UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea Magistrale in Informatica

Text-to-image diffusion models attacks in scarce resource environments and privacy issues

Tesi di laurea di: Federico Fiorio

Relatore: Prof. Stefano Ferrari

Anno Accademico 2022/2023

dedicated to . . .

dedica

Contents

Chapter 1

Introduction

This thesis aims to present possible membership inference attacks (MIAs) that can be done on big text-to-image diffusion models with no particular hardware or software constraints.
The training of a diffusion model involves two main steps: forward process and reverse process. In the forward process the network iteratively transforms data to generate increasingly noisy samples (diffusion process); in the reverse process it learns how to generate the original data from the noisy versions by iteratively reversing the diffusion steps.

MIA is an attack on machine learning models; it is the benchmark for security in the machine learning field. If a MIA succeeds, the attacker can understand which data is part of the training set and which is not. A dataset with sensitive information, like profile photos or address photos, needs to be protected from these attacks in order to preserve people's sensitive information. The need of preserving privacy is concerning, especially with the increasingly availability of these models.
The state-of-the-art MIAs are currently focused on creating shadow models and achieving results based on accuracy. In this thesis, I explore another state-of-the-art MIA not related to shadow models.
This attack was first introduced in Membership Inference Attacks Against Text-to-Image Generation Models [67].

I applied more consistent metrics than accuracy to it, as stated in Membership Inference Attacks From First Principles [7].
I also explored different ways to change and improve this attack; in particular, I noted that the attack is very efficient when background is removed from profile photos, being able to achieve a TPR of 92% with a FPR of 0.1%.
I also developed a new MIA, still based on the paper [67] but that retrieves prompts instead of images from the training set.
I also addressed how privacy is protected with the use of differential privacy and why it is not very widely used in today's solutions. I also gave new ideas for future work in order to implement a more consistent method against MIAs.

The thesis structure consists of a background 2 on MIAs and diffusion models and the analytical and mathematical fundamentals in order to have a complete knowledge of the topic. This will explore other types of generation techniques 2.1.1 2.1.2 2.1.3, more classical, and I will make clear what the architecture of diffusion models is 2.2.3 and what the difference is between denoising diffusion probabilistic models 2.2.1 (DDPMs), denoising diffusion implicit models 2.2.4 (DDIMs), and latent diffusion models 2.2.5 (LDMs).
I will explain how MIAs are performed 2.3, why they are performed, and which models are used to perform them 2.4.
After the background part, I will introduce the technologies and methodologies used 3, as well as the datasets 3.4.
The following part consists of the experiments that have been conducted and the various innovations 4.
To conclude, I will present the results and how they are related to state-of-the-art ones 5.

Chapter 2

Background

In this chapter, I lay the foundation for a comprehensive understanding of my research by delving into the essential concepts and mathematical reasoning behind diffusion models 2.2. I will explore the origins and evolution of these models, clarifying the underlying principles that guide my experimental investigations. Additionally, I will elucidate the intricate world of text-to-image models 2.2.1, 2.2.4, 2.2.5 offering clear explanations and demystifying any specialized terminology. To ensure accessibility, I will introduce concepts such as differential privacy 2.5 and provide comprehensive coverage of all the necessary prerequisites. This chapter serves not only to provide context but also to empower readers, regardless of their familiarity with these topics, to engage deeply with my research and its implications.

2.1 Text-to-image models

Text-to-image models are a class of machine learning models that aim at generating images by describing text as input.

The main models used today for generating images are diffusion models [28][59], generative adversarial networks (GANs) [24], and variational autoencoders (VAEs) [33].

This thesis focuses on diffusion models, but I will simply explain how VAEs and GANs

work and why they are becoming less used than diffusion models.

2.1.1 Autoencoders

Autoencoders [5] are composed of two main components: an encoder that maps data from a high-dimensional space to a lower latent space, and a decoder that maps data from the latent space back to the input space.
The input space can be represented as R^n, the latent space can be represented as R^p and the output space can be represented as R^n; note that the input space should be the same as the output space. An intuition can be that if the autoencoder becomes good at reproducing input data, then it is possible to say that the model understood the intrinsic meaning of data, and thus the lower-dimensional latent space (which is the output of the encoder) is a good representation of data.

The main reason autoencoders were introduced in the literature is to represent high-dimensional data with a low set of features, or, in other words, to compress data.
The encoder part of the architecture can be used alone to reduce the dimensionality of data, in this case, the decoder can be discarded.
The decoder part is responsible for the generation of content, in fact we can give a random sample of the latent domain as input to the decoder that generates a sample in the output domain, more formally:
To generate a R^n element, a sample of the R^p is randomly drawn and passed to the decoder, which compute the R^n element.

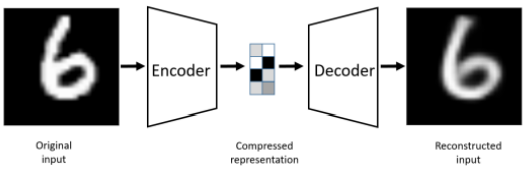

Figure 2.1: architecture of an autoencoder [5]

The main reason autoencoders do not sample well (generation) given a latent vari-

able is that their latent space is not isotropic, and thus it's difficult to get good samples from it.

For this reason, Variational autoencoders 2.1.2 were introduced in the literature.

2.1.2 Variational autoencoders (VAEs)

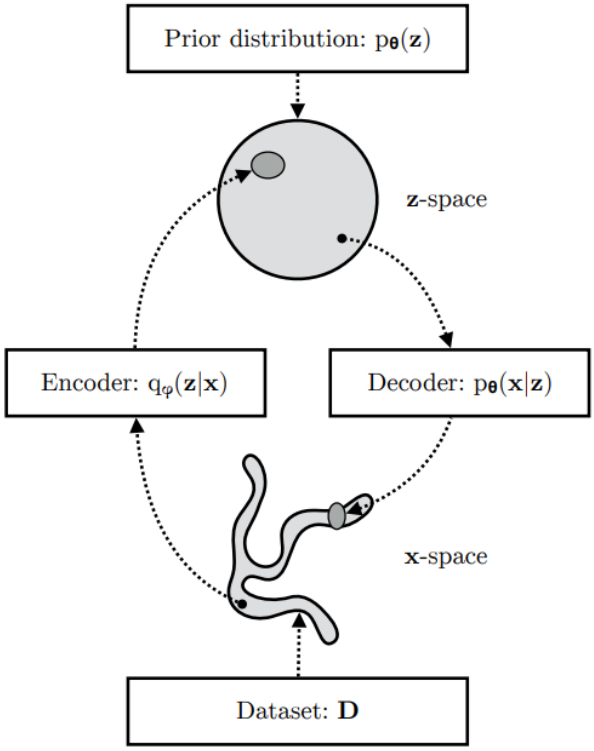

Figure 2.2: A VAE learns the mapping betweem the x space, which is usually a complicated distribution and a latent space, z-space, which is usually a simpler distribution (sphere in the image) [33].

The generative model learns a joint distribution $p_\theta(x,z) = p_\theta(z)p_\theta(x|z)$ with a prior distribution over latent space $p_\theta(z)$ and a stochastic decoder $p_\theta(x|z)$.

The stochastic encoder $q_\phi(z|x)$ is used instead of computing the intractable posterior $p_\theta(z|x)$ of the generative model.

VAEs derive from autoencoders, the main difference between these two is that VAEs apply a stochastic process to generate the output, while autoencoders are deterministic.

With VAEs, the latent space becomes closer to an isotropic space.
This is possible due to the stochastic process implemented in VAEs.
The latent variables now corresponds to the parameters of a variational distribution (usually a multivariate Gaussian distribution)
Typically, the variational distribution needs to be a unit normal (0 mean and 1 standard deviation) so it is easier for the decoder to generate new samples.
The output of the encoder is given as input to the decoder, which will then map from the latent space to the input space (as in the autoencoder architecture).

To better understand VAEs I can formalize the process [33]:
Given input data x the network gives as output x' which needs to be as similar as possible to x.
From the point of view of probabilistic modeling, one wants to maximize the likelihood of the data x by their chosen parameterized probability distribution $p_\theta(x) = p(x|\theta)$.
Distributions where a prior is assumed over the latents z results in intractable integrals.
$p_\theta(x) = \int_z p_\theta(x,z)\,dz$
where $p_\theta(x,z)$ represents the joint distribution under p_θ of the observable data x and its latent representation or encoding z. This can be re-written as: $p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)\,dz$
Now everything can be explained in terms of:

- Prior $p_\theta(z)$;
- Likelihood $p_\theta(x|z)$;
- Posterior $p_\theta(z|x)$.

$p_\theta(x)$ is intractable and must be re-written as: $q_\phi(z|x) \approx p_\theta(z|x)$.
where ϕ is defined as the set of real values that parametrize q.
For variational autoencoders, the learning process can be obtained by jointly optimize the generative model parameters θ to reduce the reconstruction error between the input and

the output, and ϕ to make $q_\phi(z|x)$ as close as possible to $p_\theta(z|x)$. As reconstruction loss, mean squared error and cross entropy are often used.
The loss is then optimized through backpropagation.
Now looking at the image **??** everything should be clear, the input data x is given to the probabilistic encoder which is responsible for the computation of the approximated posterior, the data then becomes part of the latent space on which the the conditional likelihood distribution is computed by the probabilistic decoder.
As in autoencoders, the decoder can be used alone for generation purposes, given as input a distribution from the latent space.

VAEs have more stable training than Generative Adversarial Networks (GANs), more on these in the following chapter 2.1.3, but they produce worse results compared to them [6].

2.1.3 Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN) is a machine learning architecture composed of two neural networks: the generator and the discriminator.
I will consider only GANs that are used for image generations, an example can be DCGAN [49].This network takes as input 100 random numbers drawn from a uniform distribution and outputs an image.
It is possible to formalize the training and components of a generator by looking at the image **??**, [3].
The generator is responsible for creating (in the case of images) photo-realistic images which will be the green region.
The blue region shows the part of the image space that, with a high probability (over some threshold) contains real images, the black dots are data points in the dataset representing images.
The green region is nothing but a distribution $\hat{p}_\theta(x)$ that is defined implicitly by taking points from a unit Gaussian distribution (red) and mapping them through a (deterministic) neural network (generator of the GAN).

θ represents the parameters of the network, they need to be learned to represent the generated distribution $\hat{p}_\theta(x)$ as close as possible to the true data distribution $p(x)$.

This is a classical ML problem, as usual, there is first need to define a loss function in order to understand if the generated distribution is becoming similar to the true data distribution or not.

A common measure between probability distribution is the KL divergence, from which it is possible to get the KL divergence loss that can be used during training; the smaller the loss, the better.

This was just the generator of a GAN, there is also another network inside this architecture, which is the discriminator.

The GAN is trained as a zero-sum game, where the gain of one agent means the loss of the other.

The discriminator tries to classify samples as either coming from the true distribution $p(x)$ or the generated distribution $\hat{p}(x)$.

When the discriminator is able to distinguish between the two images coming from two different distributions, then the generator needs to adjust its parameters in order to 'fool' the discriminator.

The training is completed when the discriminator is not able to tell anymore the difference between the two distributions (random guessing).

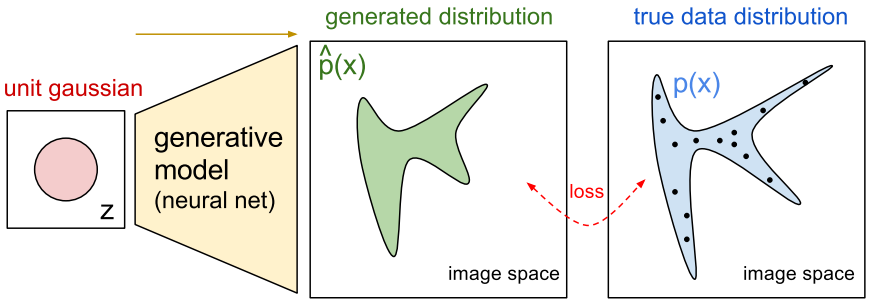

Figure 2.3: Image taken from [3] article, generative models. The green distribution is generated by the generator from a unit gaussian, while the true data is represented by the blue distribution. The objective is to minimize the distance between these two distributions.

The generator is similar to the decoder in VAEs, it maps from a latent space (parameters of a distribution) to a set of images.
After the training is completed, the discriminator is usually thrown away and the generator is kept.

These models have a 'long' history in the world of image generation, and their results are considered photo-realistic.
However, These models have a hard time in training, sometimes it results in suboptimal results, and a lot of hyperparameter fine-tuning and tricks are necessary to make them work.
For example, in the paper [4], it is emerged that a too good discriminator, might make impossible the training process for the generator, keeping it at its starting phase.
Other problems are present and tweaking operations are required, however if intrested, read the cited paper.

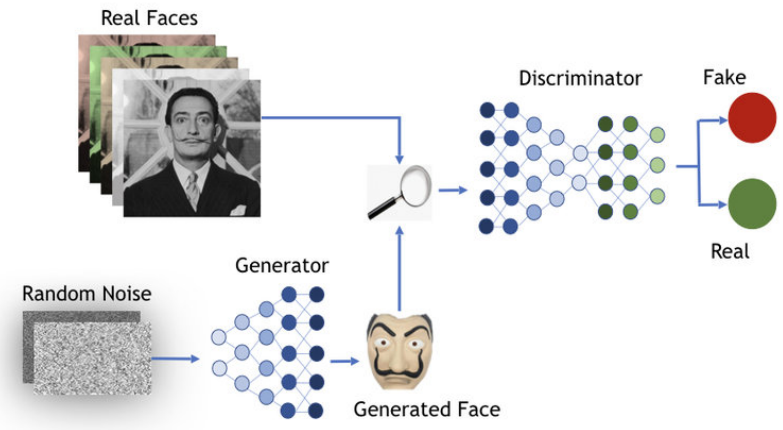

Figure 2.4: architecture of a GAN [23]

GANs are still competing in the state-of-the-art models for image generation, especially because the sampling is way faster than diffusion models.

2.2 Diffusion models

Diffusion models are a class of latent variable models, that are inspired by the diffusion process in thermodynamics [28][59].
In the context of thermodynamics, diffusion refers to the phenomenon where molecules move randomly from regions of high concentration to those of lower concentration until equilibrium is reached.
Denoising diffusion models leverage this same concept to reduce noise in signals or images by modeling noise as the random movement of particles, denoising algorithms aim to restore the original signal by simulating the diffusion process.
Diffusion models achieve state-of-the-art results beating also GANs [16].
The only downside of these models is the time it takes for sampling from them, however, some steps forward have been accomplished and some other approaches have been taken e.g. denoising diffusion implicit models (DDIM) 2.2.4.

Many different architectures of diffusion models exist, I would like to make clear the difference between denoising diffusion probabilistic models (DDPMs) 2.2.1 and denoising diffusion implicit models (DDIMs) 2.2.4.

2.2.1 Denoising diffusion probabilistic model (DDPM)

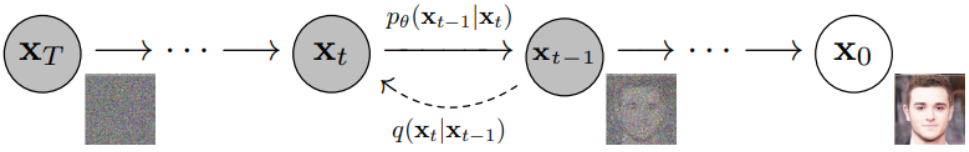

Figure 2.5: Representation of forward and reverse process as graphical model [28]. The denoising process is realized by reverting the noise diffusion process.

Formalization of DDPMs [28]:
Diffusion models are latent variable models.

In order to understand these models, three fundamentals steps needs to be formalized:

- Forward process;
- Reverse process;
- Training.

The forward process or diffusion process, is fixed to a Markov chain that gradually adds Gaussian noise to the data according to a variance schedule $\beta_1, ..., \beta_T$.
The forward process or diffusion process is formalized as:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.1}$$

The above equation, represents the transition probability that adds Gaussian noise to the previous state x_{t-1} as shown in Fig. 2.5, with β_t being a schedule of increasing noise levels.
The noise schedule is typically annealed from a high value to a low value over the course of the training.
$q(x_{1:T}|x_0)$ indicates all the possible noise steps $x_{1:T}$ given the initial state of the noise-free image x_0 .
$\prod_{t=1}^{T}$ represents the scheduler of increasing noise levels and $q(x_t|x_{t-1})$ represents the iterative process where the current state x_t derives from the previous state x_{t-1}. The transition probability of the t-th step can hence be formalized as:

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \tag{2.2}$$

where β_t indicates at each step the trade-off between information to be kept from the previous step and new noise to be added.
It is also possible to write:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\varepsilon_t \qquad \varepsilon_t \sim \mathcal{N}(0, \mathbf{I}) \tag{2.3}$$

The diffusion process above formalized allows to express the process at any step t (relating x_t and x_0) as:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.4}$$

where:

$$\alpha_t = (1-\beta_t) \qquad \text{and} \qquad \bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s = \prod_{s=1}^{t} (1-\beta_s)$$

From the Markov property, the probability of a forward chain is written as (same as the original formalization):

$$q(x_{0:T}) := q(x_0) \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{2.5}$$

The joint distribution $p_\theta(x_{0:T})$ is called the reverse process, and it is defined as a Markov chain with learned Gaussian transitions starting at $p(x_T) = \mathcal{N}(x_T; 0, \mathbf{I})$.
The reverse process is formalized as:

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{2.6}$$

Equation (2.6) describes the probability of a specific backward trajectory, parametrised by θ, which are the parameters of the model that must be learned.
The distribution of the input data, $p(x_T)$ is just an isotropic gaussian distribution that does not depend on θ, while the functions in the product represent the steps in the reverse process.
More formally:

$$p(x_T) = \mathcal{N}(x_T; 0, \mathbf{I})$$

The following equation represents a single step in the reverse process 2.5:

$$p_\theta(x_{t-1}|x_t) := \mathcal{N}(x_{t-1};\ \mu_\theta(x_t,t),\ \Sigma_\theta(x_t,t)) \tag{2.7}$$

where μ_θ and Σ_θ are two functions parametrised by θ that map the current state x_t and time t to the mean and covariance of the Gaussian transition probability.
Hence, given the state x_t, at time t, the previous state x_{t-1} is obtained by sampling the distribution in (2.7), for suitable mean and covariance that need to be learned during the training. In the original paper the covariance is fixed and only the mean is learned.

Training is performed by optimizing the usual variational bound on negative log likelihood:

$$\mathbb{E}[-\log p_\theta(x_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right] = \mathbb{E}_q \left[-\log p(x_T) - \sum_{t \geq 1} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})} \right] =: L \tag{2.8}$$

During optimization, the forward process is used to sample latent variables from the posterior distribution, and the reverse process is used to simulate trajectories of the Markov chain that start at the endpoint of the forward process and move backwards in time to estimate gradients with respect to the model parameters (θ).
The goal of training is to have reverse and forward process to follow the same distribution.

As formalized before, two different processes in diffusion models are present.
One forward process and one reverse process, both previously formalized.
In the forward process, Gaussian noise is added iteratively to a given image (a sample from the training dataset).
This process is simply a preparation of the data for training.
The goal of this model is to try to understand how much noise was added to an image during each step.
If the model can do it, then a strong tool for generation is achieved.
In order to aim for this goal a reverse process was defined.
This last one starts from the last noisy image and tries to remove noise iteratively until the starting image is re-created.
To be trained and learn how to perform a good reverse process a loss function is defined.
This last one, let the model understand if the noise that it is predicting is similar to the true noise generated during forward process or if it is really far away from the truth.
In order to train the model the negative log-likelihood is used, this is previously formalized as:

$$\mathbb{E}[-\log p_\theta(x_0)] \tag{2.9}$$

The probability of x_0 that depends on all other random variables need to be computed, this probability will need to track all other x_{t-1} variables which is not possible in practice.
The variational lower bound is used for this purpose (previously normalized as L).

After a bunch of simplifications the complexity of L can be further reduced and the final objective to minimize while training becomes:

$$L_{\text{simple}} = \mathbb{E}_{t,x_0,\varepsilon}\left[||\varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)||^2\right] \tag{2.10}$$

2.10 can also be written as:

$$L_{\text{simple}} = \mathbb{E}_{t,x_0,\varepsilon}\left[||\varepsilon - \varepsilon_\theta(x_t, t)||^2\right] \tag{2.11}$$

where $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon$ which is a random step of a random trajectory in the training algorithm context.

