Adversarial Network (GAN) [19] can reduce the time and effort needed to create deepfakes. However, training GANs requires a significant investment in terms of computational resources [26] and data preparation, including collection, organization, and cleaning. These costs make training image generators a prohibitive endeavor for many. As a consequence, generators are often trained by one *provider* and made available to many users through Machine-Learning-as-a-Service [6]. The provider wants to disclose their model responsibly and deter *model misuse*, which is the unethical use of their model to generate harmful or misleading content [36].

Problem. Consider a provider who wants to make their image generator publicly accessible under a contractual usage agreement that serves to prevent misuse of the model. The threat is a user who breaks this agreement and uses the generator to synthesize and distribute harmful deepfakes without detection. To mitigate this threat in practice, companies such as OpenAI have deployed invasive prevention measures by providing only monitored access to their models through a black-box API. Users that synthesize deepfakes are detectable when they break the usage agreement if the provider matches the deepfake with their database. This helps deter misuse of the model, but it can also lead to a lack of transparency and limit researchers and individuals from using their technology [12, 50]. For example, query monitoring which is used in practice by companies such as OpenAI raises privacy concerns as it involves collecting and potentially storing sensitive information about the user's queries. A better solution would be to implement methods that deter model misuse without the need for query monitoring.

A potential solution is to rely on deepfake detection methods [7, 13, 17, 24, 25, 30, 40, 56]. The idea guiding such *passive* methods is to exploit artifacts in the synthetic images that separate fake and real content. While these detectors protect well against some deepfakes it has been demonstrated that they can be bypassed by unseen, improved generators that adapt to existing detectors [14]. As technology advances, it is

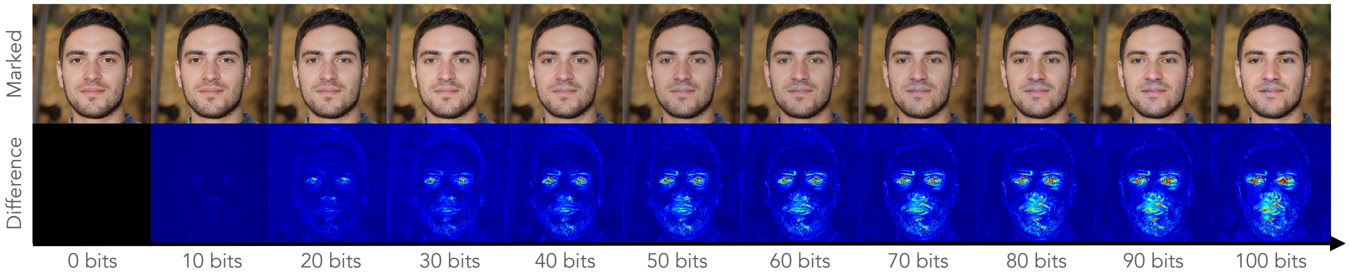


Figure 1: A demonstration of our watermark and the impact of the number of embedded bits on the visual image quality. The top row shows the watermarked, synthetic image and the bottom row shows its difference to the same image without a watermark.

possible that generators will be developed that synthesize virtually indistinguishable images, rendering passive deepfake detection methods ineffective in the long term.

A different approach to deepfake detection is watermarking [60] when the detection method can access and modify the *target generator*. This is a kind of watermarking that modifies the generator to embed an identifiable message that is later extractable from access to the generated content using a secret key. Methods such as generator watermarking [59, 60] remain applicable to unseen image generators that could be developed in the future. For deepfake detection, the provider needs a watermark that is extractable from any image synthesized by the generator. We refer to this setting as a *no-box* verification, because the verifier only requires access to the generated content, but not to the generator model. However, there are still several challenges in designing no-box watermarks. These include (i) the ability to embed long messages with limited impact on the model's utility, (ii) the undetectability of the watermark without the secret key, (iii) robustness against removal, and (iv) the method should be efficient.

Solution Overview. Existing watermarking methods are difficult to scale to high-resolution models because they require re-training the generator from scratch which is computationally intensive [59, 60]. Moreover, while some existing watermarks claim a good capacity/utility trade-off [60], these claims have been limited to relatively small generators. To address these challenges, we propose an efficient watermark embedding called Pivotal Tuning Watermarking (PTW). PTW is the first method to embed watermarks into pre-trained generators and speeds up the embedding process by three orders of magnitude from more than one GPU month when watermarking from scratch [59, 60] to less than one hour. We identify modifications to existing watermarks that speed up their embedding significantly and propose our own watermark that can be embedded using PTW with an improved capacity/utility trade-off compared to related work. Figure 1 visualizes this trade-off for our watermark.

We propose rigorous, game-based definitions for robustness and undetectability and evaluate our watermark in two threat models, where the adversary either has access to the generator only through an API (*black-box*) or has control over its parameters (*white-box*). Our results confirm that existing watermarking methods [59, 60] are robust and undetectable in the black-box threat model using existing attacks. We propose three new attacks: a black-box attack called Super-Resolution and two white-box attacks, called (1) Overwriting and (2) Reverse Pivotal Tuning. Using these attacks, our experiments show that watermarking is robust in the black-box setting, but that it cannot withstand a white-box attacker with access to only 200 images ($\approx 0.3\%$ of the generator's training dataset) who can remove watermarks at a negligible loss in the generator's image quality. Our attacks undermine robustness in the white-box setting and have implications on how watermarking for image generators could be a viable solution to deepfake detection in the future.

1.1 Contributions

- We propose a method to watermark pre-trained generators that we call Pivotal Tuning Watermarking (PTW). PTW is a method for watermarking generators that (i) does not require any training data and (ii) is three orders of magnitude faster than existing methods [59, 60].
- We modify existing watermarks [59, 60] for GANs to be compatible with pre-trained generators.
- We provide rigorous game-based definitions for robustness and undetectability for generator watermarking.
- We propose an improved watermarking scheme and experiment with three generator architectures (StyleGAN2 [27], StyleGAN3 [26] and StyleGAN-XL [45]) on multiple high-quality image generation datasets.
- We propose one black-box and two white-box watermark removal attacks. Our results show that watermarking is not robust in practice against our white-box attacks.
- We release our source code implementing all existing watermarking schemes and attacks as open source[1].

[1] https://github.com/dnn-security/gan-watermark

2 Background & Related Work

This section provides a background on generative models and Pivotal Tuning [42], followed by a description of related work on the detection and attribution of deepfakes.

2.1 Background

Generative Adversarial Network (GAN). GANs are a machine learning framework used to train deep generative models [19]. They define a generator $G: \mathcal{Z} \rightarrow \mathcal{X}$ that maps from a latent space $\mathcal{Z}$ to images $\mathcal{X}$ and a discriminator $F: \mathcal{X} \rightarrow \{0,1\}$ that maps images to binary labels. The labels represent *real* and *fake* images. Let $D \subseteq \mathcal{X}$ be an image dataset and let $\theta_{\text{F}}, \theta_{\text{G}}$ be parameters for a discriminator and generator. The unsaturated logistic loss for GANs is written as follows.

$$\begin{aligned} \mathcal{L}_{GAN} = & \underset{x \sim D}{\mathbb{E}}[\log F(x;\theta_F)] \\ & + \underset{z \sim \mathcal{N}(0,I^d)}{\mathbb{E}}[\log(1 - F(G(z;\theta_G);\theta_F)] \end{aligned} \quad (1)$$

During training, the discriminator learns to classify real and fake images and the generator learns to fool the discriminator.

StyleGAN [27]. The StyleGAN is a specific GAN architecture that introduces a mapper $f: \mathcal{Z} \rightarrow \mathcal{W}$ that maps latent codes into an intermediate latent space. This intermediate latent space contains *styles* with fine-grained control over the synthesized image. Since its inception, the basic StyleGAN [27] has been revised many times leading to the development of StyleGAN2 [28], StyleGAN3 [26] and recently StyleGAN-XL [45]. These generators achieve state-of-the-art performance on many image generation datasets including ImageNet [11] where they outperform[2] other publicly accessible generators such as Latent Diffusion models [43].

Pivotal Tuning [42]. Pivotal Tuning is a method to regularize a pre-trained generator while preserving a high fidelity to the generator before tuning. The idea is to preserve the mapping from latent codes to images by cloning and freezing the generator's parameters, referred to as the *Pivot* with parameters θ_G, and then fine-tuning a trainable, second generator θ^*_G with some regularization term $R(\cdot)$. It has been demonstrated that Pivotal Tuning achieves near-perfect image inversion while also enabling latent-based image editing [42]. The Pivotal Tuning loss $\mathcal{L}_{\text{PT}}$ is written as follows:

$$\mathcal{L}_{\text{PT}} = \mathcal{L}_{\text{LPIPS}}(x_0, x) + \lambda_{\text{R}} R(x) \quad (2)$$

where $x_0 = G(z;\theta_G)$ is an image synthesized by the Pivotal generator using a latent code $z \in \mathcal{Z}$ and $x = G(z;\theta^*_G)$ is the image generated using the tunable weights θ^*_G for the same latent code. The Learned Perceptual Image Patch Similarity (LPIPS) [63] loss $\mathcal{L}_{\text{LPIPS}}$ quantifies a perceptual similarity between images extracted using deep feature extractors.

2.2 Related Work

This section summarizes related work on deepfake detection and attribution for deep image generators.

Deepfake Detection and Attribution. Deepfake detection and attribution are the tasks of identifying fake images, that have been generated or manipulated using deep image generators. Detection focuses only on detecting whether an image is fake, whereas attribution focuses on determining the image's origin. For a given *target generator* that is used to synthesize a deepfake, we taxonomize existing work by (i) the level of access to this target generator and (ii) whether the target generator's parameters can be modified by the detection or attribution method prior to the deployment of the generator.

