Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Informatyki

Praca dyplomowa inżynierska

na kierunku Informatyka
w specjalności Sztuczna Inteligencja

Membership Inference Attacks on Large Diffusion Models

Antoni Kowalczuk
Numer albumu 303737

promotor
dr hab. inż. Tomasz Trzciński, prof. PW

WARSZAWA 2024

Membership Inference Attacks on Large Diffusion Models

Abstract. Over the last two years, the emergence of highly capable text-to-image generative models marked a new era in image content creation. Large diffusion models [1] trained on enormous collections of data downloaded from the vastness of Internet resources showed extraordinary performance on the generation task, enabling the creation of high-definition user-guided imagery in an instant. However, their development did not go without controversies. Several bodies issued copyright claims against the creators of these models. In one of the most vocal cases, Getty Images, a stock photos company, sued a creator of the Stabe Diffusion model [2] – Stability AI – for allegedly using their images to train the model without their consent.

The task of determining whether a given data sample has been used during training a given model is called in literature a *membership inference attack* (MIA). This work is centered around performing MIA on the State-of-the-Art (SOTA) open-source Stable Diffusion (SD) model. First, I line up the task and challenges regarding MIA against SD. Then, I review existing approaches to performing MIA against the Stable Diffusion model and point out the pitfalls and experimental setup mistakes in these works. Next, I propose a *fair and rigorous* experimental setting for MIA evaluation in the context of SD. Finally, I propose a set of new *membership inference attacks* against SD, evaluate them, and compare them with existing methods.

Keywords: membership inference attack, text to image generation, copyright protection

Wykrywanie przynależności obrazów do zbiorów treningowych dużych generatywnych modeli dyfuzyjnych

Streszczenie. Przez ostatnie dwa lata modele generatywne przetwarzające tekst na obraz o dużych możliwościach generacji rozpoczęły nową erę w tworzeniu obrazów. Duże modele dyfuzyjne [1] trenowane na wielkoskalowych zbiorach danych pobranych z ogromu zasobów Internetowych wykazały nadzwyczajny potencjał w generowaniu w bardzo krótkim czasie obrazów wysokiej rozdzielczości o zawartości zadanej przez użytkownika. Jednakże, ich rozwój nie przeszedł bez kontrowersji. Twórcy tych modeli stali się celem roszczeń dotyczących praw autorskich. W jednej z najgłośniejszych spraw firma Getty Images, zajmująca się sprzedażą zdjęć stockowych, pozwała twórców modelu Stable Diffusion [2] – Stability AI – oskarżając ich o użycie ich danych (obrazów) bez ich zgody do treningu modelu.

Problem czy dana próbka danych była użyta do treningu danego modelu jest znany w literaturze pod nazwą *atak inferencji członkostwa* (ang. membership inference attack). Niniejsza praca skupia się na atakach inferencji członkostwa na jednym z najlepszych dostępnych, otwartoźródłowym modelu Stable Diffusion (SD). Najpierw definiuję zadanie i wyzwania związane z atakiem na SD. Następnie, podsumowuję istniejące rozwiązania i wskazuję ich błędy oraz niepoprawnie przeprowadzane eksperymenty ewaluujące atak. Potem definiuję *uczciwy i rygorystyczny* sposób przeprowadzania ewaluacji ataku inferencji członkostwa na SD. Ostatecznie, proponuję i ewaluuję zbiór nowych ataków przeciw SD i porównuję je z istniejącymi metodami.

Słowa kluczowe: atak inferencji członkostwa, generacja obrazu z tekstu, ochrona praw autorskich

Contents

1. Introduction

Utilizing neural networks (NN) for image generation has a long history. Starting with the first approaches based on Variational AutoEncoders (VAE) [3], it has been shown that it is possible to generate images similar to real-world examples with enough compute resources and data. However, just plain random images have limited use cases in real-world applications. The invention of generative process conditioning [4] made it possible to guide the model into creating images of desired properties and contents. By inputting conditioning information (e.g., class label) additionally into the NN, the output resembled the object similar to the desired one.

Since training very capable generative models requires an enormous amount of data, usually downloaded from the internet, it is, unfortunately, a common practice to ignore the copyrights regarding the usage of this data for the training process.

1.1. Modern generative architectures

Introduction of Generative Adversarial Networks (GANs) [5], [6] allowed the generation of high-quality images. However, GANs suffered from mode collapse due to the optimization objective used to train them. It was not uncommon for the GAN to entirely and abruptly lose its generating capabilities randomly during training, forcing the developer to restart training from scratch with a different set of hyperparameters.

The emergence of Diffusion Models alleviated this ongoing struggle of GANs training by introducing an iterative (denoising) diffusion process for image generation. In short, given a clean image, a random noise is added to it to obtain a noised image. Then, NN is trained to predict the added noise, with the objective function being an L2 norm between predicted and real noise. This objective makes the optimization problem well-behaved and ensures that the training will converge without the risk of mode collapse. During inference, the model iteratively predicts noise that is then subtracted from the generated image.

The major downside of Diffusion Models comes from their denoising mechanism. Its iterative nature makes the inference and training computationally expensive. Generally, it is needed to perform 50 denoising steps to obtain high-quality generation, with each iteration being a pass through the whole neural network.

Latent Diffusion Models [7] (LDM) aim to tackle another downside of Diffusion Models: operating in an extremely high-dimensional pixel space. LDMs introduce an additional component to the whole generation stack: Variational AutoEncoders. Training and inference computational costs are significantly reduced by encoding input x into a latent representation z while retaining generative capabilities. The denoising procedure stays the same, the only difference being operating on latent vectors z instead of x. This NN architecture enabled the development of highly capable models like Stable Diffusion.

1.2. Data

Generative neural networks can only generate images depicting concepts and styles they have been familiarized with during training. To train highly performative LDMs, it is

crucial to gather data that is as diverse as possible, covering as big part of our reality as it is feasible.

LAION-5B dataset [8] is a dataset of image and text pairs created by LAION organization. It consists of **five billion** URLs to images with their descriptions obtained from Common-Crawl [9]. It has been released under permissive Creative Common CC-BY 4.0 license [10], allowing for research and commercial data usage. Creators of Stable Diffusion used a subset of this dataset to train their model.

Using much smaller and less diverse datasets to train LDMs for research is common. One work on *membership inference attacks* related to my thesis utilized the CIFAR10 [11] dataset, consisting of 60k images, to better examine this class of models in the context of MIAs. This approach allowed for faster training and hypothesis testing since the scale of both data and model is the order of magnitudes lower than for SD. However, models trained on such small datasets are substantially less capable of diverse image generation, rendering this approach less realistic.

1.3. Copyrights controversy

In contrast to Stable Diffusion developers, multiple providers of image generation services like Midjourney [12] or DALLE-3 [13], [14] do not disclose their training set sources. The main reason for that, after the competitive advantage that it gives, is to avoid copyright lawsuits from the owners of downloaded data. However, it did not stop multiple actors from suing over alleged copyright violations. Several artists' groups filed a class action lawsuit against Stability AI, Midjourney, and other key players in the generative imagery market for using artists' art without consent to train models [2]. The motivation for this work comes directly from these events. By examining the problem from the scientific perspective and pointing out the weaknesses in current approaches, I aim to shed new light on this case.

1.4. Membership inference attacks

The task of *membership inference attack* is to determine whether a given sample is a *member* or a *nonmember* of the training set of a given model. It is a binary classification task. The input to the classification method is the model's behavior for the classified sample. It is essential that the difference between samples classified as members and nonmembers comes from the model's behavior only, not from the data itself. If there is a difference in the data, the whole problem collapses to just binary classification on the data, which leads to incorrect, usually inflated results.

1.5. Technological stack

The codebase used to achieve the thesis goals is written in Python 3.8. The following Python libraries are used:

- accelerate
- black

- clip-retrieval
- datasets
- diffusers
- huggingface-hub
- img2dataset
- jupyter notebook
- matplotlib
- numpy
- pandas
- Pillow
- pyarrow
- requests
- scikit-learn
- scipy
- seaborn
- tensorboard
- tokenizers
- torch
- torchvision
- tqdm
- transformers

The experiments are run on multiple GPU-enabled machines:

- h82: provided by the Institute of Computer Science, 8xNvidia A5000 24GB VRAM
- titan4: provided by the Institute of Computer Science, 2xNvidia RTX2070 12GB VRAM + 1xNvidia A5000 24GB VRAM
- athena supercomputer: access to it is provided thanks to the PLGrid grant PLG/2023/016361. This machine provides access to 384xNvidia A100 40GB VRAM

The approximate GPU compute hours used for this thesis is 3000.

1.6. Thesis goals

My thesis has the following goals:

1. Examine current approaches to MIA on Stable Diffusion.
2. Design and implement a *fair and rigorous* experimental setting for MIA evaluation on SD.
3. Propose novel MIA against SD, implement and evaluate them in the proposed experimental setting.

2. Background

In this section, I introduce the mathematical notation used in my thesis and define the thesis' core components.

2.1. Notation

The following notation is used:

- θ: neural network's parameters
- x: input image
- E: VAE's encoder
- D: VAE's decoder
- p: probability distribution
- z: latent vector
- f: denoising neural network
- F: LDM
- τ: conditioning neural network
- y: conditioning information
- t: denoising time step
- ω: classifier-free guidance strength parameter
- α: noise scaling parameter
- T: amount of all denoising steps
- $\mathscr{L}$: objective function
- $\epsilon \sim \mathscr{N}(0, \mathbf{I})$: Gaussian noise
- M: a training set of given neural network (members set)
- NM: nonmembers set
- s: generation process strategy

2.2. Latent diffusion models

Latent Diffusion Models F are Diffusion Models that operate in latent space instead of pixel space. They consist of three main components:

1. Variational AutoEncoder, consisting of encoder E and decoder D neural networks
2. Denoising neural network f
3. Conditioning neural network τ, which guides the generative process from the user input

The general diagram proposed by [7] can be found in Fig. 2.1.

2.2.1. VAE

AutoEncoder [15] (AE) is a NN that is trained to compress image x into $z = E(x)$ and then recreate $\hat{x} = D(z)$. The training objective is to minimize

$$\mathscr{L}(x, E, D) = ||x - D(E(x))||_2^2 \tag{2.1}$$

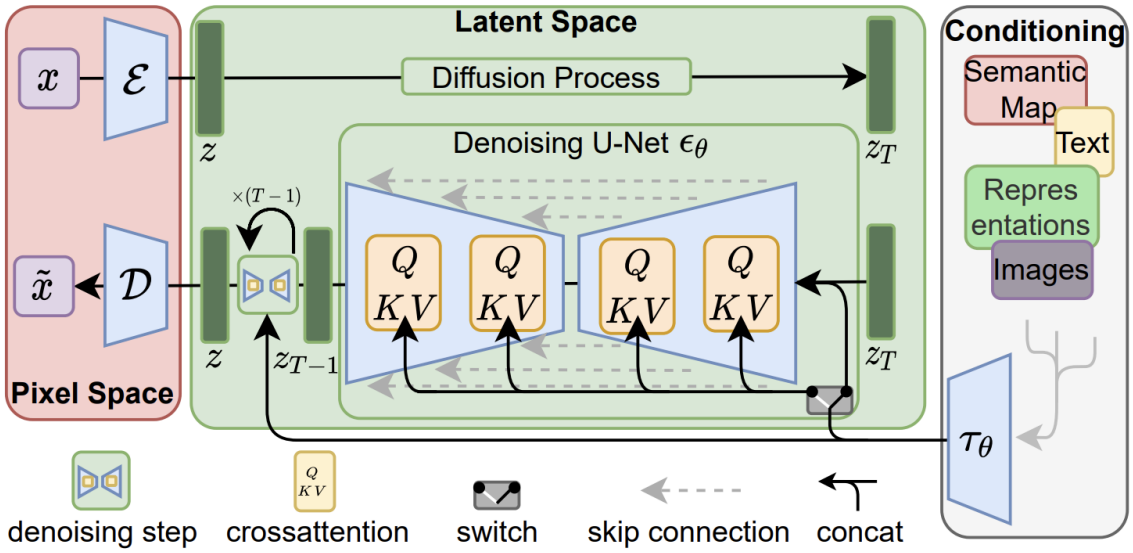


Figure 2.1. Latent Diffusion Model diagram.