The variance is fixed in the original paper, however some modifications have happened in the literature.
A fixed variance is found to be suboptimal, the variance can be learned from the noise as well.
Nichol and Dhariwal [43] proposed a hybrid objective for training both mean and variance using the weighted sum $L_{\text{simple}} + \lambda L_{\text{vlb}}$ where vlb means variational lower bound. This new approach permits to have fewer sampling steps without a big loss in quality.

Before formalizing training and sampling algorithms like in the original paper [28] and try to describe them in a less mathematical way, I will point out the parameters that a DDPM needs to learn during training.
During training the parameters to learn are μ_θ and Σ_θ by minimizing the lower bound 2.8

Algorithm 1 Training [28]

1: **repeat**
2: $x_0 \approx q(x_0)$
3: $t \approx \text{Uniform}(\{1,\dots,T\})$
4: $\varepsilon \approx \mathcal{N}(0,I)$
5: Take a gradient descent step on:
$\nabla_\theta \|\varepsilon - \varepsilon_\theta(x_t,t)\|_2^2$
6: **until** converged

The following explanation will refer to Training algorithm 1:

line 1: begin iterative process;

line 2: draw batch of samples from training sample distribution, that is, perform forward process;

line 3: perform uniform timesteps from 1 to T;

line 4: draw some noise with same dimensionality as input data;

line 5: use parametrization trick to produce samples at timestep t and then give that sample to your model and learn what was the noise used at that timestep t via gradient descent;

line 6: until convergence, repeat iterative process.

In short, a noise prediction network is being trained (predict noise that was used at timestep t).

Algorithm 2 Sampling [28]

1: $x_T \approx \mathcal{N}(0, \mathbf{I})$
2: **for** $t = T$ to 1 **do**
3: $\quad \mathbf{z} \approx \mathcal{N}(0, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = 0$
4: $\quad x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(x_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **Return** x_0

The following explanation will refer to Sampling algorithm 2:

line 1: the image x_T is a noisy image drawn from normal distribution;

line 2: begin for loop that lasts T steps;

line 3: $\mathbf{z}$ represents noise drawn from a normal distribution, unless in the last step (t = 1), where no noise is added (since it generates x_0);

line 4: the reverse process is used so, from last step T samples are drawn from normal distribution, then iteratively the sample is refined. From sample x_t the noise predicted

is removed and the normalized noise is added. The normalization is based on the standard deviation of the sample preceding x_{t-1}. The process is iterated and when it arrives at timestep 0 the final image, x_0, is generated, this one has no more noise added to it;

line 5: the loop ends;

line 6: return the sample.

2.2.2 Conditional models and guidance

The model describe in section 2.2.1 allows to generate a sample from a given distribution, which is represented by the training dataset. This collection is usually very large and include subsets with similar features. If some features of the samples are known, this information can be introduced in the training to allow the generation of new samples with a given feature.

A conditional model is a model that takes in input, not just a sample but also a class label, or in the case of diffusion models for text-to-image generation, it will take a text embedding.

The text embedding will be given as input to the model in every step of the training to be conditioned on those embeddings.

The forward process of a conditioned diffusion model stays the same.

The reverse process and the variational bound can be formally written as the following, were $\mathbf{c}$ is the input on which we want to condition the model:

$$p_\theta(x_{0:T}|\mathbf{c}) := p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t,\mathbf{c}) \tag{2.12}$$

The following equation represents a single step in the reverse process 2.5:

$$p_\theta(x_{t-1}|x_t,\mathbf{c}) := \mathcal{N}(x_{t-1};\, \mu_\theta(x_t,t,\mathbf{c}),\, \Sigma_\theta(x_t,t,\mathbf{c})) \tag{2.13}$$

The variational bound can be re-written as:

$$\mathbb{E}_q\left[-\log p(x_T|\mathbf{c}) - \sum_{t\geq 1}\log\frac{p_\theta(x_{t-1}|x_t,\mathbf{c})}{q(x_t|x_{t-1},\mathbf{c})}\right] =: \mathbb{L} \tag{2.14}$$

The conditioning applied during training might not be enough for good sampling. Two ways of guiding the sampling process were introduced in the literature: classifier guidance [16] and classifier-free guidance [29].

In the classifier guidance, an additional model is being used, a classifier.

This classifier will guide the diffusion model simply by taking the gradient on the prediction of the class of an image and using it to perturb the mean of the conditioned model.

In practice, the gradient of the two models are mixed.

By using this perturbation, in the lantent space the information is becoming closer to a situation in which the embedding of the text well represents the image.

In other words, a better spot is found to represent the two components (sample and embedding) in the latent space.

Given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$ and a gradient scale s, an increasing s improves sample quality but reduces sampling diversity:

Algorithm 3 Sampling with Classifier Guidance [16]

1: **Input:** class label y, gradient scale s
2: $x_T \leftarrow \mathcal{N}(0, \mathbf{I})$
3: **for** $t = T$ to 1 **do**
4: $\quad \mu,\ \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
5: $\quad x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma\nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
6: **end for**
7: **Return** x_0

The sampling (line 4) can be seen as taken from the following modified score:

$$\nabla_{x_t} \left[\log p(x_t|c) + \omega \log p(c|x_t)\right] \tag{2.15}$$

A larger ω gives a better overall quality but less diversity (the samples will be more similar between each other).

Given the above definitions, formally writing the classifier guided model should not be a surprise:

$$\tilde{p}(x_t|c) \propto p(x_t|c)p(c|x_t)^\omega \tag{2.16}$$

where the first term $p(x_t|c)$ is the score from the diffusion model and the second term $p(c|x_t)^{\omega}$ is the score from the classifier guidance.

In classifier-free guidance, the model's knowledge about its class conditioning is used as the classifier knowledge in the classifier guidance.
A conditioned and an unconditioned model are jointly trained.
The classifier-free guidance is sometimes referred to as "implicit classifier".
The sampling from this model is fomalized as:

$$p(c|x_t) \propto p(x_t|c)/p(x_t) \tag{2.17}$$

The model 2.16 is similar to 2.17, but with the difference of not having a classifier.
In 2.17 the term $p(x_t|c)$ is referred to the conditional diffusion model and the term $p(x_t)$ is referred to the unconditional diffusion model.
The sampling from the classifier guidance model was defined as 2.15, now the modified score of the model is defined as:

$$\nabla_{x_t}\left[(1+\omega)\log p(x_t|c) - \omega \log p(x_t)\right] \tag{2.18}$$

Another intuition: imaging having the two points in the space, the one that predicts the noise of an unconditioned image and the one that predicts the noise of a conditioned image, it is possible to sort of find the best direction to push the prediction of noise by using these points.

2.2.3 Introduction to Deep neural networks

In this chapter I will make a short introduction on deep neural networks clarifying some fundamental concepts in order to better understand the architecture of a diffusion model.
A network is considered deep if it has multiple layers, usually more than 3, but this is not a strict definition.
Deep learning was made possible thanks to the backpropagation algorithm discovered in 1986 with the publication of: Learning representations by back-propagating errors [54].
A use case of deep learning discovered in 1998 [37] and used still today is the convolutional neural network (CNN).

Convolutional neural network is a regularized type of feed-forward neural network that learns feature engineering by itself via filters (or kernel) optimization. CNNs are used for many different tasks, some examples are: image and audio classifications.

The architecture of a CNN is structured with an input layer, hidden layers, and an output layer.

In the realm of CNNs, the hidden layers encompass one or more layers that specialize in performing convolution operations. Typically, this includes a layer responsible for computing the dot product between the convolutional kernel and the input matrix of that layer. This product usually takes the form of the Frobenius inner product, and its activation function is most commonly the Rectified Linear Unit (ReLU).

As the convolutional kernel sweeps across the input matrix of the layer, it executes a convolution operation, resulting in the creation of a feature map. This feature map, in turn, contributes to the input of the subsequent layer. This process is followed by the integration of other layers, such as pooling layers, fully connected layers, and normalization layers.

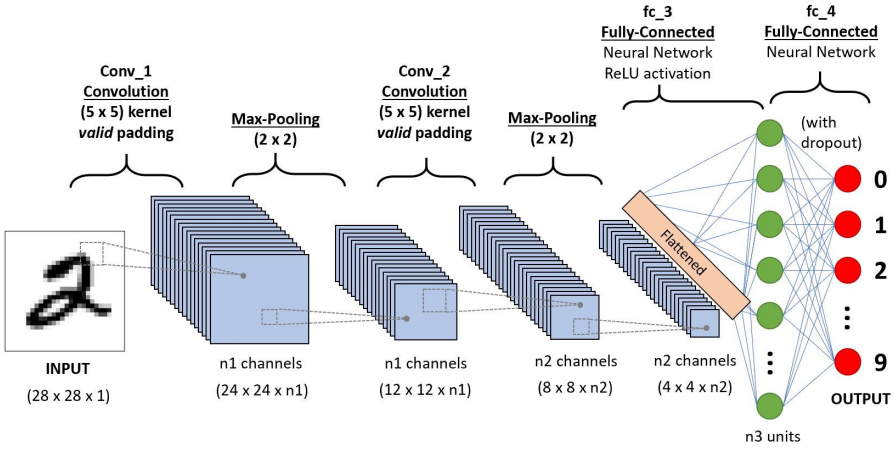

Figure 2.6: Image taken from `https://guandi1995.github.io/Classical-CNN-architecture/`, classical CNN architecture

Convolutional layers are responsible for the learning of the CNN and are able to abstract the input to feature maps. It is a process where a small matrix of numbers (called kernel or filter), is passed over the image and transforms it based on the values from the

filter.

Pooling layers are responsible for dimensionality reduction.

Fully connected layers connect every neruon in one layer to every neuron in another layer, they are used to flatter the input matrix and use it for classifications purposes.

It is possible to look at Figure 2.6 to have a visual understanding of the network.

A convolution can be formalized as:

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \tag{2.19}$$

where the input image is f, the kernel is h; the indexes of rows and columns of the result matrix are m and n respectively.

The training of a CNN happens through backpropagation here are the fundamental steps, more details in the paper [54]:

1. During the forward pass, input data is passed through the network layer by layer. Each layer performs a set of computations, which are often linear transformations (e.g., matrix multiplications) followed by non-linear activation functions (e.g. ReLU or Sigmoid). The forward pass computes the network's output, which is then compared to the actual target values to calculate a loss or error.
   This error quantifies how far off the network's predictions are from the true values.

2. Backpropagation starts by computing the gradient of the loss with respect to the output of the network. This is often done using the chain rule of calculus.
   The gradient represents how much the loss would change with small adjustments in each output value. It essentially tells the direction and magnitude to adjust each output to reduce the error.

3. The gradients calculated in the previous step are then propagated backward through the network.
   For each layer, the gradient of the loss with respect to the layer's input is computed. This involves applying the chain rule again to calculate how much each neuron in the previous layer contributed to the error.
   The gradients are accumulated and used to update the layer's parameters (weights and biases) in a way that minimize the loss.

4. After obtaining the gradients for each parameter in the network, these are updated to minimize the loss. This is typically done using optimization algorithms like Gradient Descent.

   The direction and magnitude of parameter updates are determined by the gradients. the parameters move in the opposite direction of the gradient to decrease the loss. Learning rate is a hyperparameter that controls the step size during parameter updates. It prevents overshooting the minimum of the loss function.

5. Steps 1 to 4 are repeated for multiple iterations (epochs, an epoch is a full iteration on the training set).

   During each epoch, the network refines its parameter values to make better predictions. Over time, the network's loss decreases, and its accuracy on the training data improves.

Another helpful concept are residual networks (ResNets) [26], they learn residual functions with reference to the layer inputs, instead of learning unreferenced functions. They stack residual blocks ontop of each other to form network: e.g. a ResNet-50 has fifty layers using these blocks.

Diffusion model architecture: U-net

Architectures are different in every model, some hyperparameters might change, maybe some layers or some blocks.
However, the usual architecture that is used for the reverse process is typically a U-net 2.7.
This architecture takes its name from the shape that it has.
A U-net is very similar to an autoencoder 2.1.1, it was first introduced [53] for image segmentation for biomedical images.
In the current literature, it is seen as a denoising autoencoder with a bottleneck for compressing information analogously to autoencoders.
A typical U-net has ResNet [26] blocks, which try to solve the vanishing/exploding gradient problem by using skip connections.

A skip connection connects two activations of two layers by skipping some layers in between.

The advantage is that any unnecessary layer will be skipped by regularization.

Another important block that is used inside a U-net is usually attention, typically for text-to-image diffusion models.

The cross-attention mechanism helps to concatenate the embedding of the text (conditional model) with the embedding of the image, in order to find a sweet spot for both in the latent space.

Some architectures also like to work with low resolution images and then upsample them to high quality resolution images via a super resolution procedure applied after or at the end of the U-net.

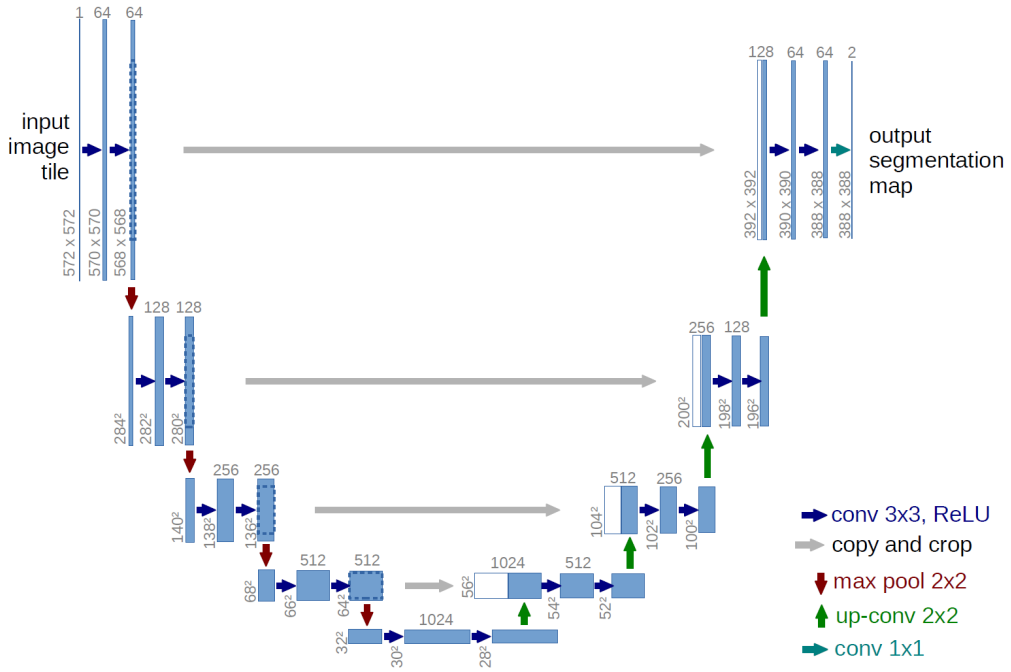

Figure 2.7: Image taken from `https://paperswithcode.com/method/resnet`, classical U-net architecture

2.2.4 Denoising diffusion implicit model (DDIM)

The problem with DDPMs is that sampling is slow; GANs achieve way faster sampling than DDPMs because they do not apply an iterative process.

By making the DDPMs not based on a Markov chain, the number of iterations can be decreased and it is possible to achieve the same or better quality in sampling.
The original paper [59] defines a new family of forward processes indexed by σ:

$$q_\sigma(x_{t-1}|x_t,x_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}x_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\frac{x_t-\sqrt{\alpha_t}x_0}{\sqrt{1-\alpha_t}}, \sigma_t^2\mathbf{I}\right) \tag{2.20}$$

where the mean is chosen to ensure that for all t:

$$q_\sigma(x_t|x_0) = \mathcal{N}\left(\sqrt{\alpha_t}x_0, (1-\alpha_t)\mathbf{I}\right)$$

The above condition on $q_\sigma(x_t|x_0)$ is very important because it guarantees the property of DDPMs that permits to recreate noise in an image at step t with just the initial noise in the image at step 0.
However in 2.20 a property of DDPMs is lost.
The process is no longer Markovian, so an image at the step t cannot be defined just by the image at step t-1.
The single forward process, can be derived from the Bayes' rule:

$$q_\sigma(x_t|x_{t-1};x_0) = \frac{q_\sigma(x_{t-1}|x_t,x_0)q_\sigma(x_t|x_0)}{q_\sigma(x_{t-1}|x_0)} \tag{2.21}$$

σ_t determines the stochasticity of the process:

- if $\sigma_t = \sqrt{(1-\alpha_{t-1})/(1-\alpha_t)}\sqrt{1-\alpha_t/\alpha_{t-1}}$ the definition matches the one of the DDPMs.
- if $\sigma_t = 0$ then, the forward process becomes deterministic.

This indicates that the DDIMs are part of the same family of functions as the DDPMs. Also the reverse process becomes determistic with a sigma of 0. This means that the same original noise leads to the same image.
The training objective is equivalent for any value of σ, this means that a model trained for the original DDPM process can be used for any process of the family.
The two generative formulas are different between DDPMs and DDIMs, but the model

trained is the same for both approaches, because the training objective is the same for both.

As it is possible to see in 2.8 since the process is not Markovian, it is possible to skip some connections and sample with fewer steps.

The generative process defined in [59] is $p_\theta(x_{0:T})$, where each $p_\theta^{(t)}(x_{t-1}|x_t)$ leverages knowledge of $q_\sigma(x_{t-1}|x_t,x_0)$, previously defined.

For some $x_0 \sim q(x_0)$ and $\varepsilon_t \sim \mathcal{N}(0,\mathbf{I})$, x_t can be obtained from the following formula, which is a linear combination of x_0 and noise ε, it can be applied also to DDPMs:

$$x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\varepsilon, \qquad \varepsilon \sim \mathcal{N}(0,I) \tag{2.22}$$

Then, the model $\varepsilon_\theta^{(t)}(x_t)$ attempts to predict ε_t from x_t without knowledge of x_0.

By rewriting 2.22 it is possible to predict x_0 given x_t.:

$$f_\theta^{(t)}(x_t) := \frac{(x_t - \sqrt{1-\alpha_t}\cdot\varepsilon_\theta^{(t)}(x_t))}{\sqrt{\alpha_t}} \tag{2.23}$$

The generative process with a fixed prior $p_\theta(x_T) = \mathcal{N}(0,\mathbf{I})$:

$$p_\theta^{(t)}(x_{t-1}|x_t) = \begin{cases} \mathcal{N}(f_\theta^{(1)}(x_1), \sigma_1^2\mathbf{I}), & \text{if } t=1 \\ q_\sigma(x_{t-1}|x_t, f_\theta^{(t)}(x_t)), & \text{otherwise} \end{cases} \tag{2.24}$$

where $q_\sigma(x_{t-1}|x_t, f_\theta^{(t)}(x_t))$ is defined from 2.20 with x_0 replaced by $f_\theta^{(t)}(x_t)$.

From $p_\theta(x_{1:T})$ in 2.24, it is possible to generate a sample x_{t-1} from a sample x_t with the following:

$$q_\sigma(x_{t-1}|x_t,x_0) = \mathbb{N}\left(\sqrt{a_{t-1}}x_0 + \sqrt{1-a_{t-1}-\sigma_t^2}\cdot\frac{x_t-\sqrt{a_t}x_0}{\sqrt{1-a_t}}, \sigma_t^2\mathbf{I}\right) \tag{2.25}$$

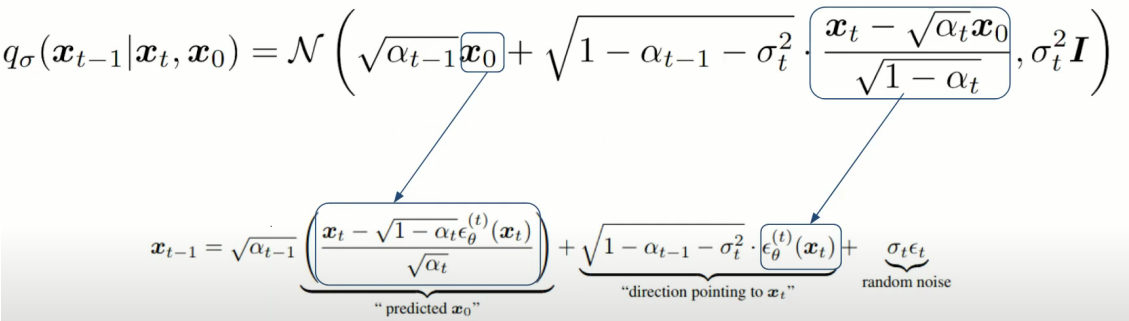

Figure 2.8: The terms from the family 2.20 are substituted in order to get a sampling formula for x_{t-1} given x_t in the general case [59].