(i) Without Generator Access: In this setting, the detection algorithm does not have access to the target generator. Existing work on detecting deepfakes trains classifiers on a public set of deepfakes with known labels for fake/real images [13, 30, 44], exploit semantic incoherence such as asymmetries [24, 34] or low-level artifacts from the generation process [7, 17, 25, 33, 40, 56]. Deepfake attribution methods without access to the target generator apply unsupervised learning methods [18, 59] or only attribute deepfakes to an architecture (and not a generator instance) [58]. Although these methods have proven effective in detecting some deepfakes, it has been shown that they can be evaded by an adversary who adapts to these detectors [14].

(ii) With Generator Access: The detection method can have some level of access to the generator, including black-box API or white-box access to its parameters. **(Fingerprinting)** Methods that do not modify the generator's parameters are referred to as *fingerprinting* methods. Recently, attribution methods have been proposed for Latent Diffusion models [46] based on training classifiers on the model's generated data. For GANs, fingerprinting methods all rely on training classifiers on the target generator's data [5,57,59]. **(Watermarking)** Methods that modify the generator prior to deployment are referred to as *watermarking* methods. Yu et al. [60] modify the generator's training data and re-train a watermarked generator. Another approach is to modify the generator's training procedure [61]. All existing methods require training the generator from scratch to embed a watermark.

3 Threat Model

Our threat model consists of a defender, who we also refer to as the model provider, and an adversary, who controls a malicious user. We define the following functions.

- TRAIN: A training function $\mathcal{T}$ trains a generator on input of a dataset D and returns trained parameters θ_G.
- EMBED: On input of a trained generator θ_G and a message $m \in \{0,1\}^n$, this function returns parameters θ^*_G for a tuned generator and a (secret) watermarking key τ.

[2] https://paperswithcode.com/dataset/ffhq

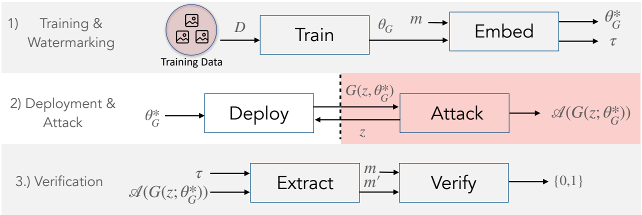


Figure 2: An overview of deepfake attribution by watermarking the generator.

- EXTRACT: Given a key τ and an image, this function returns an extracted message $m' \in \{0,1\}^n$.
- VERIFY: Given an extracted and a target message, this method computes their bit error rate (BER) and returns 1 if the BER is below a threshold (i.e., the watermark is present) and 0 otherwise.
- ATTACK: An attack $\mathcal{A}$ can use access to the generator and auxiliary data to generate and return an image. The objective of an attack is to remove a watermark.

Using the functions defined above, Figure 2 illustrates attribution through watermarking in three steps. First, the provider trains a generator and embeds a watermark. Second, the generator is deployed and a malicious user generates harmful deepfakes at any time after deployment. Finally, deepfakes are attributed to the generator by verifying their watermark.

Adversary's Capabilities. We consider two adversaries that differ in their level of access to the target generator. Our first adversary, the *black-box* adversary, has only API access to the target generator. This means they can query the generator on any latent code $z \in \mathcal{Z}$, but do not have knowledge of its parameters or intermediate activations. The black-box adversary is limited in the number of queries to the generator since queries usually incur a monetary cost to the user. Our second adversary, the *white-box* adversary, has full access to the (watermarked) target generator's parameters, meaning they can tune the generator's weights in an attempt to remove watermarks and generate any number of watermarked images. We use the same definitions for adversaries with black-box API access and white-box access as Lukas et al. [31].

Both adversaries have access to a limited set of R real, non-watermarked images from the same distribution as the defender's training data. An adversary can have limited access to real images without a watermark, which is a commonly made assumption to evaluate the robustness of watermarking [31]. We assume limited availability of real, non-watermarked data in our threat model, as an attacker with sufficient data and computational resources could train and deploy their own generator without a watermark. A black-box attacker could attempt to *extract* the generator [49], by training a surrogate on generated data, but this is out of the scope of our work since model extraction (i) requires massive computational resources and (ii) likely results in surrogates with low utility [31].

Notation	Description
$\mathcal{T}$	A stochastic training algorithm
$\mathcal{D}$	A distribution over images
$\mathcal{D}^n$	Distribution over n images
$D \sim \mathcal{D}$	Draw images D uniformly from $\mathcal{D}$
$y \leftarrow \mathcal{P}(\vec{x})$	Call $\mathcal{P}$ with $\vec{x}$ and assign result to y
$\mathcal{A}$	A procedure denoting an adversary
θ	Parameters of a generator
m	A watermarking message
τ	A secret watermarking key

Table 1: A summary of our notation.

Adversary's Objective. The common goal of our adversaries is to synthesize images (i) with high visual fidelity to real images and (ii) to generate images that do not retain the watermark. An image does not retain its watermark if the verification mistakenly outputs zero.

Defender's Capabilities. The defender has access to their own generator's parameters and the secret watermarking key. Their objective is to verify any given image whether it originated from their generator. We refer to the defender's access during verification as *no-box*, because, unlike black-box verifiable watermarking [1] that can verify the watermark using many queries to the model, with no-box access the watermark needs to be verified using only a (single) generated image.

3.1 Robustness

Algorithm 1 encodes the watermark robustness game, given a data distribution $\mathcal{D}$, a training algorithm $\mathcal{T}$, an attack $\mathcal{A}$ with access to R real images, a message length $n \in \mathbb{N}$ and a *challenge size* $K \in \mathbb{N}$. The challenge size is the number of high-quality, non-watermarked images that the adversary has to synthesize to win the game. Note that an adversary with auxiliary access to at least K non-watermarked, real images can always trivially win our security game by returning these real images in their attack. For this reason, we make the assumption that the adversary's auxiliary dataset size $R \ll K$ is much smaller than the challenge size. We choose K=50 000 which allows comparing the quality of the generated images with related work [26, 28]. Table 1 summarizes our notation.

In Algorithm 1, the defender and adversary first sample their real training and auxiliary datasets $D, D_{\mathcal{A}}$ (lines 2-3). Then, the defender trains their generator, samples a watermark message, and embeds a watermark (lines 4-6). A fair coin is flipped randomly K times (line 7) determining whether the attack targets the generator before or after watermarking (line 8). Finally, the defender verifies each of the attacker's images and should correctly predict whether an image originated from the non-watermarked or watermarked generator, called the *evasion rate* (lines 9-10). We encode the access level of the adversary to the target generator with a function $O(\theta)$ that

Algorithm 1 Watermark Robustness Game

```
1:  procedure $O(\theta)$                                  ▷ access level of the adversary
2:      return $\theta$ if white-box else $G(\cdot;\theta)$
1:  experiment ROBUSTNESS($\mathcal{D},\mathcal{T},\mathcal{A},R,K,n$)
2:      $D \sim \mathcal{D}$
3:      $D_{\mathcal{A}} \sim \mathcal{D}^R$                    ▷ real, non-watermarked
4:      $\theta_G^0 \leftarrow \mathcal{T}(D)$
5:      $m \sim \{0,1\}^n$                              ▷ watermarking message
6:      $\tau, \theta_G^1 \leftarrow$ EMBED($\theta_G^0, m$)
7:      $B \sim \{0,1\}^K$                                      ▷ K coin flips
8:      $X \leftarrow \cup_{i\in\{1..K\}} \mathcal{A}(O(\theta^{B_i}), D_{\mathcal{A}})$   ▷ images after attack
9:      $Y \leftarrow \cup_{i\in\{1..K\}}$ VERIFY(EXTRACT($X_i,\tau$), $m$)
10:     $y \leftarrow \frac{1}{|Y|}\sum_{i\in\{1..K\}} 1_{Y_i \neq B_i}$          ▷ evasion rate
11:     return $X, y$
```

returns parameters θ of the generator on white-box access and access to the synthesis function $G(\cdot;\theta)$ otherwise.

The success of the adversary is its expected evasion rate y and the visual quality of its images X, which is commonly measured by the Fréchet Inception Distance (FID) [22]. FID is a perceptual distance computed between two sets of images and a low FID score indicates a high visual similarity between both sets. Similar to the evaluation by Karras et al. [27], we measure the FID between the adversary's images X and the defender's training dataset D. The success of the adversary is a trade-off between its expected evasion rate and the visual image quality after the attack.

$$\text{Succ}_{\text{EVASION}} = \mathbb{E}\left[y - \text{FID}(X,D)\right] \quad (3)$$

3.2 Detectability

A detectable watermark poses a threat because it facilitates an adversary in locating and removing the watermark. Algorithm 2 encodes the watermark detectability game for a given data distribution $\mathcal{D}$, detection attack $\mathcal{A}_{\text{Detect}}$, the adversary's auxiliary real dataset size R, and access to labeled synthetic images R_1 and R_2 from the generator before and after watermarking. Our game challenges the adversary to determine whether an image was generated before or after watermarking.