Variational AutoEncoder builds on AE by changing the goal from input reconstruction to data distribution modeling in the latent space. The encoder E is trained to model $p(z|x)$, and the input of the decoder D is sampled from the distribution: $z \sim p(z|x)$. This approach allows for a more robust encoding of input images into the latent space. The goal is to encode as much semantic information about an image x into latent vector x as possible. This allows for better training of the denoising neural network and, consequently, better generation. For the LDM, VAEs pre-trained on large datasets like ImageNet [16] are usually used. The common approach is to freeze the weights of E and D while training the whole stack.

2.2.2. Denoising neural network

This is the core component of the LDM. Training the denoising neural network f is based on predicting the noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ that is applied to the input image's x latent representation z with a given noise scale $\alpha \in [0;1]$ at a timestep t. Noise scale for $\alpha_0 = 1$ and for $\alpha_T = 0$, $\forall t \in [0; T-1]\ \alpha_t > \alpha_{t+1}$. The noised latent is obtained with the following formula:

$$z_t = \sqrt{\alpha_t} \cdot z + \sqrt{1-\alpha_t} \cdot \epsilon \tag{2.2}$$

The NN output:

$$\hat{\epsilon} = f_\theta(z_t, t) \tag{2.3}$$

Optimization objective:

$$\min_\theta \mathcal{L}(z, t, \epsilon, f_\theta) \tag{2.4}$$

Where

$$\mathcal{L}(z, t, \epsilon, f_\theta) = ||\epsilon - f_\theta(z_t, t)||_2^2 \tag{2.5}$$

The generation process starts from $z_T \sim \mathcal{N}(0, \mathbf{I})$, and the NN f is applied to it to predict and remove the noise from the latent. The noise is gradually removed by applying the following transformation:

$$z_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(z_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_t}} f_\theta(z_t, t) \right) + \sigma_t \mathcal{N}(0, \mathbf{I}) \tag{2.6}$$

where $\sigma_t = \sqrt{1-\alpha_t}$ is a reversed noise schedule parameter, $\sigma_0 = 0$. While usually the $T = 1000$, 50 steps are performed during generation, skipping 20 timesteps at the time.

When it comes to architecture, usually the f NN backbone is based on the U-Net [17] architecture.

2.2.3. Conditioning

For the LDM to generate images corresponding to the user's input, a way to influence the generation process has to be introduced. It is done by incorporating conditioning data into the inputs of the denoising neural network f. This data is obtained from the conditioning NN τ and its input y. Then, the noise prediction from Eq. 2.3 is redefined as

$$\hat{\epsilon} = f_\theta(z_t, t, \tau(y)) \tag{2.7}$$

and optimization objective function from Eq. 2.5 as

$$\mathcal{L}(z, t, \epsilon, f_\theta) = ||\epsilon - f_\theta(z_t, t, \tau(y))||_2^2 \tag{2.8}$$

In this case, the input y is the text description of an image, and τ is the NN that encodes y into a vector representation.

2.2.4. Conditioning: classifier-free guidance

One way of improving the generation quality is to introduce classifier-free guidance [18]. It works by training f jointly on conditional and unconditional data. It is done by randomly dropping the conditioning input for 10% of the samples. During inference, the noise prediction is adjusted using the following formula:

$$f_\theta(z_t, t, \tau(y)) = \omega \cdot f_\theta(z_t, t, \tau(y)) + (1-\omega) \cdot f_\theta(z_t, t) \tag{2.9}$$

where $\omega \geq 1$. $\omega = 1$ means no guidance, increasing $\omega >> 1$ increases the strength of the guidance. It has been shown [19] that classifier-free guidance makes conditioning more effective, improving the correspondence of generated images to their text description.

2.3. Membership inference attacks

The *membership inference attack* can be defined as follows: given sample x and trained model F answer if sample x has been used to train model F. In this context, two following types of samples are defined: *members* (of the training set M) and *nonmembers* (samples

not present in M). The set of member samples is called *members set* (M), and the set of nonmember samples is called *nonmembers set* (NM).

2.3.1. Evaluation

Since MIA is a binary classification task, its performance can be evaluated using standard metrics like accuracy, precision, or recall. However, in this case, False Positive (FP) predictions (nonmember samples classified as members) are more harmful than False Negative (FN) predictions (member samples classified as nonmember). If somebody sues for copyright violation and the method returns an FP, it could lead to the sentencing of an innocent defendant, which is established to be a worse scenario than not sentencing a guilty defendant after the method returns FN.

Following this logic, the evaluation metric for MIA has been established [20] to be **TPR@FPR=1%**, where TPR (True Positive Rate) is a rate of True Positive (members classified as members) predictions with FPR (False Positive Rate) of just 1%. While it makes the task of *membership inference attack* substantially harder and more sensitive to outliers in the data, it ensures a more fair comparison between different MIAs. **TPR@FPR=1%** of 1% is equivalent to random guessing.

2.3.2. Members and nonmembers set

The data source for both classes is necessary to perform and evaluate binary classifiers. While the training (members) set for the research applications is usually known, the nonmembers set is sometimes challenging to obtain. The natural source of nonmembers is the test/validation set; however, for generative models training, one usually uses up a whole available dataset since evaluating such models does not require a set of images not used for training but is generally based on human-sourced evaluation protocols. In my work, the lack of a nonmember set is the biggest obstacle to designing and evaluating MIAs against the SOTA SD model.

2.3.3. Loss Threshold Attack

It has been shown that the mean of the loss value on the training set of any machine learning model is lower than on the test/validation set. The most basic MIA based on this intuition is *loss threshold attack*. Given M and NM, the attacker computes loss values for all the samples and sets the value of the threshold thr on the loss value. Then, every sample x with a loss value lower than thr is classified as a member. thr is set such that **FPR=1%** and on that the final **TPR@FPR=1%** is raported as the performance of the attack. Such an attack can be visualized in the Figure 2.2. While being the simplest MIA, it is also one of the worst-performing ones, achieving results barely above random guessing in a realistic scenario.

2.3.4. Members and nonmembers set: distribution match

Member and nonmember samples **must not** be distinguishable from each other, i.e., M and NM have to follow the same (or indistinguishable) distributions. The only difference between these should be the model's F behavior, not the data itself. Suppose an attacker

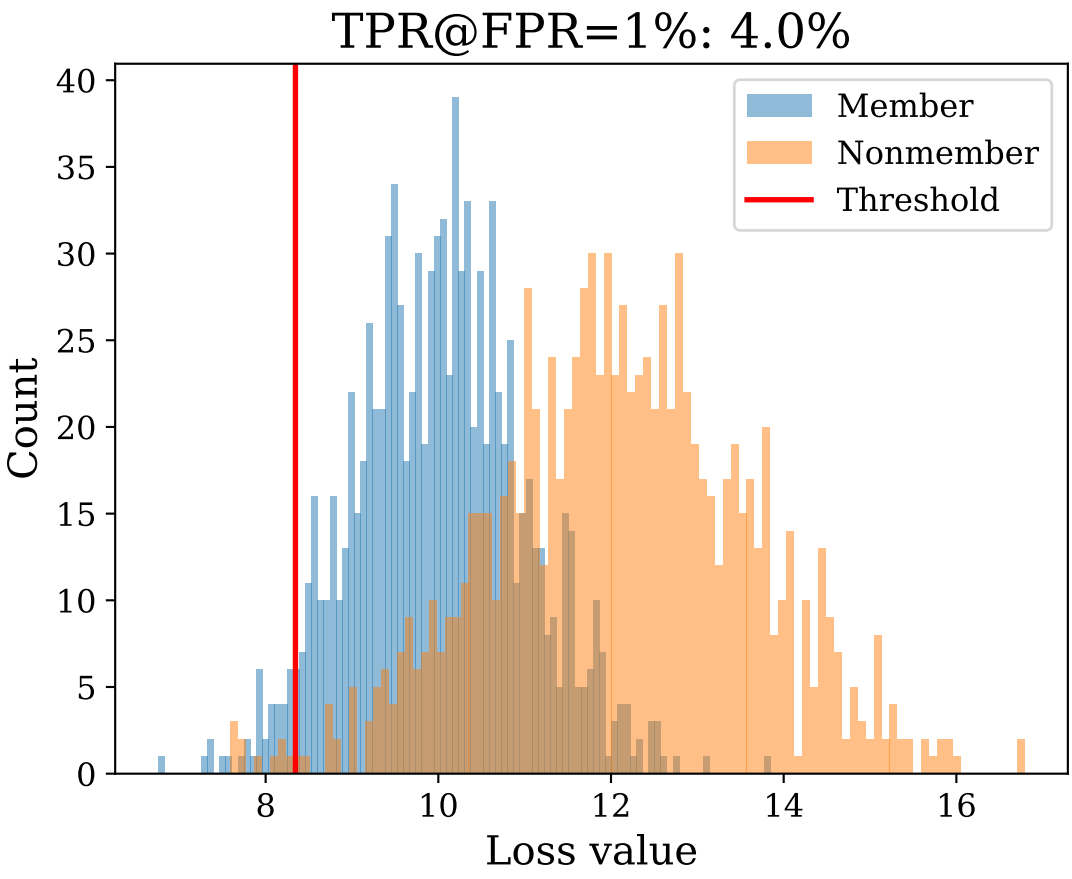


Figure 2.2. Example of the *loss threshold attack*.

can train a classifier from the M and NM while ignoring the attack model and achieve **TPR@FPR=1%** greater than 1%. In that case, this data discrepancy will inevitably inflate the final result of MIA.

For example, if the MIA is *loss threshold attack*, it is visible how it affects the results. For in-distribution data, the loss values are always lower (sometimes significantly!) than for the out-of-distribution data (in the context of the training data). This mismatch leads to situations when even the most basic MIAs achieve unreasonably good performance, which will be underlined later in this work.

2.3.5. Overfitting

As stated above, detectable differences between M and NM may lead to falsely good performance of an MIA. However, one additional important factor can influence the results: overfitting. Overfitting is a situation when a given model almost remembers each of its training samples while failing to generalize to the data distribution it has been trained on. While performing very well in loss values on the training set, it vastly underperforms on the test set, even though these sets should follow the same distribution. This is reflected in the distribution of the loss values for M and NM and may lead to situations similar to distribution mismatch between M and NM, i.e., inflated results of MIA evaluation.

2.4. Stable Diffusion

Stable Diffusion is a 890M parameters LDM trained on approximately 2.3B of image+text pairs from the LAION-5B dataset. Its training cost is estimated at 100k USD, using 150k GPU hours on Nvidia A100GPU. It comes in multiple versions: 1.x, 2.x and XL. In this work, I examine the version 1.4 [21] of SD, referenced as *SD-v1.4*. The model is licensed under an "open-source-like" CreativeML Open RAIL-M license, which is less permissive than actual open-source licenses like MIT license or GPL-3.0 license, banning "harmful" usage of the model, while allowing for gratuitous research and commercial use and providing full access to the code and model's weights.

2.4.1. Training data

The model has been trained using the following subsets of the LAION-5B dataset:

- LAION-2B-EN: subset of LAION-5B containing 2B English text+image pairs
- LAION-Aesthetics v2 5+: subset of LAION-2B-EN. Each image of LAION-2B-EN has been scored using LAION_Aesthetics_predictor_v2 [22] with scores an image from 0 (least aesthetic) to 10 (most aesthetic). The 5+ at the end indicates images with a score above 5. This dataset contains 600M English text+image pairs

First, a 1.1 version of SD was trained on all samples from the LAION-2B-EN dataset. Then, version 1.2 used data from LAION-Aesthetics v2 5+ to fine-tune version 1.1. Finally, version 1.4 has been obtained by fine-tuning on LAION-Aesthetics v2 5+ once more, this time using classifier-free guidance to improve the generation images alignment with text conditioning.

SD-v1.4, being an LDM, utilizes a VAE encoder and decoder neural networks pre-trained on the ImageNet dataset to encode image x into the latent representation z. The text encoder τ is a CLIP encoder [23].

2.4.2. Members and nonmembers set

LAION-5B is an openly-available dataset. Therefore, I can easily obtain M. However, developers of SD-v1.4 did not leave out any test/validation set that can be used as a source of NM. To my knowledge, only two papers try to alleviate this problem [24], [25], which I examine in Sec. 3.

3. Overview of Existing Approaches

In this section, I review and summarize the core publications related to my thesis. I start with existing works on *membership inference attacks*, then switch my focus to the publications tackling the lack of nonmember samples issue.

