Department of Physics and Astronomy
Heidelberg University

Master Thesis in Physics
submitted by

Sebastian Heid

born in Gelnhausen (Germany)

2026

Diffusion Model Distillation

This Master Thesis has been carried out by Sebastian Heid at the
IWR in Heidelberg
under the supervision of
Prof. Dr. Carsten Rother
and
...

Abstract

Table of Contents

List of Abbreviations

ADD Adversarial Diffusion Distillation

AI Artificial Intelligence

ALD Annealed Langevin Dynamic

CLS Classification

CNN Convolutional Neural Network

DDIM Denoising Diffusion Implicit Model

DDPM Denoising Diffusion Probablistic Model

DM Diffusion Model

ELBO Evidence Lower Bound

GAN Generative Adversarial Network

iid independent and identically distributed

KL Kullback-Leibler

LADD Latent Adversarial Diffusion Distillation

LLM Large Language Model

LPIPS Learned Perceptual Image Patch Similarity

MCMC Markov Chain Monte Carlo

MLE Maximum Likelihood Estimation

MLP Multi Layer Perceptron

MSE Mean Squared Error

NCSN Noise-Conditional Score Network

NF Normalizing Flow

NNL Negative Log-Likelihood

ODE Ordinary Differential Equation

RNN Recurrent Neural Network

RoPE Rotary Positiona Embeeding

SDE Stochastic Differential Equation

SGM Score-Based Generative Model

SOTA State-Of-The-Art

VAE Variational Autoencoder

VI Variational Inerence

ViT Vision Transformer

List of Figures

List of Tables

Chapter 1

Introduction

lksdjfklsjdflkjsdklfjdsklj

1.1 Generative Models

Generative models are a class of machine learning models that are trained to learn the true data distribution to generate new synthetic data elements. They are of high relevance for text and image generation tasks. In particular, famous models for text generation, also known as large language models (LLMs) include ChatGPT [1], Claude [4], Gemini [82] or Lama [85] whereas Stable Diffusion [54**?**], Flux [36] and Pixart [11] are widely used for image generation tasks. Both are being used more and more in the professional world and in daily life. Several different generator methods have been developed, such as variational autoencoders (VAEs) [32], generative adversarial networks (GANs) [20, 55], normalizing flows (NFs) [59], and diffusion models (DMs) [69].
The principal idea of a generative model f_θ is to learn a mapping from a simple distribution, e.g., a normal distribution $p_{\text{latent}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, often called the latent distribution, to a highly complex data distribution $p_{\text{data}}(\mathbf{x})$. A new data element $\mathbf{x}$ is obtained by sampling from the latent distribution $\mathbf{z} \sim p(\mathbf{z})$ and applying the model

$$\mathbf{x} = f_\theta(\mathbf{z}) \sim p_{\text{model}}(\mathbf{x}) \approx p_{data}(\mathbf{x}) \tag{1.1}$$

where the model's output distribution $p_{\text{model}}(\mathbf{x})$ should approximate the true data distribution as closely as possible.

In the following, the main concepts of VAEs, NFs and GANs are briefly presented to provide a comprehensive overview followed by an extensive discussion of DMs in chapter 1.2 that are the most promising models today.

1.1.1 Variational Autoencoder

Autoencoder

An autoencoder [24] is a neural network commonly used for dimensionality reduction tasks. It aims to compress high-dimensional data to lower-dimensional space by preserving important information. The architecture consists of two main parts, namely the encoder and the decoder. The encoder maps high-dimensional input data to a latent space, effectively compressing the data, and the decoder learns to reconstruct the original data from its latent space representation. Typically, as learning objective, a simple reconstruction loss, e.g. mean-squared error, is applied comparing the original input data and the output of the autoencoder.

Evidence Lower Bound Loss (ELBO)

The VAE belongs to the class of probabilistic models. A common learning objective for probabilistic models is maximum likelihood estimation (MLE). Given a set of observations $\{\mathbf{x}_i\}_{i=1}^N$ the likelihood is given under the assumption of identical and independent distributed samples

iid as

$$\mathcal{L}_{\text{MLP}} = \prod_{i=1}^{n} p_{\text{model}}(\mathbf{x}_i) \quad . \tag{1.2}$$

However, in practice, minimizing the negative log-likelihood is often preferred over maximizing the likelihood due to possible instabilities caused by the product of likelihoods

$$\mathcal{L}_{\text{NLL}} = -\sum_{i=1}^{n} \log p_{\text{model}}(\mathbf{x}_i) \quad . \tag{1.3}$$

This approach works well for simple distributions. However, for models that include latent variables $\mathbf{z} = \mathbf{z}_{1:m}$, such as VAEs, the likelihood becomes an intractable marginalization over the latent space

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \tag{1.4}$$

leading to the the so-called evidence lower bound (ELBO) loss [8] being used as training objective for VAEs.

In general, untractable probability distributions are a core challenge in modern statistics. This is especially the case for posterior distributions in Bayesian statistics. Assuming the joint probability distribution

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}) \cdot p(\mathbf{x}|\mathbf{z}) \tag{1.5}$$

is given with data $\{\mathbf{x}_i\}_{i=1}^{N}$ and latent variables $\{\mathbf{z}_j\}_{j=1}^{M}$. During inference, the in general untractable posterior $p(\mathbf{z}|\mathbf{x})$ is the quantity that needs to be computed. In the past, the dominant approach to tackle this challenge was using Monte Carlo Markov Chain simulations (MCMC) [84], which approximate the true posterior distribution by sampling. The drawback of these methods is that the sampling procedures are very slow. Therefore, in recent years, a new method has gained popularity, called variational inference (VI) [9]. In contrast to MCMC VI reformulates the problem as an optimization task

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathscr{Q}}{\arg\min}\ \text{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\right] \tag{1.6}$$

where $\mathscr{Q}$ is a family of distributions. The goal is to find the proposal density $q(\mathbf{z}) \in \mathscr{Q}$ that best approximates the true posterior distribution. To compare the proposal density and the posterior the Kullback-Leibler divergence (KL) [35] is used having the general form of

$$\text{KL}[p_0(\mathbf{x}) \,\|\, p_1(\mathbf{x})] = \int p_0(\mathbf{x}) \log \frac{p_0(\mathbf{x})}{p_1(\mathbf{x})} d\mathbf{x} \tag{1.7}$$

comparing two distributions p_0 and p_1. The key for a good choice of $\mathscr{Q}$ is to choose a family that is complex enough so that it contains a $q^*(\mathbf{z})$ close to the true posterior but simple enough to enable efficient optimization. Applying VI to the problem of finding the true posterior yields

$$\begin{aligned}
\mathrm{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\right] &= \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z} && (1.8)\\
&= \int q(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) d\mathbf{z} && (1.9)\\
&= \int q(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \log \frac{p(\mathbf{z},\mathbf{x})}{p(\mathbf{x})} d\mathbf{z} && (1.10)\\
&= \mathbb{E}\left[\log q(\mathbf{z})\right] - \mathbb{E}\left[\log p(\mathbf{z},\mathbf{x})\right] + \log p(\mathbf{x}) && (1.11)
\end{aligned}$$

From here, the ELBO loss is obtained by

$$\begin{aligned}
\log p(\mathbf{x}) &\geq \log p(\mathbf{x}) - \ \mathrm{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\right] && (1.12)\\
&= \mathbb{E}\left[\log p(\mathbf{z},\mathbf{x})\right] - \mathbb{E}\left[\log q(\mathbf{z})\right] && (1.13)\\
&= \mathbb{E}\left[\log p(\mathbf{x}|\mathbf{z})\right] - \ \mathrm{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z})\right] && (1.14)\\
&= \mathrm{ELBO} \quad . && (1.15)
\end{aligned}$$

Maximizing the ELBO loss is equivalent to minimizing the KL divergence from 1.6 and most importantly, it is tractable.
For training VAEs the ELBO objective is used. The difference compared to the MSE loss usually used for training autoencoders is that it contains an additional term from the KL divergence that enforces a specific structure on the latent space. Let θ be the decoder parameters and ϕ the encoder parameters then the objective of the VAE is given by

$$\mathcal{L}_{\mathrm{VAE}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[-\log p_\theta(\mathbf{x} \mid \mathbf{z})\right]}_{\text{Reconstruction Loss}} + \underbrace{\mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\right]}_{\text{Regularization}} \quad . \tag{1.16}$$

In practice, the latent distribution is often set to a standard normal distribution $p_{\text{latent}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{1})$ and the reparameterization trick [32] enabling gradient computation is applied. After training, a new data point can be generated by sampling from the latent distribution $\mathbf{z} \sim p_{\text{latent}}(\mathbf{z})$ and applying the decoder of the trained model.