The reverse process can be generalized to any subsample of steps and make it faster:

$$p_\theta(x_{0:T}) := p_\theta(x_T) \prod_{i=1}^{S} p_\theta^{(\tau_i)}(x_{\tau_{i-1}}|x_{\tau_i}) \times \prod_{t \in \bar{\tau}} p_\theta^{(t)}(x_0|x_t) \tag{2.26}$$

Most high-level features are similar, regardless of the generative trajectory.

In many cases, samples generated with only 20 steps are already very similar to others generated with 1000 steps in terms of high-level features, with only minor differences in details.

2.2.5 Latent diffusion model (LDM)

This section is an introduction to latent diffusion models (LDMs) [51]. These models have been introduced to make the inference and training steps in text-to-image generation less expensive than in previous DDPM 2.2.1 and DDIM 2.2.4 models, by using a latent space to perform computations.

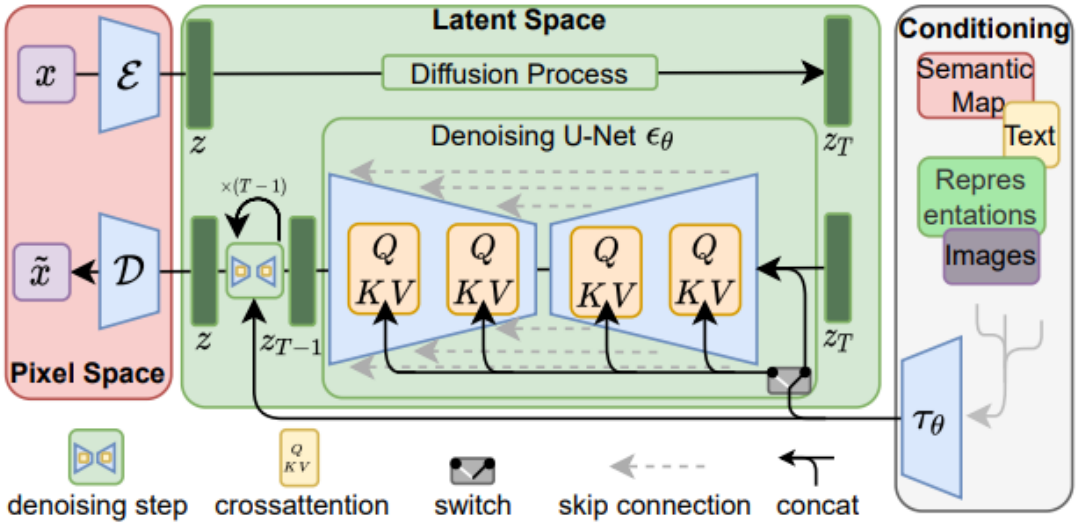

Figure 2.9: Visual explanation of LDMs [51]. In red is represented the input space as x and output space as $\tilde{x}$, their distributions will be as similar as possible, the x input is transform to a latent space, while the output is re-transformed from the latent space. In green the latent space with diffusion and reverse process, in white the possible conditionings.

Starting from 2.10 which is the usual training objective for text-to-image diffusion models, in [51] a more appealing training objective is proposed, which is specific for this architecture.

The trained encoder $\mathcal{E}$ and decoder $\mathscr{D}$ play an important role 2.9.

They are able to compress and decompress data so that, the model can focus on only important semantic bit of information and also work in a lower-dimensional space.

The model takes advantage of image-specific inductive biases to build a more specific U-net with 2-D convolutional layers to further focus on important information inside images.

The objective for training is here focused on relevant bits:

$$L_{LDM} := \mathbb{E}_{\varepsilon(x),\varepsilon\sim\mathcal{N}(0,1),t}\left[||\varepsilon - \varepsilon_\theta(z_t,t)||_2^2\right] \tag{2.27}$$

The model can also incorporate conditioning. It can be achieved with a conditional denosing autoencoder: $\varepsilon_\theta(z_t,t,y)$

A domain-specific encoder τ_θ is introduced, it is able to project a conditioning (like textual information) to an intermediate representation: $\tau_\theta(y) \in \mathbb{R}^{M\times d_\tau}$

The representation is mapped to intermediate layers of the U-net using cross-attention layers.

The conditional LDM objective becomes:

$$L_{LDM} := \mathbb{E}_{\varepsilon(x),y,\varepsilon\sim\mathcal{N}(0,1),t}\left[||\varepsilon - \varepsilon_\theta(z_t,t,\tau_\theta(y))||_2^2\right] \tag{2.28}$$

Summarizing: LDMs are powerful architectures biased towards text-to-image generation, they use powerful encoders to encode images and text and use attention layers in the U-net to process them.

The information is compressed and thus faster to compute, with the advantage of focusing only on specific details.

The final output is then decompressed from the latent space to the input space of pixels, this guarantee a fast processing but still an high-resolution image as result.

2.3 Attacks on Diffusion models

This section will be dedicated to the possible attacks that can be achieved on diffusion models.
I will not go into details of the attacks since every attack is different.
An attack depends on the type of architecture that is being attacked and if there is a white box or black box approach.
In a black box approach the settings of the model are not known; on the other hand, in a white box approach, the model hyperparameters are known.
The main types of attack that are known in the literature are [8]:

- Membership inference attacks (MIAs): these attacks permit to understand if a sample is present or not in the training set;
- Model inversion attacks: these are designed to extract sensitive information about the training data or the model itself. By analyzing the output of a machine learning model, attackers can infer information about the training data that was used to train the model;
- Attribute inference attacks: these attacks aim at reconstructing attributes of training samples;
- Extraction attacks: these attack aim at completely recover training samples.

Most work in the literature against diffusion models is done on membership inference attacks [67] while some work has been done also to retrieve training samples[8].
An extraction attack is very dangerous for a text-to-image model.
In many cases, these models are very large and trained on a lot of web images; some of them might have some privacy or copyright issues.
An attacker might be able to steal sensitive information without anyone noticing it.
It is important to not underestimate these attacks.
Some precautions have been proposed; the most important one is the use of differential privacy 2.5.

It is a mathematical way to address privacy in ML training, but since there is not a specific threshold for defying what is at risk and what is not at risk, differential privacy is used in many cases in a form that is not optimal and thus is not sufficient to block maleficent attacks [31].
In other cases, differential privacy is not used because it is basically a trade-off between performance and privacy.
Researchers often prefer to achieve state-of-the-art results instead of defending personal information; in fact, the focus is on the presentation of new concepts and techniques that achieve better results than previous ones; however, privacy is not usually taken into account for practical purposes.
In the following subsections I will describe the most common used MIA 2.3.1 and the role of overfitting in such attacks 2.3.2

2.3.1 Shadow models configuration in membership inference attack

A common pipeline for membership inference attacks is used in the case of a black box setting; this setup requires the use of shadow models.
A black box setting in the context of an attack is defined as not knowing the parameters or the dataset information of the network; this means that only the inputs are known and any other parameter regarding training or weights is not known. If any parameters or dataset information is known, we talk about white box settings [57].
The shadow models approach is one of the only useful ones in a black box setting, but it requires a lot of resources to train and create these shadow models.
In this thesis, the aim is to be able to access a white box model or at least a part of the training set (not the full dataset) in order to compensate for the lack of resources.

The pipeline using shadow models to perform a black box membership inference attack is described in [57].
An attacker that wants to perform a MIA on a specific model M knows the inputs to M and the outputs of M.
The user doesn't have direct access to the model, but through APIs.

The attacked model is a classifier which outputs probability as confident measure for the classification.
Since access to the attacked model is very restricted, information about the training set needs to be learned not from the model itself but from the so-called shadow models.
Some assumptions need to be made anyway.
The type and architecture of the target model need to be known. In the case where the architecture is not known, since the model is accessed through an API, it will provide a very similar or identical model if the user asks to perform the same task. It's then possible to perform training using the same API.
Here are two possibilities:

- There is also a known data distribution that mimics the underling data of the training set;
- There is not a known data distribution that mimics the training set.

In the first case, it is already possible to start training the shadow models to become copies (or very similar) of the attacked model.
This shadow model will mimic the behavior of the attacked model on the shadow model's training set; however, if there is no knowledge of the training data distribution, the solution is to use synthetic data.
If the API is very confident in its output on the synthetic data given as input, then, it probably comes from the same distribution as the training set.
Now there is everything needed to train as many shadow models as are necessary.
A shadow model is created for each class of the training set of the attacked model, the shadow models predict if a sample is part of the training set 'in' or not part of it 'out', the information gained from these models is then used to train an attack model, which is a simple binary classifier, which can then be queried to understand if a specific input was part or not of the original training set of the attacked model. 2.10.

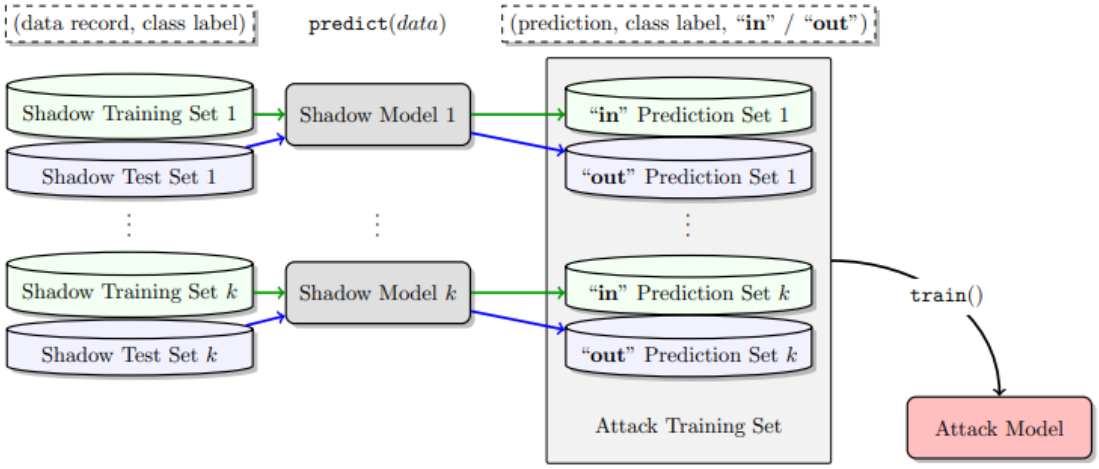

Figure 2.10: A specific training set and test set is created for every class of the dataset of the attacked model; a shadow model is created for every class of the dataset; the outputs of the shadow models are used as labels for the training of a binary classifier [57].

2.3.2 Role of overfitting in MIA

In the paper from Bogdan Kulynych [35], vulnerability to membership inference attacks is discussed.
The vulnerability to MIAs is measured by the normalized advantage [68] of the adversary A over random guessing.
A ML model is said to overfit, or poorly generalize, when its average loss on the training set differs from its loss on new samples from the population.
Previous work [68], shows that while overfitting is sufficient for MIAs, it is not necessary.

The following image 2.11 shows that for the common notion of generalization, that is, average loss (area) on training set S and test set $\bar{S}$, the model does not overfit.
The average loss is comparable numerically in training and test set, but the distributions are different.

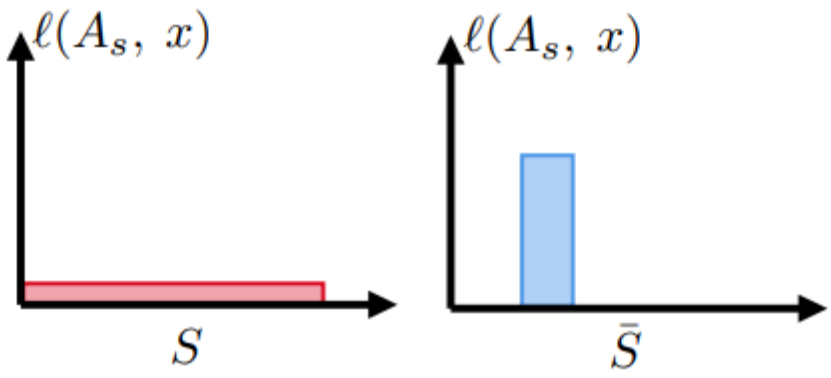

Figure 2.11: On the left loss function on training set S and on the right on test set $\bar{\text{S}}$, the average loss has the same value but the two distributions are different [35].

The authors decided to define a more strict definition of generalization so that it can be a sufficient and **necessary** condition for MIAs.
It covers the difference in the distributions of any given property of a model on the training data and outside.
A property is defined as a function that takes as input a model and a sample and gives as output a numeric vector:

$$\pi(A_{\text{S}}, x) \tag{2.29}$$

Where x is a sample, A_{S} is a model and π is a property.
A property function can be a loss function, gradient or prediction from the model.
To make a definition to be necessary for MIAs, the authors decided to look for the distributions of properties on examples x from training set and test set, proposing the following definition of distributional-generalization:

For any property function $\pi(A_{\text{S}}, x)$:

$$d(P_{x\sim S}(\pi(A_{\text{s}}, x)), P_{x\sim\bar{\text{S}}}(\pi(A_{\text{S}}, x))) \tag{2.30}$$

where $d(a, b)$ is a measure of dissimilarity between probability distributions.
It looks at the distance between the distribution of data in the training set and in the test set of any possible property, which implies also the loss function. This means that

this definition is a more general definition of generalization where the standard one can be applied by using as property the loss function and the mean discrepancy function as measure of dissimilarity (d).

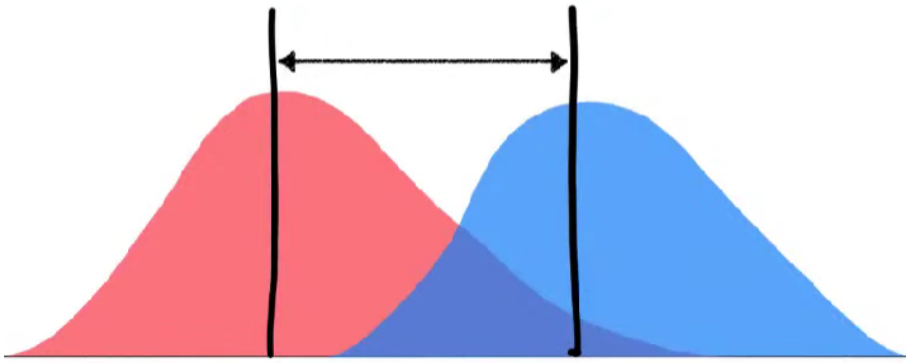

Figure 2.12: Image representing the distribution of any property $\pi(A_S, x)$ (on the x-axis) and how it differs from training and test set

Whereas standard generalization quantifies how much the training algorithm tends to memorize the training dataset through the lens of its performance (loss), distributional-generalization can do so (1) through the lens of other properties beyond losses, and (2) considering distributional information instead of only the difference between the means. Distributional-generalization is able to capture any discrepancy in the population, whereas standard generalization does not.
To summarize:

- Poor standard generalization is sufficient for MIAs;
- Poor distributional-generalization is sufficient and necessary for MIAs.

2.4 Attack models

Every attack to machine learning models is built in a different and unique way, however in many cases there is a classification model that is trained and is used to perform the actual attack on a specific targeted model.

In this thesis binary classifiers has been used to distinguish between what's **in** the training set of the attacked model and what's **out**.
The classifiers will hence assign each sample to one of the label classes "in" or "out".
In the following subsections, with analytical references, I will explain the fundamental concepts behind the most used ones.

2.4.1 Support vector machine

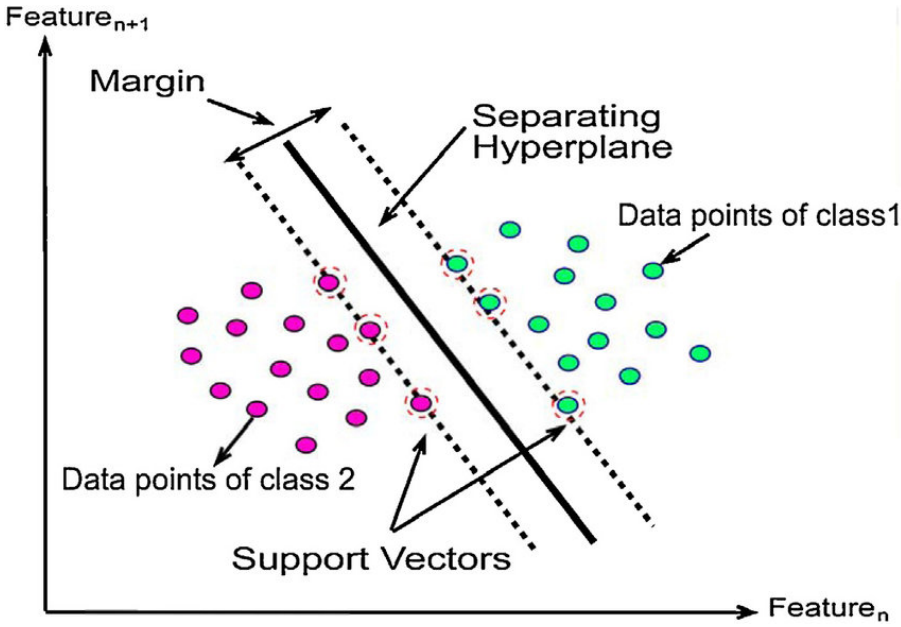

Figure 2.13: Support vector machines and their components

Support vector machines (SVMs)[64] [14] in machine learning, are a type of supervised learning used for classification and regression.
The goal of SVMs is to create the best line or decision boundary that can separate n-dimensional space into classes, so that when a new data point needs to be classified, it is put in the right space.
The best decision boundary is an hyperplane in n-dimensions where n is > 2, in the case of binary classification, this boundary is simply a line.
The decision boundary needs to be created so that it has the maximum margin which is the distance between the decision boundary and the data points.

By intuition, if the margin is big it means that the data is well separated, in the opposite case, with a small margin it is hard to distinguish between the classes.

In figure 2.13 it is possible to see the different components in this algorithm.
The support vectors are defined formally as:

$$\mathbf{wx} - b = \pm 1 \tag{2.31}$$

where the - 1 means that the support vector is on the left of the margin and + 1 means the opposite.
The decision boundary will have equation:

$$\mathbf{wx} - b = 0 \tag{2.32}$$

These equations are nothing but hyperplanes, that in 2 dimensions become lines.
The objective is maximize this margin.
By some simple linear algebra it is possible to take a point on the decision boundary and prove that the margin is equals to:

$\mathbf{x}$ on decision boundary.
The unit vectors needed to come from $\mathbf{x}$ and arrive at the support vector (in this case support vector on the right):

$$\mathbf{w}(\mathbf{x} + k\frac{w}{||\mathbf{w}||}) - b = 1$$
$$k = \frac{1}{||\mathbf{w}||}$$

The margin is:

$$\frac{2}{||\mathbf{w}||} \tag{2.33}$$

Maximizing the margin means minimizing the denominator so $||\mathbf{w}||$.
This optimization problem can be written as:

$$\min_{w,b} ||w|| \tag{2.34}$$
$$\text{s.t.} \quad (wx_i - b \leq 1) \quad i = 1, \ldots, N$$

The problem is solved by finding w,b that solve this minimization problem.

It is useful to introduce also the kernel trick in SVMs [14].
In many real-world scenarios, data is not linearly separable in the original feature space. The kernel trick is a way to deal with this problem by implicitly mapping the data into a higher-dimensional space.
The kernel function computes the dot product (or a similarity measure) between two data points in this higher-dimensional space without explicitly computing the transformation itself.
This is computationally efficient and allows SVMs to work in high-dimensional spaces.
Common Kernel Functions:

- Linear Kernel: $K(x,y) = x \cdot y$;
- Polynomial Kernel: $K(x,y) = (x \cdot y + c)^d$;
- Radial Basis Function (RBF) Kernel: $K(x,y) = \exp(-\gamma \cdot \|x-y\|^2)$
- Sigmoid Kernel: $K(x,y) = \tanh(\alpha \cdot x \cdot y + c)$.