3.1. Membership inference attacks

3.1.1. Membership Inference Attacks From First Principles [20]

This paper aims to unify the underlying components of the whole *membership inference attack* problem. The authors highlight pitfalls in the existing approaches and evaluate the efficacy of the attack methods. They provide intuition on the necessity of evaluation of such methods in the extremely low (i.e., $\leq 0.1\%$) FPR regime, supporting claims from 2.3, and point out why formerly used accuracy and Receiver Operating Characteristic (ROC) metrics fall short in a fair evaluation protocol. They also suggest that all ROC curves representing the performance of an MIA should have logarithmic scales to compare the effectiveness better in such a low FPR regime (see Fig. 3.1).

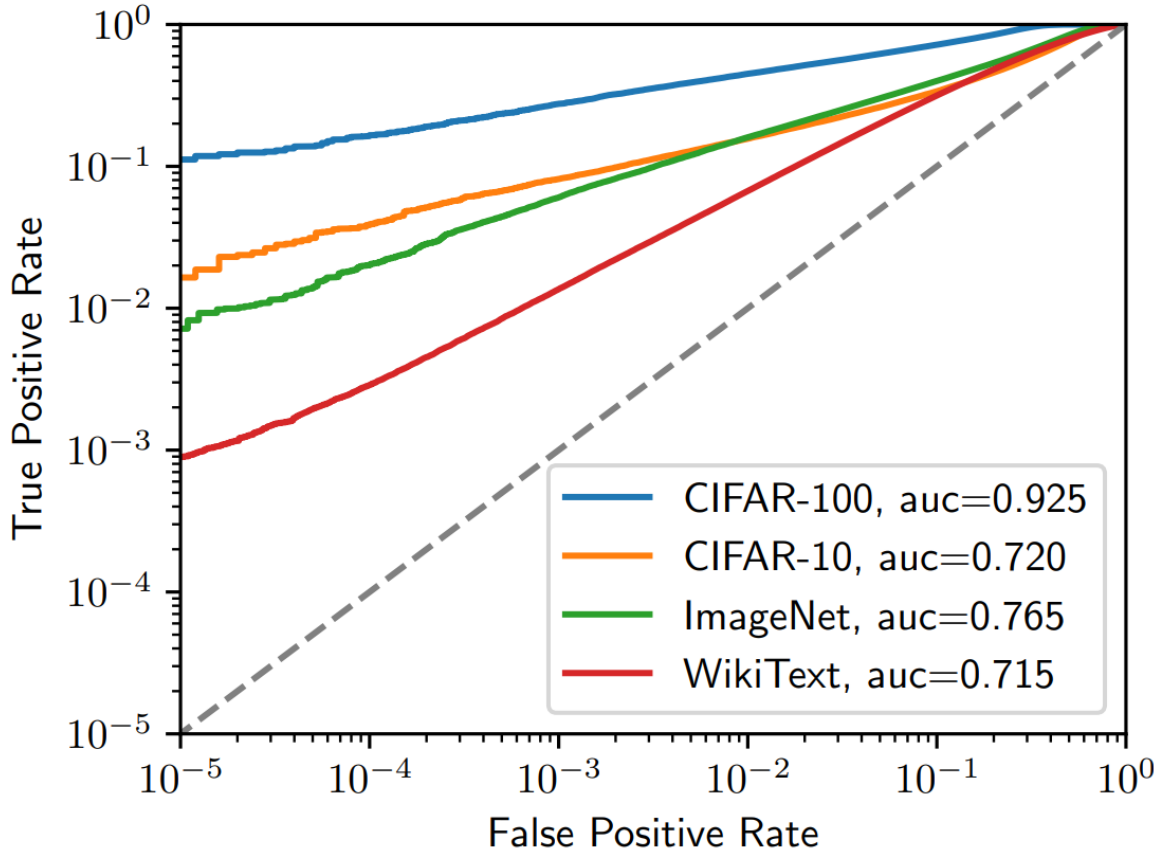


Figure 3.1. An example of Receiver Operating Characteristic curve with logarithmic x and y scales suggested to visualize the efficacy of an MIA in the low FRP regime. Source: [20]

However, the main contribution of this work is *LiRA* (Likelihood Ratio Attack), an attack based on *shadow models*. A shadow model is one in which the training set and architecture

are similar (or identical) to the attacked model. Given dataset D, an attacked model f, an attacked sample $x \in D$ they train N shadow models fin_i on N a random sets $Din_i \subset D$ for $i = 1,2,\ldots,N$, such that $x \in Din_i$, and N shadow models $fout_i$ on N a random sets $Dout_i \subset D$ for $i = 1,2,\ldots,N$, such that $x \notin Dout_i$.

Then, for the sample x they compute the models' confidences $\phi\big(f(x)\big) = log\left(\frac{f(x)}{1-f(x)}\right)$, the one for the attacked model: $conf_{obs} = \phi\big(f(x)\big)$, and two sets of confidences: $confs_{in}$ and $confs_{out}$ for the shadow models fin_i and $fout_i$. All confidence computations are done for the attacked sample x. The final prediction Λ is obtained using the following formula:

$$\mu_{in} \leftarrow mean(confs_{in}) \tag{3.1}$$

$$\mu_{out} \leftarrow mean(confs_{out}) \tag{3.2}$$

$$\sigma^2_{in} \leftarrow variance(confs_{in}) \tag{3.3}$$

$$\sigma^2_{out} \leftarrow variance(confs_{out}) \tag{3.4}$$

$$\Lambda = \frac{p\big(conf_{obs}|\mathcal{N}(\mu_{in},\sigma^2_{in})\big)}{p\big(conf_{obs}|\mathcal{N}(\mu_{out},\sigma^2_{out})\big)} \tag{3.5}$$

Authors claim SOTA results of their method on multiple MIA tasks in the classification domain, reaching more than 10 times better **TRP@FPR=0.1%** than previous works. However, LiRA is extremely unrealistic and inapplicable even for research purposes in the context of Stable Diffusion. LiRA achieves excellent results when $2N$ (shadow models overall count) is in the region of 64 to 128, which for the SD amounts to over 6.4M USD in compute costs.

3.1.2. Extracting Training Data from Diffusion Models [26]

One of the contributions of this paper is the evaluation of the MIAs on the small Latent Diffusion Model trained on the CIFAR10 dataset.

They evaluate two MIAs: *loss threshold attack* (see 2.3.3) and LiRA. Both attacks are based on some *loss* measurement captured for a given attacked sample. They examine the most optimal timestep t to record the reconstruction loss (2.5) specific to the LDM. Authors find that $t \in [50;300]$ corresponds to the best performance of MIA. They evaluate attacks on $t = 100$, with LiRA reaching **TPR@FPR=1%** of 71%, while the baseline *loss threshold attack* reaches only **TPR@FPR=1%** of 2.1%. This shows how extremely hard and low-performing the baseline approach is compared to the LiRA.

One additional contribution is a *Strong LiRA*. The only difference between Strong LiRA and LiRA is that for each sample, they run the evaluation over the mean of 5 runs through the models, boosting the results to almost 90% **TPR@FPR=1%**. I utilize this insight to improve the efficacy of MIAs evaluated on the SD-v1.4.

3.2. Absence of nonmember samples

As stated in Sec. 2.4, there is no natural source of nonmember samples for SD-v1.4. Despite that, there are two works that I am aware of that try to tackle this problem to allow

for MIA evaluation without nonmembers set. In this section, I analyze and identify pitfalls in their approaches.

3.2.1. Membership Inference Attacks Against Text-to-image Generation Models [24]

In this paper, authors propose to source the NM from other online datasets for text-to-image generation: MSCOCO-Face [27] and VG-Face [28]. For the M set, they use the LAION-Face [29] dataset, limiting the scope of the work to face image generation using LDM. Their proposed attacks are based on image classification NN; one even ignores the output of the LDM (!!!). Unsurprisingly, the performance of their attacks is sometimes almost perfect (however, they use accuracy as their metric) since the distribution mismatch between M and NM is enormous.

3.2.2. Are diffusion models vulnerable to membership inference attacks? [25]

Another proposed approach is using a distinct text-to-image dataset to fine-tune the SD-v1.4. In this case, the M and NM can be easily controlled by fine-tuning a subset of the dataset. The goal of such fine-tuning is to tailor the SD-v1.4 to the distribution of the new dataset while keeping its generative capabilities. However, such an approach is prone to overfitting, again rendering the MIA evaluation results incorrect.

More specifically, authors fine-tune SD-v1.4 on half of the POKEMON [30] dataset, i.e., 416 samples, while holding out 417 samples as NM. To evaluate the effect of such an approach on the MIAs' performance, I fine-tuned my version of SD-v1.4 on the same dataset. In Fig 3.2, I show that the overfitting during fine-tuning gradually improves the performance of the simplest *loss threshold attack*, reaching up to 80% **TPR@FPR=1%**, which is an astounding, unrealistic result.

The work's main contribution is their MIA; however, at the time of development of this work, I could not implement it from scratch to compare it with my MIAs.

Additionally, to evaluate their MIA on SD-v1.4 facing lack of NM, they also sample COCO2017-val [27] as a NM and run evaluation of their method, an approach already described as incorrect in Section 3.2.1.

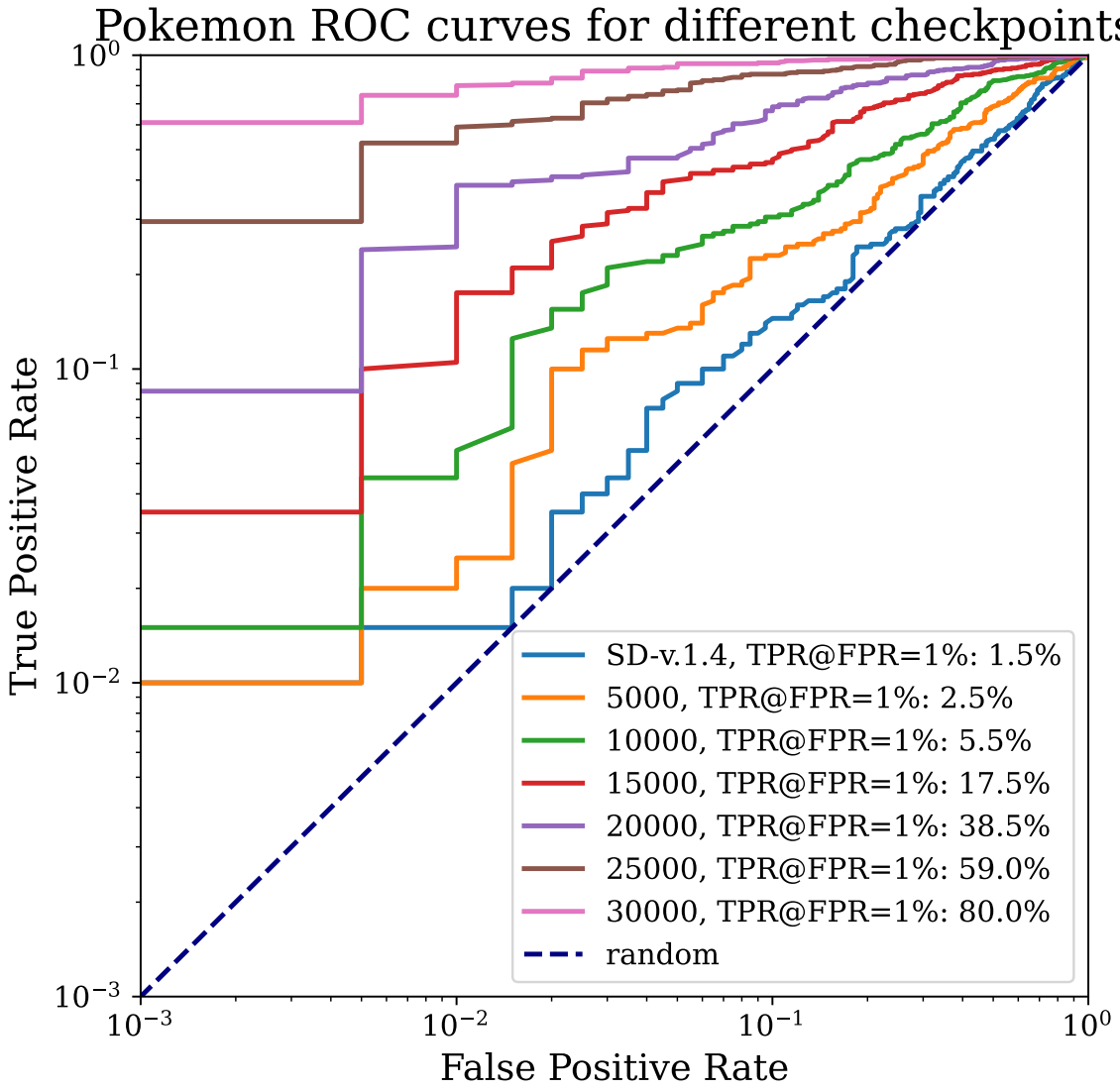


Figure 3.2. *Loss threshold attack* performance on the POKEMON dataset for the fine-tuned SD-v1.4 checkpoints saved every 5000 training steps.