1.1.2 Normalizing Flow

Normalizing flows (NFs) [59] are another class of propablistic models, similar to VAEs, which learn a mapping between a latent distribution to the data distribution. However, NFs have some key differences compared to VAEs. First, NFs are fully invertible meaning the network F_θ consists of a sequence of (simpler) bijective operations so that an inverse transformation $\overline{F_\theta}$ mapping back from data to latent distribution exists. As a consequence, the latent space and the data space need to have the same dimension.
A second decisive advantage is that NFs allow the exact and tractable computation of the

likeihood using the change of variables formula

$$p_{\text{latent}(\mathbf{z})} = p_{\text{model}}(\mathbf{x}) \left| \frac{\partial F_\theta(\mathbf{z})}{\partial \mathbf{z}} \right| = p_{\text{model}}(F_\theta(\mathbf{z})) \left| \frac{\partial F_\theta(\mathbf{z})}{\partial \mathbf{z}} \right| \tag{1.17}$$

for the latent distribution and

$$p_{\text{model}}(\mathbf{x}) = p_{\text{latent}}(\mathbf{z}) \left| \frac{\partial F_\theta(\mathbf{z})}{\partial \mathbf{z}} \right|^{-1} = p_{\text{latent}}(\overline{F}_\theta(\mathbf{x})) \left| \frac{\partial \overline{F}_\theta(\mathbf{x})}{\partial \mathbf{x}} \right| \quad . \tag{1.18}$$

It is worth noting that computing the likelihood (1.18) includes computing the determinant of a jacobian matrix. Therefore, NFs rely on carefully designed invertible layers (e.g., affine coupling layers [17]) where the determinant of the jacobian can be easily computed. This enables training via exact maximum likelihood estimation and does not rely on the ELBO loss like VAEs.

1.1.3 Generative Adversarial Network

In contrast to VAEs, NFs or DMs, GANs [20] belong to the class of non-probablistic generators which do not require an explicit likelihood function. The model, also called the generator G, is trained in an adversarial training setup that contains, on the one hand, the generator that should generate new data points being as realistic as possible and, on the other hand, a discriminator that should be able to distinguish if a data point is generated by the generator ("fake") or from the original dataset ("real"). The generator and the discriminator are alternately trained, where the generator tries to fool the discriminator by generating more realistic data points, and the discriminator D becomes better at distinguishing "fake" from "real". This is formalized in a MinMax condition

$$\min_G \max_D \; \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{1.19}$$

where $G(\mathbf{z})$ is a sample generated by the generator and $\mathbf{x}$ is an element from the original dataset. The prediction of the discriminator is close to one if it identifies its input as "real" and close to zero if it is identified as "fake".

GANs were the dominant class of network architectures for image generating [29, 43, 67] but were overtaken by diffusion models in the last years. The GANs training process turned out to be often instable, and extensive hyperparameter tuning and engineering is required [41, 64]. One challenge is to carefully balance the learning dynamics of the generator and the discriminator. If one overpowers the other, e.g., the discriminator always perfectly distinguishes "real" from "fake" samples, the generator stops learning. Another difficulty is known as mode collapse [83], meaning that the generator just produces samples from a part of the original data distribution but fails to completely capture it. A third challenge is the problem of exploding and vanishing gradients [5] which prevents effective learning of the generator.

1.2 Diffusion Models

Diffusion models [25, 61, 70, 71, 73, 77] are probabilistic generative models that have achieved remarkable success in image generation tasks [2, 36] demonstrating exceptional performance in terms of image quality and image diversity. While they require significant computational resources and have slower inference compared to some alternatives [20, 32, 59], they have become the dominant generative modeling paradigm in the research community. The main principals of diffusion models are reflected in two processes motivated by principles of non-equilibrium thermodynamics, namely the forward process and the reverse process. In the forward process, the original images are gradually perturbed by systematically adding Gaussian noise until pure noise is left. The reverse process describes the iterative noise removal by the diffusion model until a clean image is obtained. During image generation by a diffusion model, one starts from randomly sampled noise and applies the diffusion model iteratively, progressively denoising at each step. The repeated application gradually produces a clean, high-quality image.

1.2.1 Foundations of Diffusion Models

Over time, several different formulations of diffusion models were proposed. However, there are three dominant formulations recognized in the literature [14]: Denoising diffusion probabilistic models (DDPMs) [25, 70], score-based generative models (SGMs) [73] and stochastic differential equations (SDEs) [77]. Recently, in models like Flux-dev [36], a different formulation called rectified flow matching [42, 44] emerged. In the following, all four formulations are presented, whereby DDPM and rectified flow matching is discussed in depth due to their significant influence on widely known models such as Flux-dev [36] or Stable Diffusion [54, 60] which are partly used in this work.

Denoising Diffusion Probabilistic Models

The discussion of DDPM is based on the lecture notes form Inga Strümke and Helge Langseth [80] if not stated otherwise.
In DDPMs [25] the forward and reverse process are modeled as Markov Chains, meaning that each step only depends on the immediate previous step. This key concept is illustrated in fig. 1.1. Let $\mathbf{x}_0, ...\mathbf{x}_T$ be the states of the Markov Chain with $\mathbf{x}_0$ representing the clean image and $\mathbf{x}_T$ representing pure noise, then one step of the reverse process is given by $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$. This shows that the slightly denoised state $\mathbf{x}_{t-1}$ just depends on state $\mathbf{x}_t$ and θ denotes the parameters of the trained diffusion models which is used to predict $\mathbf{x}_{t-1}$. The forward process, which progressively adds noise proceeds from left to right in fig. 1.1 and is given by $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ where no learned parameters are required.

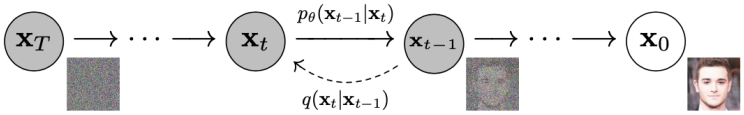

Figure 1.1: The graphical model of DDPM from [25].

Forward Process In the forward process, noise is gradually added to the original data to transform the data distribution p_{data} to a new often simpler distribution p_{prior}, such that for a large number of steps T, approximately $\mathbf{x}_T \sim p_{\text{prior}}$ holds. It is important that this approximation holds because during the generation process, a sample from p_{prior} is drawn and the process is reversed. To add the right amount of noise at every step and ensure a smooth transition between p_{data} and p_{prior}, being crucial for the performance of the diffusion model in the reverse process, a variance schedule needs to be defined $\{\beta_t\}_{t=1}^T$, e.g., a linear [25], a cosine [49] or a learnable [31] schedule. Normally, the variance schedule is chosen such that the variance β_t is small for early steps t which provides a gradual corruption of the input data. Concretely, the next state $\mathbf{x}_t$ within the Markov Chain is computed by

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = K(\mathbf{x}_t; \mathbf{x}_{t-1}, \beta_t) \tag{1.20}$$

with the Markov kernel K, which is chosen as a Gaussian Markov diffusion kernel in DDPM. The joint probability of the entire sequence $\mathbf{x}_{1:T}$, given a sampled $\mathbf{x}_0$ from the data distribution, is obtained by

$$q\left(\mathbf{x}_{1:T}|\mathbf{x}_0\right) = \prod_{t=1}^{T} q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right) \tag{1.21}$$