SVM is very helpful method in case not much information is given about the data. It can be used for the data such as image, text, audio etc.
It can be used for the data that is not regularly distributed and have unknown distribution, it also doesn't suffer from overfitting and outliers have not much importance.

2.4.2 Logistic regression

Logistic regression is a classifier that estimates the probability of an event to occur.
In regression analysis, the value of the dependent variable can be inferred through the values of the independent variables, the dependent variable in logistic regression is a dichotomous variable, it can take only two possible values, while in linear regression the dependent variable is a continuous one.
Since the values of the dependent variable need to be between 0 and 1 (for probability

estimate), the logistic regression function will look like this:

$$f(z) = \frac{1}{1+e^{-z}} \tag{2.35}$$

2.35 is called logistic function. The value z is substituted with the linear regression function which is:

$$\hat{y} = b_1 x_1 + b_2 x_2 ... b_n x_n + b_0 \tag{2.36}$$

Where x is the n-dimensional input, **b** is the n+1 parameter vector where b_0 is the intercept of the line and the rest is part of the slope of the line.

The logistic regression function for predicting that the value of dependent value is 1 is:

$$f(\hat{y}|x_1, x_2 ... x_n) = \frac{1}{1+e^{-(b_1 x_1 + b_2 x_2 ... b_n x_n + b_0)}} \tag{2.37}$$

The parameter vector **b** can be learned through an optimization algorithm by defining a loss function.

A common case is to define an error of 1 when the prediction is wrong and 0 when the prediction is right and then update the parameters through gradient descent algorithm.

2.4.3 Multi layer perceptron with softmax

A multilayer perceptron (MLP) is a fully connected feedforward artificial neural network (NN).

A detailed description of the topic is out of the scope of this thesis, but the main concepts will be here illustreted.

A MLP is constituted of units called neurons, each one of them has different inputs, weights and biases.

The weights of the neurons characterize the function computed by the MLP and are tuned during learning.

Inputs, weights and biases are combined in a weighted sum that is then given as input to an activation function. The output of a k_{th} neuron will look like this:

$$y_k = \varphi(\sum_{i=0}^{n} w_{ki} x_i) \tag{2.38}$$

where n is the number of the neurons of the previous layer.

The x_0 input is assigned the value +1, which makes it a bias input with $w_{k0} = b_k$.

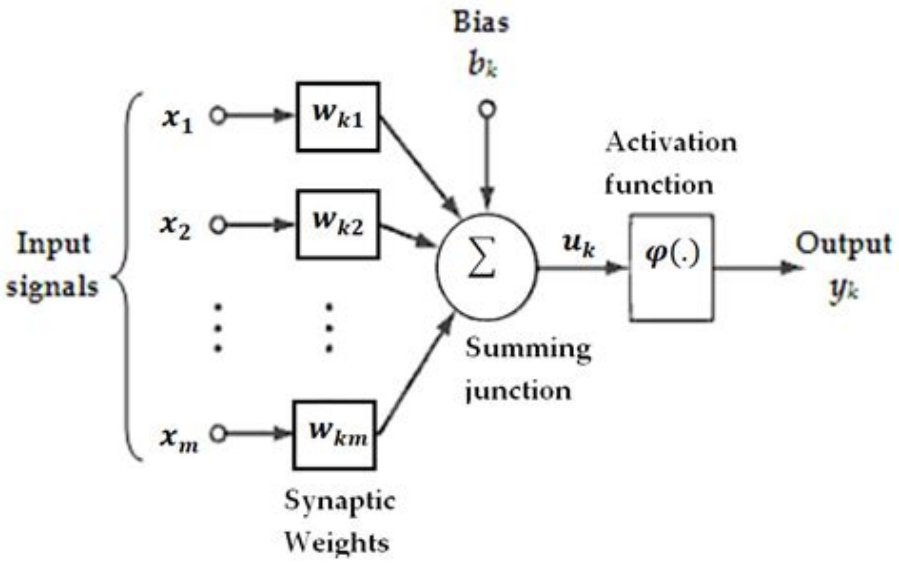

Figure 2.14: Artificial neuron and its components

Fig. 2.14 depicts the scheme of a single neuron, but many neurons can be combined to form layers and many layers can be combined to form a neural network. In fact, in a MLP, neurons are organized in groups, called layers. The neurons of a layer receive the input from the neurons of the previous layer and provide their output as the input of the neurons of the next layers.

A 3-layers fully connected feed forward NN is usually referred to MLP.

The activation functions are non linear and they exist of many types, their use is related to the network specific architecture.

For training these models the usual optimization algorithm that is used is gradient descent 2.2.3, which updates the weights of all neurons in all layers by backpropagating the signal from the output layer to the input layer.

In order to use a MLP as a binary classifier, the output layer needs to have a softmax activation function, which converts a vector of j real numbers into a probability distribution of j possible outcomes.

For the need of this thesis, $j = 2$ (number of output classes).

2.5 Differential privacy

Differential privacy is a privacy framework, it is a mathematical way to define privacy.
Data anonymization is not an easy task, a common example to illustrate the failure of data anonimization is the Netflix prize dataset.
In 2007, Netflix announced a prize of one million dollars to anyone who could increase the prediction rate of their recommendation system.
In order to work with real data and real cases, Netflix released a large dataset of users and their movie preferences.
The data were anonymized, and sensitive information like name and address were deleted in the publication of the data.
However, a study [42] revealed that anonymity is not enough to hide someone's identity.
In fact, some of the users' names were reconstructed, and their movies and political preferences were exposed to the world.
The attack consists of a linkage attack, in simple words, by the union of different databases all easy to access, anyone identity can be reconstructed even if their personal information is anonymized.
The need of the differential privacy arised since other techniques, such as data anonymization, cannot provide privacy protection for some distribution of the data.

The key idea behind differential privacy is to learn from the general trend of data without releasing personal information about individuals.
The dataset should not be influenced by the presence of a specific individual, and the general trend of the data should be the same with or without a specific field of data.
After one outcome becomes much more likely after adding one particular individual's data, information is leaked.
On the other hand, if someone's information is added and the outcome remains similar, it is possible to say that the machine learning model is more confident because it has more data, but we are not leaking information about the individuals in the dataset.
It is possible to look at the log of the ratios of the two probabilities, before and after adding a specific field to have an idea of the privacy loss that is going on.

Differential privacy aims at reducing this privacy loss.
As formalized in the paper [15] a privacy loss for a mechanism M, a dataset D where D' is the same dataset with one more example, a set of outcomes$S$ and a privacy budget ε is defined as:

$$\log \frac{P(M(D) \in S)}{P(M(D') \in S)} \leq \varepsilon \tag{2.39}$$

A smaller privacy budget equals a more private dataset and a smaller learning effect on that same dataset, on the other hand a bigger budget means less privacy and more learning.

A more traditional way to define differential privacy is:

$$P(M(D) \in S) \leq e^{\varepsilon} P(M(D') \in S) + \delta \tag{2.40}$$

where δ is the failure probability.
δ needs to be much less than the probability of a given individual in the dataset.
The previous definition is also referred to as: (ε-δ)-differential privacy.
This is effective because now, it is proven that the dataset is not leaking no more of what the bound proves, no matter of the post-processing that an attacker can do.
This protection can be achieved by adding noise such that no outcome is more likely than another. For example in images the noise is translated in blurriness.
Also another way to perform this mechanism is by doing random sampling.
There is one another important thing that needs to be addressed, as the fundamental law of information recovery states, it is possible to erode privacy by asking many overly-accurate questions, as the number of queries increases, the privacy loss also becomes greater.
This can be easily represented with differential privacy by: an algorithm (ε_1-δ_1) differentially private followed by an algorithm (ε_2-δ_2) differentially private, which is: (ε_1+ε_2)-(δ_1+δ_2) differentialy private.

Differential privacy has been used in deep learning by modifying the gradient descent algorithm [1]. The cited paper modifies stochastic gradient descent to become differentially private (DP-SGD).
The idea is to limit privacy loss per gradient update by adding noise, so for each gradient

update there is the guarantee to have $(\varepsilon + \delta)$ privacy.
Sampling with probability q and with T timesteps, the total differential privacy applied during DP-SGD is:

$$(Tq\varepsilon - Tq\delta) \tag{2.41}$$

There are other bounds that tighten this constraint, but the first one given, which is more naive, should give an idea of what is going on.

The moments accountant comes into play as the differential privacy is modeled as a normal random distribution; this gives the possibility to apply concentration inequalities that study the behavior of the distribution when moving away from the mean.
It is possible to get a tighter bound by looking at higher moments like the variance (moment for X^2) and so on, up to X^λ.
Through the training, it keeps track of what bound is associated with each of the moments and chooses the tightest one (from the first to the 32^{nd}moment).

This was how differential privacy is formalized and used in deep learning, we will see an application of this during further paragraphs of the thesis 3.1.4, and we will also see why it is not used in the context of text-to-image models.

Chapter 3

Technologies and tools

This thesis focuses on diffusion models, on low availability of resources and on a training dataset which is at least in part open-source, so it can be used for the attack, otherwise it is going to be a black box setting, not treated in this thesis.
The experiments will aim at attacking diffusion models 3.1, the attack consists of understanding if a specific instance is part or not of the training set via the use of binary classifiers 2.4.
They will be trained through the use of a small part of the training set ('in') of the diffusion model and a small part of another dataset ('out') [3.4].
In order to create a dateset usable by the binary classifier for training, the instances of both 'in' and 'out' will follow the same treatment:

1. An image-to-text model 3.2.1 will be used for extracting the prompt from one image;
2. The prompt will be used to generate another image;
3. The features from both images will be extracted through a feature-extracting model 3.2.2;
4. The feature vectors will be subtracted so that there is going to be only one instance concatenating both the original image and the generated image;

5. The instances will be labeled according to the original image used; in case it belonged to the training set of the diffusion model, the final label will be 'in', and if the opposite is true, the label will be 'out'.

6. The newly created dataset will be used for training the binary classifier that will be used for the attack.

CLIP [48] (Contrastive Language-Image Pretraining) is a powerful neural network model developed by OpenAI that can understand and connect images and text.
It is an important tool used in the models presented. CLIP embeddings are the representations of images and text in a shared vector space, allowing for direct comparisons and matching between them.
In order to simulate the small resource environment, the free software from Google 3.2.5 will be used.

3.1 Attacked models

The attacked models are the ones that receive the attack: an outsider will try to gather important information about their training set. This operation is done after the model is trained and is put into production.
The models presented are all pre-trained, since we are in a scarce resource environment, it is not possible to train them from scratch.
I will go through the type of models attacked, training setup and then explain in details the attack in the following chapter (Sect. 4).
The following information is related to the architecture of the models and how they were trained.

3.1.1 stable-diffusion v1-5

stable-diffusion is open-source latent diffusion model introduced in [51].
It is the main model attacked in the tests of this thesis, because it is close to a state-of-the-art model and it is open source.
The latter is critical, since the attack developed needs to have at least a preview of the training set to be open source.
This point will be further clarified in the following chapter (Sect. 4.1.2).
stable-diffusion belongs to the latent diffusion models, which have been introduced in Sect. 2.2.5.
This model is very popular also because it can run on a consumer GPU-equipped PC (with 12 GB RAM).
The mechanism behind latent diffusion models gives the possibility to fine-tune and make inferences by using the weights of a pre-trained model on a consumer GPU.
This, along its performances, is one of main reason for the stable-diffusion popularity.
This thesis focuses on diffusion models, on low availability of resources and on a dataset which is at least in part open-source, so this model is the best one for this purpose.
Besides its text-to-image generation capabilities, which are exploited in the present thesis, the model can also be used for inpainting [69].
stable-diffusion-v1-5 was generated through updating previous model's versions starting from stable-diffusion-v1-1.
The following schema 3.1.1 present the various versions that have concurred in the creation of stable-diffusion-v1-5.
The following concepts might need an in-depth explanation before accessing the schema:

- Steps: it is sometimes addressed as epochs, it refers to one complete cycle of training a neural network on a given dataset. During each epoch, the entire dataset is passed forward and backward through the neural network exactly once. The purpose of an epoch is to update the model's parameters (weights and biases) based on the error or loss calculated during that cycle, with the goal of improving the model's ability to make accurate predictions, more details in 2.2.3;

- Resolution: it identifies how many pixels are present in the images of the datasets, usually low quality images have lower pixels than high quality ones.
  Training on low quality images usually brings low quality results, however some considerations must be done.
  There is the possibility to train a model on low quality images and then upscale [63] the result in order to transform it in a higher quality image.
  The trade-off is usually the same, lower quality images make a faster training but lower quality results which might need the use of upscaling as final part of the network.
  Higher resolution images might require too much use of resources but the results are usually better than low quality images;

- Aesthetics score: The aesthetic score is simply a measure of the aesthetic quality of an image [10];
  It is computed with a linear neural network that takes as input the CLIP embeddings of the images and gives as output an aesthetic score.

- Watermark probability: probability of having watermark, which is a way to prevent copyright in online content;

- The datasets: they can be accessed at the following Sect. 3.4.

A schema of the training steps done on the pretrained model (It can be found on the technical card of the model at [52]) is the following:

- stable-diffusion-v1-1: 237,000 steps at resolution 256×256 on LAION-2B-en.
  194,000 steps at resolution 512×512 on LAION-high-resolution (170M examples from LAION-5B with resolution $\geq$ 1024×1024);

- stable-diffusion-v1-2: Resumed from stable-diffusion-v1-1. 515,000 steps at resolution 512×512 on "LAION-improved-aesthetics" (a subset of LAION-2B-en, filtered to images with an original size $\geq$ 512×512, estimated aesthetics score > 5.0, and an estimated watermark probability < 0.5. The watermark estimate is

from the LAION-5B metadata, the aesthetics score is estimated using an improved aesthetics estimator);

- stable-diffusion-v1-3: Resumed from stable-diffusion-v1-2 - 195,000 steps at resolution 512×512 on "LAION-improved-aesthetics" and 10% dropping of the text-conditioning to improve classifier-free guidance sampling 2.2.2;
- stable-diffusion-v1-3: Resumed from stable-diffusion-v1-2 - 195,000 steps at resolution 512×512 on "LAION-improved-aesthetics" and 10% dropping of the text-conditioning to improve classifier-free guidance sampling;
- stable-diffusion-v1-5 Resumed from stable-diffusion-v1-2 - 595,000 steps at resolution 512×512 on "LAION-aesthetics v2 5+" and 10% dropping of the text-conditioning to improve classifier-free guidance sampling.

3.1.2 Karlo v1 alpha

Karlo is a text-conditional image generation model based on OpenAI's unCLIP architecture [50], as reported in its model-card on Hugging Face [18].
Karlo is a diffusion model but its architecture is different from stable-diffusion's, it is in fact based on unCLIP which is also known as DALL-E 2.
DALL-E 2 is able to upscale images up to 1024×1024 pixels while Karlo is only capable of reaching 256×256. It is possible to use Karlo as an open-source alternative to DALL-E 2.

In Fig. 3.1, the unCLIP architecture is reported [50].
It is an overview of the unCLIP architecture.
The CLIP training process gives a joint representation space for images and text.
The CLIP embedding is then given to a diffusion prior in order to create an image embedding that will be used by the decoder for the image generation task.

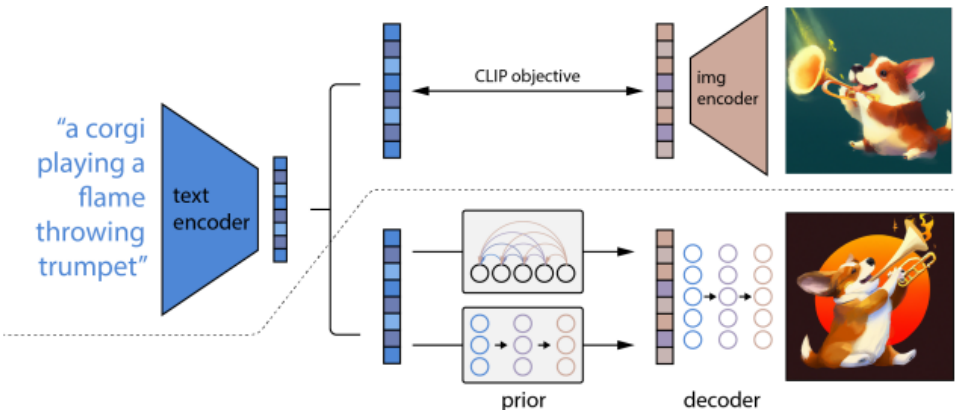

Figure 3.1: Let's provide a broad perspective on unCLIP. The diagram is divided by a dotted line into two main segments.

Above the dotted line, the training phase where CLIP undergoes a learning process to establish a unified representation space for both textual and visual data is represented.

Below the dotted line, the text-to-image generation process is depicted. Initially, a CLIP text embedding is inputted into either an autoregressive model or a diffusion prior. This step results in the creation of an image embedding. Subsequently, this newly-formed embedding is utilized to condition a diffusion decoder, which is responsible for generating the final image, as detailed in the work by [50].

Karlo does have a prior and a decoder, as stated in 3.1 but it also has super-resolution modules.

In a more specific context, the standard super-resolution (SR) module, which has been trained using the DDPM objective, serves the purpose of upscaling an original 64×64 pixel image to a 256×256 pixel image resolution in the first six denoising steps, utilizing a technique known as respacing. This initial phase focuses on achieving a substantial increase in image size.

Subsequently, an additional SR module, which has undergone fine-tuning through training with the VQ-GAN-style loss, takes on the critical role of executing the final step. This step is dedicated to the meticulous recovery of high-frequency details, contributing to the overall enhancement of image quality and fidelity.

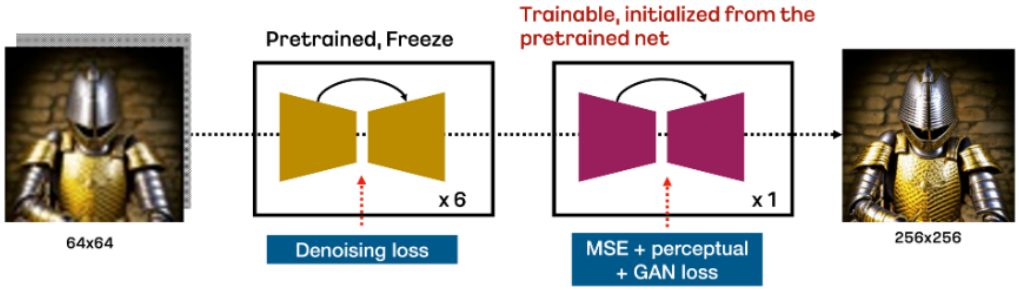

Figure 3.2: The image represents the SR modules of Karlo [18]

Here a schema of how the model was trained:

- All components are trained from scratch on 115M image-text pairs including COYO-100M, CC3M, and CC12M 3.4;
- In the case of Prior and Decoder, ViT-L/14 provided by OpenAI's CLIP repository is used;
- In the case of the SR module, first it is trained by using the DDPM objective in 1M steps, followed by additional 234K steps to fine-tune the additional component.

3.1.3 Dreamlike photoreal 2.0

Dreamlike photoreal 2.0 is a photorealistic model based on Stable Diffusion v1-5.
I opted for this model because it is a fine-tuned version of stable-diffusion model and it works also with 1024×768 pixels images.
The idea is to try the same type of attack in order to evaluate the effect of fine-tuning in countering the attack (i.e. how it is actually helpful for 'forgetting' the previous training sets on which it was trained).
as reported in its model-card on Hugging Face [11].
The training schema is pretty much the same as stable-diffusion 1.5 except from the fine-tuning procedure that has not been properly described.