4. Solution

In this section, I outline the development process of a critical component of the proposed *fair and rigorous* experimental setting for MIA evaluation on the SD-v1.4 model: **LAION-mi** dataset. This dataset addresses the problem of the absence of the natural nonmembers set for the model. The core contribution is a dataset with members and nonmembers selected being indistinguishable, allowing the correct evaluation of MIAs. My approach solves two common problems regarding the creation of such datasets for MIA evaluation on Large Diffusion Models: duplicate samples and distribution mismatch between M and NM.

4.1. Source of members set

The source of M is LAION-Aesthetics v2 5+. It has been used in training the SD-v1.4 model (see Sec. 2.4).

4.2. Source of nonmembers set

One of the subsets of LAION-5B is LAION-2B-Multi, which contains text+image pairs, with text being in all languages but English. LAION-5B authors used machine translation to translate these text descriptions to English and released a LAION-2B-Multi-Translated version of this subset. This dataset is further used as a source of the nonmember samples because it should not overlap with the LAION-2B-EN, a superset of LAION-Aesthetics v2 5+.

4.3. Challenge 1: duplicates

LAION-5B dataset is a dataset of images scrapped from all over the Internet, which may contain duplicates. In this case, a duplicate is a sample in which an image is present more than once in the whole dataset. It is intuitive that some images, e.g., depicting famous politicians, are present under more than one unique URL. Indeed, [31] shows that just in LAION-2B-EN, there are approximately 30% of duplicate samples!

4.3.1. Why is it a problem?

Simply enough, if a sample from LAION-2B-EN is also present in LAION-2B-Multi-Translated, the source of NM, it would contaminate the NM with a member sample. This directly impacts the correctness of the MIA evaluation protocol I propose in this work, undermining it.

4.4. Solution 1: deduplication

Fortunately, LAION-5B authors share a KNN-index-based API [32], allowing duplicate search through the entire LAION-5B dataset. Providing CLIP embedding of an image returns a set of duplicate candidates' URLs. I want to de-duplicate a nonmember's set since the duplicates affect only this set. This API returns samples from the whole LAION-5B;

however, I am interested only in duplicates present in the source of members' samples – LAION-2B-EN. I use CLD3 [33] machine learning language identifier to classify samples to this set and obtain a final set of duplicate candidates for a given sample. I then download the images from these URLs and compute L2 distances between their CLIP embeddings and a tested image CLIP embedding. Finally, I set a threshold on the L2 distance above which I classified the tested nonmember sample to have no duplicates in the LAION-2B-EN dataset.

4.4.1. Deduplication querying details

I sample 160k nonmember candidates from the LAION-2B-Multi-Translated. Then, for each of them, I query the KNN API and receive up to 40 in duplicate candidates. Next, I compute L2 distances (on CLIP image embeddings) between nonmember candidates and their corresponding duplicate candidates. Then, I choose the one with the lowest L2 as the final duplicate candidate for each nonmember candidate. The distribution of L2 distances can be found in the plot 4.1. The data roughly follows a normal distribution.

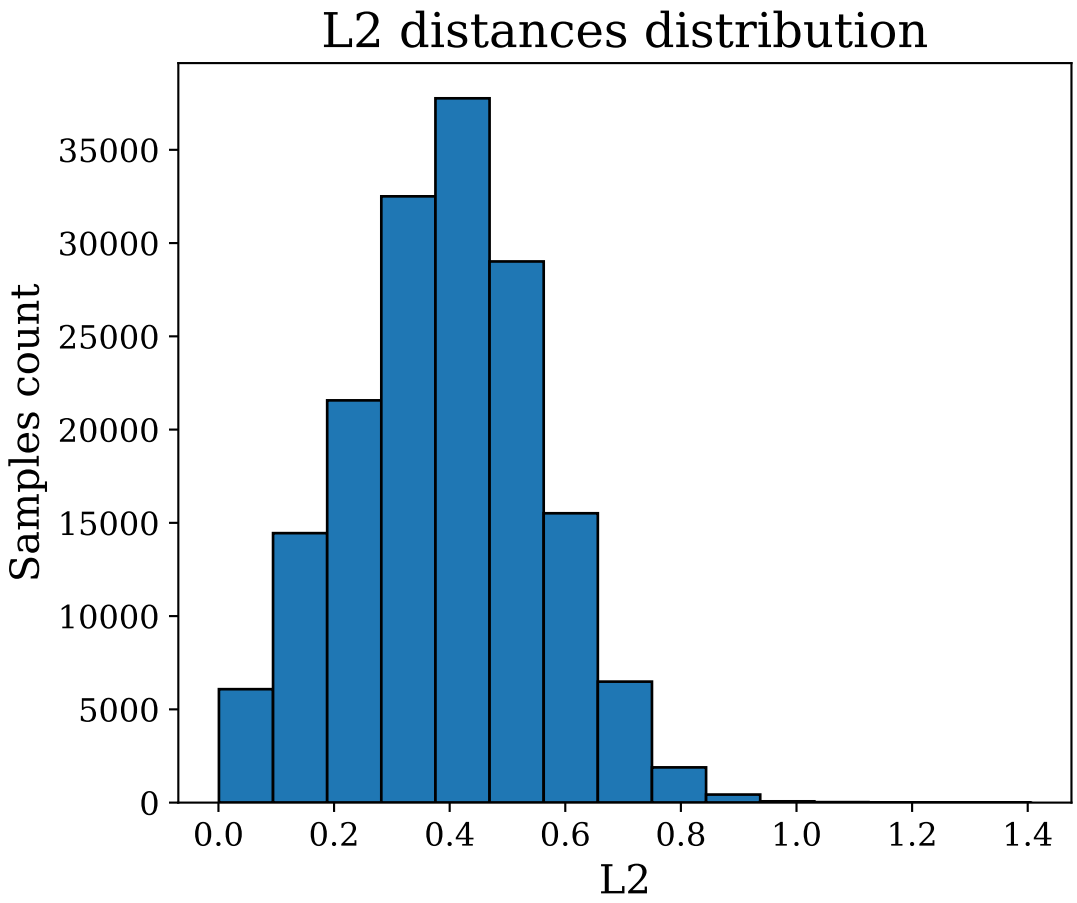


Figure 4.1. The distribution of L2 distances between CLIP image embeddings of nonmembers and corresponding deduplication candidates.

4.4.2. Threshold selection

The goal of the deduplication process is to minimize the possibility of a member sample being present in the NM. To pick the threshold value that removes the most duplicates while leaving the most nonmember samples, I examine different threshold values and the duplicate ratio. I manually check 100 images per bucket with L2 distances in $[threshold; threshold + 0.1)$ interval up until threshold 0.5, for which, after examining 300 samples, I do not find any duplicates. Using the *rule of three* [34], I infer that there are less than 1% of duplicates in the data with 95% of confidence, which is a satisfying result for this application. The duplicates ratios are in the plot 4.2. For the threshold of 0.5, I filter out around 75% of samples, arriving at 40k nonmember samples in NM.

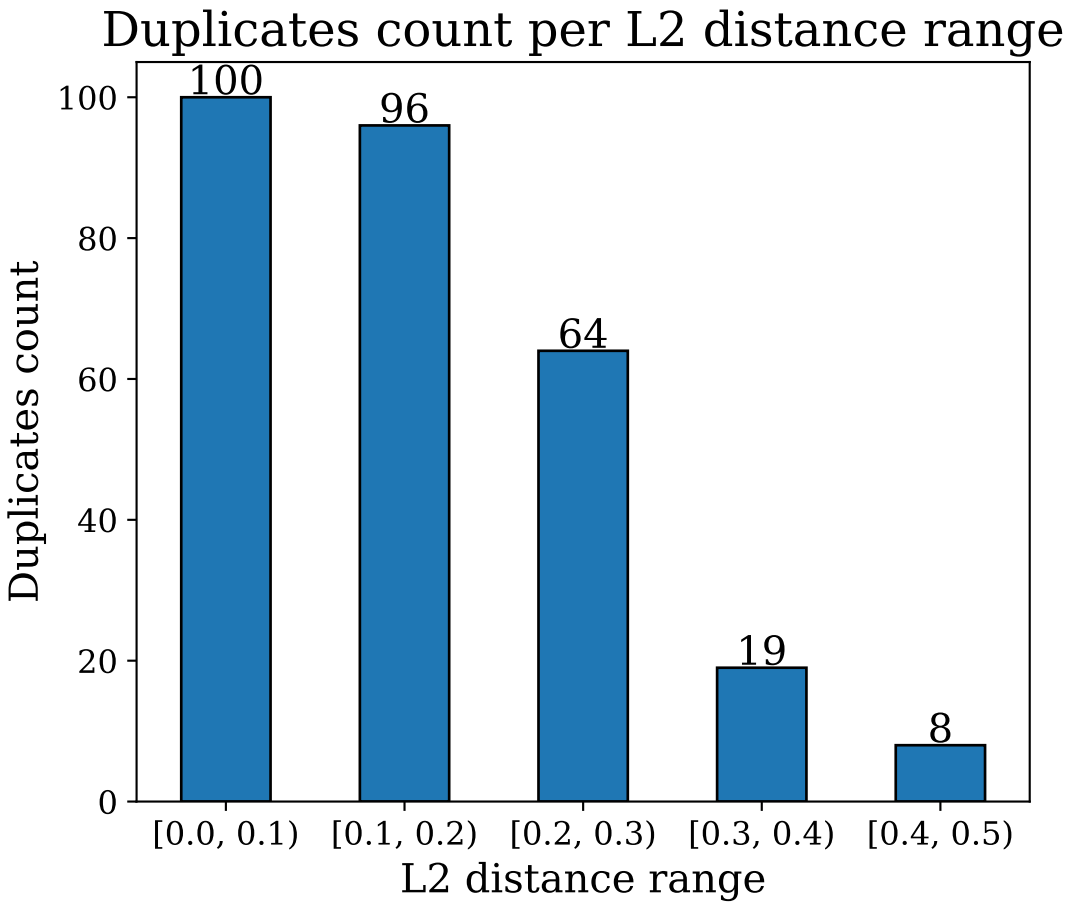


Figure 4.2. The number of duplicates confirmed in the L2 distance bucket.

4.5. Challenge 2: distribution mismatch

After tackling the duplication problem and obtaining a clean nonmembers set, one final obstacle prohibits the correct evaluation of an MIA: mismatch between M and NM. As lined up in 3.2.1, ignoring this issue leads to wrong results. To avoid it, I propose the following mismatch evaluation metrics:

- Fréchet Inception Distance (FID) [35]: this is a metric of internal set complexity used e.g., for GAN generations evaluation. Suppose the internal complexity of a set of k

samples from M (or NM) is significantly lower than for a set of $k/2$ samples from M and $k/2$ samples from NM. In that case, I assume that the distribution mismatch is significant.

- Visual analysis of PCA [36] 2D projection: this approach utilizes Principal Component Analysis (PCA) to decompose a set of M and NM into 2D space and plot the obtained points on a scatterplot. The distribution mismatch is significant if the samples from M occupy a different surface region than NM.
- Classification: I train a binary classifier directly to distinguish M from NM. If I can achieve non-random accuracy, the datasets will be easily distinguishable.

The source of the distribution mismatch, in my case, is the machine translation of the text descriptions of samples in M and NM. The above metrics utilize CLIP embeddings of text data to compute their values. Using CLIP image embeddings, I also evaluate FID, PCA, and classification performance on image data.

4.5.1. Problem scale assessment

To assess how different the distributions of NM and a source of M, I randomly sample 40k samples from LAION-Aesthetics v2 5+ and run evaluations using the above metrics. Unsurprisingly, the mismatch is very significant due to the machine translation-induced distribution shift of the text data in NM. In Fig. 4.3, it is clear that the data points from M are differently distributed from NM. Secondly, FID for the set of 5k members and 5k nonmembers is significantly higher than the internal FID values of members and nonmembers set, see Tab. 4.1. Lastly, classification evaluation also confirms the scale of the problem: a simple linear layer after just one epoch reaches almost 90% accuracy.