First, the training and auxiliary datasets are sampled (lines 2-3) for training a non-watermarked model (line 4). The defender samples a message and embeds a watermark (lines 5-6). Then, the attacker gets access to R_1, R_2 random, synthetic non-watermarked, and watermarked images (lines 7-9). A fair coin is flipped randomly, deciding whether the detection attack sees a watermarked or non-watermarked image (line 10), the attack is executed (line 13) and the attacker's prediction is returned (line 14). The success of the detectability attack is its expected classification accuracy.

$$\text{Succ}_{\text{DETECTION}} = \mathbb{E}\left[p\right] \quad (4)$$

Algorithm 2 Watermark Detection Game

```
1:  experiment DETECTABILITY($\mathcal{D},\mathcal{T},\mathcal{A}_{Detect},R,R_1,R_2$)
2:      $D \sim \mathcal{D}$
3:      $D_{\mathcal{A}} \sim \mathcal{D}^R$
4:      $\theta_G^0 \leftarrow \mathcal{T}(D)$
5:      $m \sim \{0,1\}^n$
6:      $\tau, \theta_G^1 \leftarrow$ EMBED($\theta_G^0, m$)
7:      $Z_1, Z_2 \sim \mathcal{Z}^{R_1}, \mathcal{Z}^{R_2}$
8:      $D_{\mathcal{A}_1} \leftarrow \{G(z;\theta_0) | z \in Z_1\}$          ▷ non-watermarked
9:      $D_{\mathcal{A}_2} \leftarrow \{G(z;\theta_1) | z \in Z_2\}$              ▷ watermarked
10:     $b \sim \{0,1\}$                                          ▷ coin flip
11:     $z \sim \mathcal{Z}$
12:     $x \leftarrow G(z;\theta_b)$
13:     $p \leftarrow 1_{\mathcal{A}_{Detect}(x, D_{\mathcal{A}}, D_{\mathcal{A}_1}, D_{\mathcal{A}_2}) = b}$
14:     return $p$
```

Related work has proposed other methods to measure detectability that sample synthetic images from generators trained on the same dataset with different seeds [59]. However, in these approaches, it is unclear whether the detection was successful because the watermark has been detected or because there are some other patterns that make each generator instance identifiable (e.g., a fingerprint). Our notion of detectability can be attributed solely to the impact of the watermark in the synthesized image.

4 Conceptual Approach

This section describes our proposed embedding method for watermarking image generators. We describe improvements of our embedding method over existing methods. Then, we modify two existing watermarks for GANs to enable their embedding into pre-trained generators and propose our own improved GAN watermark. Finally, we propose three attacks against the robustness of watermarking.

4.1 Pivotal Tuning Watermarking

Pivotal Tuning Watermarking (PTW) is a method for watermarking a pre-trained generator. On input of a pre-trained generator θ_G and *watermark decoder* M, an n-bit watermarking message $m \in \{0,1\}^n$, a number of iterations N, a regularization parameter λ_R and a learning rate α, PTW returns a watermarked generator θ_G^* with high output fidelity to the generator before watermarking for the same latent codes.

The watermark decoder neural network extracts messages from images and is used to regularize the generator. Let $M : \mathcal{X} \rightarrow \{0,1\}^n$ be a decoder neural network that extracts n-bit messages from images and $m \in \{0,1\}^n$ is a message that should be embedded. Let λ_R be the strength of the watermark regularization term, α is the learning rate and N be the number of steps to optimize the generator θ_G.

Algorithm 3 Pivotal Tuning Watermarking

```
1: **experiment** PTW($\theta_G, M, m, N, \lambda_R, \alpha$)
2:     $\theta^*_G \leftarrow$ copy parameters from $\theta_G$
3:     **for** $i \in \{1..N\}$ **do**                                   ▷ *embedding loop*
4:         $z \sim \mathcal{Z}$
5:         $x_0 \leftarrow G(z; \theta_G)$
6:         $x \leftarrow G(z; \theta^*_G)$
7:         $g_{\theta^*_G} \leftarrow \nabla_{\theta^*_G} \mathcal{L}_{LPIPS}(x_0, x) + \lambda_R H(M(x), m)$
8:         $\theta^*_G \leftarrow \theta^*_G - \alpha \cdot \mathrm{Adam}(\theta^*_G, g_{\theta^*_G})$           ▷ *update*
9:     **return** $M, \theta^*_G$
```

Algorithm 3 creates a copy of the pre-trained generator's parameters, called the *pivot*, and enters a loop (lines 2-3). A random latent code is drawn and passed through both variants of the generators to produce two images x_0, x (lines 4-6). The loss is computed according to Equation (2) where we compute a binary cross-entropy H on the extracted and target messages (line 7). The optimization tries to encode as many bits of the message as possible into x while minimizing the LPIPS loss to x_0 generated by the pivot. The model parameters are updated iteratively (line 8) and the tunable generator's parameters and the trained decoder are returned. The decoder represents the secret watermarking key (line 9).

Overview. Any image generator that maps from a latent space to images (see Section 2.1) is compatible with PTW. PTW can also be used to embed any image watermarking method [3,64] for that can be learned by a watermark decoder network. We highlight three advantages of PTW over existing embedding methods for GANs [59,60].

1. **Speed**: PTW enables watermarking a pre-trained generator up to three orders of magnitude faster than watermarking from scratch.
2. **No Training Data**: PTW only needs access to the generator, but not to any training data nor the discriminator.
3. **Post-Hoc**: PTW allows embedding watermarks as a post-processing step into any pre-trained generator.

As stated in Algorithm 3, PTW requires access to a watermark decoding network M. This decoding network is crucial for the embedding procedure and we describe different methods of training such a watermark decoder for GANs. We propose our own watermark and then we modify and improve two existing watermarks for GANs to allow embedding them into pre-trained generators as a baseline comparison to our watermark.

4.2 Training a Watermark Decoder

Problem. The goal is to train a decoder neural network $M : \mathcal{X} \rightarrow \{0,1\}^n$ that extracts messages from images. Our decoder should be trained with knowledge of the generator's functionality and learn which pixels can be modified easily to hide messages with minimal impact on the visual image quality of the generator. For example, a generator trained to synthesize faces likely allows encoding more bits per pixel for central pixels in the image that belong to a face, as opposed to pixels at the edge which encode the background. The challenge is that none of the existing GAN architectures allow the input of a message, hence the problem becomes on how to modulate a message to a generator.

Overview. We identify three methods to modulate a message to a generator without changing its architecture. All three options are based on training a deep neural network *encoder*, that takes a message as an input and predicts a perturbation (i) for the generator's parameters [61] or (ii) its inputs [38] which are the latent codes. The first two options are a *weights mapper* M_W and a *bias mapper* M_B, which maps a message to a subset of the generator's weights and biases ($\leq 1\%$ in practice). The weights and bias projectors make some assumptions about the generator's architecture, namely that an intermediate layer defines weight and bias parameters. This assumption is true for generators consisting of convolutional layers, but not all generative image models contain convolutional layers [62].

Algorithm 4 Training our Watermark Decoder

```
**experiment** DecoderTraining($\theta_G, n, N, \lambda_R$)
    $M, M_Z \leftarrow$ random initialization
    **for** $i \in \{1..|\textsc{Conv}(\theta_G)|\}$ **do**
        $M^i_W, M^i_B \leftarrow$ random initialization
    $\theta \leftarrow$ parameters from $\{M, M^i_W, M^i_B, M_Z\}$
    **for** $i \in \{1..N\}$ **do**
        $m \sim \{0,1\}^n$
        $z \sim \mathcal{Z}$
        $x_0 \leftarrow G(z; \theta_G)$
        $\theta^*_G \leftarrow$ copy parameters from $\theta_G$
        $i \leftarrow 0$
        **for** $W, b \in \textsc{Conv}(\theta^*_G)$ **do**
            $W \leftarrow W \cdot M^i_W(m, z)$                          ▷ *weight mapper*
            $b \leftarrow b + M^i_B(m, z)$                             ▷ *bias mapper*
            $i \leftarrow i + 1$
        $\tilde{z} \leftarrow z + M_Z(m, z)$                          ▷ *latent mapper*
        $x \leftarrow G(\tilde{z}; \theta^*_G)$
        $g_\theta \leftarrow \nabla_\theta \mathcal{L}_{\mathrm{LPIPS}}(x_0, x) + \lambda_R H(M(x), m)$
        $\theta \leftarrow \theta - \alpha \cdot \mathrm{Adam}(\theta, g_\theta)$                 ▷ *joint update*
    **return** $M$
```

Hence, our method of training a decoder is architecture-dependent and applicable only to generators containing convolutional layers. However, this procedure can be adapted to generators without convolutional layers by the same principles but using a different set of mappers which we leave to future work. We now describe our decoder training algorithm for an image generator $G : \mathcal{Z} \rightarrow \mathcal{X}$ containing at least one convolutional layer. Let $\textsc{Conv}(\theta)$ be a function that extracts weights and bias parameters from each convolutional layer from a given set of parameters θ.

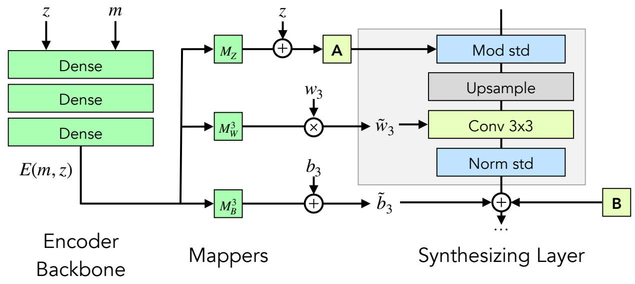


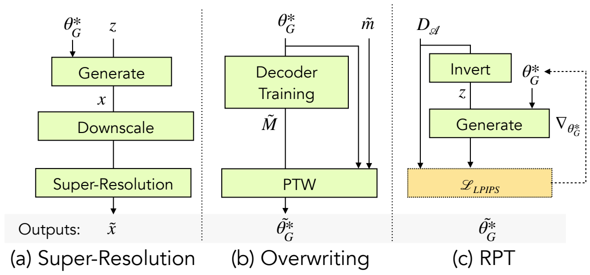
Figure 3: An exemplary illustration of the mappers for a single generator synthesis layer (adapted from [28]). On input of a latent code z and a message m, the mappers (in green) modulate the generator's weights and inputs. $\boxed{\text{A}}$ is an affine transform and $\boxed{\text{B}}$ is random noise sampled during inference.