3.1.4 Differentially private diffusion models

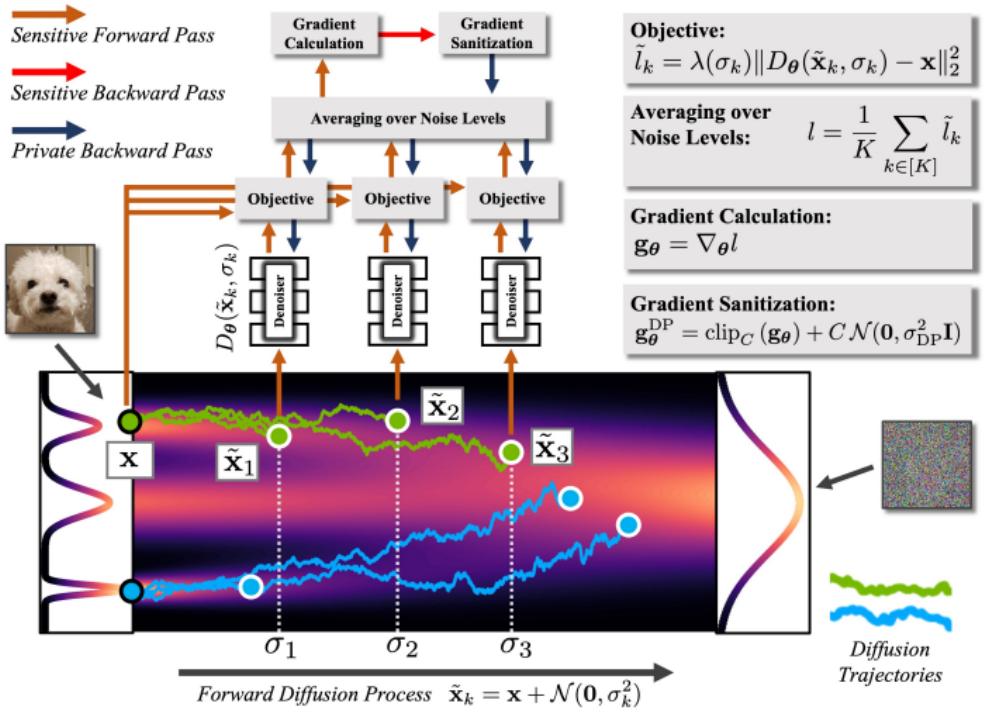

Figure 3.3: The information flow during training in the Differentially Private Diffusion Model (DPDM) is depicted in the graph. It illustrates the flow for a single training sample when the batch size is set to 1 (represented in green), with another sample shown in blue. The privacy of the model is ensured through the use of DP-SGD, and noise multiplicity with a value of K=3. The diffusion process is visualized here using a one-dimensional toy distribution, where the marginal probabilities are represented in purple. It's worth noting that primary experiments involve high-dimensional images [17].

Differentially private diffusion models (DPDM) [17], are an innovation in the field of diffusion model 2.2 for image generation and privacy concerns, trying to achieve state-of-the-art results in diffusion models trained using differential privacy 2.5.

The authors introduce differentially private stochastic gradient descent (DP-SGD) 2.5 to enforce privacy during training, they thoroughly investigate the diffusion models parameterization and the sampling algorithm, which are crucial ingredients in DPDMs.

For more information, the DPDM training algorithm is presented in the paper.

In Fig. 3.3 it is possible to observe the information flow during training and core formulas explained in the paper.
The architecture is not specifically explained but it follows other state-of-the-art architectures for DDPM, specifically for image synthesis.
The neural network backbone of DPDMs uses the DDPM++ architecture [61].
For class-conditional generation, a learned class-embedding is added to the σ-embedding as in [16].

These are the only diffusion models that I found trained on differentially private setups, the reasons why companies do not use differential privacy in text-to-image models, and the main one, is for quality reasons.
Differential privacy adds noise to the images which result in a lower quality product.
Since the aim of companies that produce text-to-image models is quality, they prefer to have less private models to achieve better performances and resolutions giving the users what they want.

3.2 Tools

3.2.1 Image captioning model

BLIP is a vision-language pre-trained (VLP) model, introduced by Salesforce research [38], that is able to perform various multi-modal tasks including:

- Visual Question Answering (VQA) is a task where a model is asked a question about an image and must provide a textual answer.
  Typically, the inputs include an image (visual input) and a textual question (text input).
  The output is a textual answer (text output) that the model generates in response to the question.

- Image-Text retrieval or matching aims to find semantically related pairs of images

and text from a collection.
It involves a pair of inputs, a query image (visual input) or a textual query (text input), and a dataset containing images and corresponding text descriptions.
The output can vary depending on the specific subtask. In retrieval, it might be a ranked list of images or text descriptions that are most relevant to the query.

- Image captioning is the task of automatically generating a descriptive textual caption for an input image.
  The input is typically an image (visual input). The output is a textual caption (text output) that describes the content of the image in natural language.

For the aims of the present thesis, it has been used only as an image captioning tool. This model is an answer to the previously not optimal configurations of models in the VLP world:

- Previously studied encoder-based models are less straightforward to directly transfer to text generation tasks and encoder-decoder architectures haven't been successfully used in VLP tasks.
- From data perspective, state-of-the-art models are trained on web scraped images with text-images pairs not always of the best qualities.
  Noisy web text is suboptimal for vision-language learning, as shown in [38].
  In fact, in the design of BLIP, Captioning and Filtering (CapFilt) a new method to improve the quality of the text corpus is proposed as solution.

BLIP is a multimodal mixture of encoder-decoder, a unified vision-language model which can operate in one of the three functionalities:

- **Unimodal encoder**
- **Image-grounded text encoder**
- **Image-grounded text decoder**

For the purpose of this thesis, the Image-grounded text decoder is the functionality used.

In particular, the decoder is trained with a language modeling (LM) loss to generate captions given images (Fig. **??**).

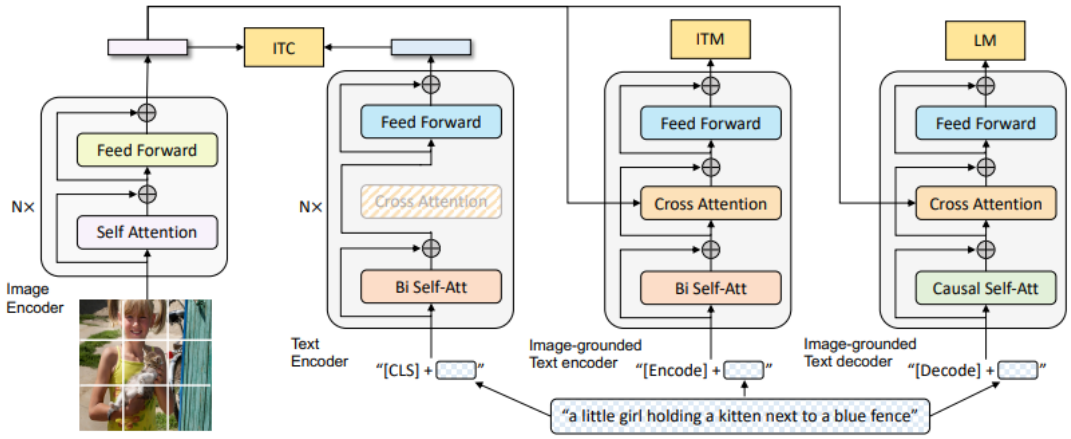

Figure 3.4: Blip architecture and its different modules [38]. In this thesis, the Image-grounded text decoder is employed, which involves the substitution of bi-directional self-attention layers with causal self-attention layers. Additionally, it maintains the same cross-attention layers and feed-forward networks as the encoder. The training objective for the decoder is centered around language modeling (LM) loss, aiming to produce captions when provided with images.

This model can be seen as a dynamic way to achieve different state-of-the-art results in different VLP tasks.

3.2.2 Feature extraction model

A feature extraction model is a computational framework or algorithm used in machine learning and data analysis to automatically identify and extract relevant information (features) from data.

The primary goal of a feature extraction model is to transform the input data into a more compact and meaningful representation that captures essential patterns or characteristics,

making it easier for downstream machine learning algorithms to process and learn from the data effectively.
In this thesis, this type of model is used to extract relevant information from one or more images in the form of vectors so that mathematical operations can be computed on the feature vectors.
This can be helpful for example to compute how different the two images are by subtracting fundamental features.

VGG16 is a convolutional neural network (CNN), proposed by K. Simonyan and A. Zisserman [58].
I will use this model as feature extraction model, in particular to retrieve relevant information from an input image.
This model achieves 92.7% top-5 test accuracy on the ImageNet dataset which contains 14 million images belonging to 1000 classes.
VGG16 was the winner of the ILSVRC-2014 competition, it improved
AlexNet (another CNN, [34]) in the use of kernels.
AlexNet has the kernel size of 11 for the first convolutional layer and 5 for the second layer, while VGG16 obtained the same processing using a longer sequence of 3x3 kernels.
This solution makes use of less parameters and hence avoid overfitting. 3x3 kernels are the smallest possible to capture vertical and horizontal features at the same time.
Another benefit of using multiple smaller layers rather than a single large layer is that more non-linear activation layers accompany the convolution layers, allowing the network to converge quickly.

VGG16 is a 16-layers deep CNN, with 138 million parameters, which makes it a quite large-sized model even for today standards.
A lot of parameters do not always mean complexity: in fact, the architecture is pretty simple and has the right number of layers to avoid gradient vanishing.

For better understanding the network, here are some details about its components:

- VGG16 receives as input a 224×224 image.
  In order to have an input of 224x224 cropping the center of the image or resizing different images sizes is necessary;

- The convolutional filters of VGG16 use the smallest possible receptive field of 3×3.
  VGG16 also uses a 1×1 convolution filter as the input's linear transformation.
  VGG16 uses a stride of 1 to preserve spatial information (the stride indicates how many pixels the kernel is shifted);

- The Rectified Linear Unit Activation Function (ReLU) component, ReLU is a linear function that with a matching output for positive inputs and a zero output for negative inputs;

- The VGG network's hidden layers use ReLU instead of Local Response Normalization like AlexNet.
  The latter increases training time and memory consumption with little improvement to overall accuracy;

- A pooling layer follows several convolutional layers.
  Pooling is fundamental to decrease the number of features on which the network is working with, it decreases the dimensionality of the outputs of the convolutional layers;

- VGG16 includes three fully connected layers.
  The first two layers each have 4096 channels, and the third layer has 1000 channels, one for every class.

In the following 3.5 image, the depth of the configurations increases from the left (A) to the right (E).
The number of parameters also increases with the depth of the network.
The last layer is a soft-max since the network has been designed for classification purposes.
In this thesis, it will be used for feature extraction, Hence, the softmax layer has been

removed, and the last layer is the FC-1000 layer.

The output of the network is then a 1000-feature vector.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 3.5: Image taken from [58], different setups for VGG16

3.2.3 Python

Python is a high-level programming language. It emphasizes code readability by the use of indentation.

Thanks to the support to many programming paradigms and its simplicity in instructions, Python is one of the most used programming languages today.

The use of this programming language is very extensive in this thesis due to the many machine learning libraries that are implemented with it.

Some examples are Tensorflow [2], Pytorch [46], Sklearn [47] and the more common ones for analytics and mathematics Pandas [44] and Numpy [25].

3.2.4 Sklearn

The library scikit-learn (sklearn) [47] is a free software machine learning library for the Python programming language, it features many ML common tasks like:

- Classification;
- Regression;
- Cleaning and transforming data;
- model selection;
- clustering;
- Dimensionality reduction.

This library was very useful for the thesis because it implements the attack models 2.4.
This library provides many functionalities and is simple to be used due to its high-level abstraction.
Other libraries, e.g. Pytorch [46], allow low level operations to modify the models.
However this was not the case for the needs of this theis, so sklearn has been chosen for its simplicity.

3.2.5 Google Colab

Google Colab [12] is an executable webpage that resembles Jupyter notebook.
In Colab anyone can write python code and run it through the cell-system, in fact, there are various cells each one contains a piece of code which is then run to make the results

available for the other cells.
Google Colab is extremely useful because it gives GPU, CPU and RAM for free while using it.
This platform permits to run big models through the given resources, if the requirements are too big, it is possible to opt for a paid account.
Usually the RAM given for free by Google is enough for running big models like stable-diffusion 3.1.1 in an inference setting, and that what matters for MIAs.
Software like this permit to easy reproduce the experiments made in this thesis without having hardware limitations.

3.2.6 Hugging Face

Hugging Face, is an American company that develops tools for building applications using machine learning.
It is well-known for its repository of machine learning models that people freely upload on their website ([20]) so that they can be used by other machine learning enthusiasts.
Hugging Face offers models, datasets, spaces and computational power for machine learning purposes.
It is also well-known for its broad transformers library for natural language processing.
Hugging face hosts all kind of models that people create, from computer vision to multimodal models.
I personally used this framework in order to import the previously described models in: 3.1, 3.2.2, 3.2.1.
I was able to simply import them in my working environment thanks to the easy to use libraries.
In just a few lines of code, it is possible to effortlessly bring in the libraries and models available on the Hugging Face platform.
Additionally, there is the oprion to choose the computing device to deploy your models on, typically between two options: the CPU or GPU.
Machine learning practitioners tend to favor GPUs due to their Single Instruction, Mul-

tiple Data (SIMD) architecture. This architecture allows the workload of instructions to be effectively divided and executed concurrently across multiple cores, making GPUs a preferred choice over CPUs.

3.2.7 Downloading images and file formats

The datasets used can all be categorized as big data datasets, these are typically downloaded as smaller compressed files that contain only a small part of the dataset.
Typically, the format for the compression is the .parquet.
Apache Parquet is an open-source, column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. I used an application called TAD [62] that extracted the parquet file and to avoid to download the entire dataset, an indirect represention was provided as a table containing the list of the single images and their location.
This information is stored in a parquet file.
Alternatively, the parquet file can be imported directly in Python by using
pandas.read_parquet function, a viable option when the file size is fair.

3.3 FID

The Fréchet inception distance (FID) is a metric used to assess the quality of images created by a generative model.
Unlike the earlier inception score (IS), which evaluates only the distribution of generated images, the FID compares the distribution of generated images with the distribution of a set of real images ("ground truth").

During FID computation, the sets of images are mapped to the corresponding sets of activation vectors of a classification network, and the mean and variance of the Gaussian distributions that best describe these sets are calculated. The distribution of each of the

two sets is then approximated by the Gaussian distribution of the vectors calculated by the network. The FID is then calculated between the two Gaussian distributions.
It can indeed be demonstrated that the FID for two Gaussian distributions has a relatively easy-to-calculate form (see [19] [27]).
For this thesis, the classification network used for feature-extraction is VGG16 3.2.2.

Let D_1 and D_2 be two feature distributions in the feature space extracted from VGG16. These distributions represent the features of real images and generated images, respectively.
To compute FID, first calculate the mean of features extracted from D_1 and D_2, denoted as μ_1 and μ_2, respectively. Calculate the covariance matrices of features extracted from D_1 and D_2, denoted as Σ_1 and Σ_2, respectively. The Fréchet distance between these two distributions is calculated using the following formula:

$$\mathrm{FID}(D_1, D_2) = \|\mu_1 - \mu_2\|^2 + \mathrm{Tr}(\Sigma_1 + \Sigma_2 - 2\sqrt{\Sigma_1 \Sigma_2}) \tag{3.1}$$

where $\mathrm{Tr}()$ represents the trace of a matrix.

FID has been used to compare subsets of images in datasets 3.4 to their generated versions using a text-to-image model like stable-diffusion 3.1.1. In certain instances, as indicated by the results in 5.4, it is observed that the FID between 'out' images and the generated ones closely approximates the FID calculated between 'in' images (from the actual training set) and their corresponding generated counterparts. This similarity can occur when the data collection process in the two datasets is notably alike.

3.4 Datasets used

In this section, the datasets used to experiment the attacks are described.
To perform the MIA attacks 4.1.1 presented in this thesis, first there is the need to retrieve the original training sets of the attacked models, then, retrieve other similar datasets to the original ones so that the results are comparable.
The attack is successful if the attack model 2.4 trained for a specific attacked model 3.1,

is able to distinguish which images come from the original training set and which come from similar datasets.
When I mention "similar datasets", I'm talking about those that exhibit a comparable range of content as the training set. For instance, both VGG-face and LAION-aesthetic v2 5+ faces datasets contain faces.

The following datasets have been used in the training procedure of the attacked models:

- LAION-aesthetic v2 5+ 3.4.1 was used as part of training of stable-diffusion v1-5 3.1.1 and Dreamlike photoreal 2.0 3.1.3;
- CC3M was used as part of training of karlo v1 alpha 3.1.2;
- LAION-aesthetic v2 5+ faces 3.4.1 is part of LAION-aesthetic v2 5+ and was used as part of training of stable-diffusion and Dreamlike photoreal 2.0.

The following datasets have been used as similar datasets to the original training sets:

- MS-COCO 3.4.3 was used to compare the attacked models' performances on LAION-aesthetic v2 5+ and CC3M;
- VGG-face 3.4.2 was used to compare the attacked models' performances on LAION-aesthetic v2 5+ faces.

3.4.1 LAION

The Large-scale Artificial Intelligence Open Network (LAION) is a German non-profit which provides open-source artificial intelligence models and datasets.
It is known for releasing big datasets used in high-profile text-to-image models like stable-diffusion [52] and Imagen [56].

The data for some of the datasets at LAION is originally taken from Common Crawl [13], a dataset of web scraped pages.
The images and their textual description (provided as ALT attribute in the HTML page) are filtered in order to discard non matching data.
The initial cleaning is thus made by Common Crawl, after that, the matching test is performed using CLIP. [48].
The datasets available at LAION contain various information about the images, for example a description of the image in string format.
These datasets however, do not contain images themselves but URLs that point to them, so researchers have to download them.

LAION-aesthetic v2 5+

LAION-aesthetic v2 5+ is a subset of LAION-5B. The developers decided to use an MLP to evaluate aesthetic scores of the images, initially trained on simulacra-aesthetic-captions dataset.
They found good results and refined the results with a second version of the MLP and that's why this dataset is called version 2.
The 5+ means that the aesthetic score predicted from the MLP is 5 or higher.
These are the subsets of data with 5+:

- 600M image-text pairs with predicted aesthetics scores of 5 or higher;
- 12M image-text pairs with predicted aesthetics scores of 6 or higher;
- 3M image-text pairs with predicted aesthetics scores of 6.25 or higher;
- 625k image-text pairs with predicted aesthetics scores of 6.5 or higher.

Since this thesis is focused on a low-resource budget scenario, only a subset of 1k images from the first 600M subset has been selected.
The images information in LAION-aesthetic v2 5+ does not follow a specific order, so it was safe to extract the first 1k images of the dataset.

LAION-aesthetic v2 5+ faces

LAION-aesthetic v2 5+ faces is not a public dataset, but I renamed it from the previous LAION-aesthetic v2 5+, it consists of around 300 face images.
This dataset is a subset of the first 600M with aesthetic score 5+ (which is a part of LAION-aesthetic v2 5+), where only a few faces were extracted by hand by me.
The manual selection is motivated by two reasons: there was no need of a large face dataset and it should be composed of only good quality pictures, to avoid spurious results.

3.4.2 VGG-face

VGG face dataset [45] was created for faces recognition purposes.
It has over 2.6M images to be able to train a model that can recognize faces in images or videos.
I only used around 300 images from this dataset as I needed them to compare the results of the attacked models on LAION-aesthetic v2 5+ faces 3.4.1 dataset in order to perform the membership inference attack on images at risk of privacy.
The data in VGG-face is not organized at random, so I had to extract 300 images uniformely at random through the entire dataset.

3.4.3 MS-COCO

The MS COCO [39] (Microsoft Common Objects in Context) dataset is a large-scale object detection, segmentation, key-point detection, and captioning dataset. The dataset consists of 328K images.
I used around 1k images from this dataset.
I mainly used it to carry on the membership inference attack, using LAION and COCO as the two main datasets in order to distinguish between 'in' and 'out' samples.
COCO is a top choice for this attack because:

- Very popular dataset with proven scientific reliability, number of citations can be visualized at [66];
- It is used for both object-detection and image-captioning, it means that there are captions as in LAION and that there are many different objects and images as in LAION (many objects for detection);
- It was used [67], which is the work that inspired this thesis.
  Hence, using the same data, the experimental results can be directly compared.

3.4.4 CC12M

Conceptual 12M [9] is a dataset with around 12 million image-text pairs meant to be used for vision-and-language pre-training.
It is larger and covers a much more diverse set of visual concepts than the Conceptual Captions (CC3M).
CC3M was used in the creation of one of the attacked models, karlo v1 alpha (see Sect. 3.1.2).