4.6. Solution 2: sanitization

The proposed solution utilizes classifier-based distribution mismatch evaluation to filter out member samples significantly different from the NM distribution. Since the classifier is trained with the task of modeling differences in distribution between NM and M, it can be used to create a new set of member samples that it fails to classify as members. This new set M has a significantly closer distribution to the NM set than the previous M. By applying this procedure iteratively, the classifier can be forced to model more and more subtle differences between sets. At some point, the M and NM will be indistinguishable. A detailed solution is available in Algorithm 1.

4.6.1. Sanitization run details

I run the Algorithm 1 for three iterations. To obtain 40k member samples as close to NM as possible, I filter 5M samples from the LAION-Aesthetics v2 5+ dataset. The code runs for approximately 2h on a single Nvidia RTX 2070. The classifiers are trained only on the CLIP text embeddings of the samples.

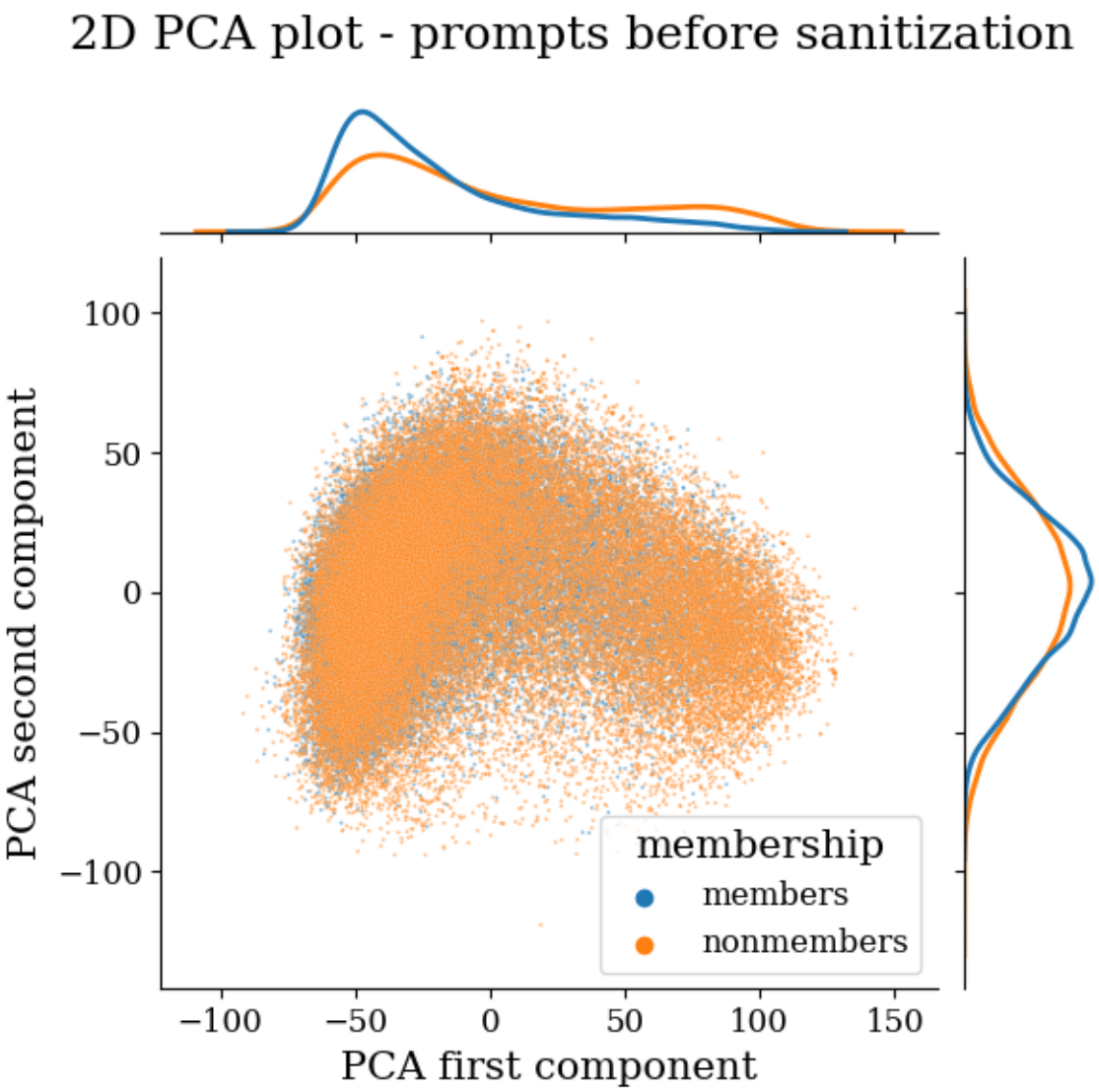


Figure 4.3. 2D PCA visualization of the CLIP text embeddings of the random 40k member samples and NM set.

4.6.2. Distribution match evaluation

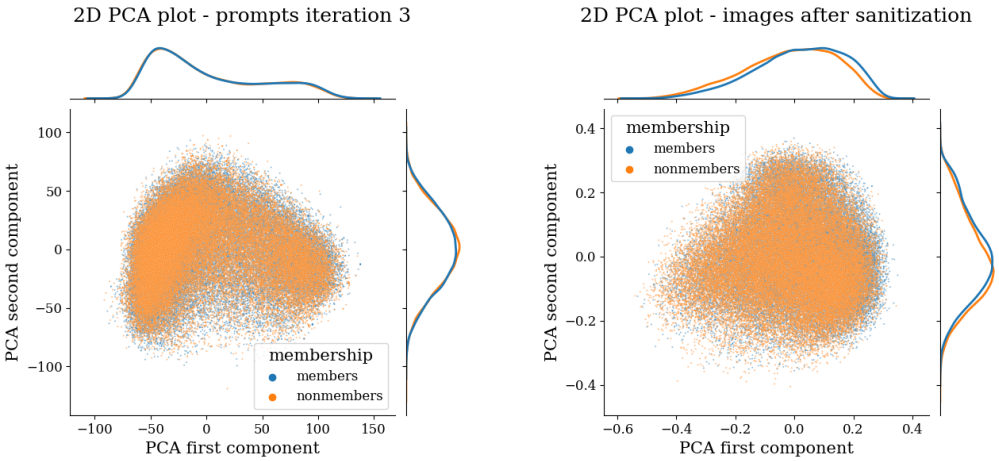
I run all three evaluation methods on the new M and NM set. As seen in Fig 4.4 for both text (Fig. 4.4a) and image (Fig. 4.4b) data, the 2D distributions between M and NM align. Additionally, the FID score drops to almost the same as internal (ref. Tab. 4.1). Finally, I train a binary classifier on this data, which arrives at roughly 52% accuracy, which is almost random. Improvement of the image data alignment while focusing only on the text is due to the direct correlation between the image and its description.

4.7. LAION-mi dataset

After tackling the duplicates and distribution mismatch challenge, I obtain a M and a NM set that allows for a fair evaluation of an MIA on SD-v1.4. The dataset is released

Algorithm 1 Sanitization algorithm

$F \leftarrow \emptyset$ ▷ trained binary classifiers
$NM \leftarrow$ deduplicated nonmembers
$M \leftarrow$ global set of members
$M_i \leftarrow \emptyset$ ▷ sanitized members after i-th iteration
$TrainSet \leftarrow \emptyset$ ▷ training dataset
for $i \leftarrow 1, 2, ..., n$ **do**
    $TrainSet \leftarrow \emptyset$
    if $i = 1$ **then**
        $TrainSet \leftarrow$ sample of size $|NM|$ from M
    else
        $TrainSet \leftarrow M_{i-1}$
    end if
    $TrainSet \leftarrow TrainSet \cup NM$
    $F_i \leftarrow$ trained classifier on $TrainSet$
    while $|M_i| < |NM|$ **do**
        $M_{tmp} \leftarrow$ sample from M
        for $j \leftarrow 1, 2, ..., i$ **do**
            if F_j predicts member label for sample **then**
                $M_{tmp} \leftarrow M_{tmp} \setminus sample$
            end if
        end for
        $M_i \leftarrow M_i \cup M_{tmp}$
    end while
end for
$SM \leftarrow M_n$ ▷ final sanitized members set



(a) CLIP text embeddings of the M and NM after sanitization.

(b) CLIP image embeddings of the M and NM after sanitization.

Figure 4.4. 2D visualization for text and image embeddings after three iterations of the sanitization algorithm.

publicly at huggingface.com[1] under the same permissive license as LAION-5B. It consists of 40k member and 40k nonmember samples.

[1] https://huggingface.co/datasets/antoniaaa/laion_mi

Table 4.1. FID comparison for 10k samples.

	FID	
DATA SUBSET	TEXT	IMAGES
MEMBERS INTERNAL - RANDOM	9.84	7.00
MEMBERS INTERNAL - SANITIZED	9.77	7.06
NONMEMBERS INTERNAL	9.73	7.01
COMPARATIVE - RANDOM	66.43	13.90
COMPARATIVE - SANITIZED	**13.54**	**8.87**

5. Experimental Setup

With the lack of nonmembers set problem tackled in Section 4, I then define how to evaluate the MIAs I use in my work. I also aim to show how my approach to dealing with the lack of NM is superior to other methods described in Section 3.

5.1. Datasets

In this section, I list the datasets used to evaluate the MIAs.

5.1.1. POKEMON

As described in Sec 3.2.2, fine-tuning a SD model on a small text-to-image dataset can be tried to obtain a NM set directly. To present how misleading results such an approach can produce, I fine-tune an SD-v1.4 on the POKEMON dataset for 30k train steps utilizing code and hyperparameters from [37]. I leave out 200 samples as NM and train on the remaining 633.

5.1.2. LAION-mi

For the *fair and rigorous* evaluation I utilize my LAION-mi dataset.

5.2. Attacks

In this section, I outline the setup for the *membership inference attacks*: scenarios, measured model's behavior, and usage of the measured data for performing the attack.

5.2.1. Scenarios

To test different real-world applications of text-to-image generative services, I run experiments for three scenarios:

- white-box: full access to model's weights and code, possibility to manipulate the generation process
- grey-box: denoising model is hidden behind an API, the attacker has access to the VAE and text encoder, can submit latent of image and text embedding to the API
- black-box: the whole model hidden behind an API; an attacker can only submit prompts

5.2.2. Measured behavior

I propose three different measurements (each is timestep t-dependent):

- Pixel Error: $||x_t - \hat{x}_t||_2^2$
- Latent Error: $||z_t - \hat{z}_t||_2^2$
- Model Loss: $\mathscr{L}(z, t, \epsilon, f_\theta) = ||\epsilon - f_\theta\left(z_t, t, \tau(y)\right)||_2^2$ (Eq. 2.8)

For the black-box scenario, the only measurement I can register is the final Pixel Error between the original and the generated image. For the grey-box scenario, I can also collect the Latent Error between the latent of the original image and the generated image's latent. More interesting approaches are possible in the white-box scenario. During inference

through the model F on each timestep t, I can collect all three measurements and use them for performing an MIA. The novel MIAs proposed are considering this scenario only.

5.2.3. Generation process alteration

Each proposed attack has a defined order of denoising steps and timesteps to guide the generation process. Manipulation of this process aims to extract more information about *membership* of a given sample by testing the model behavior under different inference settings. The hypothesis is that the generation process will behave more robustly for the member samples than the nonmember samples. This will lead to lower errors for members than nonmembers, benefitting the performance of a MIA since the difference between members and nonmembers will be magnified.

5.2.4. Data collection procedure

This procedure is defined differently for different scenarios. Generally, for a given sample x and a generation strategy s, I run a denoising process through the model F. It is done 5 times with a different Random Noise Generator seed for each run, following findings from 3.1.2. The strategy for the black-box and grey-box scenarios is simple: I do not modify the generation process since I cannot access it.

For the white-box scenario, the procedure is as follows. At each defined in s timestep t, I compute all three measurements and save their values. After the generation process of F finishes, I obtain a matrix of $t_s \times 3 \times 5$ measurements per sample, with the last dimension corresponding to the re-runs through the model. I then calculate the mean over the last dimension and obtain the final matrix of features of size $t_s \times 3$.

In all scenarios, I run this procedure for 5k member and 5k nonmember samples from the LAION-mi or all 633 members and 200 nonmembers for the POKEMON.

5.2.5. Threshold attack

Similar to the *loss threshold attack*, I define the threshold so that the **FPR=1%** and report the **TPR**. For each measurement type, I pick the timestep t with the highest **TPR** and report it as the attack's performance on such measurement. In further comparison of different strategies, I differentiate between these types of measurements, pointing out that some of the strategies benefit one measurement type more than the others.