As a Gaussian kernel is chosen the explicit form of $q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right)$ is given by

$$q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}\right) \tag{1.22}$$

which leads to the prior distribution $p_{\text{prior}}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{I})$. A property of this formulation is that there is a closed-form solution for every $\mathbf{x}_t$ in the Markov Chain. In general, sampling from a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ can be achieved by $\mathbf{x} = \boldsymbol{\mu} + \mathbf{A}\mathbf{z}$ with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\boldsymbol{\Sigma} = \mathbf{A}\mathbf{A}^T$. Defining $\alpha_t = 1 - \beta_t$ and setting $\mathbf{A} = \sqrt{1-\alpha_t}\mathbf{I}$ any $\mathbf{x}_t$ of the Markov chain can be computed by

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{z}_t && (1.23)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\,\mathbf{z}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{z}_t && (1.24)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\,\mathbf{z}_{t-1:t} && (1.25)\\
&= \dots && (1.26)\\
&= \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\mathbf{z}_{0:t}. && (1.27)
\end{aligned}$$

where $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$ which shows that $q(\mathbf{x}_t|\mathbf{x}_0)$ follows a normal distribution

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x_t}; \sqrt{\bar{\alpha}_t}\mathbf{x_0}, (1-\bar{\alpha}_t)\mathbf{I}\right) \tag{1.28}$$

This discussion underlines that during the forward process the mean of $\mathbf{x}_t$ is gradually moved towards zero and the variance increases towards one. Therefore, the closed-form solution for $\mathbf{x}_t$ is

$$\mathbf{x_t} = \sqrt{\bar{\alpha}_t}\mathbf{x_0} + \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon} \tag{1.29}$$

with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This possibility to directly sample every $\mathbf{x}_t$ is crucial for efficient training.

Reverse Process The reverse process recovers the clear image by iteratively removing noise. The authors of [19] showed that in the limit of infinitesimal small steps the reverse process has the same distributional from as the forward process, a Gaussian. Particularly, in the reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is the quantity of interest. However, it is not tractable due to

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t) = q(\mathbf{x}_t|\mathbf{x}_{t-1})\frac{q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)} \tag{1.30}$$

using Bayes law where $q(\mathbf{x}_{t-1})$ and $q(\mathbf{x}_t)$ are marginal distributions requiring the integration over the whole distribution $q(\mathbf{x}_0)$. Therefore, it is modeled by a neural network parameterized by θ that should learn the Gaussian distribution for each step. Although $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is not tractable $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is which is leveraged during training. The joint distribution learned by the model is given by

$$p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{1.31}$$

where $p(\mathbf{x}_T) =$ is pure noise and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is a gaussian with learned mean and covariance

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\right) \tag{1.32}$$

As seen in the forward process, the variance of the noise follows a predefined schedule which means that the neural network just has to learn the mean $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t)$ and the variance is fixed by $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2\mathbf{I}$.

Objective Function Diffusion models belong to the class of probabilistic generative models. A natural choice for the objective of these models is the negative log-likelihood $-\log p_\theta(\mathbf{x}_0)$. As discussed in sec. 1.1.1 the likelihood is not always tractable. Therefore, the ELBO loss is used as learning objective. In the following the ELBO loss is derived for diffusion models. This derivation is mainly based on [53] and [25]. Starting with the expectation of the negative log-likelihood

$$
\begin{aligned}
&- \mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0) \right] && (1.33)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T \, p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right) \right] && (1.34)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T \, p(\mathbf{x}_T) \, q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right) \right] && (1.35)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \, \mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0)} \left[p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] \right] && (1.36)
\end{aligned}
$$

using Jensen's inequality

$$
\begin{aligned}
-\mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0) \right] &\leq -\mathbb{E}_{q(\mathbf{x}_0)} \left[\mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right) \right] \right] && (1.37)\\
&= -\mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right) \right] && (1.38)\\
&= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] && (1.39)\\
&=: \mathcal{L}_{\text{DDPM}} && (1.40)
\end{aligned}
$$

one obtains the loss for training a diffusion model. This can be further transformed

$$
\begin{aligned}
\mathcal{L}_{\text{DDPM}} &= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=2}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] && (1.41)\\
&= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=2}^{T} \log \left(\frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}{q(\mathbf{x}_t \mid \mathbf{x}_0)} \right) \right.\\
&\quad \left. - \log \frac{p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1)}{q(\mathbf{x}_1 \mid \mathbf{x}_0)} \right] && (1.42)\\
&= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T \mid \mathbf{x}_0)} - \sum_{t=2}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right] && (1.43)\\
&= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[\mathrm{KL} \left(q(\mathbf{x}_T \mid \mathbf{x}_0) \,\|\, p(\mathbf{x}_T) \right) + \sum_{t=2}^{T} \mathrm{KL} \left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right) \right.\\
&\quad \left. - \log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right] && (1.44)
\end{aligned}
$$

For training, only non-constant terms are relevant. That is why the loss further simplifies to

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_t)} \left[\text{KL} \left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right) \right]$$
$$- \mathbb{E}_{q(\mathbf{x}_0,\ldots,\mathbf{x}_T)} \left[\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right] + \text{const} \tag{1.45}$$
$$\approx \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_t)} \left[\text{KL} \left(q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right) \right] + \text{const} \tag{1.46}$$

assuming that the term $\mathbb{E}_{q(\mathbf{x}_0,\ldots,\mathbf{x}_T)} \left[\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) \right]$ is negligible. Note that in eq. 1.46 only Gaussians, $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)\right)$ with $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$ and $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \hat{\mu}(\mathbf{x}_t, \mathbf{x}_0), \hat{\beta}_t \mathbf{I}\right)$ are compared in the KL-divergence meaning there is a closed-form solution. The loss for the diffusion model boils down to

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_t)} \left[\frac{1}{2\sigma^2} ||\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2 \right] \tag{1.47}$$

The mean and the variance of the forward process $\hat{\mu}$ is obtained by

$$\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t \tag{1.48}$$

$$\hat{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t \tag{1.49}$$

However, instead of predicting the mean, diffusion models are more often trained on predicting the noise $\boldsymbol{\epsilon}_\theta$ because [25] empirically found that this lead to more stable training

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T} \frac{1}{2\tilde{\beta}_t} \frac{(1 - \alpha_t)^2}{\alpha_t (1 - \bar{\alpha}_t)} \cdot \mathbb{E}_{q(x_t|x_0)} \left[\|\boldsymbol{\epsilon}_\theta(x_t, t) - \boldsymbol{\epsilon}_t\|_2^2 \right] \tag{1.50}$$

which is achieved by reparameterization of eq. 1.47 via

$$\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \hat{\alpha}_t}} \boldsymbol{\epsilon}_t \right) \tag{1.51}$$

In addition, they found empirically that ignoring the weighting term of eq. 1.50 worked better

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t\|_2^2 \right] \tag{1.52}$$

DDPM Sampler The training process of the DDPM approach is depicted in algorithm 1 and fig. 1.2). In each training iteration, an image $\mathbf{x}_0 \sim p_{\text{data}}$ is sampled from the data distribution. Next, a timestep t is uniformly sampled from a predefined interval $[0, ..., T]$ and Gaussian noise is drawn according to $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Using eq. 1.29 the noisy image $\mathbf{x}_t$ at timestep t is

computed. The diffusion model is applied to predict the noise from this noisy input, and the loss function (eq. 1.52) measures the difference between the predicted and true noise. Finally, the model parameters are updated using gradient descent or a more advanced iterative optimization algorithms.