Chapter 4

Experiments and innovations

In this chapter, I will present the main experiments and innovations.
First, I need to talk more specifically about the papers that I used the most 4.1, then I will explain how I continued the work done in these two papers.
I will focus on the attacks that I have done 4.2, the datasets used for these attacks, and reproducibility.
At the end of the chapter, I will talk about privacy and how DPDM impacted my research 4.4.
The results of the experiments will be shown in the next chapter 5.

4.1 Literature works

The works presented in this section are those that mostly inspired the present thesis.
Many others are here cited and were used in the experiments, but these two are the ones from which I took the most inspiration.
The paper [67] see Sect. 4.1.1 is the one from which I took inspiration for the MIAs, it present attacks that can be done in scarce resource environment.
The paper [7] see Sect. 4.1.3 present a common way to evaluate MIAs, with motivations that make it preferable to accuracy or other measures.

4.1.1 Membership Inference Attacks Against Text-to-image Generation Models

Membership Inference Attacks Against Text-to-image Generation Mode [67] was published on October 2022.
At the best of my knowledge, it is the only paper that does not make use of shadow models to perform a membership inference attack. This is an important feature to operate in a scarce resource environment and to reproduce its attacks.
Among the several attack techniques described in this paper, I took inspiration from one of them, and then I tried to further improve it.
I will describe which parts of the paper were the most useful and how they impacted my work.

The main attack

To construct a text-to-image generation model M, a huge collection D of text and image data pairs is needed as the training set, $D = \{(t_i, x_i)\}$.
The model is then optimized via minimizing a predefined loss function.
The goal of the adversary is to understand if a specific image x was used in the training set of the attacked text-to-image model M.
The adversary can utilize M through APIs for the inference process, but does not know the model parameters.
The adversary has access to a small subset from training data of target model $D_{\text{subtarget}}$ and also to a small set of non-member data $\bar{D}$.
$\bar{D}$ contains data that is similar to what is present in $D_{\text{subtarget}}$ but it is not part of the training set.
The two small datasets are put together to create another dataset
$D_{\text{auxiliary}} = \{D_{\text{subtarget}} \cup \bar{D}\}$, that can be used for training a binary classifier that will be the attack model A.

With $D_{\text{auxiliary}}$ I can perform an attack in a scarce resource environment without the

need of leveraging shadow techniques (see Sect. 2.3.1).

The attack used is referred in the paper [67] as II-S.

The working hypothesis behind this attack is that the reconstruction error for an image that belongs to the training set is less than the error of an image that does not belong to the training set.

To implement the attack pipeline, another tool needs to be used, namely a tool to obtain a suitable caption text given an image. This is due to the fact that the target model can be queried only using a textual input. Hence, a suitable textual description of the query image must be obtained.

The adversary only holds the image x and in order to create another image x', the adversary needs a captioning model, C, to generate a caption t' for the image x : $t' = C(x)$.

Then the generated caption t' is fed into the target text-to-image generation model M to obtain a generated image x'. In this way, the query image is connected to the generated image explicitly.

Attack II-S, also applies a pre-trained visual language model to extract the embeddings of the generated image and its corresponding query image, respectively, and then measures the distance between them.

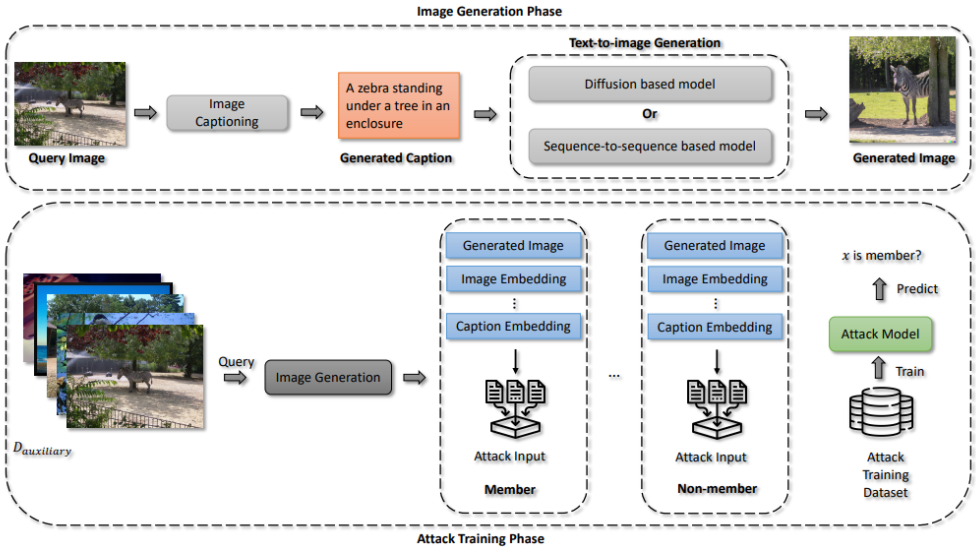

Figure 4.1: Overview of the attack pipeline, image generation and attack phase [67],

Attack II-S operates both on pixel- and semantic-level. Hence it does not only rely on the similarity of the involved images (query image and generated image), but also try to constrain the test with the semantic content of the images (by evaluating the distance in the text embedding of the two images).
Finally, the adversary feeds the distances into the attack model to distinguish between members and non-members 4.1.

Among the several attacks presented in [67], II-S seems to be the most promising one, achieving a reported accuracy of more than 0.97, at least 10% more than the other approaches.

Captioning and features extraction

The paper [67] shows that denoising steps, captioning tools and feature extraction tools have negligible effects on the final results.
Many different models for these purposes have been tried out: CLIP [48], ClipCap [41] and BLIP [38].
The results are consistent with all the different kind of attacks and with the different tools used. Attack II-S still have the advantage over the others, thus, remains the one on which is worth investigating more.

Main flaw of MIA Against Text-to-image Generation Models paper

In this paper some good intuitions were made, however the first thing that catches the eye when looking at results is accuracy.
Accuracy have been used in all results of the attacks of the paper, it is said to be the most used metric and so to compare with other prior attack, the creators also used it in their experiments. Accuracy does not tell if an attack is good or bad, so it's not a good indicator of the effectiveness of the attacks.
The attacks have to be tested again in order to understand their true effectiveness.
Further in results 5 I will show what I obtained through the attack II-S of this paper.

4.1.2 Dataset disclosure

The dataset disclosure is referred to $D_{\text{subtarget}}$.
For an in-depth explanation of the 'blackness' of an attack, please refer to [30].
In the case of this thesis, some training data is disclosed in order to make the attack work, which is a good compromise to not use shadow models 2.3.1.
In a black-box setting the model's parameters, the internal parameters, architecture and functioning of the model are not given.
In my case, it is not needed to know the model parameters, which brings me in a black-box-blind situation according to Hu et Al. [30], however a part of the training dataset is known to the attacker, which might not be enough to downgrade the attack from black-box-blind to whitebox, but is a concern.
I would say that the partial knowledge of the training set is part of partially knowing the training distribution, which is tolerated in a black-box setting.
In general, knowing the training distribution (knowledge of all domain images) does not mean to know the training data (a subset of the domain).

The attack described in Sect. 4.1.1 can be tolerated as black-box given that:

- The model's parameters are not known and neither is the architecture;
- Given limited resources and the vastness of the image domain, partial knowledge of the training set is a necessity to make the attack feasible;
- It is realistic to think that some knowledge about the dataset exists; for example, the model could have been generated by updating a previous model for which the training set was known. Or, perhaps many images were explicitly collected to create the model, but the training set was supplemented with public datasets.

Reasoning on the advantages and disadvantages of having a disclosed dataset, it is possible to point out that in case of a small model and a fully disclosed dataset (white-box setting) the attack 4.1.1 is not useful since the images of the training set can be searched inside the disclosed dataset.

However if the model faced is a state-of-the-art model, the number of parameters will be in the order of millions or billions.
With such a big model the training set will also be in the order of millions or billions of images making the search through the dataset for the original images a tough task, especially in a scarce resource environment.
It is reasonable to say that with a fully disclosed dataset and a big model to attack, in a scarce resource environment, the MIA [67] remains useful even with this setting, because of the complexity, hardware and timeframe constraints to download and archive millions or billions of images and search through them for a specific one.

A possible dataset search

Previously in this Sect. 4.1.2 I have explained that a MIA can be replaced with a search on the dataset if the latter is fully disclosed and if the model to attack is not too big.
In this subsection I want to explore a search though the dataset samples that works in case of big models but that also requires large amount of resources which are not available in the case of this thesis.
An example of such search con be performed on the LAION dataset [36].

In case of abundance of resources, it is possible to search through an enormous dataset through CLIP [48] search.
CLIP is a neural network that is able to predict, given an image and text descriptions, the most suitable prompt for that image. CLIP combines computer vision and natural language processing domains to achieve this task.
CLIP matches the performance of the original ResNet50 on ImageNet “zero-shot” without using any of the original 1.28M labeled examples, overcoming several major challenges in computer vision.
CLIP search works in this way:

1. Create embeddings of the dataset you want to search in;

2. Embed the query image/text through CLIP;

3. Retrieve the similar images to the query image through KNN algorithm.

The main problem is that to run a similar pipeline (see [36]) there needs to be a backend that contains all the data from the dataset.
All the images need to be downloaded, stored and the embeddings of each one computed, then the search algorithm can be performed.
These operations are impossible in a scarce resource environment, thus, this is not a viable option for this thesis.

In the following subsection 4.1.2 some numerical examples to motivate the impossibility of use in a scarce resource environment.

Local repository unfeasability

Considering a consumer setup with an optic fiber internet connection, it is realistic to have around 65 Mpbs for download.
Images in large datasets are not high-quality ones, usually 200×200 to 800×800 or, in rare cases 1400×1400. They vary in file size from 10 Kb to 100 Kb.
Consider that for downloading a dataset we are actually downloading URLs and text pairs not images.
LAION 5B has around 40 TB of data with just URLs and text data.
Considering a 65 Mbps downloading speed it will take about two months to finalize the operation.

Now consider a very optimistic time of 0.3 second for searching an image via the URLs through the internet and download that image.
Considering 5 billions of images, the download will require 1.500.000.000 billions of seconds which is almost 50 years.
Also ignoring the the searching time, but only the 350 TB of the images (by considering the average image of 70 kB), the result is still around 548 days. Which is a lot less, but

is also unrealistic.

Considering 350 TB of data and 548 days to download all the images, the processing time for getting results is another obstacle, this is the proof of why it is not possible to perform a search via CLIP on a big dataset on a local machine.
The attack through searching is thus ruled out in a scarce resource environment.

4.1.3 Membership Inference Attacks From First Principles

The evaluation of the effectiveness of a MIA can be realized using several metrics.
Among them, the average-case accuracy is often used. However, in [7] its suitability is questioned and another metric, the True Positive rate at a low False Positive Rate (TPR at low FPR) is considered.
Besides, in [7] a new MIA, called Likelihood Ratio Attack (LiRA), is proposed. LiRa is an attack that requires several shadow models and resources, such attack is at odds with the thesis scope and thus will not be further considered.
For the experiments in the present thesis, the TPR at low FPR metric will be used.
MIAs are very often evaluated using average-case "accuracy" metrics that fail to characterize whether the attack can confidently identify any members of the training set.
In other words, the ability of identifying even a small minority of the members of the training set with high confidence is enough to state the success of the attack (even is the vast majorty of the members cannot be identified).

With TPR at low FPR it was found that most prior attacks perform poorly when evaluated in this way. In essence, when devising an attack, the goal is to optimize the true-positive rate, ensuring that a substantial number of members are correctly identified, while simultaneously minimizing the occurrence of false-positives, which are inaccurate membership identifications.
The metric can be expressed as TPR of $A(D_{\text{auxiliary}})$ considering FPR ≤ 0.1, for symbols explanation refer to Sect. 4.1.1.

There are at least two main reasons to motivate the inappropriateness of accuracy:

1. **Balanced accuracy is symmetric.** False-positives and false-negatives are considered with the same importance, but actually false positives reduce the attack efficacy while false negatives are just an indication of an unsuccessful data extraction;

2. **Balanced accuracy is an average-case metric.** Balanced accuracy is an average-case metric. The average does not capture the effectiveness of an attack. In fact, two attacks with same accuracy might be very different. One attack might perfectly recognize a small subset of 0.1% of the members and recognize at 0.5 all the other cases, reaching an average accuracy of 0.5005. A second attack might recognize at 0.5005 all the given cases. While the first attack is very potent, the second one is useless and nonetheless they have the same accuracy.

A natural choice for attacks measure is to consider the tradeoff between the true-positive rate (TPR) and false-positive rate (FPR).
Intuitively, an attack should maximize the true-positive rate (many members are identified), while incurring few false positives (incorrect membership guesses).

For the same reason accuracy does not work, also AUC does not work, since the AUC averages over all false-positive rates, including high error rates that are irrelevant for a practical use.
The final takeaway is to test an attack with TP rates at low FP rates; this is emphasized by the log-log ROC representation.
This way there is more emphasis on the low FP part of the attack which is the most important thing to consider.

4.2 Innovations

In this chapter I will discuss the main innovations that I bring to MIAs with this thesis.
Part of the work of the thesis has been focused in replicate the results in [67], but also some innovative approaches have been experimented.

In the context of face images, removing the background from profile pictures seem to bring better results 4.2.1.
Another approach, based on attack (see Sect. 4.1.1) have been tried. With the scope of finding similar prompts to the ones present in the training set instead of trying to obtain images 4.2.2.

4.2.1 No background faces MIA

An innovation from this thesis is a smarter use of face images in the MIA attack. Faces can be considered a private asset that people are not willing to give for sale.
Many people would not agree to sell their profile images for research or commercial purposes, even though this is what actually happens today with images released on social media and websites.
Images of faces are considered private because it is a critical information that allows to recognize a human being among billions of others with that given image (or collection of images).
Face recognition is one of the most concerning topics in the contemporary world. This information can be used by organizations for improper surveillance.
Although people are becoming more aware of the importance of their privacy, billions of face images are floating around the internet every day. This motivate the interest to protect personality rights by being able to recognize if a given image was used or not in the training process of a text-to-image generation model.

The attack follows the same principle of the main attack that can be summarized in figure 4.1.
In [67] the researchers also tried to recognize faces with the MIA described in Sect. 4.1.1.
To improve the effectiveness of the attack, this experiment have been replicated with further preprocessing to eliminate the background in the face images.
Profile images usually have a background, this one can represent different environments, when the difference between embeddings of the original image and generated image is

performed, the information regarding backgrounds is redundant and is preferred to only have features of the faces themselves.
The background is considered redundant because the caption generated via BLIP usually captures the main features of the subject but does not really capture the background information; the result is that after the generation, the original and generated images will have different backgrounds with different features.
By removing backgrounds of the original profile image and the generated profile image (after the generation) the redundant information is eliminated and the feature extracting model can retrieve more representative information by using the same images.

Usually profile pictures have a distinct foreground and background, this is not guaranteed for other type of images that can be found in any text-to-image dataset.
Comparing Figs. a and b, it is clear why the background plays a different role in profile and non-profile pictures. For non-profile pictures, the elements of the background can carry important information of the picture for its semantic that will be lost with its removal.

(a) Non-profile picture (b) Profile picture

Figure 4.2: The importance of the background in images of different kind. (a) Non-profile picture (from LAION-5v+) contains several elements and the background in not easily identifiable. The removal of some elements can affect the semantic content of the image. (b) Profile picture (from VGG face) the background is not needed for semantic purposes, the only thing that matters is the face representation.

Removing backgrounds from images different than profile pictures can be challeng-

ing and not easy to automate, while with faces is pretty easy to achieve and many Python libraries can already do it with no effort.
In particular, for this experiment I used the rembg library [22].

4.2.2 Prompt attack

The idea behind this attack is similar to II-S [67], but instead of considering differences between the original image x and the generated image x' to understand if x is part of the training set D, This attack focuses just on the generated images.
The pipeline 4.1 stays pretty much the same; for creating the classifier A, I will still need to have a part D disclosed.
The idea is to get captions t' from the images through BLIP or by using the original captions t and then generate new images with those captions.
After the images have been generated, I will get the embeddings of the images and use them to train the classifier.
This time the embeddings are not the difference between the original and generated images but are just the ones referring to the generated images of the 'in' training set and 'out' training set images.
After A is trained, the attack will consist of giving the model an image, and it will say if the image was part or not of the training set. In this case, it's possible that the image is not a perfect copy of an image in the training set since we are speculating on just the generated images.
The attack is helpful, because instead of focusing on the image, it is possible to get the caption from the image using the same captioning model used for training and get an idea of a prompt that is in the training set.
Of course, it is very unlikely that the caption will be a perfect copy of one of the captions in the training set; however, it is possible that the caption will be similar or will generate similar content to one in the training set. In this way, we can disclose not the images in the training set, but the captions that generate similar content to the ones in the training set. In other words, we discover a caption that has same value of a caption in the training

set.

This is very helpful to gather prompts and use them to generate synthetic data that can be used to train shadow models 2.3.1.

Following formalization of 4.1.1, the differences with this attack are:

- The input of the binary classifier A is no longer the difference between the embeddings of x and x' but it is just x';
- The output of the binary classifier A is still true or false, but it is not the final output. The user needs to refer to the previously created prompt t' for the image x' in order to have a similar prompt to t, which is present in the training set.
- The attack is not a MIA but is useful to gather data to train shadow models and perform MIAs in a non-scarce resources environment.

The cosine similarity on the two prompts 4.3 and 4.4 is : 0.0 and thus, should not be a good indicator on how similar content the prompts generate.

To better understand if two prompts generate similar content, the best idea is to compare the two images with FID 3.3, this way it is possible to directly compare two generated images.

Figure 4.3: Originial prompt:
Rusty Bell on Old Lodge Building,
Searcy County, Arkansas

Figure 4.4: BLIP prompt:
rusty bell hanging from a brick wall
with a brick wall behind it

4.3 Implementation

The implementation does not follow a sequential flow, it follows a more experiment oriented flow, with only specific parts of code run for specific experiments.
To replicate the attacks done in this thesis, follow the pipeline in Sect. 4.1.1 or the others presented in the other attacks of this thesis 4.2.1, 4.2.2.
I will now explain the implementation following 4.1.1, the others are similar to it.
In order to replicate the experiments, there is first the need to import the various models to attack, the image captioning tools and the features extracting tools.
After everything is imported, either from Hugging Face [20] or other sources, there is the need to download the images of the different datasets 3.4, one dataset should be the training set of the attacked model, the other one should be similar but not the same dataset.
The captions from the images of the imported datasets need to be extracted with the previously imported captioning tool.
After the caption of each image of both datasets have been extracted, use them to generate new images by giving the captions as input to the text-to-image model to attack. This will require some time but will generate all of the images needed to train the binary classifier and perform the attack, if a diffusion model is used, in the image generation task I used 20 inference steps.
After the images are generated, it is possible to create a dataset to use for training a binary classifier.
This dataset should have the difference between the features of the original images and the generated images (in vector form) and the label corresponding to 'in' if the features belong to training set, 'out' otherwise.
In order to create this dataset, there is the need to use the imported features extracting tool on both original and generated images and then make the difference.
The difference is a simple element wise absolute difference, but another method can be chosen aswell, refer to Wu et Al. [67].
Finally, it is possible to use the dataset to train a binary classifier of choice and perform the attack, the training split for the experiments was 60% of the total number of samples and the test split was 40%.

The best binary classifier used in the experiments was a MLP classifier imported from Sklearn [47] with default hyperparameters except for max_iter which is = 1000.

For more information, please refer to the implementation code [21].

4.4 Privacy and DPDM

Differential private diffusion models [17], (Sect. 3.1.4) are a good way to test the use of differential privacy in diffusion models.
The presented models on the GitHub page [17] are pretrained, there is the possibility to sample from them or to use the existing code to train new DPDMs.
The idea for this experiment was to sample from the pretrained DPDMs (see table 4.1) and then compare the outputs to the original images, however, since the output is not linked to a specific original image, it is not possible to compare the output image to a specific training set image, so I could not recreate the attack II-S of the paper [67].

Dataset	Resolution	Epsilon
MNIST	28	10.0
CelebA	64	10.0
ImageNet	32	10.0

Table 4.1: The dataset field represents the dataset on which the model was trained, resolution is the output resolution (e.g. 28×28), and epsilon (ε) is a key parameter that quantifies the level of privacy protection provided by a differentially private algorithm or mechanism. It measures the trade-off between privacy and data utility. A smaller epsilon value indicates a stronger guarantee of privacy but may result in a larger loss of data utility, while a larger epsilon value provides less privacy protection but preserves more data utility.
A complete table can be found in the GitHub page [17].