Training the Decoder. Algorithm 4 encodes the algorithm for training a decoder M given a pre-trained generator θ_G, a watermark message length n, N training steps and a regularization weight λ_R. We randomly initialize the decoder and mapper neural networks (lines 2-4) and set the learnable parameters θ to contain parameters from the watermark decoder and all mappers (line 5). Then, in each step, we sample a message and latent code randomly and generate an image with the provided generator (lines 7-8). We create a copy of the provided generator, iterate through all its convolutional layers and perturb the copied generator's weights using the weight and bias mappers (lines 10-14). Then, we perturb the latent code using the latent mapper and generate an image using the copied generator and the perturbed latent code (lines 16-17). Finally, we compute the loss as stated in Equation (2) and update the parameters of all mappers and the watermark decoder (lines 18-19). After training, we return the watermark decoder (line 20). Figure 3 illustrates the modulation of a message to a synthesizing layer in a generator.

The returned decoder can extract messages from any image. Notably, the decoder learns the number of bits that can be encoded per pixel that causes the least degradation in visual image quality (according to the LPIPS loss). We refer to Figure 1 to observe this effect, where the most perturbed pixels are pixels with high semantic value such as the eyes and nose of a face. Since the decoder depends on the generator instance, a different decoder should be trained for each instance for the best visual quality, but decoders can be re-used if the model architectures are similar. The decoder can be used to embed any message of length n using PTW. Next, we describe modifications to existing watermarks [59,61] that allow embedding them into pre-trained generators.

4.3 Modifying Existing Watermarks

Two existing watermarks for GANs require re-training the generator from scratch to embed a watermark [59, 60]. This section describes modifications to both methods to allow watermarking pre-trained generators as a baseline comparison to our method. First, we briefly summarize both watermarks and then describe our modifications.



(a) Super-Resolution (b) Overwriting (c) RPT

Figure 4: An illustration of our three attacks against the robustness of generator watermarking. RPT stands for Reverse Pivotal Tuning. The function INVERT maps images to latent codes and DOWNSCALE reduces the resolution of an image.

Summary. The first watermark, which we call `Yu1` [59], trains an encoder-decoder network on real images by marking the images with imperceptible patterns. The GAN is trained from scratch on the marked training data. The second watermark, which we call `Yu2` [60], modifies the GAN's training objective and is trained from scratch. Some of their modifications were invented to mitigate training instabilities such as mode collapse [51] or consistency losses, which are problems that do not appear during fine-tuning. The authors modulate a watermark through a weight mapper M_W and embed the watermark by adding the predicted parameter perturbation to the generator, allowing them to generate many differently watermarked generators after the (expensive) re-training procedure. We refer to the author's papers for a more detailed description of their works.

Modifications. The required modifications for `Yu1` are straightforward: Instead of training on real training data, we stamp synthetic data and use their decoder network for embedding a watermark with PTW. For `Yu2`, we ignore all additional losses that address training instabilities or consistency losses and train their weight mapping network instead via fine-tuning on synthetic data while freezing the generator's weights. We embed the `Yu2` watermark using the author's proposed approach by applying the prediction of the trained weight mapper to the generator's parameters.

4.4 Attacks against the Robustness

This section proposes three novel adaptive attacks against the robustness of model watermarking for image generators. Recall from Section 3 that we consider two adversaries: a black-box and a white-box adversary. Previous work [59, 60] assumes only a black-box attacker who can execute any of these five attacks: blurring, cropping, image noising, JPEG compression, or quantization. We refer to Appendix A for a detailed description of all attacks and parameters including those from previous work. Figure 4 illustrates our proposed black box and two white-box attacks.

4.4.1 Black-box Attacks

Our black-box attacker first scales the resolution of an image down by a factor ρ and then uses *super-resolution* models [43] to upscale the image to its previous resolution. A super-resolution model can upscale images by interpolating details that are not sharp in the low-resolution image. Such super-resolution models enable an attacker to apply stronger perturbations to images in an attempt to remove their watermark with a smaller impact on the image quality. Super-Resolution models have been demonstrated to generalize well to out-of-domain data, meaning that the attacker does not need any access to the generator's training data. While our attacks use pre-trained models from related work [43] to achieve super-resolution, we are the first to apply super-resolution models to undermine watermarking in image generators. Previous attacks [59–61] use image augmentation techniques such as blurring or noising to remove the watermark.

4.4.2 White-box Attacks

We propose two adaptive white-box attacks called *overwriting* and *Reverse Pivotal Tuning* (RPT). In the overwriting attack, the attacker trains their own watermark decoder (see Algorithm 4) and then uses PTW for watermarking the generator using a random message. The success of the overwriting attack depends on the similarity between the defender's watermarking key τ and the attacker's watermarking key $\tilde{\tau}$. The overwriting attack can be successful in removing the watermark if both keys modulate similar pixels in the input.

Algorithm 5 Reverse Pivotal Tuning (RPT) Attack

```
1: procedure RPT($\theta^*_G, D^R, N, \alpha$)
2:     $Z \leftarrow \{\text{INVERT}(x, \theta^*_G) | x \in D^R\}$
3:     for $i \in \{1..N\}$ do
4:         $j \sim \{1..|Z|\}$
5:         $g_{\theta^*_G} \leftarrow \nabla_{\theta^*_G} \mathcal{L}_{\text{LPIPS}}(G(Z_j; \theta^*_G), D^R_j)$
6:         $\theta^*_G \leftarrow \theta^*_G - \alpha \cdot \text{Adam}(\theta, g_{\theta^*_G})$
7:     return $\theta^*_G$
1: procedure INVERT($x, \theta^*_G$)
2:     return $\arg\min_{z \in \mathcal{Z}} \mathcal{L}_{\text{LPIPS}}(G(z; \theta^*_G), x)$
```

Algorithm 5 implements our RPT attack. The attacker has access to a watermarked generator θ^*_G, a limited set of R non-watermarked, real images D^R and performs the RPT attack for N steps with a learning rate α. Their goal is to regularize the generator to synthesize images that are visually similar to their real images (i.e., they have high visual quality), but do not retain a watermark. The RPT attack consists of two stages: (1) inversion of the real images (line 5) and (2) Pivotal Tuning so that the inverted images have a high visual similarity to the real images (lines 5-6). RPT should be successful with the availability of many non-watermarked images.

5 Evaluation

We describe our experimental setup and specify our measured quantities, namely the capacity, utility, detectability, and robustness. Then, we measure detectability and robustness (see Section 3) and compare our watermarking method to the modified methods from related work (see Section 4.3).

5.1 Setup

Datasets. We experiment with three datasets for which pre-trained, high-quality generators have been made publicly available. FFHQ [27] consists of 70k human faces in various poses. We experiment with a lower-resolution version of the dataset at 256^2 pixels, which we refer to as FFHQ-256, and the high-quality version FFHQ at $1\,024^2$ pixels. In addition, we experiment with AFHQv2 [8] which consists of roughly 16k animal images and has a resolution of 512^2 pixels.

Pre-Trained Generators. We experiment with three StyleGAN-based architectures: StyleGAN2 [27], StyleGAN3 [26] and StyleGAN-XL [45]. We select the StyleGAN architectures because (1) this architecture achieves state-of-the-art FID values on the surveyed datasets and (2) many high-quality model checkpoints are publicly available that have been trained with different seeds[3][4]. State-of-the-art image generator architectures are evaluated using the same checkpoints that we are using. Therefore, the image quality of our watermarked generated images can be compared to the image quality in ongoing research on image generation models.

Framework. We implement all watermarking methods from scratch in PyTorch 1.13. While implementations for `Yu1`[5] and `Yu2`[6] exist, we could not reproduce their results with the provided implementation. `Yu1` never converges and `Yu2` is implemented in Tensorflow version 1, which is no longer supported by modern GPUs, meaning we cannot re-use their source code or load the provided generator checkpoints.

5.2 Evaluation Criteria

Utility. Similar to existing work [27] we measure the utility of a generator by its Fréchet Inception Distance (FID) [22]. Lower FID indicates a higher utility. Similar to Karras et al. [27], we measure FID between $50,000$ generated and real images. For AFHQv2 we use only $16,000$ real images due to the limited dataset size.

[3] https://github.com/NVlabs/stylegan3

[4] https://github.com/autonomousvision/stylegan-xl

[5] https://github.com/ningyu1991/ArtificialGANFingerprints

[6] https://github.com/ningyu1991/ScalableGANFingerprints

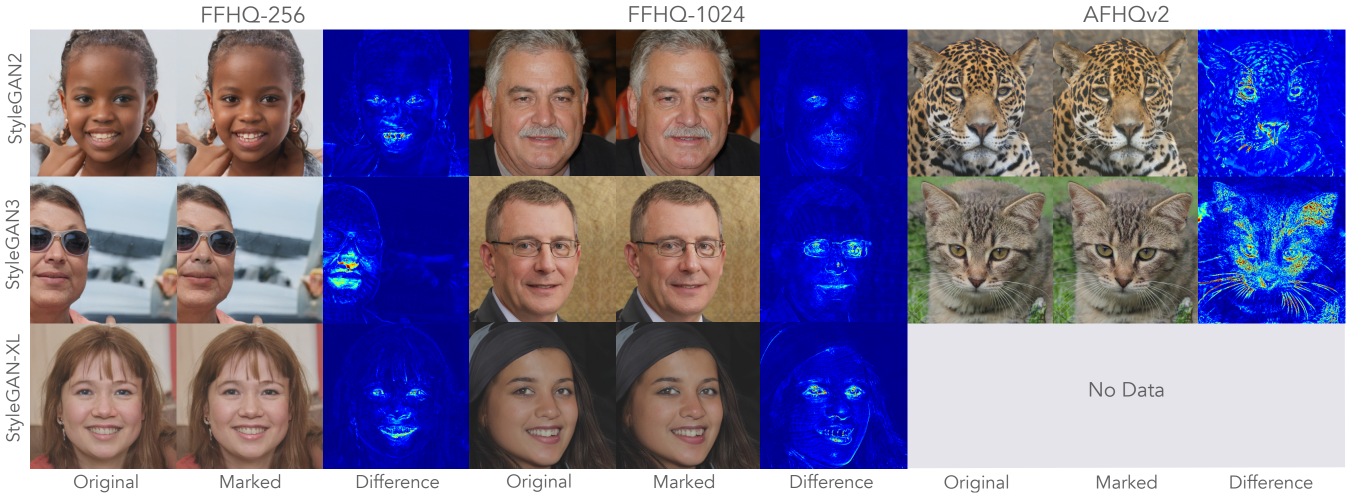


Figure 5: Images synthesized using our watermarked generators on different datasets and model architectures. We show the image synthesized by the generator (i) before and (ii) after watermarking, and (iii) the difference between the watermarked and non-watermarked images. StyleGAN-XL does not provide a pre-trained model checkpoint for AFHQv2.