Algorithm 1 Training of DDPM using learning rate η [80]

repeat
- $\mathbf{x}_0 \sim p_{\text{data}}$
- $t \sim \text{Uniform}(\{1, \dots, T\})$
- $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \boldsymbol{\epsilon}$
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \|\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) - \boldsymbol{\epsilon}\|_2^2$

until converged

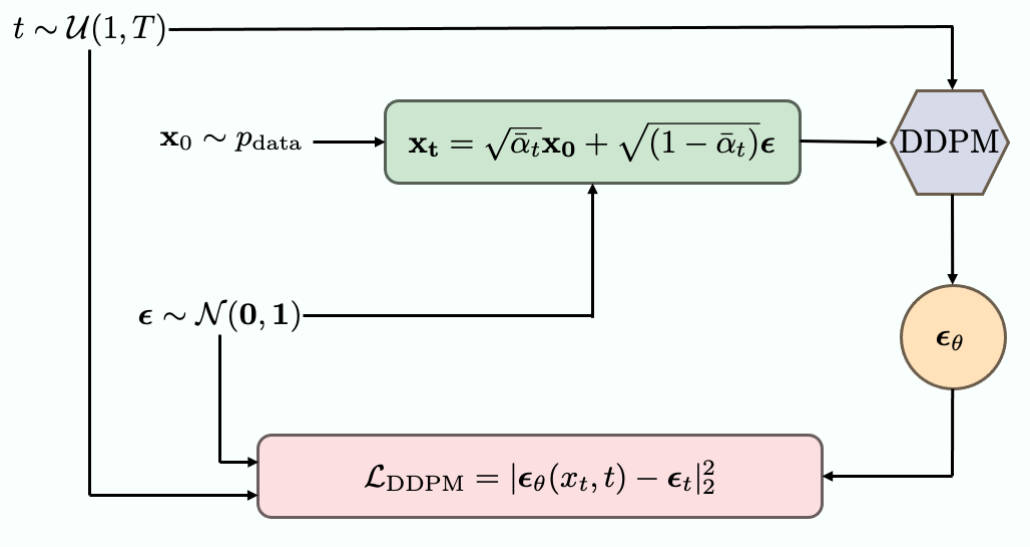

Figure 1.2: DDPM training [10].

Algorithm 2 and fig. 1.3 show schematically the sampling process which reverses the diffusion process to generate images form noise. Starting with pure Gaussian noise $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the process iteratively denoises the image. At each timestep t, the slightly denoised image $\mathbf{x}_{t-1}$ is computed using

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(x_t, t) \right) + \sigma_t \mathbf{z} \tag{1.53}$$

until $t = 2$. In the last step, the final clear images is obtained by

$$\mathbf{x}_0 = \frac{1}{\sqrt{\alpha_1}} \left(\mathbf{x}_1 - \frac{1 - \alpha_1}{\sqrt{1 - \bar{\alpha}_1}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_1, 1) \right) + \sigma_1 \mathbf{z} \tag{1.54}$$

Here, it becomes clear that the model has to predict the noise T times which makes inference of such diffusion model slow.

Algorithm 2 Sampling of DDPM [80]

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \dots, 1$ **do**
  if $t > 1$ **then**
    $\lambda \leftarrow 1$
  else
    $\lambda \leftarrow 0$
  end if
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_{t-1} \leftarrow \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \lambda \cdot \sigma_t \cdot \mathbf{z}$
end for
return $\mathbf{x}_0$

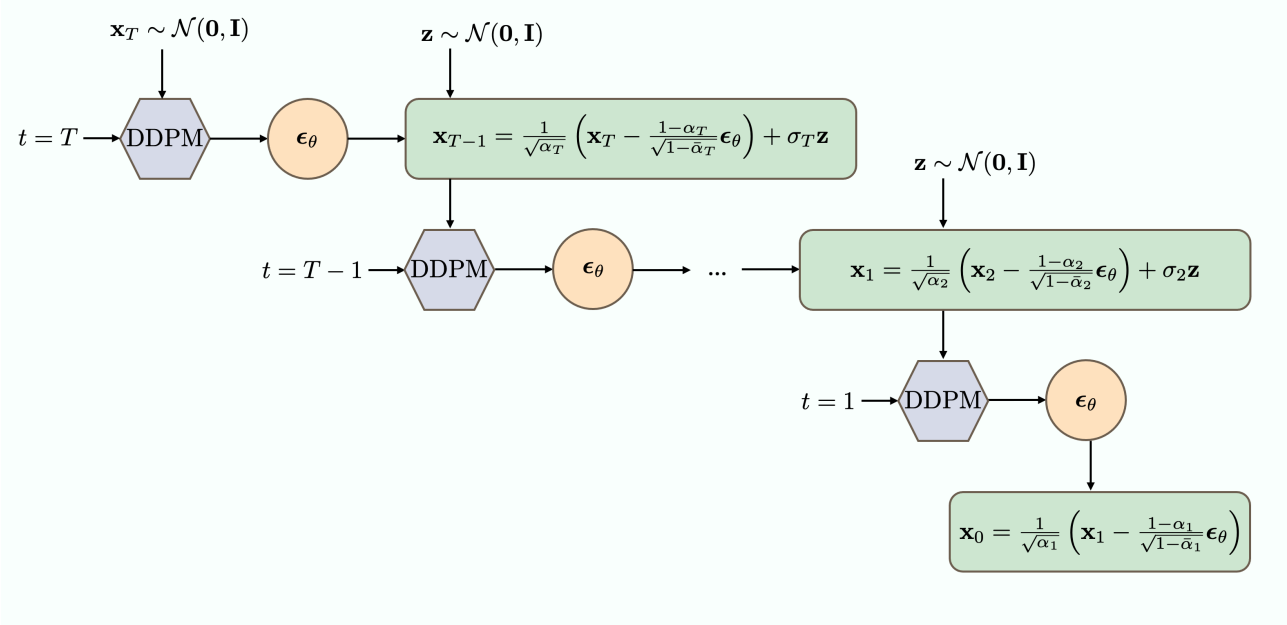

Figure 1.3: DDPM sampling [10].

Denoising Diffusion Implict Models The DDPM sampling process can be accelerated by replacing the Markovian diffusion process with a non-Markovian alternative resulting in denoising diffusion implicit models (DDIMs) [71]. Unlike DDPM, DDIM defines the joint distribution as

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = q(\mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=2}^{T} q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \tag{1.55}$$

It becomes clear that the forward process $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{x}_0)$ is not Markovian anymore due to its explicit dependence on $\mathbf{x}_0$. The authors of [71] parameterize the reverse process

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \tilde{\sigma}_t^2} \cdot \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t} \mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \tilde{\sigma}_t^2 \cdot \mathbf{I} \right) \tag{1.56}$$

The corresponding sampling equation becomes

$$\mathbf{x}_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\cdot\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)}_{\text{Predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\tilde{\sigma}_t^2}\cdot\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}_{\text{Direction of } \mathbf{x}_t} + \underbrace{\tilde{\sigma}_t\cdot\mathbf{z}}_{\text{Noise}} \tag{1.57}$$

where $\tilde{\sigma}_t$ is a parameter to choose and does not necessarily follow a predefined variance schedule. $\mathbf{x}_{t-1}$ consists of three parts. The first part is a prediction of $\mathbf{x}_0$ based on $\boldsymbol{\epsilon}_\theta$. The second part is a directional term describing the transition between $\mathbf{x}_t$ and $\mathbf{x}_0$. And the last part adds Gaussian noise scaled by $\tilde{\sigma}_t$. The key advantage of DDIM is the flexibility in choosing $\tilde{\sigma}_t$, which does not need to follow a predefined variance schedule. Different values of $\tilde{\sigma}_t$ yield different diffusion processes while using the same trained model. In particular, for the choice

$$\sigma_t = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}\sqrt{1-\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}} \tag{1.58}$$

the DDIM approach becomes Markovian again and here, the deep connection between DDPM and DDIM can be seen. Another aspect to note is that if setting $\tilde{\sigma} = 0$ the whole process become deterministic.