5.2.6. Classifier attack

For each sample and generation strategy in the white-box scenario, I obtain a $t_s \times 3$ matrix of features. I use these features to train a binary classifier C on the membership inference task for the samples with the known membership label. The output of the trained C being a probability that a sample x is a member is treated as a *measurement* and I perform a *threshold attack* described in 5.2.5 on this value.

The binary classification algorithms I utilize are:

- Logistic Regression (LR)
- Decision Tree Classifier (DTC)
- Random Forest Classifier (RFC)

- Neural Network binary classifier (NN), 1 hidden layer of size 10 with ReLU [38] activation, Sigmoid activation for output layer. Adam [39] optimizer, learning rate of 0.001, binary cross entropy loss, 32 batch size, 100 epochs

For LR, DTC, and RFC, I perform a Grid Search over their respective hyperparameters with 5-fold cross-validation for each fit. I perform early stopping for each of the four model classes and use the best model for the final evaluation.

5.2.7. Bootstrapping

Because of the definition of the evaluation metric being **TPR@FPR=1%**, the results can be greatly influenced by outliers, with the worst results on the randomized sample being on average 10 times worse than the best results (see Sec. 6.3). To alleviate this problem, I propose the following evaluation method:

1. Randomly sample 1k members and 1k nonmembers from the 10k samples used to collect behavior measurements.
2. Perform the attack on this data.
3. Evaluate the attack, raport **TPR@FPR=1%**.
4. Repeat 100 times.
5. Raport mean and standard deviation of the **TPR@FPR=1%** obtained as the final score.

The results' variance is significantly reduced, and the evaluation procedure is more reliable. The bootstrapping process applies to both *threshold* and *classifier* attacks.

5.2.8. Baseline attack

The baseline attack is proposed in Section 3.1.2, evaluated on the $t = 100$ timestep. The measured value used for attack here is the Model Loss.

5.2.9. Generation from prompt

In this attack, I simulate a default generation procedure by performing 50 denoising steps on the full LDM for the given text input. The measured value used for the attack here is the Pixel Error.

5.2.10. Proposed novel MIAs

The following is the set of proposed and evaluated generation process strategies used to collect measurements for the white-box *threshold attack* and *classifier attack*:

1. **Partial denoising** Following findings described in Sec. 3.1.2, I start the denoising process at the timestep 300 for the steps 26 up to 50. z is noised once, with scale α_{300}, and then for each step is calculated from the f noise prediction.
2. **Partial denoising non-iterative** is similar to the *Partial denoising* method, but at each timestep, I pass a newly noised z to f instead of the output of the previous timestep's inference through f. The main reason behind this is to see whether the information about membership gets lost or accumulates during the iterative denoising process.
3. **Partial denoising latent shift** follows the *Partial denoising* attack method, but at the middle timestep 170 I shift z by a scaled random noise. The intuition behind this

method is that the member samples should return to the correct trajectory after the shift more efficiently than the nonmember samples, enabling better identification of the member samples.

4. **Partial denoising text shift** is similar to the *Partial denoising* latent shift, but instead of shifting the latent representation of the image, I shift the text embedding by a random noise $N(0, \mathbf{I}) \cdot 0.1$ before passing it to the f. The intuition here is the same as in the partial denoising latent shift method, but I want to see if the text embedding is more critical than the latent representation of the image.
5. **Partial denoising bigger text shift** is identical to the *Partial denoising text shift*, but the noise added to the text embedding is scaled by 0.5 instead of 0.1. I want to test if adding more noise to the text embedding benefits the performance of the attack.
6. **Partial denoising wrong start** In this method I follow the *Partial denoising*, but the starting z is noised using α_t from the wrong timestep, $t = 100$ instead of $t = 300$. I want to test whether I can extract more information about the membership if I apply lower noise to the latent representations at the beginning of the process than f expects.
7. **Full denoising start 100** is similar to the *Partial denoising wrong start* method, but I perform an almost full denoising process, starting on timestep and 900 ending at timestep 0, denoising every 100 step and starting from the latent noised with α_{100} noise scale.
8. **Full denoising start 100 text shift** is close to the *Full denoising start 100*, but I shift the text embedding by a random noise $N(0, \mathbf{I}) \cdot 0.1$ before passing it to f.
9. **Full denoising start 50** is similar to the *Full denoising start 100*, but the starting latent is noised using α_{50} instead of α_{100}.
10. **Full denoising start 300** is like the *Full denoising start 100*, but the starting latent is noised using α_{300} instead of α_{100}.
11. **Short denoising start 300** In this method, I perform a short denoising process, starting from timestep 200 and ending at timestep 0 every 100 step instead of the full one. The latent representation of the image is noised using α_{300}.
12. **Reversed noising** I perform 10 denoising steps for timesteps from 900 to 0. z is noised using noise scales α_t in reverse order, e.g. at timestep 800 I pass the latent noised with α_{100} to the f. Additionally, during inference, I use classifier-free guidance with $\omega = 100$. The text embedding passed to f remains unchanged. The intuition behind the attack is that member samples will behave more robustly than nonmember samples under such conditions.
13. **Full denoising start 300 no cfg** is similar to the *Full denoising start 300*, but I do not use the classifier-free guidance (i.e. $\omega = 1$). I want to see if the classifier-free guidance benefits the attack's performance.
14. **Full denoising start 100 non-iterative**. This method is identical with the *Full denoising start 100*, but at each t I pass newly noised z with scale α_{100} of the image to the f, instead of the output of the previous timestep.
15. **Reversed noising regular cfg** resembles the *Full denoising start 100 non-iterative* attack method. The latent representations I pass to the f are noised using α_t from the

timesteps in the reversed order, i.e., from 0 to 900, so the first latent passed to the f is noised using α_0, while f receives timestep 900 as input. The classifier-free guidance is applied with a scale of 7.5 (default).

16. **Reversed denoising** In this method, I reverse the order of timesteps at which I perform denoising using f. I start from the timestep 0 and then go up to the timestep 900, for 10 steps. Similarly, as in the baseline *loss threshold method*, I apply noise to z once, but this time using noise scale α_{100}. The intuition is similar to in the *Reversed noising* method, but here I test if the iterative nature of the diffusion denoising process will magnify this effect.

6. Results

This section presents results from evaluating different MIAs performed on the SD-v1.4 model. In the first part, I summarize the best generation strategies for the *threshold attack* and compare them to the baseline method. I also discuss the results obtained for the incorrect fine-tuning approach presented in 3.2.2. Then, I show the results for all generation strategies for both *threshold* and *classifier* attacks and discuss the insights. Finally, I underline the necessity of bootstrapping (5.2.7) by presenting the differences between the worst and the best **TPR@FPR=1%** obtained from the 100 random subsets.

6.1. The best generation strategies

I present the evaluation results in Table 6.1. The best-performing generation process alteration strategies are *Reversed noising*, *Partial denoising*, and *Reversed denoising*. All of these outperform the *Baseline loss threshold* attack on all three measurements: Model Loss, Latent Error, and Pixel Error for the LAION-mi dataset. However, the performance is underwhelming, with the best result being 2.51% **TPR@FPR=1%**.

In the non-realistic scenario, evaluating these attacks on the SD-v1.4 fine-tuned on the POKEMON dataset results in orders of magnitude better performance than on LAION-mi. The *Partial denoising* attack on the Latent Error almost perfectly classifies all member samples as members, reaching outstanding 99.5% **TPR@FPR=1%**. The *baseline loss threshold* attack also reaches an impressive 80.9% **TPR@FPR=1%**. These results confirm that the fine-tuning approach proposed in 3.2.2 leads to misleading results and should not be used to handle the lack of a nonmember set.

6.1.1. Generation from prompt

Unfortunately, attacking the model in black-box and grey-box scenarios leads to less than 1% **TPR@FPR=1%** for LAION-mi, which performs worse than random guessing. This underlines the difficulty of attacking a model hidden behind an API, highlighting challenges in real-world applications of such attacks.

Once again, the results for the flawed POKEMON setting are much better, yet modest compared to white-box performance, reaching up to 12% **TPR@FPR=1%** for black-box. Surprisingly, the grey-box scenario results in a worse attack than the black-box, but since the whole setting is incorrect, I feel there is no meaningful insight to be deducted from these results.

6.2. All generation strategies

Surprisingly enough, more complex *classifier* attacks utilizing machine learning algorithms usually fall short of the *threshold* attacks in the **TPR@FPR=1%** regime. Different strategies to obtain measurements rarely significantly boost performance, usually hurting rather than improving it.

Table 6.1. Comparison of the best white-box attacks to the grey- and black-box scenarios for LAION-mi and POKEMON datasets in the *threshold attack* setting.

SCENARIO	LOSS	METHOD	TPR@FPR=1%. ↑ LAION-MI	POKEMON
WHITE-BOX	MODEL LOSS	BASELINE LOSS THR.	1.92%±0.59	80.9%±2.27
		REVERSED NOISING	2.51%±0.73	97.3%±0.93
		PARTIAL DENOISING	2.31%±0.61	94.5%±1.34
		REVERSED DENOISING	2.25%±0.64	91.5%±1.63
	LATENT ERROR	REVERSED NOISING	1.26%±0.62	11.5%±1.84
		PARTIAL DENOISING	2.42%±0.62	99.5%±0.4
		REVERSED DENOISING	2.17%±0.64	61.1%±2.74
	PIXEL ERROR	REVERSED NOISING	1.90%±0.51	8.36%±1.66
		PARTIAL DENOISING	1.75%±0.68	25.38%±2.55
		REVERSED DENOISING	2.03%±0.55	12.0%±1.97
GREY-BOX	LATENT ERROR	GENERATION FROM PROMPT	0.93%±0.41	7.15%±1.5
BLACK-BOX	PIXEL ERROR	GENERATION FROM PROMPT	0.35%±0.19	12.0%±1.9

6.2.1. Threshold attack

Most of the proposed generation strategies from 5.2.10 end up around 2.0% for Model Loss, 1.5% **TPR@FPR=1%** for Pixel Error, while Latent Error performance differs significantly between different strategies. Generally, the best results are obtained from the Model Loss, which aligns with Sec. 2.3.3. Intuitively, this measurement should be the best way of capturing the differences between member and nonmember samples.

The Pixel Error leads to the least impressive results, barely outperforming the *baseline loss threshold* attack in the best *Reversed noising* setting. However, it is more robust to the generation process manipulation than the Latent Error measurement, which for *Full denoising start 100* drops to almost random guessing performance. In defense of Latent Error, the performance of *Partial denoising* is almost on par with the best Model Loss results.

6.2.2. Classifier attack

Only one classifier got better **TPR@FPR=1%** than the best *threshold attack* setting. Random Forest Classifier trained using data from the *Reversed noising* strategy reached 2.75% **TPR@FPR=1%**. The results of all other classifier and strategy combinations are close to the 2.0% level or significantly worse.

The worst classifier algorithm is the Decision Tree Classifier. It barely reaches performance of 1.1% **TPR@FPR=1%** for *Partial denoising non-iterative*, while it falls below the random guessing performance of 1% for all other strategies.

Neural Network stayed around the underwhelming performance of 1.3% on average, while Logistic Regression occupied the whole spectrum from random guessing performance to 2.4% **TPR@FPR=1%** for *Reversed noising regular cfg*.

The results for the RFC are by far the best, crossing the 2.0% **TPR@FPR=1%** in 11 out of 16 sampling strategies.

6.3. Bootstrapping

I highlight the importance of bootstrapping by comparing the best, worst, and average **TPR@FPR=1%** computed throughout 100 evaluation random samples. Figure 6.1 and 6.2 shows that for all of the generation strategies and attack types, the differences can reach an order of magnitude between the worst and the best random subset of M and NM.

This is especially important when evaluating the *classifier attack* since the split into train, validation, and test set is heavily influenced by randomness. Because of that, bootstrapping strategy 5.2.7 is so vital for the *fair* evaluation of this kind of *membership inference attacks*.

Table 6.2. Comparison of all of the generation strategies in the*threshold attack* setting. White-box scenario only.