Capacity. We measure the capacity of a watermark in bits by the difference in the expected number of correctly extracted bits from watermarked and non-watermarked images. The expected rate of correctly extracted bits equals 0.5 for non-watermarked images assuming messages are sampled uniformly at random. Let $m \in \{0,1\}^n$ be a message, τ the secret watermarking key, and θ are the parameters of a generator. The capacity of the generator is computed as follows.

$$C_\theta = n \cdot \mathop{\mathbb{E}}_{z \in Z}\left[\textsc{Verify}(\textsc{Extract}(G(z;\theta),\tau)) - 0.5\right] \qquad (5)$$

It is straightforward to achieve a high capacity by overwriting a significant portion of the host image. However, this approach also decreases the visual image quality which can be measured and visualized as the capacity/utility trade-off.

Decision Threshold. We consider a watermark to be *removed*, if we can reject the null hypothesis H_0 with a p-value less than 0.05. The null hypothesis states that k matching bits were extracted from the synthetic images by random chance. Quantitatively, the probability of this event is calculated as $\Pr(X > k|H_0) = \sum_{i=k}^{n} \binom{n}{i} 0.5^n$. In practice, for a watermark with $n = 40$ bits, we need to extract at least 26 bits correctly, meaning that we verify the presence of a watermark by correctly extracting $C_\theta \geq 6$ in bits.

5.3 Runtime Analysis

To calculate the speed-up of PTW over existing watermarking methods [60, 61], we compare it with training non-watermarked generators from scratch. This comparison is fair, as watermarking is not expected to decrease a generator's training time. We estimate the total runtimes in GPU hours using the suggested hyper-parameters in the relevant GAN papers [26, 28, 45] on 8xA100 GPUs.

Model	StyleGAN2	StyleGAN3	StyleGAN-XL
FFHQ-256	158h	482h	552h
FFHQ-512	384h	662h	1285h
FFHQ-1024	929h	1161h	1456h

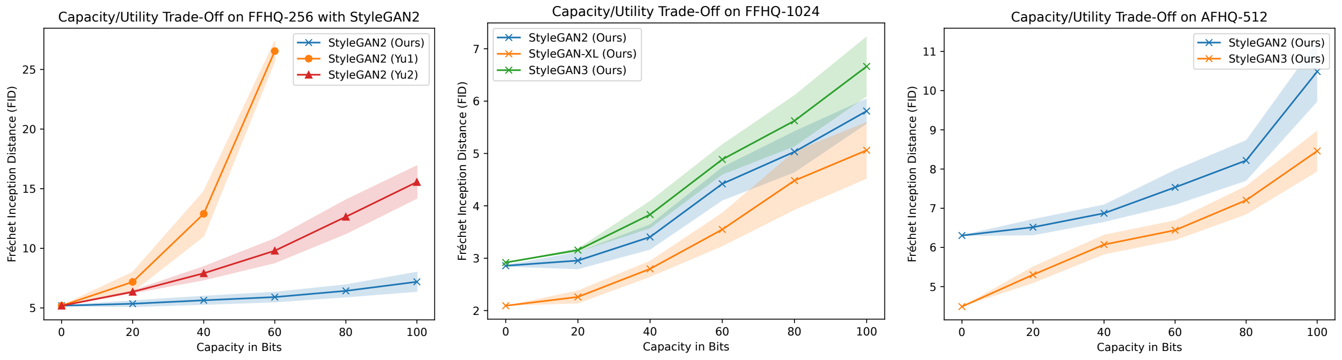
Table 2: GPU hours required for training generators without watermarking from scratch on FFHQ [27] on 8xA100 GPUs.

Table 2 shows the estimated training runtimes from scratch for each generator on FFHQ [28] at varying pixel resolutions. For instance, training a StyleGAN-XL model on FFHQ at a resolution of 256^2 pixels requires 552 GPU hours. With PTW, watermarking a pre-trained generator on FFHQ requires only about 0.5 GPU hours which is a three-orders of magnitude improvement for high-resolution generators. Our approach also requires training the watermarking decoder (see Algorithm 4), which is a one-time upfront cost of about 2 GPU hours.

5.4 Capacity/Utility Trade-off

This section summarizes our results on the capacity/utility trade-off on various datasets, model architectures, and in comparison to two existing, modified watermarks: `Yu1`, and `Yu2`.

Visual Inspection. Figure 5 shows images synthesized by our watermarked generators on all three surveyed datasets. The columns show the original image synthesized before watermarking, the image synthesized after watermarking and their differences in the form of a heatmap. Heavily modified regions are highlighted in yellow and red. In both versions of the facial image datasets, we observe that our watermark focuses on pixels located on the face of the generated person, most prominently its eyes. Upon closer inspection, the net-



(a) FFHQ-256.

(b) FFHQ-1024.

(c) AFHQv2-512.

Figure 6: The capacity/utility trade-off. Figure 6a shows our watermark in comparison to two existing, modified watermarks on FFHQ-256 for StyleGAN2. Figure 6b shows our watermarking for on FFHQ-1024 using three different generator architectures. Figure 6c shows our watermark on a different domain than faces (AFHQv2 [8], wild animals) using two different generator architectures. The shaded area represents the standard deviation (N=3) in all Figures.

work modifies the eyes and mouth area of a face strongest and is invariant to the location of the face in the image. For AFHQv2, we observe that the pixels are more spread out onto the entire image. In the next subsection, we compare our watermark quantitatively to other existing watermarks.

Comparison to Existing Watermarks. Figure 6a shows the trade-off on FFHQ-256 for different watermarking methods using a StyleGAN2 architecture. We plot the embedded bits C_θ against the FID. Our method outperforms the other two approaches substantially, as we can embed 100 bits with a similar loss in utility as embedding 20 bits using Yu2. In contrast, Yu1 is not competitive, even though it employs PTW as its embedding strategy. Upon analyzing the generated images, we observed that Yu1 is not sensitive to the capacity per pixel and attempts to encode many bits of the message into background pixels, which noticeably deteriorates the generator's quality. We believe this type of watermarking strategy works better when the entire generator is re-trained from scratch, as the generator can learn to allocate capacity to arbitrary pixels. For FFHQ-256, using our watermark, encoding 40 bits only worsens the FID by approximately 0.3 points.

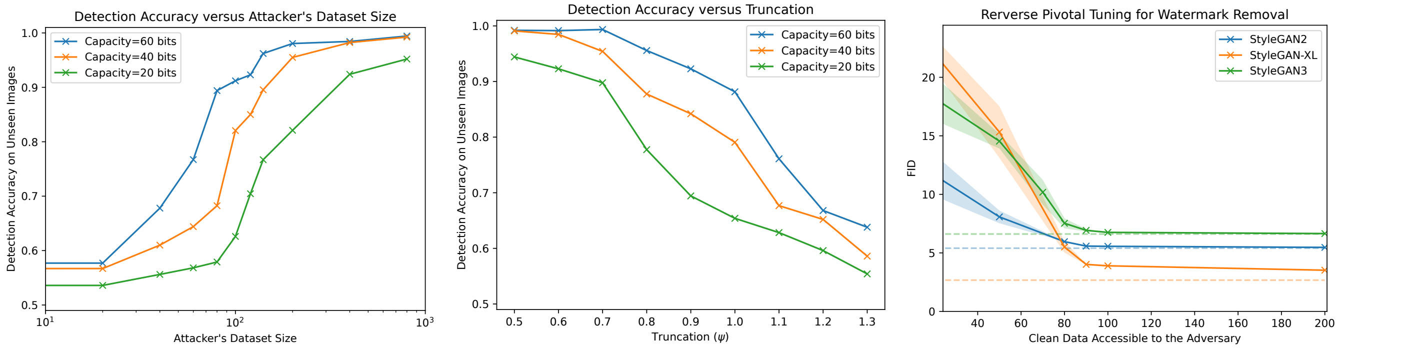
Watermarking High-Quality Generators. Figure 6b shows the trade-off using our watermark across three generator architectures on FFHQ-1024. Compared to FFHQ-256 which has a FID of over 5, the generators trained on FFHQ-1024 have a much lower FID of less than 3. Our watermark embeds up to 40 bits with little loss in utility, but the FID deteriorates quickly when embedding more than 40 bits. For StyleGAN2, we measure a FID deterioration of almost 3 points when embedding 100 bits. We believe the effect on the FID is greater for the high-quality dataset due to two factors: (1) high-quality images with a low FID may be more sensitive to modifications, and (2) our watermark decoder downscales images to 224^2 pixels which means that our decoder cannot extract more information from larger images. Our decoder is a ResNet18 [21] model designed for this resolution. Nonetheless, we demonstrate that watermarking high-quality generators is possible using our method.

Watermarking Different Domains. Figure 6c shows the capacity/utility trade-off across two generator architectures for the domain of animal images. We cannot evaluate StyleGAN-XL on AFHQv2 because no pre-trained checkpoint was made available for this dataset. Our goal is to demonstrate that our watermark is not restricted to just the facial image domain. Figure 6c shows that our approach can embed watermarks up to 100 bits, although we observe a strong deterioration in FID of more than 4 points, at which point the watermark is (barely) visually perceptible. While it is possible to embed 100 bits, given our results, we believe that 40 bits are more practically relevant as the deterioration in FID is less than one point for StyleGAN2 and the watermark is not easily perceptible. Interestingly, the deterioration in FID is stronger for the animal domain which we attribute to a larger output diversity. AFHQv2 contains images of multiple different animal species in diverse poses.