Scored-Based Generative Models

This disscussion is mainly based on [14, 87]. Score-Based Generative Models (SGMs) [74, 75] do not learn directly the data distribution $p_{\text{data}}(\mathbf{x})$ but the score of the probability function $\nabla_x \log p_{\text{data}}(\mathbf{x})$. The score represents a vector field that points to the steepest increase in log probability density, thereby, guiding the data generation process to regions of higher data density. There are several approaches on how the score function can be learned [21, 76, 78]. Here, the one closest to the DDPM approach is presented [74]. During training, the data is perturbed by Gaussian noise at different noise levels and a neural network is trained to estimate the score function based on the noise level $s_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q(x_t)$. These networks are called Noise-Conditional Score Networks (NCSNs). The learning objective boils down to

$$\mathbb{E}_{T\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[\lambda(t)\sigma_t^2\left\|\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t) - s_\theta(\mathbf{x}_t,t)\right\|^2\right] \tag{1.59}$$

$$= \mathbb{E}_{T\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[\lambda(t)\sigma_t^2\left\|\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t|\mathbf{x}_0) - s_\theta(\mathbf{x}_t,t)\right\|^2\right] + \text{const} \tag{1.60}$$

$$= \mathbb{E}_{T\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[\lambda(t)\left\|-\frac{\mathbf{x}_t-\mathbf{x}_0}{\sigma_t} - \sigma_t s_\theta(\mathbf{x}_t,t)\right\|^2\right] + \text{const} \tag{1.61}$$

$$= \mathbb{E}_{T\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\lambda(t)\left\|\boldsymbol{\epsilon} + \sigma_t s_\theta(\mathbf{x}_t,t)\right\|^2\right] + \text{const} \tag{1.62}$$

In the final line in eq. 1.62 the relation to DDPM loss becomes visible by identifying $\boldsymbol{\epsilon}(\mathbf{x}_t, t) = -\sigma_t s_\theta(\mathbf{x}_t, t)$ [87], showing that learning to predict noise is equivalent to learning the score func-

tion.
For sampling a new element from the data distribution [74] proposed Annealed Langevin Dynamics (ALD) which produces samples by only using the score function being approximated by the learned neural network

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \alpha \nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1}) + \sqrt{2\alpha}\mathbf{z}_t, \quad 1 \leq t \leq T \tag{1.63}$$

with $\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and α regulates the step size of the update in the direction of the score. By applying eq. 1.63 iteratively to initial Gaussian noise, a new element of the data distribution is generated.

Stochastic Differential Equations

This discussion is mainly based on [14, 87]. Stochastic Differential Equations (SDEs) [78] are a generalization of SGMs and DDPMs to continuous time where the forward and the reverse process are modeled by the solution of stochastic differential equations. The forward process is described by the SDE

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \tag{1.64}$$

where $\mathbf{f}(\mathbf{x}, t)$ is the drift coefficient, $g(t)$ the diffusion coefficient and $\mathbf{w}$ a standard Wiener process (aka Brownian Motion). [3] proved that the diffusion process in eq. 1.64 can be reversed leading to the reversed-time SDE given by

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla_x \log q(\mathbf{x}_t)\right] dt + g(t)d\tilde{\mathbf{w}} \tag{1.65}$$

with a standard Wiener process $\tilde{\mathbf{w}}$ where the time flies backward. Both, forward and reverse SDE, share the same marginals. A score neural network s_θ is trained similarly to SGMs by generalizing the objective function (see eq. 1.59) to continuous time

$$\mathbb{E}_{t\sim\mathcal{U}[0,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)} \left[\lambda(t) \left\| s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)\right\|^2\right] \tag{1.66}$$

such as $s_\theta(\mathbf{x}, t) \approx \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)$. For converting noise to data, a solution of eq. 1.65 has to be computed using the score neural network and a SDE solver. Another finding of [78] is that for eq. 1.65 an ODE exists having the same marginal as the SDE being known as probability flow ODE in the literature. It's given by by the deterministic version of eq. 1.65

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla_x \log q(x_t)\right] dt \tag{1.67}$$

Both reversed-SDE or probability flow ODE can be used to generate new samples using SDE solvers [27, 78] or respectively ODE solvers [28, 46, 78, 91].

Flow Matching

The concept of flow matching was introduced for normalizing flows [42] to find transport trajectories between two probability distributions. In contrast to diffusion models, which follows a noisy trajectory flow matching directly learns a probability path $p(t, x)$ connecting two distributions π_1 and π_2. In particular, the corresponding vector field $u(t, x)$, often referred to as velocity, that generates the probability path is learned via the flow matching loss function

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t\sim\mathcal{U}(0,1=),x\sim p(t,x)}\left[\|v_\theta(t, x) - u(t, x)\|^2\right] \tag{1.68}$$

by a neural network $v_\theta(t, x)$. The physical analogy is that particles are transport from one location to another and the vector field specifies at every time and location in which direction and how fast the particles move, in order to reach a final state. However, in flow matching neither the probability path $p(t, x)$ nor the vector field $u(t, x)$ is directly accessible. Therefore, the marginal probability path is constructed by a mixture of simpler conditional probability paths $p(t, x \mid x_1)$ such as $p(0, x, \mid x_1) = p(0, x) \approx q(x)$ and $p(1, x \mid x_1) = \mathcal{N}(x; \mathbf{0}, \mathbf{I})$ with x_1 being sampled from the data distribution

$$p(x, t) = \int p(t, x \mid x_1)q(x_1)dx_1 \tag{1.69}$$

where $q(x)$ denotes the data distribution. Furthermore, a marginal vector field can be formulated

$$u(t, x) = \int u(t, x \mid x_1)\frac{p(t, x \mid x_1)q(x_1)}{p(t, x)}dx_1 \tag{1.70}$$

based on the conditional vector field $u(t, x \mid x_1)$ which generates the marginal probability path. The trick is not to use the marginal distributions, because the integrals are still intractable, instead reformulate the loss function (eq. 1.68) using the conditional vector field.

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1=),x\sim p(t,x)}\left[\|v_\theta(t, x) - u(t, x \mid x_1)\|^2\right] \tag{1.71}$$

The key insight is that $\mathcal{L}_{\text{CFM}}$ leads to the same gradients w.r.t θ as $\mathcal{L}_{\text{FM}}$ and can actually be computed due to the tractable conditional vector field.

Rectified Flow [44] proposed rectified flow, a specialized variant of flow matching [42] that aims to straighten the trajectories between source and target distributions π_0 and π_1 by iteratively refining the learned ODE. This approach offers significant computational advantages, specifically, the straightened paths create a more direct mapping between distributions, making the velocity field easier to learn and enabling efficient sampling with as few as 1-2 Euler integration steps, compared to the 50-1000 steps typically required by diffusion models.

The ODE of the rectified flow for $x_0 \sim \pi_0$ and $x_1 \sim \pi_1$ is given by

$$dx_t = v_\theta(x_t, t)dt \tag{1.72}$$

where v_θ is the vector field approximated by a neural network driving the flow such that it follows the direction $x_1 - x_0$. This is enforced by the learning objective

$$\min_v \int_0^1 \mathbb{E}\left[\|(x_1 - x_0) - v_\theta(x_t, t)\|^2\right] dt \tag{1.73}$$

where $x_t = tx_1 + (1-t)x_0$ is the linear interpolation between x_0 and x_1. After training, the ODE in eq. 1.72 can be solved by sampling x_0, thus obtaining a sample from the data distribution π_1. One main property of the flow obtained through this learning procedure is that trajectories of the rectified flow do not cross. This is depicted in fig. 1.4. (a) shows the linear interpolations from pairs of $(x_0, x_1) \sim \pi_0 \times \pi_1$. In this case, trajectories intersect. Next, in (b), the vector field is learned. The rectified flow induced by (x_0, x_1) results in rewired trajectories such that they do not cross. This procedure, obtaining the linear interpolation between the elements from the previously learned rectified flow x_0^{k-1}, x_1^{k-1} and learning the rectified flow x_t^k, can be iteratively applied and is called Reflow. The second iteration is depicted in (c) and (d). With every iteration the trajectory becomes more straightened (see algorithm 8). An optional step to enable fast inference is to learn the k-rectified flow by a neural network $\hat{T}$ which directly maps x_0^k to x_1^k. Note, the difference to rectified flow learning is that here, the network approximates the mapping between (x_0^k, x_1^k) whereas rectification yields a different mapping with lower transport cost and straightened trajectory (x_0^{k+1}, x_1^{k+1}).

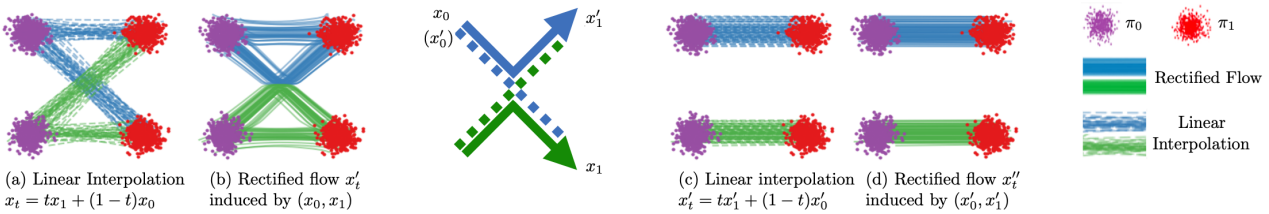

Figure 1.4: Rectified Flow [44].