LOSS METHOD	MODEL LOSS	LATENT ERROR	PIXEL ERROR
PARTIAL DENOISING	2.3%±0.61	**2.4**%±0.62	1.7%±0.68
PARTIAL DENOISING NON-ITERATIVE	2.3%±0.68	1.1%±0.46	1.39%±0.59
PARTIAL DENOISING LATENT SHIFT	2.3%±0.62	2.24%±0.9	1.6%±0.62
PARTIAL DENOISING TEXT SHIFT	2.2%±0.57	2.28%±0.57	1.7%±0.64
PARTIAL DENOISING BIGGER TEXT SHIFT	2.3%±0.62	2.22%±0.75	1.74%±0.6
PARTIAL DENOISING WRONG START	2.2%±0.69	2.13%±0.85	1.99%±0.56
FULL DENOISING START 100	2.08%±0.64	1.1%±0.41	1.84%±0.6
FULL DENOISING START 100 TEXT SHIFT	2.01%±0.6	1.1%±0.43	1.8%±0.6
FULL DENOISING START 50	2.0%±0.6	1.15%±0.38	1.95%±0.55
FULL DENOISING START 300	1.99%±0.61	1.34%±0.49	1.72%±0.64
SHORT DENOISING START 300	2.32%±0.67	2.06%±0.72	1.57%±0.67
REVERSED DENOISING	2.25%±0.64	2.17%±0.64	**2.03**%±0.55
FULL DENOISING 300 NO CFG	2.07%±0.62	1.45%±0.52	1.7%±0.6
FULL DENOISING 100 NON-ITERATIVE	2.2%±0.55	2.18%±0.75	1.91%±0.53
REVERSED NOISING REGULAR CFG	**2.5**%±0.74	2.03%±0.6	1.92%±0.5
REVERSED NOISING	**2.51**%±0.73	1.26%±0.52	1.9%±0.51

Table 6.3. Comparison of all of the generation strategies in the*classifier attack* setting. White-box scenario only.

Classifier class method	LR	DTC	RFC	NN
Partial denoising	1.83%±0.87	0.67%±1.09	2.27%±0.76	1.50%±0.92
Partial denoising non-iterative	2.24%±0.86	**1.10**%±1.12	2.35%±0.88	1.52%±1.03
Partial denoising latent shift	1.94%±0.92	0.86%±0.94	2.35%±0.86	1.40%±0.91
Partial denoising text shift	1.71%±0.87	0.49%±0.84	2.43%±0.93	1.40%±1.00
Partial denoising bigger text shift	2.14%±1.01	0.80%±1.13	2.30%±0.73	1.49%±0.90
Partial denoising wrong start	1.50%±0.70	0.74%±1.08	2.26%±0.87	1.51%±0.92
Full denoising start 100	1.09%±0.56	0.54%±0.74	1.89%±0.85	1.30%±0.76
Full denoising start 100 text shift	1.10%±0.62	0.62%±0.76	1.88%±0.80	1.24%±0.69
Full denoising start 50	1.16%±0.68	0.70%±0.68	1.89%±0.78	1.31%±0.78
Full denoising start 300	1.21%±0.58	0.42%±0.85	1.87%±0.87	1.39%±0.80
Short denoising start 300	1.94%±0.91	0.54%±0.83	2.31%±0.91	1.49%±0.85
Reversed denoising	1.96%±0.96	0.80%±1.17	2.37%±0.98	1.26%±0.78
Full denoising 300 no cfg	1.31%±0.73	0.31%±0.65	1.78%±0.78	1.37%±0.82
Full denoising 100 non-iterative	1.60%±0.77	0.68%±1.05	2.19%±0.85	1.60%±0.73
Reversed noising regular cfg	**2.41**%±1.09	0.47%±1.03	2.17%±0.84	1.40%±0.82
Reversed noising	1.98%±0.97	0.41%±1.02	**2.75**%±1.03	**1.62**%±0.86

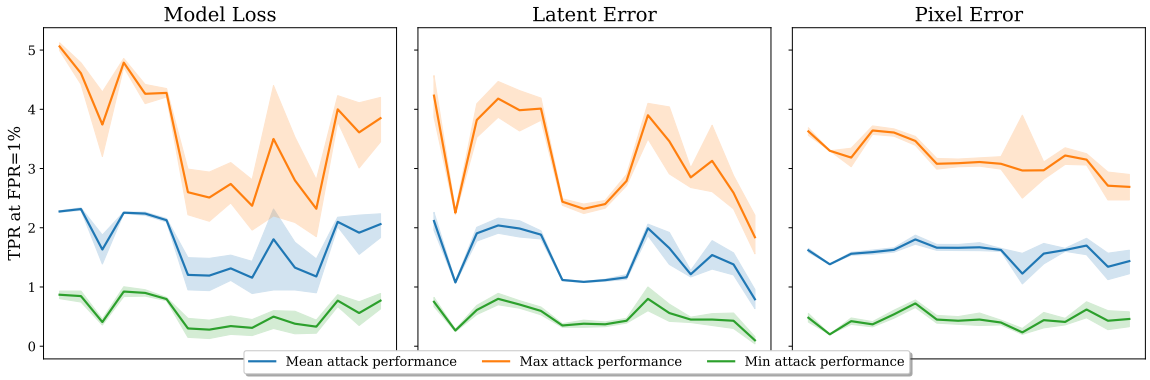


Figure 6.1. Difference between the worst, average, and best attack performance from the 100 bootstrapping runs, *threshold attack*.

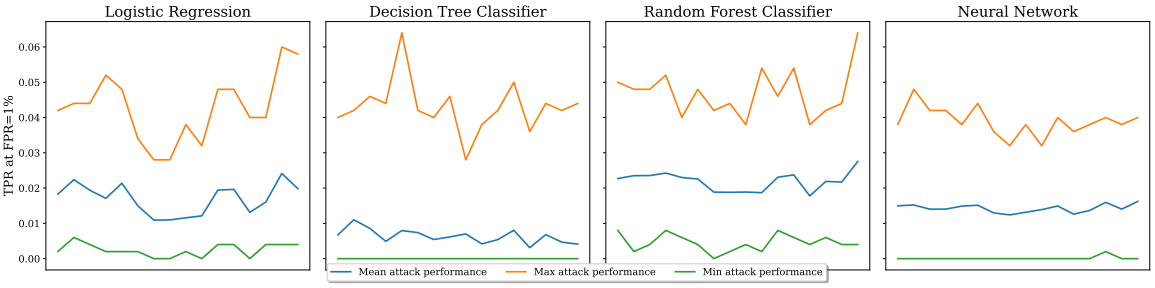


Figure 6.2. Difference between the worst, average, and best attack performance from the 100 bootstrapping runs, *classifier attack*.

7. Implementation Details

The following is a detailed description of the implemented software used to perform all experiments in this work. The repository with code is available on GitHub[2]. The folder structure resembles three main tasks I faced during work on my thesis:

1. SD-v1.4 finetuning on POKEMON dataset.
2. Creation of the **LAION-mi** dataset.
3. Evaluation of multiple MIAs in various scenarios.

The repository contains the following directories:

- attacks: code used to perform MIA Evaluation.
- data: to store datasets and outputs of scripts.
- laion-mi: software used to develop LAION-mi.
- logs: logs of ran jobs.
- models: fine-tuned checkpoints of SD-v1.4.
- pokemon-finetuning: source code used to fine-tune SD-v1.4.
- scripts: bash scripts with configured input arguments to Python scripts and *slurm* job definitions.

7.1. attacks directory

The following is the summary of contents of the *attacks* directory. This folder is responsible for evaluating MIAs against different models, datasets, and attacks.

7.1.1. analysis.py

This script generates plots and tables related to MIAs used in this thesis. It compiles the results obtained from the *mia.py* script runs, renders bootstrapping and POKEMON ROC plots, and creates tables with results of all MIAs for *threshold* and *classifier* attacks. The entry point is **.npz* files containing raw outputs of the *mia.py* script.

7.1.2. attack_utils.py

It contains helper functions needed to perform MIAs with different configurations in the *mia.py* script. It implements model and data loading utilities, inference, results calculation, and saving of the final results. It supports the configuration of the starting parameters of the attack and data transformations needed to run the attack.

7.1.3. attacks_config.py

This configuration file contains definitions of the attacks proposed in my thesis. It contains three constants:

- NAMES: list of all working names of the attacks.
- LABELS: list of all official names of the attacks.
- ATTACKS: dictionary of the parameters of all the attacks.

[2] https://github.com/Ankowa/inz

In the *ATTACKS* dictionary, attacks are defined by a set of parameters. Its structure is as follows:

Listing 1. *ATTACKS* dictionary's structure

```
{
    str: {  # attack name
        "start": int,    # starting timestep
        "stop": int,     # end timestep
        "step": int,     # step size
        "iterative": bool,  # True = noise every step
        "shift-latent-idx": int,     # timestep of latent shift
        "shift-text": bool, # True = noise text embedding
        "shift-by": float,  # text noise strength
        "start-noise-step": int,     # starting a_t
        "fix-noise-step": bool, # True = noise with a_t always
        "reverse-noise": bool,  # True = reverse timesteps
        "classifier-free-guidance": float,  # cfg strength
    }
}
```

7.1.4. classifiers_evaluation.py

This script performs steps described in Sec. 5.2.6 for all classifier classes but Neural Network. It loads the raw results for each attack and performs a Grid Search with 5-fold cross-validation. It heavily utilizes multiprocessing to speed up computations.

7.1.5. compile_experiments_data.py

The *mia.py* script is configured to support multi-GPU processing. However, the results of such runs are split into multiple files. The *compile_experiments_data.py* allows to combine these files into one file.

7.1.6. mia.py

This is the main entry point of this directory. It allows running all the MIAs implemented for any dataset and model combination relevant to this work. The input is the model, dataset, and attack name to perform. Then, utilizing functionality implemented in *attack_utils.py*, it runs the specified attack and saves the results. As mentioned, it supports multi-GPU processing by partitioning the samples between multiple processes.

7.1.7. run_nn_classification.py

Similarly to the *classifiers_evaluation.py* it evaluates the *classifier* attack using an NN classifier. To utilize the GPU to its maximum potential here, I rely on multiprocessing to split the training and obtain results quickly.

7.1.8. utils.py

This script provides general-use helper functions for raw data loading and plot rendering configuration.

7.2. laion-mi directory

The code for creating and analyzing the **LAION-mi** is implemented in this folder. Starting with downloading required data, through deduplication and sanitization algorithms, and ending with evaluation and plot rendering.

7.2.1. compute_img_embeddings.py

To evaluate the distribution mismatch (Sec. 4.5), the CLIP embeddings of images from M and NM are needed. This script computes and saves them in the specified location.

7.2.2. config.py

As with the *attacks* directory, the *laion-mi* directory's scripts share some common constants. In this case, the *config* file contains the output directories' names for various scripts, e.g., *L2_DISTANCES_DIR = "out/laion_2b_multi_l2_distances"* is the directory where the L2 distances needed for deduplication are stored.

7.2.3. deduplicate_and_extract.py

This script performs the deduplication procedure described in Sec. 4.4. Knowing the deduplication L2 threshold (here 0.5) is required beforehand, which is set based on the analysis performed in *deduplication_analysis.ipynb*. It takes nonmember candidates and their duplicate candidates and performs deduplication.

7.2.4. deduplication_analysis.ipynb

This Jupyter Notebook is used for L2 distance deduplication threshold selection and general analysis of the duplicates problem. It implements manual duplicate confirmation to pick the threshold that filters out the most duplicates possible.

7.2.5. download_data.py

Helpful script to download images from the set of URLs.

7.2.6. get_non_members.py

Useful script to download nonmembers from the metadata the *deduplicate_and_extract.py* script returns.

7.2.7. iterative_sanitization_results.py

This script computes the distribution match metrics from Sec. 4.5 on random M and NM and on sanitized M and NM for image and text data.

7.2.8. iterative_sanitization.py

Implementation of the sanitization Algorithm 1. It is required to perform the deduplication step first to run it. It returns the final members set.

7.2.9. load_and_filter_laion.py

Helpful script for downloading a specific subset of LAION dataset and applying filtering by the *aesthetic score*. The output is then used to extract the members and nonmembers sets.

7.2.10. query_knn_index.py

This script implements sourcing the duplicate candidates via LAION-5B KNN-index-based API [32]. It utilizes multithreading to run multiple requests parallelly, ultimately boosting performance. The resulting responses are aggregated and saved for further processing. The input is the nonmember candidates list.

7.2.11. utils.py

Similarly to the *attacks/utils.py*, this script provides helper functions and classes for raw data loading, multi-thread API querying, and configuration of the plot rendering parameters.