5.5 Detectability

Our first experiment measures the detectability of our watermark at different capacities. We recall from our security game in Algorithm 2 that the attacker has access to R_1, R_2 non-watermarked and watermarked images. In our experiments, we set $R_1 = R_2$ and evaluate the detectability of images synthesized by generators that have been watermarked with varying capacities. We use a standard, pre-trained ResNet-18 and fine-tune it for the detection task using an Adam optimizer.

Detectability versus Dataset Size. Figure 7a shows the detection accuracy p of our attack against the number of labeled



(a) (b) (c)

Figure 7: (a) The detection accuracy in relation to the adversary's dataset size for different capacities without truncation. (b) The detection accuracy plotted against truncation when fixing the adversary's dataset size to ≤ 100 labeled images. (c) An ablation study for the number of real, non-watermarked data used during our adaptive Reverse Pivotal Tuning attack. The shaded areas denote the standard deviation (N=3) and the dashed horizontal lines show the generator's FID before the attack.

images available to the adversary. As expected, a higher watermark capacity results in a higher detectability of watermarked images. We observe that an adversary with access to ≥ 400 labeled images has a classification accuracy of over 90% in classifying watermarked/non-watermarked images for any capacity. An adversary with access to ≤ 100 images cannot reliably detect watermarked images if the capacity is at most 40 bits. Our results show that the detection algorithm is not successful at detecting our watermark unless the attacker has access to a relatively large set of labeled, non-watermarked images. Next, we evaluate the influence of the latent code's sampling strategy on the watermark detectability.

Truncation Trick. Generating an image from a GAN requires sampling a latent code. The *truncation trick* [4] is a technique used for generative models to limit the range of values for a latent code and allows for controlling the diversity and quality of the generated images. If the truncation threshold ψ is low, samples will have a high similarity to the real training data, but limited diversity meaning they may appear similar to each other. We identify that truncation plays a significant role in the ability of an adversary to detect watermarks. Figure 7b shows that the detection accuracy decreases with an increasing diversity of the generator's output when fixing the number of samples available to the adversary.

5.6 Robustness

We evaluate the watermark's robustness evaluated against several types of attacks, including (1) black-box attacks like cropping, blurring, quantization, noising, JPEG compression, and our super-resolution attack, and (2) two white-box attacks, overwriting, and Reverse Pivotal Tuning (see Section 4.4). We refer to Appendix A for a description of the attacks and parameters we use during our evaluation. All attacks are evaluated against generators that have been watermarked with a capacity of at least $C_\theta \geq 40$ bits. Embedding 40 bits only deteriorates the generator's FID by about 0.3 points on average for StyleGAN2 on FFHQ.

5.6.1 Black-box Attacks

Latent Space Analysis. We examine whether there are points in the generator's latent space that synthesize high-quality images without a watermark. If such points exist, an attacker could attempt to find them and sample the generator on these points. We test for such latent subspaces using three sampling methods: (i) truncation, (ii) latent interpolation, and (iii) style-mixing (for the StyleGAN architectures). Truncation restricts the distance of a latent code to the global average. Latent interpolation samples points on a line between latent codes and style-mixing combines intermediate latent codes $w \in \mathcal{W}$ and feeds them to the generator [27]. Our results show that the mean capacity remains unaffected by the sampling method, meaning we were able to successfully extract the watermark message in all cases. We conclude from these results that the watermark generalizes to the generator's entire latent space.

Removal Attacks. Next, we perform all surveyed removal attacks against all watermarked generators and measure the evasion rate and FID with $K = 50,000$ synthetic images. A summary of all black-box attacks is shown as a scatter plot in Figure 8. The Figure shows the remaining capacity after an attack on the x-axis and utility (measured by the FID) on the y-axis. The Pareto front, which represents the optimal trade-off between capacity and utility, is highlighted and represents the best attack out of all surveyed attacks that an adversary could choose. We find that none of the black-box attacks are effective at removing a watermark, but our super-resolution attack is always part of the Pareto Front.

The Pareto front represents the best capacity/utility trade-

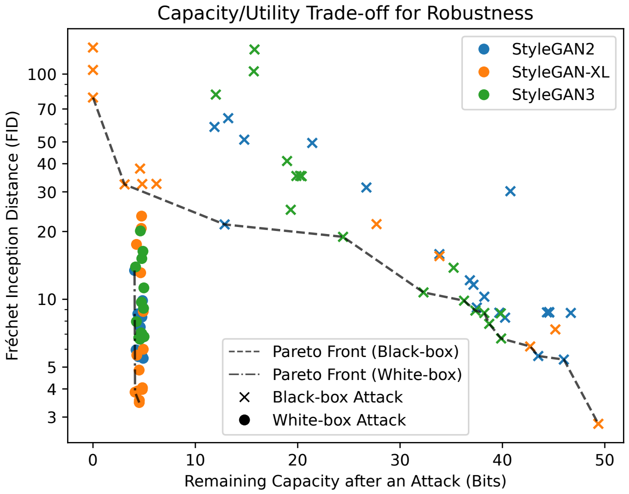


Figure 8: This Figure shows the robustness of our watermark against all surveyed attacks. We highlight black-box and white-box attacks that are members of the Pareto front.

	StyleGAN2		StyleGAN-XL		StyleGAN3	
	C_θ	FID	C_θ	FID	C_θ	FID
Attacks	43.05	5.4	48.79	2.67	40.33	6.61
Black-box Attacks						
Crop	39.73	8.72	42.71	6.18	38.23	8.69
Blur	38.82	36.84	12.12	10.32	35.12	11.73
JPEG	42.12	8.70	38.43	9.12	38.23	9.33
Noise	40.26	8.29	45.17	7.35	32.29	10.73
Quantize	37.17	11.60	43.27	5.61	39.72	8.71
SR	32.86	11.51	34.52	11.62	30.12	11.34
White-box Attacks						
Overwrite	4.78	8.34	4.91	8.83	4.73	9.71
RPT_{200}	4.91	5.47	4.52	3.52	4.59	6.65
RPT_{100}	4.44	5.56	4.21	3.90	4.47	6.75
RPT_{50}	4.38	8.07	4.38	15.32	4.16	14.47

Table 3: The capacity and FID of all surveyed attacks. We ablate over multiple parameters for each attack and this table shows the points with the best (i.e., lowest) FID. RPT_R stands for the Reverse Pivotal Tuning attack using R real samples.

off a black-box attacker can achieve using these attacks. For example, a black-box attacker can reduce the capacity by 10 bits from 50 to 40, but in doing so reduces the FID by over 6 points. Our super-resolution attack is on the Pareto front but cannot remove the watermark. Removal is only possible when the FID drops to 30, at which point the image quality has been compromised. Table 3 summarizes the best-performing black-box attacks for the three evaluated generator architectures. Each attack has a single parameter that we ablate over using grid search. We refer to Appendix A for a detailed description of all attacks and parameters we used in this ablation. Table 3 lists those data points that either remove the watermark ($C_\theta <$ 5) or, if the watermark cannot be removed, the data point with the lowest FID. None of the black-box attacks, including our super-resolution attack, are successful in removing the watermark while preserving the generator's utility.

5.6.2 White-box Attacks

Overwriting. Table 3 shows that overwriting can remove watermarks but deteriorates the generator's image quality, measured using FID, by approximately 3 points for StyleGAN2 and 6 points for StyleGAN-XL. Such a deterioration in FID likely prevents attacks in practice because low-quality deepfakes are more easily detectable. Our overwriting attack also implicitly assumes knowledge of the defender's watermarking method which may not be the case in practice. Overwriting could cause a greater decline in FID if the attacker's and defender's watermarking methods differ.

Reverse Pivotal Tuning. Our Reverse Pivotal Tuning (RPT) attack is substantially more effective than the overwriting attack as it preserves the FID of the generator to a greater extent. We found that an attacker with access to 200 real, non-watermarked images is capable of removing any watermark without causing a noticeable deterioration in FID. This means that with access to less than 0.3% of the training dataset, a white-box adversary can remove any watermark. In the case of StyleGAN-XL, using 200 images leads to a decrease in FID of less than one point (from 2.67 to 3.52).

Ablation Study for RPT. Figure 7c shows an ablation study over the amount of real, non-watermarked training data required by an attacker to remove a watermark. We measured these curves as follows: We randomly sample a set of R real images and run the RPT attack encoded by Algorithm 5 with gradually increasing weight λ_{LPIPS} on the LPIPS loss until the watermark is removed. Then we compute the FID on $K = 50{,}000$ images. In all experiments, the watermark is eventually removed but access to more data has a significant impact on the FID that is retained in the generator after the attack. For StyleGAN2, we find that 80 images ($\approx 0.1\%$ of the training data) are sufficient to remove the watermark at less than 0.3 points of deterioration in FID, which represents a visually imperceptible quality degradation. Our results demonstrate that an adaptive attacker with access to the generator's parameters can remove any watermark using only a small number of clean, non-watermarked images and can pose a threat to the trustworthiness of watermarking.

6 Discussion

This section discusses the limitations of watermarking and our study, the extension of our work to other image generators, and ethical considerations from releasing our attacks.

Non-Cooperative Providers. Our study demonstrates that watermarking for image generators can be robust under certain threat models (as detailed in Section 3). For watermarking to effectively control misuse, every provider has to cooperate and watermark their generators prior to disclosing them. However, this is unlikely to occur in practice [55]. A capable adversary with access to sufficient training data and computational resources can always train their own generator without a watermark and provide it to others. It is important to acknowledge this inherent limitation of watermarking methods, as they cannot prevent this scenario. Nonetheless, they can act as a deterrent to adversaries who lack the ability to train their own generators from synthesizing harmful deepfakes.

Watermarking from Scratch. We focus on the robustness and undetectability of watermarking methods on pre-trained image generators due to the substantially higher scalability compared to watermarking from scratch. Further research is needed to explore the potential impact of watermarking from scratch on robustness and undetectability.