Algorithm 3 Rectified Flow: Main Algorithm

Procedure: $x' = \text{RectFlow}((x_0, x_1))$
Inputs: Draws from a coupling (x_0, x_1) of π_0 and π_1; velocity model $v_\theta : \mathbb{R}^d \to \mathbb{R}^d$ with parameter θ.
Training: $\hat{\theta} = \arg\min_\theta \mathbb{E}\left[\|x_1 - x_0 - v(tx_1 + (1-t)x_0, t)\|^2\right]$, with $t \sim \text{Uniform}([0,1])$.
Sampling: Draw (x'_0, x'_1) following $dx_t = v_{\hat{\theta}}(x_t, t)dt$ starting from $x'_0 \sim \pi_0$ (or backwardly $x'_1 \sim \pi_1$).
Return: $Z = \{Z_t : t \in [0,1]\}$.

Reflow (optional): $x^{k+1} = \text{RectFlow}((x_0^k, x_1^k))$, starting from $(x_0^0, x_1^0) = (x_0, x_1)$.
Distill (optional): Learn a neural network $\hat{T}$ to distill the k-rectified flow, such that $x_1^k \approx \hat{T}(x_0^k)$.

1.3 Fundamentals of Neural Network Architectures

In this section, the focus lies on the key fundamental building blocks of the neural networks which are used in the experiments in section **??**.

1.3.1 Transformer

The introduction of transformer architecture [86] was a decisive moment for the whole AI community. It significantly pushed the development in natural language and image processing forward. Therefore, it is no surprise that the latest models are based on this architecture type [2, 4, 36, 82].

The transformer was introduced as an alternative to recurrent neural networks [63] for processing sequential data like text. In the original paper, the transformer architecture leverages an encoder-decoder structure, however, there are many variants like decoder-only [1, 85] or encoder-only transformer [16].

Transformers, as introduced in the original paper (Fig. 1.5), take text as input, which must first be converted into numerical form through a process called tokenization. This produces a sequence of tokens that are a numerical representation of words or subwords of the text. To provide the model with information about the position of each token in the sequence, positional encodings are added. These preprocessing steps occur before the input is fed into the transformer architecture. Each encoder block consists of two sub-layers: (1) a multi-head self-attention mechanism followed by a residual connection [22] and layer normalization [6]. The decoder blocks contain three sub-layers: (1) a masked multi-head self-attention where the masking prevents that future tokens are seen during the training process, (2) a cross-attention mechanism with the output of the encoder being and (3) an additional feedforward network. All three sub-layers are followed by a residual connection and layer normalization. The final output is projected to vocabulary-sized logits representing a probability distribution over possible next tokens.

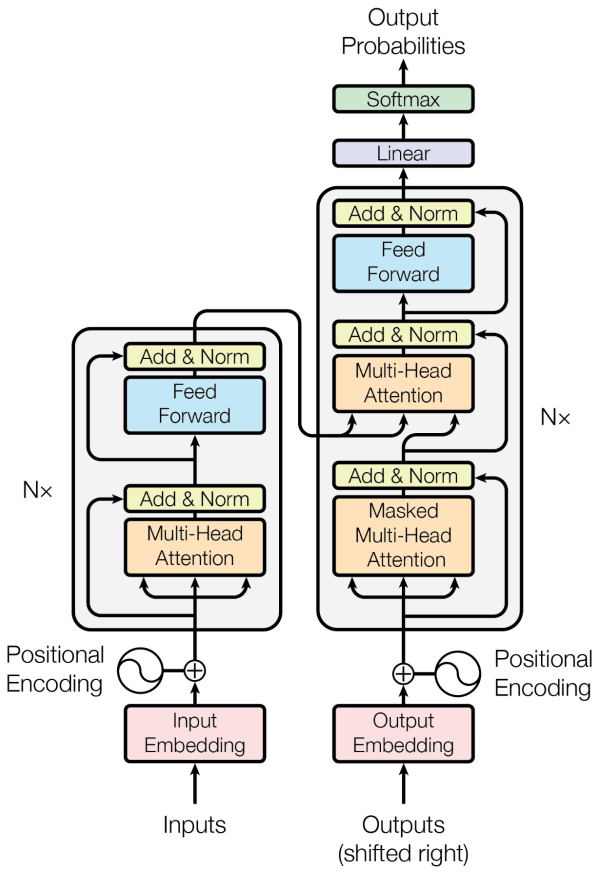

Figure 1.5: Transformer architecture from [86].

1.3.2 Attention Mechanism

The attention mechanism is an integral part of the transformer architecture. It enables the model to focus on the input elements that are most important for the task at hand and ignore the less important ones. This mechanism was first introduced for RNNs [7] to overcome the information bottleneck, as traditionally in RNNs only the last hidden state is given as input for the next prediction. In [7], all previous hidden states were presented to the model as additional input, and the task of the attention mechanism was to filter out those that were relevant for the prediction of the next state. The authors of [86] leveraged the formalism and reformulated it for the transformer architecture that will be discussed in the following.

In the attention mechanism, so-called queries, keys and values are computed. The attention mechanism can be understood as a soft lookup table where queries search for relevant keys, and the corresponding values are retrieved and weighted by similarity. A distinction is made between self-attention (fig. 1.6) and cross-attention (fig. 1.7). In self-attention, the interaction of the tokens within a sequence is calculated. In contrast, in cross-attention you have two input sequences with the goal that the model aligns and relates information between the two different sequences.

Self-Attention

Let $X \in \mathbb{R}^{n \times d}$ be the embedding vector of the tokens. Then the queries Q, keys K and values V are computed by linearly projection of the embedding vector

$$Q = X \cdot W_q^T \tag{1.74}$$

$$K = X \cdot W_k^T \tag{1.75}$$

$$V = X \cdot W_v^T \tag{1.76}$$

with $W_q^T \in \mathbb{R}^{d \times d_k}$, $W_k^T \in \mathbb{R}^{d \times d_k}$ and $W_v^T \in \mathbb{R}^{d \times d_v}$. Now, so-called scaled-dot product attention is used where the attention matrix A that assigns how much focus is put on the different parts of the token sequence is computed by the softmax of the scaled dot product between the queries and the keys

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{1.77}$$

The softmax function is defined as the following for a vector $\mathbf{z} = (z_1, z_2, ..., z_N)$

$$\text{softmax}\,(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1.78}$$

The scaling of the dot product QK^T with the factor $\frac{1}{\sqrt{d_k}}$ is done in order to prevent vanishing gradients in the softmax function which occurs if the values become too small or too big. Finally, the attention matrix is applied with the values. This can be summarized in the following formulation

$$Z = \text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{1.79}$$

One main challenge of the discussed attention mechanism is that it has quadratic complexity $\mathcal{O}(n^2)$ in computation and memory consumption which is tackled by several approaches [12, 13, 15, 33].