I set the sampling procedure to sustain 1000 inference steps and at random (sampler.stochastic=true).
Since the models are trained with differential privacy, the final outputs of the models are blurred images, which make comparison with the training set data almost impossible. This is for sure a good trait of differential privacy for diffusion models; however, the blurriness, which is a good trait for privacy, is also a big downside for the usage of the model; with outputted blurry images, the users will be very unsatisfied, mainly because of today's quality standards.

The blurriness comes from the noise that is added through differential privacy, making it impossible to recognize a specific instance inside the dataset.
This follows the main principle of differential privacy, which is that the dataset should maintain the same effectiveness with or without a specific instance.
While differential privacy is a good solution for privacy issues, I would say that it is not applicable in the case of text-to-image models. The main reason is that the outputs quality is greatly reduced and therefore their utility.
Researchers should come up with more effective ways to preserve privacy in these models, which should not impact the final outputs; in the current state, the trade-off between privacy and performance is too heavy, and companies will always choose performance.

Chapter 5

Results

This chapter provides an overview of the experiments conducted with the introduced MIAs 4.1.1 4.2.
A FID 3.3 score comparison is presented to understand how well stable-diffuion 3.1.1 recognise the different datasets.
To conclude this chapter, the results of the MIAs are contrasted with those of state-of-the-art methods 5.5.

Referring to the previous formalization of an attack 4.1.1, an experiment is performed on a text-to-image model M trained on a dataset $D = (t_i, x_i)$.
The objective is to understand if a sample image x is part of the training set.
Since the attack is not in a blackbox-blind setup [30] (see Sect. 4.1.2), a subset of the training set is disclosed which is formalized as $D_{\text{subtarget}}$; a subset of a similar dataset to the training set is also disclosed, called $\bar{D}$.
The disclosed data is put together to create a dataset $D_{\text{auxiliary}} = \{D_{\text{subtarget}} \cup \bar{D}\}$ used for training the attack model A, which is a binary classifier.
$D_{\text{auxiliary}}$ has 50% of total samples belonging to $D_{\text{subtarget}}$ and 50% of total samples belonging to $\bar{D}$.
A captioning model C and a feature extracting model F are also fundamental to perform an attack.

The captioning model extracts captions from images $t' = C(x)$, the caption is given to the attacked model to create a new image that is linked to x, $x' = M(t')$.
Features of the images x and x' are extracted through F so that the absolute difference between the two can be computed and given as input feature along with the label 'in', 'out' to the attack model A for training phase.
After A is trained the model can be used on the test split which is 40% of the total number of samples to obtain the results of the attack.
In cases of the other two MIAs 4.2.1, 4.2.2 the are small adjustments in the procedure but the required elements are the same.

The experiments can be summarized in:

- II-S attack 5.1:
  - Stable-diffusion 5.1.1;
  - Karlo 5.1.2;
  - Dreamlike 5.1.3;
- Privacy attack 5.2:
  - BG 5.2.1;
  - No-BG 5.2.2;
- Prompt attack 5.3.

5.1 II-S attack

In this section the II-S attacks on different models are presented, in particular these models are:

- Stable-diffusion 5.1.1;
- Karlo 5.1.2;

- Dreamlike 5.1.3;

The experiments were conducted using a subset of LAION-aesthetic v2 5+ 3.4.1 as $D_{\text{subtarget}}$ and a subset of MS-COCO 3.4.3 as $\bar{D}$ for Stable-diffusion. The experiments were presented using different sizes of $D_{\text{auxiliary}}$ (400, 1000, 2000)
For Karlo, $D_{\text{subtarget}}$ was represented by CC3M, which is a subset of CC12M 3.4.4 and $\bar{D}$ is represented by MS-COCO 3.4.3, $D_{\text{auxiliary}}$ was only tested with 1000 total samples.
For Dreamlike, $D_{\text{subtarget}}$ was represented by LAION-aesthetic v2 5+ 3.4.1, since it was obtained by fine-tuning Stable-diffusion, and as $\bar{D}$ MS-COCO 3.4.3 was used. $D_{\text{auxiliary}}$ was only tested with 1000 total samples.

During the experiments of Stable-diffusion, different attack models A were used 2.4. These models are:

- SVM 2.4.1;
- Logistic regression 2.4.2;
- MLP 2.4.3.

After confirming that an MLP was the best model to use for classification I used the same attack model A to perform the other attacks on the remaning text-to-image models.

5.1.1 Stable-diffusion

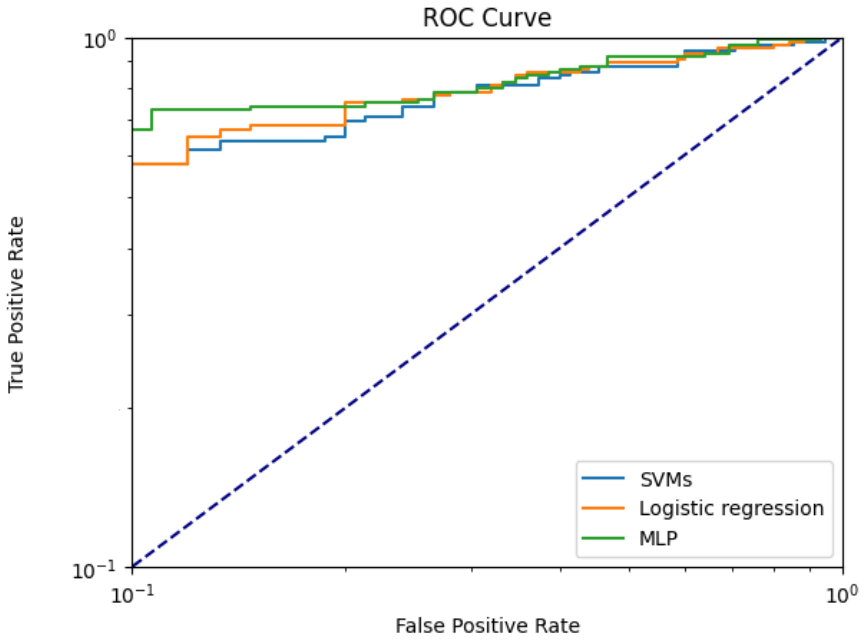

Figure 5.1: Log ROC curve of LAION vs COCO 400 images, stable-diffusion

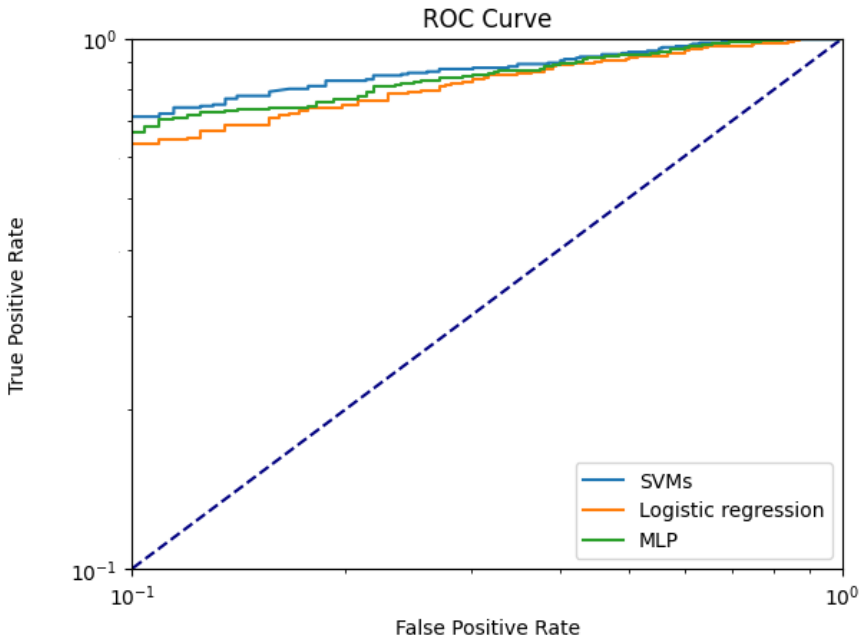

Figure 5.2: Log ROC curve of LAION vs COCO 1000 images, stable-diffusion

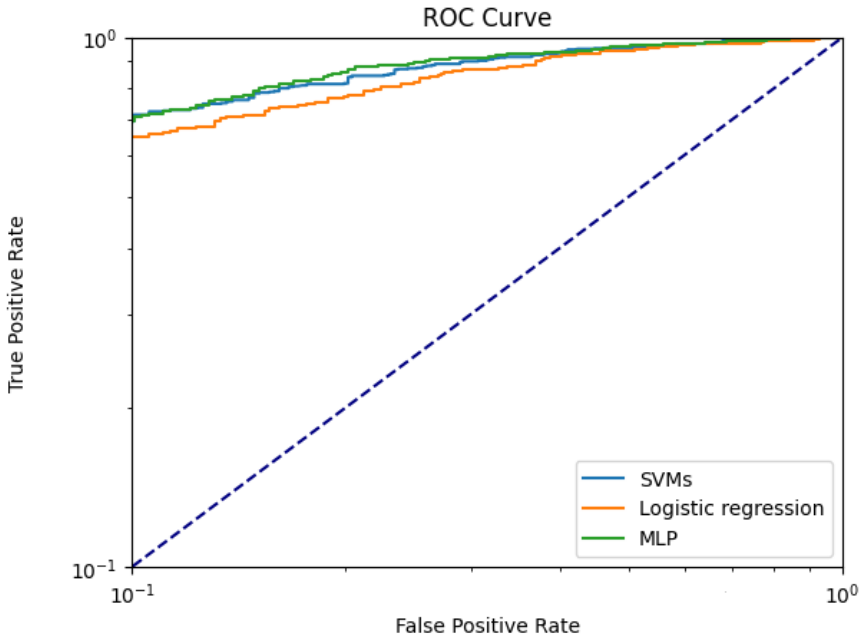

Figure 5.3: Log ROC curve of LAION vs COCO 2000 images, stable-diffusion

TPR with low FPR in 5.1 is:

- Tpr for fpr <= 0.1 in SVM is: 0.57
- Tpr for fpr <= 0.1 in Logistic regression is: 0.57
- Tpr for fpr <= 0.1 in MLP: 0.67

TPR with low FPR in 5.2 is:

- Tpr for fpr <= 0.1 in SVM is: 0.71
- Tpr for fpr <= 0.1 in Logistic regression is: 0.63
- Tpr for fpr <= 0.1 in MLP: 0.66

TPR with low FPR in 5.3 is:

- Tpr for fpr <= 0.1 in SVM is: 0.69

- Tpr for fpr <= 0.1 in Logistic regression is: 0.64
- Tpr for fpr <= 0.1 in MLP: 0.69

The information is summarized in the following table:

TABLE of TPR of different classifiers with FPR <= 0.1

	400 images	1000 images	2000 images
SVM	0.57	0.71	0.69
Log. Regr.	0.57	0.63	0.64
MLP	0.67	0.66	0.69

5.1.2 Karlo

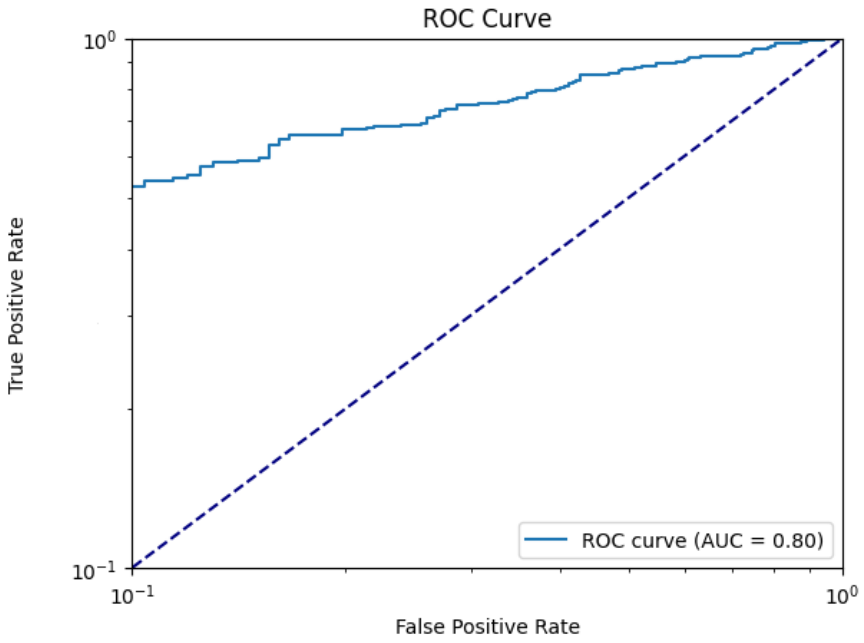

Figure 5.4: Log ROC curve of CC3M vs MS COCO, 1000 images, karlo

TPR with low FPR in 5.4 is:

- Tpr for fpr <= 0.1 using MLP is: 0.52

5.1.3 Dreamlike

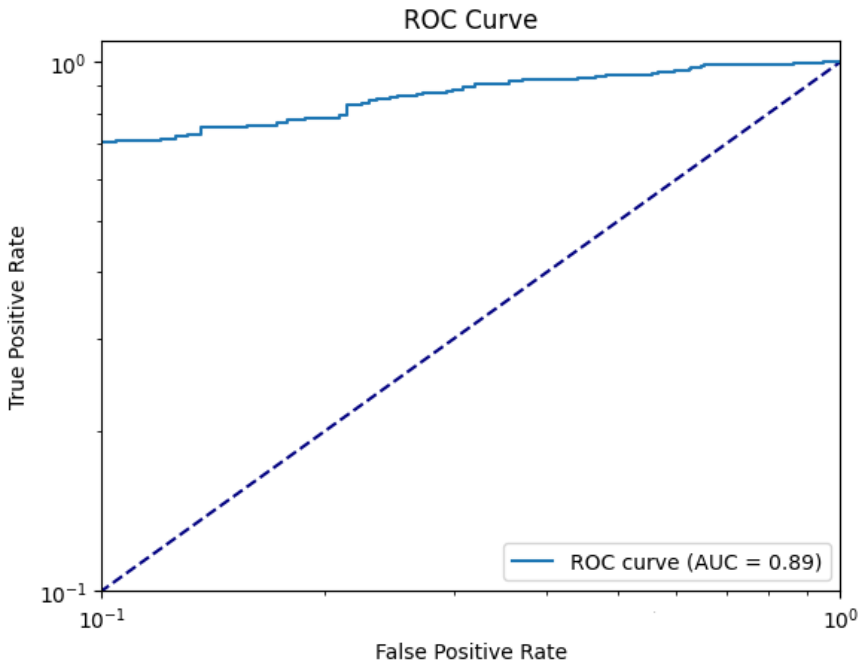

Figure 5.5: Log ROC curve of LAION5v+ vs MS COCO, 1000 images, dreamlike photoreal 2.0

TPR with low FPR in 5.5 is:

- Tpr for fpr <= 0.1 using MLP is: 0.70

5.2 Privacy attack

In this section, the privacy attack on Stable-diffusion is presented 4.2.1.
The experiments focus on two different settings: one where profile images have a background (BG) 5.2.1 and the other where the background is removed from images (NO BG) 5.2.2.
In both cases, $D_{\text{subtarget}}$ was represented by LAION-aesthetic v2 5+ faces 3.4.1 while $\bar{D}$ was represented by VGG-face 3.4.2.
$D_{\text{auxiliary}}$ have a total sample size of 1200 images.

5.2.1 Profile faces with background

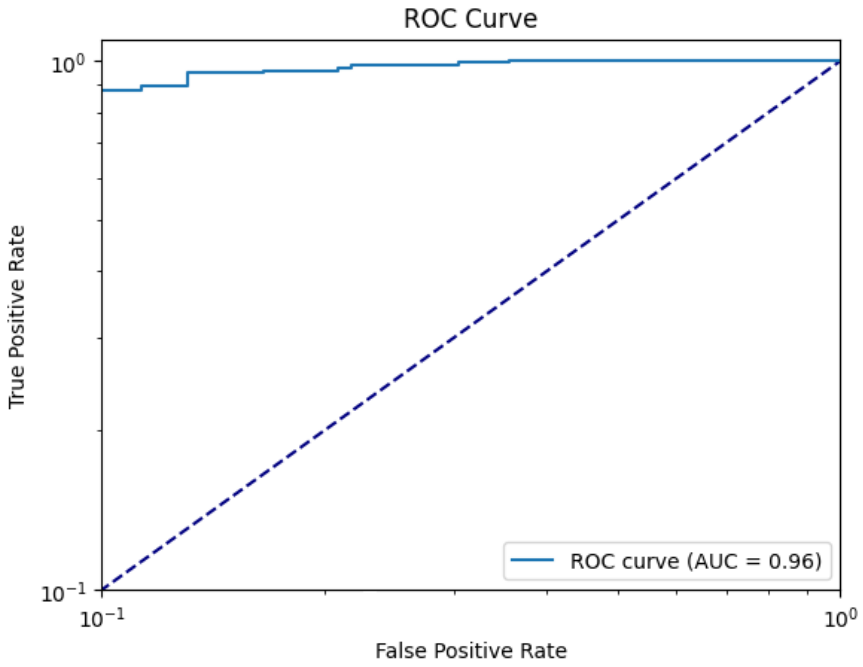

Figure 5.6: Log ROC curve of LAION5v+ faces vs VGG-face, with BG, 1200 images, stable-diffusion

TPR with low FPR in 5.6 is:

- Tpr for fpr <= 0.1 using MLP is: 0.88

5.2.2 Profile faces with no background

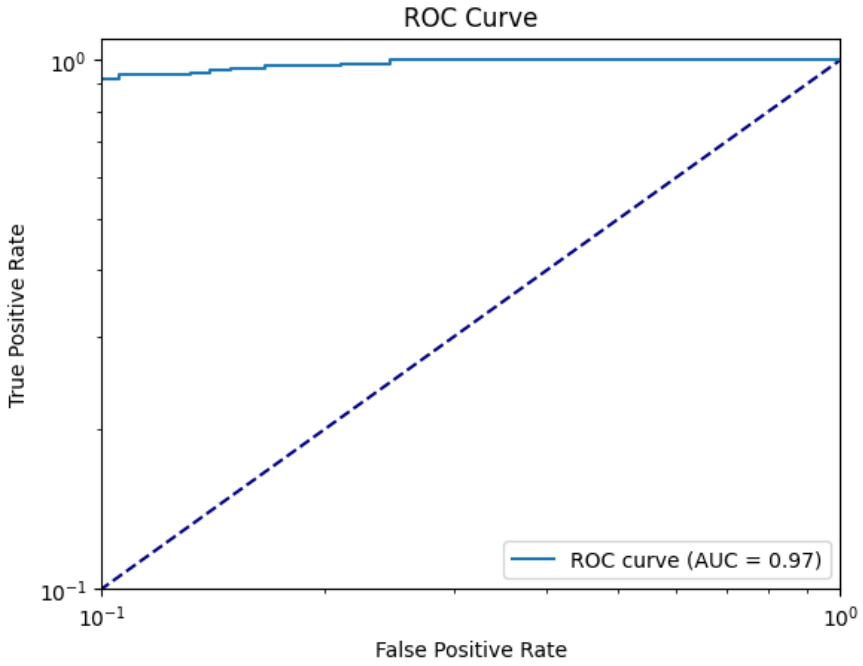

Figure 5.7: Log ROC curve of LAION5v+ faces vs VGG-face, NO BG, 1200 image, stable-diffusion

TPR with low FPR in 5.7 is:

- Tpr for fpr <= 0.1 using MLP is: 0.92

The removal of background brought a 0.04 increase in TPR with fpr <= 0.1 using just 1200 images.

5.3 Prompt attack

In this section, the prompt attack is presented 4.2.2.

This attack was conducted on Stable-diffusion 3.1.1, The aim of the attack is to retrieve similar prompts to the training set of the attacked model.

The attack follow the same format to the II-S attack 4.1.1, however, the input to the

classifier A is no longer the absolute distance of the extracted features of the original and generated images through F, but the input is constituted just of the features of the generated images.

Once A gives as output 'True' it means that it is possible to extract the prompt from the image x' through F to obtain a prompt which generates similar content to the ones present in the training set.