Watermarking for Intellectual Property Protection. Watermarking has also been used in the context of Intellectual Protection (IP) protection of neural networks [32,37,52]. The IP protection threat model typically assumes white-box access of the adversary to the target generator and black-box API access of the defender to the adversary's generator to verify a watermark. For example, Ong et al. [37] embed a *backdoor* into a generator that synthesizes images containing a watermark when the generator is queried with certain latent codes. All watermarks evaluated in this paper assume no-box access to the target generator, meaning that our watermarks can be extracted from any of the generator's synthetic images. We show that existing no-box watermarks are not robust in the white-box setting, meaning that they are likely not suitable candidates for IP protection of generators in practice.

Other Generator Models. Recently, different image generators such as DALL·E 2 [41] based on the Transformer architecture [53], and latent diffusion models [43] have been shown to synthesize high-quality deepfakes. While OpenAI's DALL·E 2 generator is only accessible through a black-box API[7], model checkpoints for latent diffusion are publicly available as a white-box[8]. The utility of these checkpoints on FFHQ is comparable to that of the StyleGAN model checkpoints used in this paper (e.g., latent diffusion reports a FID score of 4.98). DALL·E 2 and Latent Diffusion models also map from a latent space to images, but they accept auxiliary input such as text that controls the synthesis. While PTW is compatible with any image generator that maps latent codes to images, extending our work to other models requires developing a watermarking method that can map to their parameters (see Section 4.2), which we leave to future work. Our work demonstrates for one state-of-the-art generator model architecture (StyleGAN-based generators with a FID score of 2.1) that they can be watermarked effectively using our method.

Summary of Deepfake Detection. Our research reveals that existing watermarks are not robust against an adversary with white-box access to the generator, but can withstand a black-box adversary. This means that watermarking can be a viable solution for deterring deepfakes if the provider acts responsibly and the generator is provided through a black-box deployment. The provider's goal is to deter model misuse which can be accomplished through several means. These include (1) monitoring and restricting queries, (2) relying on passive deepfake detection methods or (3) implementing proactive methods such as watermarking. Monitoring lacks transparency and can deter usage of the model if the user does not trust the provider [12,50]. Passive detection methods may be unable to detect deepfakes as the quality of synthesized images improves or the adversary adapts to existing detectors [14]. Active methods enable a different type of deployment that (1) does not require query monitoring and (2) remains applicable to future, higher-quality generators. The provider and the user agree on a mutually trusted third party to deploy the generator, who does not tamper with the watermark nor monitor the queries. Our research suggests that such a black-box deployment represents a viable option in practice to prevent model misuse using existing watermarks.

Ethical Consideration. Deep image generators can have potential negative societal impacts when misused, for instance, when generating harmful deepfakes. Our contributions are intended to raise awareness about the limited trustworthiness of watermarking in potential future deployments of image generators, rather than to undermine real systems. While the attacks presented in our paper could be used to evade watermarking thereby enabling misuse, we believe that sharing our attacks does not cause harm at this time, since there are no known deployments of the presented watermarking methods. We aim to advance the development of watermarking methods that cannot be broken by our attacks.

7 Conclusion

We propose Pivotal Tuning Watermarking (PTW), which is a scalable method for watermarking pre-trained image generators. Watermarking can be a promising long-term solution to deepfake detection if the model providers are cooperative and deploy only watermarked generators. PTW is three orders of magnitude faster than related work and enables watermarking generators 4× larger than without the need for any training data. We find that our watermark is undetectable to an adversary without the secret watermarking key. Watermarking is robust against all surveyed black-box attacks, but not against an adaptive white-box attacker. Such an adaptive attacker can remove watermarks with almost no impact on image quality using less than 0.3% of the training data with our adaptive Reverse Pivotal Tuning (RPT) attack. Our results challenge that watermarking prevents model misuse when the parameters of a generator are provided. We hope that PTW advances the development of trustworthy watermarking methods.

[7] https://openai.com/dall-e-2/

[8] https://github.com/CompVis/latent-diffusion

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1615–1631, 2018.

[2] Shruti Agarwal, Hany Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. Protecting world leaders against deep fakes. In *CVPR workshops*, volume 1, page 38, 2019.

[3] Ali Al-Haj. Combined dwt-dct digital image watermarking. *Journal of computer science*, 3(9):740–746, 2007.

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[5] Tu Bui, Ning Yu, and John Collomosse. Repmix: Representation mixing for robust attribution of synthesized images. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XIV*, pages 146–163. Springer, 2022.

[6] Zhipeng Cai, Zuobin Xiong, Honghui Xu, Peng Wang, Wei Li, and Yi Pan. Generative adversarial networks: A survey toward private and secure applications. *ACM Computing Surveys (CSUR)*, 54(6):1–38, 2021.

[7] Han Chen, Yuezun Li, Dongdong Lin, Bin Li, and Junqiang Wu. Watching the big artifacts: Exposing deepfake videos via bi-granularity artifacts. *Pattern Recognition*, 135:109179, 2023.

[8] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197, 2020.

[9] Dustin T Crystal, Nicholas G Cuccolo, Ahmed Ibrahim, Heather Furnas, and Samuel J Lin. Photographic and video deepfakes have arrived: how machine learning may influence plastic surgery. *Plastic and reconstructive surgery*, 145(4):1079–1086, 2020.

[10] Adrienne De Ruiter. The distinct wrong of deepfakes. *Philosophy & Technology*, 34(4):1311–1332, 2021.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Jennifer Ding, Christopher Akiki, Yacine Jernite, Anne Lee Steele, and Temi Popo. Towards openness beyond open access: User journeys through 3 open ai collaboratives. *arXiv preprint arXiv:2301.08488*, 2023.

[13] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset. *arXiv preprint arXiv:2006.07397*, 2020.

[14] Chengdong Dong, Ajay Kumar, and Eryun Liu. Think twice before detecting gan-generated fake images from their spectral domain imprints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7865–7874, 2022.

[15] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Ting Zhang, Weiming Zhang, Nenghai Yu, Dong Chen, Fang Wen, and Baining Guo. Protecting celebrities from deepfake with identity consistency transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9468–9478, 2022.

[16] Sandy Engelhardt, Lalith Sharan, Matthias Karck, Raffaele De Simone, and Ivo Wolf. Cross-domain conditional generative adversarial networks for stereoscopic hyperrealism in surgical training. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2019: 22nd International Conference, Shenzhen, China, October 13–17, 2019, Proceedings, Part V*, pages 155–163. Springer, 2019.

[17] Joel Frank, Thorsten Eisenhofer, Lea Schönherr, Asja Fischer, Dorothea Kolossa, and Thorsten Holz. Leveraging frequency analysis for deep fake image recognition. In *International conference on machine learning*, pages 3247–3258. PMLR, 2020.

[18] Sharath Girish, Saksham Suri, Sai Saketh Rambhatla, and Abhinav Shrivastava. Towards discovery and attribution of open-world gan generated images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14094–14103, 2021.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] Drew Harwell. Scarlett johansson on fake ai-generated sex videos:'nothing can stop someone from cutting and pasting my image'. *Washington Post*, 31:12, 2018.

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

[22] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[23] Emmie Hine and Luciano Floridi. New deepfake regulations in china are a tool for social stability, but at what cost? *Nature Machine Intelligence*, 4(7):608–610, 2022.

[24] Shu Hu, Yuezun Li, and Siwei Lyu. Exposing gan-generated faces using inconsistent corneal specular highlights. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2500–2504. IEEE, 2021.

[25] Yonghyun Jeong, Doyeon Kim, Seungjai Min, Seongho Joe, Youngjune Gwon, and Jongwon Choi. Bihpf: bilateral high-pass filters for robust deepfake detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 48–57, 2022.

[26] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[27] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[28] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[29] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] Lingzhi Li, Jianmin Bao, Ting Zhang, Hao Yang, Dong Chen, Fang Wen, and Baining Guo. Face x-ray for more general face forgery detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5001–5010, 2020.

[31] Nils Lukas, Edward Jiang, Xinda Li, and Florian Kerschbaum. Sok: How robust is image classification deep neural network watermarking? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 787–804. IEEE, 2022.

[32] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. Deep neural network fingerprinting by conferrable adversarial examples. *International Conference on Learning Representations*, 2021.

[33] Francesco Marra, Diego Gragnaniello, Luisa Verdoliva, and Giovanni Poggi. Do gans leave artificial fingerprints? In *2019 IEEE conference on multimedia information processing and retrieval (MIPR)*, pages 506–511. IEEE, 2019.

[34] Falko Matern, Christian Riess, and Marc Stamminger. Exploiting visual artifacts to expose deepfakes and face manipulations. In *2019 IEEE Winter Applications of Computer Vision Workshops (WACVW)*, pages 83–92. IEEE, 2019.

[35] Jaron Mink, Licheng Luo, Natã M Barbosa, Olivia Figueira, Yang Wang, and Gang Wang. Deepphish: Understanding user trust towards artificially generated profiles in online social networks. In *Proc. of USENIX Security*, 2022.

[36] Yisroel Mirsky and Wenke Lee. The creation and detection of deepfakes: A survey. *ACM Computing Surveys (CSUR)*, 54(1):1–41, 2021.

[37] Ding Sheng Ong, Chee Seng Chan, Kam Woh Ng, Lixin Fan, and Qiang Yang. Protecting intellectual property of generative adversarial networks from ambiguity attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3630–3639, 2021.

[38] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2085–2094, 2021.

[39] Fabi Prezja, Juha Paloneva, Ilkka Pölönen, Esko Niinimäki, and Sami Äyrämö. Deepfake knee osteoarthritis x-rays from generative adversarial neural networks deceive medical experts and offer augmentation potential to automatic classification. *Scientific Reports*, 12(1):18573, 2022.