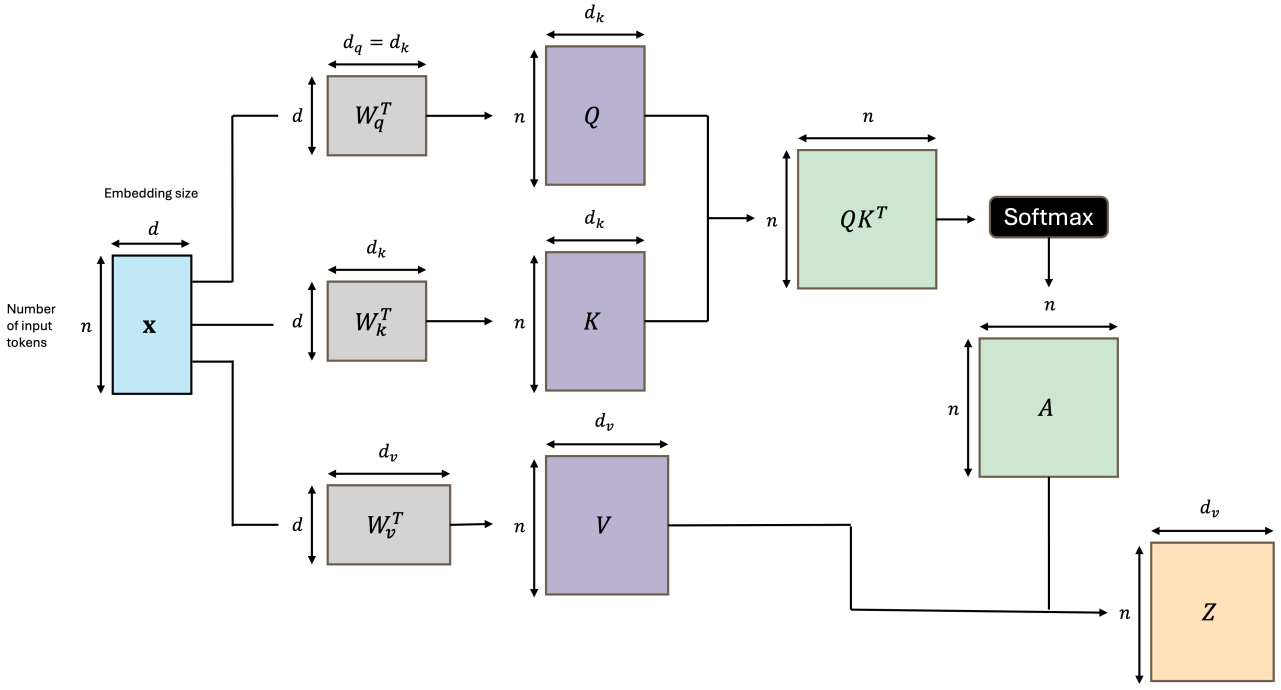

Figure 1.6: Self-attention mechanism inspired from [58].

Cross-Attention

The goal of cross-attention is to relate two different sequences of tokens $\mathbf{x}_1$ and $\mathbf{x}_2$. Commonly, the queries come from the target sequence ,e.g., decoder of the transformer, while the keys and values come from the source sequence, e.g., encoder output of transformer. Besides that the computation is similar to the computation of self-attention.

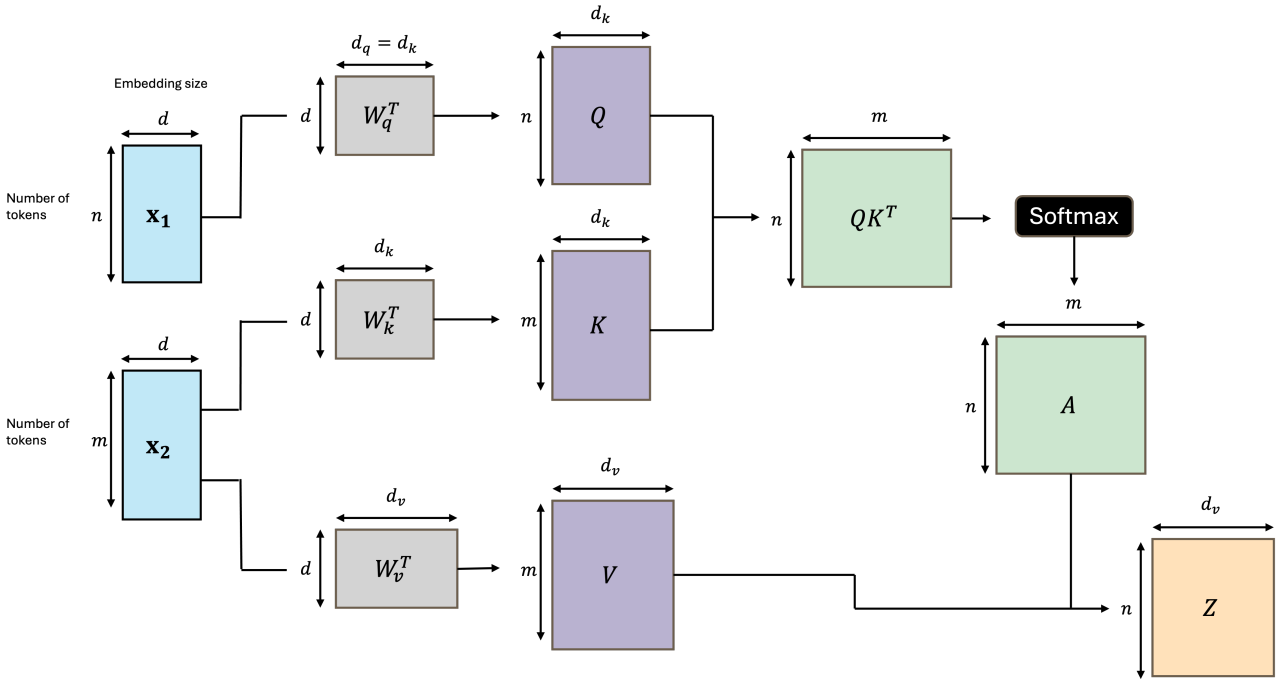

Figure 1.7: Cross-attention mechanism inspired from [58].

Multi-Head Attention

Multi-head attention is another trick used in transformers to make the results of the attention operations even more expressive. Instead of only performing individual self- or cross-attentions in a transformer block, so-called multi-head attentions are used. Here, based on the input embeddings, not just one but N times keys, queries and values are calculated, each with different weight matrices $W_{q_i}^T \in \mathbb{R}^{d \times d_k}$, $W_{k_i}^T \in \mathbb{R}^{d \times d_k}$ and $W_{v_i}^T \in \mathbb{R}^{d \times d_v}$ with $i = 1, ..., N$, and the attention mechanism is executed. Each attention that is carried out is designated as a head_i. The results of the N attentions are concatenated and combined with another linear projection $W^O \in \mathbb{R}^{Nd_v \times d}$.

$$\text{MultiHead}\left(Q, K, V\right) = \text{Concat}\left(\text{head}_1, ..., \text{head}_N\right) W^O \tag{1.80}$$

$$\text{head}_i = \text{Attention}\left(XW_{q_i}^T, XW_{k_i}^T, XW_{v_i}^T\right) \tag{1.81}$$

This is advantageous because the individual heads can concentrate on different tasks, thus reducing the complexity for each head compared to the case where only a single attention has to capture everything.

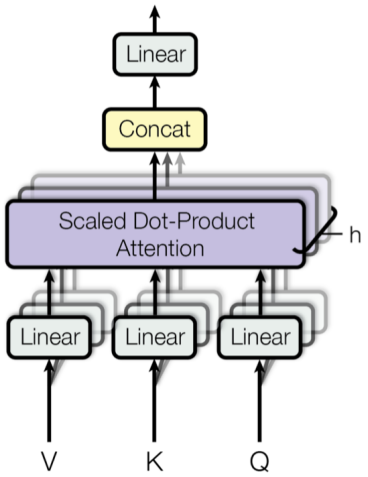

Figure 1.8: Multi-head attention from [86].

1.3.3 Positional Encoding

Before the embedded token sequence is fed as input to the transformer, a positional information is added to each token to provide the network with information about the order of the tokens. There are different methods, which can be divided into the categories relative [38, 51, 81] vs absolute [34, 39, 45, 86] positional embedding and deterministic [34, 39, 51, 81, 86] versus learnable [38, 45] positional embedding.

Sinusoidal Positional Encoding

Sinusoidal positional encoding is used in the original transformer paper [86] which leverages the sin and cosine function to inject positional information into the model. It belongs to the category of absolute and deterministic positional encoding.

$$PE\left(pos, 2i\right) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{1.82}$$

$$PE\left(pos, 2i+1\right) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{1.83}$$

where d is the embedding dimension, pos is the position of the token in the sequence and i is the dimension index which is used to compute different frequencies for different parts of the embedding vector. The computed positional embedding has the same dimension like the token embedding vector and is be added to it.