This process is not a MIA since it does not tell if an image is part or not of the training set, however, to have a comparison between the other performed MIAs, I stopped the experiment before obtaining the prompt from x' and just considered the output of A, if it is 'True' the image can be treated as part of the training set, in the opposite case it is not part of the training set 5.3.1. In this way I can actually compute the TPR at a low FPR like in the other performed attacks. Then, using FID I can understand if the prompts obtained from the classified 'True' images (which are the images 'in' the trainng set) generate similar content to the ones present in the real training set 5.3.2.

5.3.1 Stable diffusion

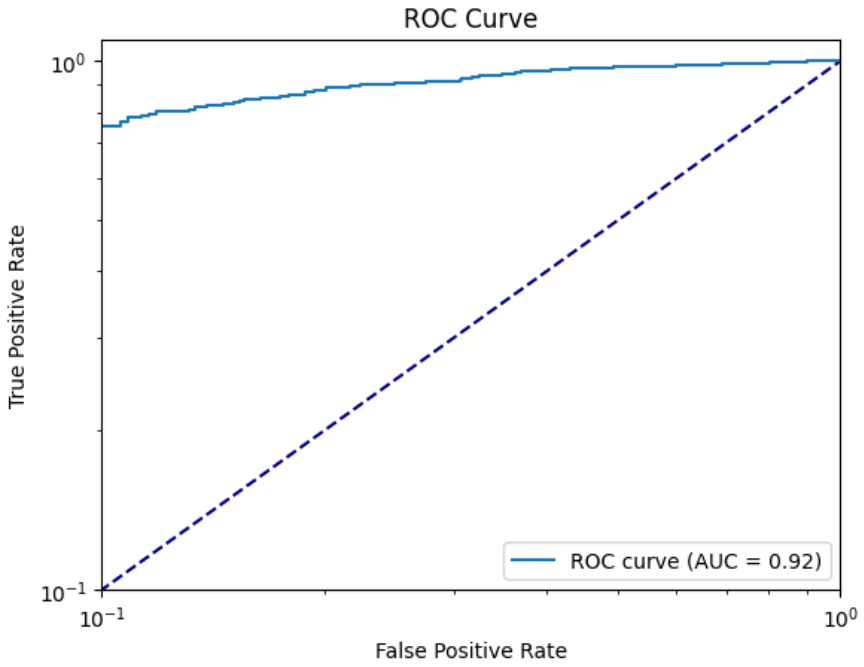

Figure 5.8: Log ROC curve of LAION5v+ vs MS COCO 2000 image, stable-diffusion

TPR with low FPR in 5.8 is:

- Tpr for fpr <= 0.1 using MLP is: 0.75

5.3.2 Generation of similar content from extracted prompts

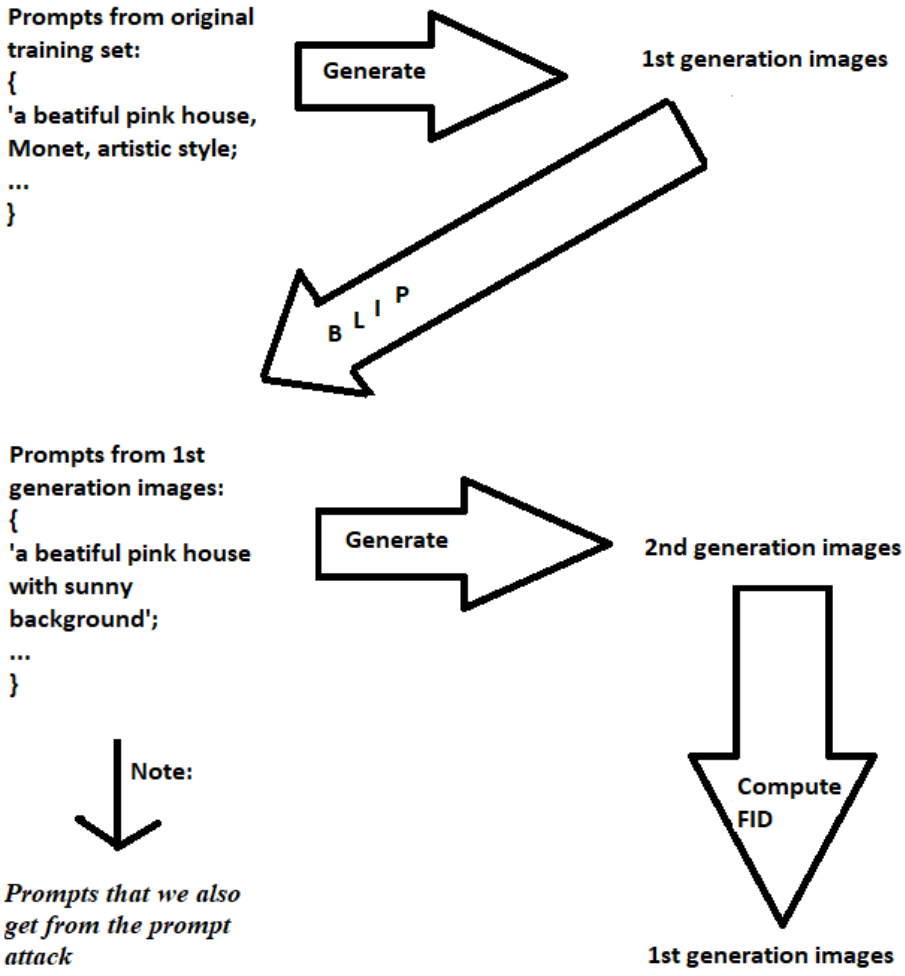

Figure 5.9: A schema on the validation of the 'prompt' attack

In this attack 5.3.1 is important to consider also the FID between the images generated with stable-diffusion from the original prompt of the training set (1st generation) and the images generated from the prompt captioned directly from the 1st generation images (2nd generation).

To understand if the FID between the two generation of images is small and thus, if the prompt generate similar content between the two generations, the images need to belong to a similar distribution. If that is the case, the prompts from the original training set generate similar content to the prompts extracted with BLIP on the classified 'in' images of the attack.
The FID obtained on 500 1st generation images and 500 2nd generation images is: **3736.850**.
Compared to the FIDs in 5.4, it is pretty small, this confirms the attack utility.

This attack is also useful to gather more prompts to generate synthetic data and create shadow models in order to perform other types of attack.

5.4 FIDs

The Fréchet Inception Distance 3.3 is a metric used to evaluate the quality and diversity of generated images. It measures the similarity between the feature distributions of real and generated images by comparing their multivariate Gaussian distributions.
A lower FID score indicates that the generated images are more similar to real ones, suggesting better image generation quality. FID has become a widely adopted metric for assessing the performance of text-to-image models.
In the following computations 5.1, FID has been computed on the feature vectors extracted from VGG16 3.2.2 on both original and generated images of the same dataset. The generated images were created using Stable diffusion 3.1.1.

Stable diffusion is a model trained on the LAION dataset, in table 5.1 it is possible to observe that the FID between original and generated images from LAION obtain the lowest FID.
The other datasets have a slightly lower increase in FID, if the number of images would have been higher than 500, then probably this gap would be bigger, however, even with this small number of images, it is possible to understand why the attacks work, the main reason is that the attack model is able to recognise that the images come from two differ-

ent distributions.

	FID
LAION 3.4.1	4018.502
VGG-face 3.4.2	5874.106
MS-COCO 3.4.3	4192.455
CC3M 3.4.4	4162.855

Table 5.1: FIDs computations between original and generated images of the corresponding datasets, text-to-image model used is Stable diffusion 3.1.1

5.5 Comparison with state-of-the-art

In this section, I will conduct a comparative analysis of the outcomes achieved in contrast to the latest state-of-the-art results.

Directly comparing my results to those of state-of-the-art Membership Inference Attacks (MIAs) poses a challenge due to significant variations in the attacked models and datasets utilized. Unlike many state-of-the-art approaches that typically necessitate the creation of a shadow model for MIA, my attack configurations consistently aimed to avoid the use of such models.

Given the distinctive characteristics of my approach, I could not identify any state-of-the-art MIAs that closely resemble my setup to the best of my knowledge. Consequently, comparing results across different setups becomes challenging.

One noteworthy exception is the work by Wu et al. [67], in which the setup aligns precisely with mine. However, it's worth noting that their results are presented in terms of accuracy, whereas I primarily focus on True Positive Rate (TPR) at low False Positive Rates (FPR). Nonetheless, I will still include these results, despite the difference in the evaluation metric.

It's important to acknowledge that the other MIA techniques typically employ more con-

strained setups compared to mine. Consequently, their results may appear lower due to their reliance on blackbox-blind [30] settings and the necessity to construct shadow models for effective attacks.

TPR @ 0.1% FPR

	C-10	C-100	WT103	Others
Yeom et al.[68]	0.0%	0.0%	0.1%	-
Shokri et al.[57]	0.3%	1.6%	-	-
Jayaraman et al.[32]	0.0%	0.0%	-	-
Song and Mittal. [60]	0.1%	1.4%	-	-
Sablayrolles et al. [55]	1.7%	7.4%	1.0%	-
Long et al. [40]	2.2%	4.7%	-	-
Watson et al. [65]	1.3%	5.4%	1.1%	-
Carlini et al. [7]	8.4%	27.6%	1.4%	-
Attack II-S	-	-	-	**69%**
Privacy attack, no BG	-	-	-	**92%**
Prompt attack	-	-	-	**75%**

Table 5.2: Results from Carlini et al. [7], with in bold, my results on the previously described datasets 3.4

Accuracy

	MS-COCO face	VG face
II-S attack LDM	98.0%	98.0%
II-S attack DALL-E mini	99.9%	99.9%

Table 5.3: Results from Wu et al. [67] with $\bar{D}$: MS-COCO face, VG face, on LDM and DALL-E mini models

Chapter 6

Conclusions

In this thesis, I have expounded upon various concepts related to Membership Inference Attacks (MIAs) and diffusion models, all within the context of a resource-constrained environment. Starting from the foundational background, I've provided concise summaries of various types of diffusion models, with a particular emphasis on both classical and emerging models 2.1.1, 2.1.2, 2.1.3, 2.2.

Within the realm of diffusion models, I've taken a more detailed approach by distinguishing between denoising diffusion probabilistic models (DDPMs) 2.2.1, denoising diffusion implicit models (DDIMs) 2.2.4, and latent diffusion models (LDMs) 2.2.5.

Furthermore, I've elucidated the methodologies and rationales behind MIAs, exploring how they are conducted 2.3, why they are undertaken, and which models are commonly employed to carry out such attacks 2.4. I've provided meticulous insights into the attacks performed within the scope of this thesis, including detailed descriptions of the techniques employed and the novel contributions introduced 4.

To ensure the reproducibility of results, I've also referenced the datasets 3.4 and technologies 3 utilized in these attacks.

In the experiments chapter 4, I've offered an exhaustive account of the attacks, their intricacies, and how they relate to state-of-the-art techniques 5.

To be more specific about the MIAs, I reimplemented the II-S attack outlined in the paper by Wu et al. [67], with a particular emphasis on log-scale Receiver Operating Characteristic (ROC) and True Positive Rate (TPR) at a 0.1% False Positive Rate (FPR), achieving a TPR of 69% 5.1 on the stable-diffusion model.

I also applied the same attack to different diffusion models, including Karlo v1 alpha, which employs a distinct architecture and dataset compared to stable-diffusion, yielding a TPR of 52% 5.1.2, and Dreamlike photoreal 2.0, a fine-tuned variant of stable-diffusion, with a TPR of 70% 5.1.3.

In terms of improvements, I enhanced the results of an attack against profile images, increasing the TPR from 88% 5.2.1 using the methodology from Wu et al. [67] to a TPR of 92% with my new approach 5.2.2.

Additionally, I developed an alternative MIA with a focus on the content generated by prompts, achieving a TPR of 75% 5.1.1 while maintaining a low Fréchet Inception Distance (FID) between the generated images from the original prompts and those obtained during the attack.

It's worth noting that the MIAs presented in this thesis 4 can be executed in a resource-constrained environment, albeit with the caveat that a portion of the training set must be disclosed. This, in turn, relaxes the conventional blackbox setting 4.1.2, presenting both advantages and disadvantages. On one hand, it enables MIAs with fewer resource requirements, while on the other hand, it necessitates specific conditions for attack execution. Thus, the decision of whether to disclose the dataset or opt for a more stringent blackbox-blind setting depends on the attacker's evaluation of resource allocation and attack objectives.

I hope this research will be useful in better understanding the foundations of MIAs and how to develop them even without much resources.

6.1 Future work

Emerging concepts for pushing the boundaries of Text-to-Image Models:

- Differential privacy holds great promise for safeguarding privacy in the context of text-to-image models. However, further research is warranted to ascertain its practicality in image generation scenarios, as it may present challenges that adversely impact the quality of results.
- An intriguing avenue to fortify the security of text-to-image models involves training them on synthetic datasets. By doing so, even if a MIA is executed, the data extracted may pose no threat to individuals' privacy since it is not derived from real-world individuals. Nonetheless, it is imperative that the generated data does not create synthetic but indistinguishable replicas of the originals. The potential incorporation of differential privacy within this context remains a subject for exploration.
- While the feature extraction model employed herein is VGG16 3.2.2, it is noteworthy that the approach aligns with the findings presented in [67], which suggests that virtually any feature extraction model can be utilized for the purpose of the MIA attack described in Section 4.1.1. Further investigation could shed light on whether this assertion holds true or if superior performance can be attained through the use of alternative extraction models.

Bibliography

[1] ACM, oct 2016. Available at https://doi.org/10.1145%2F2976749.2978318.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Available at https://www.tensorflow.org/.

[3] Open AI. research/generative-models, June 16, 2016. Available at https://openai.com/research/generative-models.

[4] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks, 2017.

[5] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders, 2021.

[6] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G. Willcocks. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7327–7347, 2022.

[7] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, A. Terzis, and Florian Tramèr. Membership inference attacks from first principles. *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914, 2021. Available at https://api.semanticscholar.org/CorpusID:244920593.

[8] Nicholas Carlini, Jamie Hayes, Milad Nasr, Matthew Jagielski, Vikash Sehwag, Florian Tramèr, Borja Balle, Daphne Ippolito, and Eric Wallace. Extracting training data from diffusion models. *CoRR*, abs/2301.13188, 2023. Available at https://doi.org/10.48550/arXiv.2301.13188, version 1.6.0.

[9] Soravit Changpinyo, Piyush Sharma, Nan Ding, and Radu Soricut. Conceptual 12M: Pushing web-scale image-text pre-training to recognize long-tail visual concepts. In *CVPR*, 2021.

[10] christoph Schuhmann. improved-aesthetic-predictor, 2022. Available at https://github.com/christophschuhmann/improved-aesthetic-predictor.

[11] christoph Schuhmann. dreamlike-photoreal-2.0, 2023. Available at https://huggingface.co/dreamlike-art/dreamlike-photoreal-2.0.

[12] Google Colab, 2018. Available at https://colab.research.google.com/.

[13] Common Crawl, 2008. Available at https://commoncrawl.org/.

[14] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[15] Roxana Danger. Differential privacy: What is all the noise about?, 2022.

[16] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021.

[17] Tim Dockhorn, Tianshi Cao, Arash Vahdat, and Karsten Kreis. Differentially private diffusion models, 2022. Additional URL: https://nv-tlabs.github.io/DPDM/.

[18] Jisu Choi Jongmin Kim Minwoo Byeon Woonhyuk Baek Donghoon Lee, Jiseob Kim and Saehoon Kim. Karlo-v1.0.alpha on coyo-100m and cc15m. https://github.com/kakaobrain/karlo, 2022.

[19] D.C Dowson and B.V Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455, 1982.

[20] Hugging Face, 2016. Available at https://huggingface.co/models.

[21] Federico Fiorio, 2023. Available at https://github.com/fedeflowers/Thesis.

[22] Daniel Gatis, 2023. Available at https://pypi.org/project/rembg/.

[23] Bhaskar Ghosh, Indira Dutta, Albert Carlson, Michael Totaro, and Magdy Bayoumi. An empirical analysis of generative adversarial network training times with varying batch sizes. 10 2020.

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[25] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. Available at https://doi.org/10.1038/s41586-020-2649-2.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[27] 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. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[28] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[29] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

[30] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey, 2022.

[31] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice, 2019.

[32] Bargav Jayaraman, Lingxiao Wang, Katherine Knipmeyer, Quanquan Gu, and David Evans. Revisiting membership inference under realistic assumptions, 2021.

[33] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. Available at https://doi.org/10.1561%2F2200000056, version 1.6.0.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. Available at https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[35] Bogdan Kulynych, Mohammad Yaghini, Giovanni Cherubin, Michael Veale, and Carmela Troncoso. Disparate vulnerability to membership inference attacks. *Proceedings on Privacy Enhancing Technologies*, 2022:460 – 480, 2019. Available at https://api.semanticscholar.org/CorpusID:237513943.

[36] LAION, 2023. Available at https://rom1504.github.io/clip-retrieval/?back=https%3A%2F%2Fknn.laion.ai&index=laion5B-H-14&useMclip=false.

[37] Yann LeCun, Koray Kavukcuoglu, and Clement Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.

[38] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation, 2022.

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[40] Yunhui Long, Lei Wang, Diyue Bu, Vincent Bindschaedler, Xiaofeng Wang, Haixu Tang, Carl A. Gunter, and Kai Chen. A pragmatic approach to membership inferences on machine learning models. In *Proceedings - 5th IEEE European Symposium on Security and Privacy, Euro S and P 2020*, Proceedings - 5th IEEE European Symposium on Security and Privacy, Euro S and P 2020, pages 521–534, United States, September 2020. Institute of Electrical and Electronics Engineers Inc. Funding Information: This work was supported in part by NSF CNS 13-30491 (THaW). The views expressed are those of the authors only. Publisher Copyright: © 2020 IEEE.; 5th IEEE European Symposium on Security and Privacy, Euro S and P 2020 ; Conference date: 07-09-2020 Through 11-09-2020.

[41] Ron Mokady, Amir Hertz, and Amit H. Bermano. Clipcap: Clip prefix for image captioning, 2021.

[42] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125, 2008.

[43] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[44] The pandas development team. pandas-dev/pandas: Pandas, February 2020. Available at `https://doi.org/10.5281/zenodo.3509134`.

[45] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015. Available at `https://api.semanticscholar.org/CorpusID:4637184`.

[46] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[48] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[49] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

[50] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

[51] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2021. Available at `https://api.semanticscholar.org/CorpusID:245335280`.

[52] 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 (CVPR)*, pages 10684–10695, June 2022.

[53] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[54] David E. Rumelhari, Geoffrey E. Hintont, Ronald, J., and Williams. Learning representations by backpropagating errors. 2004. Available at `https://api.semanticscholar.org/CorpusID:237368852`.

[55] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, Yann Ollivier, and Herve Jegou. White-box vs black-box: Bayes optimal strategies for membership inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5558–5567. PMLR, 09–15 Jun 2019. Available at `https://proceedings.mlr.press/v97/sablayrolles19a.html`.

[56] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, S. Sara Mahdavi, Rapha Gontijo Lopes, Tim Salimans, Jonathan Ho, David J Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding, 2022.

[57] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models, 2017.

[58] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[59] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[60] Liwei Song and Prateek Mittal. Systematic evaluation of privacy risks of machine learning models, 2020.

[61] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[62] TAD, 2023. Available at https://www.tadviewer.com/.

[63] Farzad Toutounchi and Ebroul Izquierdo. Advanced super-resolution using lossless pooling convolutional networks, 2018.

[64] Vladimir Naumovich Vapnik. Statistical learning theory. 1998. Available at https://api.semanticscholar.org/CorpusID:28637672.

[65] Lauren Watson, Chuan Guo, Graham Cormode, and Alex Sablayrolles. On the importance of difficulty calibration in membership inference attacks, 2022.

[66] Papers with code, 2023. Available at https://paperswithcode.com/dataset/coco.

[67] Yixin Wu, Ning Yu, Zheng Li, Michael Backes, and Yang Zhang. Membership inference attacks against text-to-image generation models, 2022.

[68] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting, 2018.

[69] Yu Zeng, Zhe Lin, Jimei Yang, Jianming Zhang, Eli Shechtman, and Huchuan Lu. High-resolution image inpainting with iterative confidence feedback and guided upsampling, 2020.