7.3. pokemon-finetuning directory

In this directory, two scripts are located:

- split_dataset.py: splits the POKEMON dataset into training (members) set and validation (nonmembers) set. The POKEMON dataset originally is not split.
- train_text_to_image.py: implements fine-tuning of the SD-v1.4

7.4. scripts directory

The experiments that require a high amount of computing power are run on the computing nodes that use the *slurm* scheduling system to distribute access to GPUs between users. This folder contains bash scripts that submit jobs to the slurm system and bash scripts to run the software with specified arguments directly on the GPU for less compute-intensive tasks like *classifier* attack.

7.4.1. finetune

This subdirectory contains bash scripts to run the finetuning operation on the SD-v1.4 on the POKEMON dataset on one GPU using slurm.

7.4.2. mia

This subdirectory contains bash scripts to run all membership inference attacks on multiple GPUs.

7.4.3. calc_fid_images.sh

The script to run the FID evaluation on the image data.

7.4.4. run_tests.py

Script to run *pytest* unit tests and *mypy* type checks on the source code.

7.4.5. split_pokemon.sh

Script to split the POKEMON dataset into train and validation sets.

7.5. Formatting and testing

The code is formatted using *black* formatter. For the type checking, I use the *mypy* tool, and for the unit tests and test coverage, I use *pytest* and *pytest-cov*.

8. Summary

This work achieved the following thesis goals:

1. Examination of the previous works in the context of membership inference attacks, focusing on the Stable Diffusion model. I point out the core contributions in these publications, relate them to my thesis topic, and identify pitfalls and flaws in the proposed methodologies.
2. Design and development of a *fair and rigorous* experimental setting allows for correctly evaluating the MIAs against SD-v1.4. The critical component of this setting is the **LAION-mi** dataset consisting of 40k member and 40k nonmember samples. Another essential component is the bootstrapping strategy, which allows fair and outlier-resistant comparison between different MIAs.
3. Proposal of novel MIA strategies directly tailored to the SD's architecture and the underlying generation process, most notably an approach of the diffusion process manipulation for more efficient membership information extraction from the model. Additionally, the introduction of *classifier attack* provides more insight into the membership inference problem.
4. Evaluation of such strategies in the proposed settings and comparison between a fair experimental setting and a flawed, fine-tuning base setting.
5. Analysis of the results, insights on different approaches' strengths and weaknesses, identification of the best-performing strategies.

This work has been published [40] at the Core A-grade conference: WACV2024, worth 140 points in the List of Conferences of the Polish Ministry of Science and Higher Education.

8.1. Conclusions

While achieving significantly better results than the *baseline loss threshold* attack, the proposed attacks' performance is far from practical. Unfortunately, the best result of 2.75% **TPR@FPR=1%** cannot apply to real-world scenarios like copyright claim lawsuits. Moreover, in the case of black-box LDMs hidden behind an API, the application of such methods is even more limited. First of all, the training set is usually private. Therefore, an attacker cannot construct members and nonmembers sets, which prohibits any evaluation of MIAs against such models. Secondly, even if the M and NM can be obtained, the performance of a black-box attack is worse than random guessing.

From the other perspective, the best-performing MIAs from literature need a set of *shadow models* to achieve their extraordinary results. Unfortunately, in the context of large LDMs like Stable Diffusion, it is prohibitively expensive to perform such an attack. In the real-world scenario, I argue that withdrawing the lawsuit is more cost-efficient than spending over 6.4M USD to prove that the defendant violated the copyright terms and get compensation from them.

While the *membership inference attack* against large diffusion models stays an open problem, I hope this work and the released dataset and experimental setting will help the scientific community evaluate their MIAs in a *fair and rigorous* manner.

8.2. Limitations

LAION-5B dataset, the source of data used both to train the Stable Diffusion model and to create the LAION-mi dataset, does not contain the images, just URLs to them. Therefore, to prepare a training set, it is necessary first to download the pictures using the URLs and then to train the model. Since the content that a URL points to is controlled by the owner of it, it might be the subject of unexpected editions or deletions, which cannot be handled reliably. Because of that, I cannot guarantee with 100% certainty that every member sample in LAION-mi is an actual member of the Stable Diffusion model's train set.

Approximately 10% of links are dead, i.e., it is impossible to download images from them. This limitation is inherent to the LAION-5B dataset and cannot be alleviated. Additionally, a URL to an image from LAION-mi has to be alive for this image to be used for the attack evaluation. I argue that it is unlikely that a URL from the LAION-mi members subset that was dead during the Stable Diffusion model training is now alive. However, such a scenario cannot be ruled out. Fortunately, since it affects only the members set, it makes the membership inference task harder, saving from the pitfalls described in Section 2.3.

8.3. Acknowledgements

I gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Centers: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2023/016361

załącznik nr 3 do zarządzenia
nr 28 /2016 Rektora PW

Politechnika Warszawska

Warszawa, 29.01.2024r.
……..
miejscowość i data

Antoni Kowalczuk
……..
imię i nazwisko studenta
303737
……..
numer albumu
Informatyka
……
kierunek studiów

OŚWIADCZENIE

Świadomy/-a odpowiedzialności karnej za składanie fałszywych zeznań oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie, pod opieką kierującego pracą dyplomową.

Jednocześnie oświadczam, że:

- niniejsza praca dyplomowa nie narusza praw autorskich w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.) oraz dóbr osobistych chronionych prawem cywilnym,
- niniejsza praca dyplomowa nie zawiera danych i informacji, które uzyskałem/-am w sposób niedozwolony,
- niniejsza praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów lub tytułów zawodowych,
- wszystkie informacje umieszczone w niniejszej pracy, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami,
- znam regulacje prawne Politechniki Warszawskiej w sprawie zarządzania prawami autorskimi i prawami pokrewnymi, prawami własności przemysłowej oraz zasadami komercjalizacji.

Oświadczam, że treść pracy dyplomowej w wersji drukowanej, treść pracy dyplomowej zawartej na nośniku elektronicznym (płycie kompaktowej) oraz treść pracy dyplomowej w module APD systemu USOS są identyczne.

Antoni Kowalczuk
.....
czytelny podpis studenta

References

[1] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, 2020. arXiv: 2006.11239 [cs.LG].

[2] Reuters, *Getty images lawsuit says stability ai misused photos to train ai*, https://www.reuters.com/legal/getty-images-lawsuit-says-stability-ai-misused-photos-train-ai-2023-02-06/, 2023.

[3] D. P. Kingma and M. Welling, *Auto-encoding variational bayes*, 2022. arXiv: 1312.6114 [stat.ML].

[4] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models", in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[5] J. Schmidhuber, *Generative adversarial networks are special cases of artificial curiosity (1990) and also closely related to predictability minimization (1991)*, 2020. arXiv: 1906.04493 [cs.NE].

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, 2014. arXiv: 1406.2661 [stat.ML].

[7] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, 2022. arXiv: 2112.10752 [cs.CV].

[8] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev, *Laion-5b: An open large-scale dataset for training next generation image-text models*, 2022. arXiv: 2210.08402 [cs.CV].

[9] CommonCrawl, *Commoncrawl*, https://commoncrawl.org/, 2023.

[10] C. C. org., *Cc by 4.0 deed*, https://creativecommons.org/licenses/by/4.0/deed.en, 2023.

[11] A. Krizhevski, *Learning multiple layers of features from tiny images*, https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf, 2009.

[12] M. Team, https://www.midjourney.com/, 2022.

[13] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation", in *International Conference on Machine Learning*, 2021.

[14] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, "Hierarchical text-conditional image generation with CLIP latents", *arXiv preprint arXiv:2204.06125*, 2022.

[15] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, vol. 313, no. 5786, pp. 504–507, 2006. DOI: 10.1126/science.1127647. eprint: https://www.science.org/doi/pdf/10.1126/

science.1127647. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.1127647.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *Imagenet large scale visual recognition challenge*, 2015. arXiv: 1409.0575 [cs.CV].

[17] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV].

[18] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi, *Photorealistic text-to-image diffusion models with deep language understanding*, 2022. arXiv: 2205.11487 [cs.CV].

[19] J. Ho and T. Salimans, *Classifier-free diffusion guidance*, 2022. arXiv: 2207.12598 [cs.LG].

[20] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, *Membership inference attacks from first principles*, 2022. arXiv: 2112.03570 [cs.CR].

[21] stability.ai, *Stable diffusion v1-4 model card*, https://huggingface.co/CompVis/stable-diffusion-v1-4, 2022.

[22] C. Schuhmann, *Clip+mlp aesthetic score predictor*, [Online], 2022. [Online]. Available: https://huggingface.co/datasets/FacePerceiver/laion-face.

[23] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, *Learning transferable visual models from natural language supervision*, 2021. arXiv: 2103.00020 [cs.CV].

[24] OpenReview.net, *Review of "membership inference attacks against text-to-image generation models"*, https://openreview.net/forum?id=J41IW8Z7mE, [Online], 2023.

[25] J. Duan, F. Kong, S. Wang, X. Shi, and K. Xu, *Are diffusion models vulnerable to membership inference attacks?*, 2023. arXiv: 2302.01316 [cs.CV].

[26] N. Carlini, J. Hayes, M. Nasr, M. Jagielski, V. Sehwag, F. Tramèr, B. Balle, D. Ippolito, and E. Wallace, *Extracting training data from diffusion models*, 2023. arXiv: 2301.13188 [cs.CR].

[27] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV].

[28] Q. Cao, L. Shen, W. Xie, O. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age", May 2018, pp. 67–74. DOI: 10.1109/FG.2018.00020.

[29] LAION, *Laion-face dataset*, [Online], 2022. [Online]. Available: https://github.com/christophschuhmann/improved-aesthetic-predictor.

[30] J. N. M. Pinkney, *Pokemon blip captions*, https://huggingface.co/datasets/lambdalabs/pokemon-blip-captions/, 2022.

[31] R. Webster, J. Rabin, L. Simon, and F. Jurie, *On the de-duplication of laion-2b*, 2023. arXiv: 2303.12733 [cs.CV].

[32] R. Beaumont, *Clip retrieval: Easily compute clip embeddings and build a clip retrieval system with them*, https://github.com/rom1504/clip-retrieval, 2022.

[33] D. T. Vo and R. Khoury, *Language identification on massive datasets of short message using an attention mechanism cnn*, 2019. arXiv: 1910.06748 [cs.CL].

[34] Wikipedia, *Rule of three (statistics) — Wikipedia, the free encyclopedia*, [Online], 2022. [Online]. Available: https://en.wikipedia.org/wiki/Rule_of_three_(statistics).

[35] 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*, 2018. arXiv: 1706.08500 [cs.LG].

[36] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)", *Computers and Geosciences*, vol. 19, no. 3, pp. 303–342, 1993, ISSN: 0098-3004. DOI: https://doi.org/10.1016/0098-3004(93)90090-R. [Online]. Available: https://www.sciencedirect.com/science/article/pii/009830049390090R.

[37] P. von Platen, S. Patil, A. Lozhkov, P. Cuenca, N. Lambert, K. Rasul, M. Davaadorj, and T. Wolf, *Diffusers: State-of-the-art diffusion models*, [Online], 2022. [Online]. Available: %5C%5Chttps://github.com/huggingface/diffusers/blob/%5C%5C79df50388df09d9615e3c067695a453bb0a694c0/examples/text_to_image/train_text_to_image.py#L4.

[38] A. F. Agarap, *Deep learning using rectified linear units (relu)*, 2019. arXiv: 1803.08375 [cs.NE].

[39] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[40] J. Dubiński, A. Kowalczuk, S. Pawlak, P. Rokita, T. Trzciński, and P. Morawiecki, "Towards more realistic membership inference attacks on large diffusion models", in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Jan. 2024, pp. 4860–4869.

List of Symbols and Abbreviations

MIA – Membership Inference Attack
SOTA – State-of-the-Art
SD – Stable Diffusion
NN – Neural Network
VAE – Variational AutoEncoder
GAN – Generative Adversarial Networks
LDM – Latent Diffusion Model
GPU – Graphics Processing Unit
GB – GigaByte
VRAM – Video Random Access Memory
AE – AutoEncoder
M – Members set
NM – Nonmembers set
FP – False Positive
FPR – False Positive Rate
FN – False Negative
TP – True Positive
TPR – True Positive Rate
ROC – Receiver Operating Characteristic
LiRA – Likelihood Ratio Attack
FID – Fréchet Inception Distance
PCA – Principal Component Analysis
LR – Logistic Regression
DTC – Decision Tree Classifier
RFC – Random Forest Classifier
ReLU – Rectified Linear Unit
API – Application Programming Interface

List of Figures

List of Tables