[40] Yuyang Qian, Guojun Yin, Lu Sheng, Zixuan Chen, and Jing Shao. Thinking in frequency: Face forgery detection by mining frequency-aware clues. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII*, pages 86–103. Springer, 2020.

[41] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[42] Daniel Roich, Ron Mokady, Amit H Bermano, and Daniel Cohen-Or. Pivotal tuning for latent-based editing of real images. *ACM Transactions on Graphics (TOG)*, 42(1):1–13, 2022.

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[44] Andreas Rossler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Faceforensics++: Learning to detect manipulated facial images. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1–11, 2019.

[45] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH 2022 conference proceedings*, pages 1–10, 2022.

[46] Zeyang Sha, Zheng Li, Ning Yu, and Yang Zhang. Defake: Detection and attribution of fake images generated by text-to-image diffusion models. *arXiv preprint arXiv:2210.06998*, 2022.

[47] Jessica Silbey and Woodrow Hartzog. The upside of deep fakes. *Md. L. Rev.*, 78:960, 2018.

[48] Vera Sorin, Yiftach Barash, Eli Konen, and Eyal Klang. Creating artificial images for radiology applications using generative adversarial networks (gans)–a systematic review. *Academic radiology*, 27(8):1175–1185, 2020.

[49] Sebastian Szyller, Vasisht Duddu, Tommi Gröndahl, and N Asokan. Good artists copy, great artists steal: Model extraction attacks against image translation generative adversarial networks. *arXiv preprint arXiv:2104.12623*, 2021.

[50] Alex Tamkin, Miles Brundage, Jack Clark, and Deep Ganguli. Understanding the capabilities, limitations, and societal impact of large language models. *arXiv preprint arXiv:2102.02503*, 2021.

[51] Hoang Thanh-Tung and Truyen Tran. Catastrophic forgetting and mode collapse in gans. In *2020 international joint conference on neural networks (ijcnn)*, pages 1–10. IEEE, 2020.

[52] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pages 269–277, 2017.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[54] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

[55] Mika Westerlund. The emergence of deepfake technology: A review. *Technology innovation management review*, 9(11), 2019.

[56] Xi Wu, Zhen Xie, YuTao Gao, and Yu Xiao. Sstnet: Detecting manipulated faces through spatial, steganalysis and temporal features. In *ICASSP 2020-2020 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 2952–2956. IEEE, 2020.

[57] Tianyun Yang, Juan Cao, Qiang Sheng, Lei Li, Jiaqi Ji, Xirong Li, and Sheng Tang. Learning to disentangle gan fingerprint for fake image attribution. *arXiv preprint arXiv:2106.08749*, 2021.

[58] Tianyun Yang, Ziyao Huang, Juan Cao, Lei Li, and Xirong Li. Deepfake network architecture attribution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4662–4670, 2022.

[59] Ning Yu, Larry S Davis, and Mario Fritz. Attributing fake images to gans: Learning and analyzing gan fingerprints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7556–7566, 2019.

[60] Ning Yu, Vladislav Skripniuk, Sahar Abdelnabi, and Mario Fritz. Artificial fingerprinting for generative models: Rooting deepfake attribution in training data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14448–14457, 2021.

[61] Ning Yu, Vladislav Skripniuk, Dingfan Chen, Larry Davis, and Mario Fritz. Responsible disclosure of generative models using scalable fingerprinting. *arXiv preprint arXiv:2012.08726*, 2020.

[62] Bowen Zhang, Shuyang Gu, Bo Zhang, Jianmin Bao, Dong Chen, Fang Wen, Yong Wang, and Baining Guo. Styleswin: Transformer-based gan for high-resolution image generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11304–11314, 2022.

[63] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[64] Xin Zhong, Pei-Chi Huang, Spyridon Mastorakis, and Frank Y Shih. An automated and robust image watermarking scheme based on deep neural networks. *IEEE Transactions on Multimedia*, 23:1951–1961, 2020.

[65] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain gan inversion for real image editing. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVII 16*, pages 592–608. Springer, 2020.

A Attack Description

This section describes the attacks and shows which parameters we explored in our grid search. We refer to the image before attacking as the *base* image x and as the *attacked* image after the attacker performs their attack $\tilde{x} = \mathcal{A}(x, I)$ with auxiliary information I. We find the range of evaluated parameters through experimentation by limiting the degradation of the attack on the visual quality of the images.

A.1 Black-box Attacks

Black-box attacks assume only black-box API access to the *target generator*, meaning that they can query the generator on arbitrary latent codes $z \in \mathcal{Z}$, but they have no knowledge of or control over the generator's parameters or its intermediate activations.

Algorithm 6 Super-Resolution Attack

```
procedure SUPER-RESOLUTION($x, \rho$)
    $\tilde{x} \leftarrow$ RESIZE($x, \lfloor$RESOLUTION$(x) \cdot \rho\rfloor$)
    while RESOLUTION($\tilde{x}$) $<$ RESOLUTION($x$) do
        $\tilde{x} \leftarrow$ SR($\tilde{x}$)                ▷ apply SR model
    return RESIZE($\tilde{x}$, RESOLUTION($x$))
procedure RESOLUTION($x$)
    return resolution of x in pixels
procedure RESIZE($x, d$)
    return downsized image $x$ with resolution $d$
```

The black-box attacks can be described as follows.

- **Crop**: First, the base image is center-cropped with a given cropping ratio $\rho \in (0, 1]$ and then resizes the cropping back to the base image's original size. We experiment with cropping ratios $\rho \in [0.9, 1]$.
- **JPEG Compression**: This attack performs JPEG compression [54] on the base image with a quality q. A higher quality better preserves the visual quality of the image and we experiment with $q \in [80, 200]$.
- **Noise**: This attack adds Gaussian noise $\mathcal{N}(0, \sigma^2)$ to the image. We experiment with $\sigma \in (0, 0.05]$.
- **Quantize**. This attack quantizes the number of states that a pixel can have. We compute quantization by the following formula for a quantization $q \in [0, 1]$.

$$\textsc{Quantize}(x) = q \cdot \lfloor x/q \rfloor \tag{6}$$

  We experiment with quantization strengths $q \in [0.5, 1]$.
- **Super-Resolution**. Our Super-Resolution attack uses Latent Diffusion models [43] provided through HuggingFace[9]. The used model increases an image's resolution by a factor of 4 through an optimization process. We use Super-Resolution as a removal attack summarized by Algorithm 6. We experiment with scaling factors $\rho \in [0.125, 0.5]$.

A.2 White-box Attacks

White-box attacks assume full access to the target generator's parameters, meaning that the adversary can issue virtually unlimited queries to the target generator and can control the generator's parameters and intermediate activations.

The white-box attacks can be described as follows.

- **Overwriting**. The overwriting attack trains a decoder according to Algorithm 4 and overwrites the existing watermark using PTW described in Algorithm 3. We experiment with different weights of the watermark embedding λ_R for PTW (see Algorithm 3). Increasing the weight of the decoder's loss λ_M results in a stronger perturbation of all images synthesized by the target generator. We experiment with a weight $\lambda_M \in [0.5, 1.5]$.
- **Reverse Pivotal Tuning** (RPT). Our RPT attack is parameterized by the number of real, non-watermarked images R available to the adversary. We invert images in the generator's latent space by backpropagating the LPIPS loss between the currently generated image and the base image and updating the current latent code. While other methods [42, 65] to invert real images can yield better results, backpropagation is simple and works well in practice. During training, we iteratively synthesize images from a randomly sampled batch of inverted latent codes and optimize the LPIPS similarity between the generated and corresponding base images. Algorithm 5 encodes our RPT attack.

B Implementation Details

This section describes the implementation details of our approach such as the hyperparameters we used to embed our watermarks or the reference to the (publicly available) pretrained generator checkpoints. We make a fully working implementation of all methods surveyed in this paper available as open source.

[9] https://huggingface.co/CompVis/ldm-super-resolution-4x-openimages

B.1 Generator Checkpoints

We experiment with the following checkpoints. All checkpoints were made publicly available by the authors [26,28,45]. On **FFHQ-256**, we use the following models: StyleGAN2[10], StyleGAN-XL[11], StyleGAN3[12]. On **AFHQv2** we use the following models: StyleGAN2[13], StyleGAN3[14]. On **FFHQ-1024** we use the following models: StyleGAN2[15], StyleGAN-XL[16], StyleGAN3[17]

B.2 Watermarking Parameters

All watermarks in this paper are embedded with the following parameters. We use an Adam optimizer [29] and we use a learning rate of 10^{-4} for the generator during PTW (see Algorithm 3). We use the same generator learning rate for training a watermark decoder (see Algorithm 4) and a learning rate of 10^{-3} for the watermark decoder. The watermark decoder training from Algorithm 4 contains a similarity loss and a binary cross-entropy loss for the watermark decoder. We scale the similarity loss with a weight $\lambda_{LPIPS} = 1$ and the loss for the decoder with $\lambda_M = 0.1$. We train with a batch size of 128 on FFHQ-256, a batch size of 64 on AFHQv2, and a batch size of 32 for FFHQ-1024.

[10] https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada/pretrained/paper-fig7c-training-set-sweeps/ffhq70k-paper256-ada.pkl

[11] https://s3.eu-central-1.amazonaws.com/avg-projects/stylegan_xl/models/ffhq256.pkl

[12] https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/stylegan3-t-ffhqu-256x256.pkl

[13] https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan2/versions/1/files/stylegan2-afhqv2-512x512.pkl

[14] https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/stylegan3-t-afhqv2-512x512.pkl

[15] https://nvlabs-fi-cdn.nvidia.com/stylegan2-ada-pytorch/pretrained/ffhq.pkl

[16] https://s3.eu-central-1.amazonaws.com/avg-projects/stylegan_xl/models/ffhq1024.pkl

[17] https://api.ngc.nvidia.com/v2/models/nvidia/research/stylegan3/versions/1/files/stylegan3-t-ffhq-1024x1024.pkl