Rotary Position Embedding

In recent years, the focus shifted more and more from absolute positional embedding to relative positional embedding because one hypothesized that relative order of tokens matter and that these methods are better in extrapolating compared to absolute positional embedding methods [93]. Rotary Positiona Embeeding (RoPE) [81] is a a relative positional embedding method used in FLUX-dev [36]. The idea is to encode the position via rotation in the attention mechanism, to be more explicit in the dot product between the queries Q and the keys K. Starting with the 2d case, they propose that queries Q_m which are soley based on the mth embedding vector x_m and its position m via

$$Q_m\left(x_m, m\right) = \begin{bmatrix}\cos\left(m\theta\right) & -\sin\left(m\theta\right) \\ \sin\left(m\theta\right) & \cos\left(m\theta\right)\end{bmatrix} \begin{bmatrix}W_q^{11} & W_q^{12} \\ W_q^{21} & W_q^{22}\end{bmatrix} \begin{bmatrix}x_m^1 \\ x_m^2\end{bmatrix} \tag{1.84}$$

$$= R_{\theta,m}^d W_q \mathbf{x_m} \tag{1.85}$$

with a preset non-zero constant θ. **Hier wurde die andere Konvention zum Schreiben des Attention Mechanismus verwendet als oben -> vereinheitlichen???** The keys K_m are computed in exactly the same way. A visualization can be found in fig. 1.9. In the multi-dimensional case the rotation matrix becomes

$$R^d_{\theta,m} = \begin{bmatrix} \cos(m\theta_1) & -\sin(m\theta_1) & \cdots & 0 & 0 \\ \sin(m\theta_1) & \cos(m\theta_1) & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cos\left(m\theta_{d/2}\right) & -\sin\left(m\theta_{d/2}\right) \\ 0 & 0 & \cdots & \sin\left(m\theta_{d/2}\right) & \cos\left(m\theta_{d/2}\right) \end{bmatrix} \tag{1.86}$$

By computing the dot product between the queries and the keys one obtain a single rotation matrix just depending on the relative positions

$$Q_m^T K_n = \left(R^d_{\theta,m} W_q \mathbf{x_m}\right)^T R^d_{\theta,n} W_k \mathbf{x_n} = \mathbf{x_m^T} W_q^T R^d_{\theta,n-m} W_k \mathbf{x_n} \tag{1.87}$$

because $R^d_{\theta,n-m} = (R^d_{\theta,m})^T R^d_{\theta,n}$. In contrast to sinusoidal positional encoding being additive, rotary positional encoding is multiplicative. Moreover, it has the property of long-term decay that means the further away two embeddings are the smaller the dot product of the keys and queries will be.

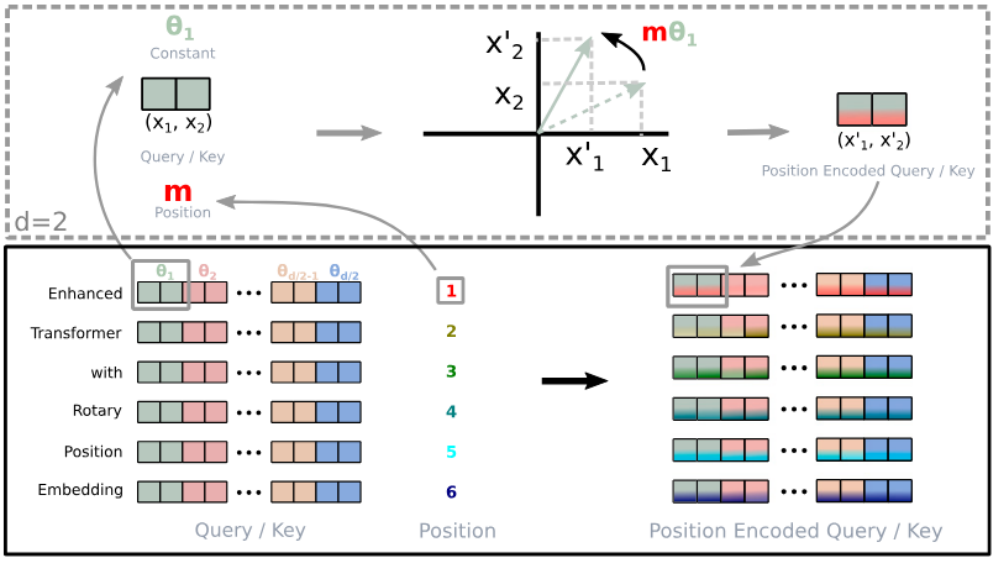

Figure 1.9: Scheme of implementation of rotational position encoding from [81].

1.3.4 Vision Transformer

Original transformers were introduced for language processing tasks. There were some attempts to transfer the methodology of the transformer architecture to computer vision problems [26, 52, 56], but for a long time, convolutional neural networks (CNNs) [57] remained the first choice. CNNs have an inherent advantage over transformers because they possess an image-related inductive bias. On the one hand, CNNs are translationally equivariant by con-

struction and, on the other hand, the concept of locality is embedded in the way they are built. In contrast, these two concepts must be learned from scratch by a transformer, as they are not predefined by the architecture, which makes a sufficiently large dataset essential to achieve competitive performance.

The introduction of vision transformer (ViT) [18] showed that given a sufficiently large dataset transformers can match or even surpass CNNs for vision tasks. The first key aspect in ViT is that not every single pixel is related to all other pixels because this would be computationally infeasible due to the quadratic complexity of the attention mechanism. Instead, the image is split $\mathbf{x} \in R^{N \times W \times C}$ into a sequence of smaller patches $\mathbf{x_p} \in R^{N \times (P^2 \cdot C)}$ where C is the number of channels, H the height of the image, W the width of the image, (P, P) the resolution of a patch and $N = \frac{HW}{P^2}$ the number of patches. At the beginning of the token sequence an additional classification (CLS) token is prepended. The patches are flattened and linearly projected to the encoder dimension. After that a learned positional encoding is added before they are fed into the encoder-only transformer architecture. On top of the transformer encoder a MLP head processes the embedded version of the classification token and provides probability for the different classes from the classification problem.

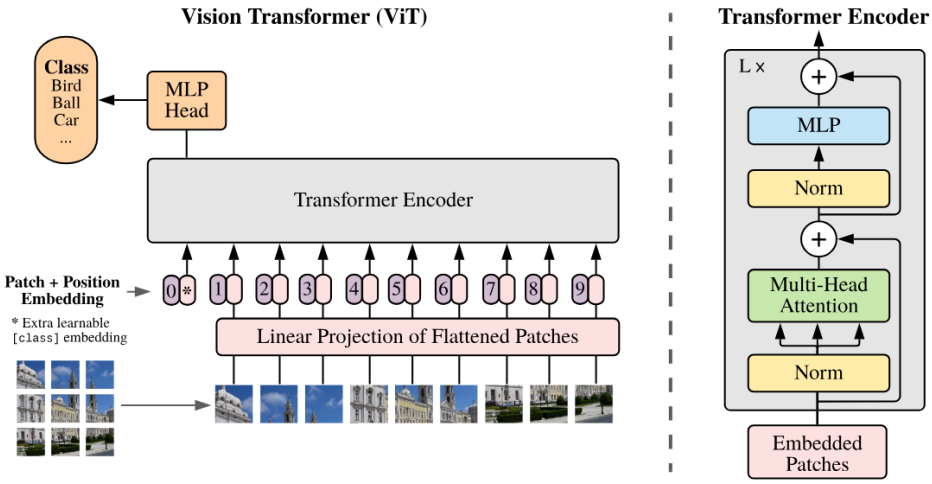

Figure 1.10: Vision transformer from [18].

1.4 Diffusion Model Distillation

Diffusion models excel in generating high-quality and diverse images, showcasing remarkable capabilities in image synthesis. However, their main disadvantage is their long inference time. This stems from two factors: First, diffusion models have an iterative generation process removing noise step-by-step. To obtain high-quality images often 50-100 steps are required. Second, recent diffusion models like Flux [36] or HiDream [2] are very large, with billions of parameters, making each step computationally and memory intensive. These issues limit their use for real-time applications or in memory constraint environments such as edge devices.

Diffusion model distillation addresses these challenges. The principal idea of all distillation methods is to transfer the knowledge and image generation capacity from a fully trained diffusion model often referred to as teacher model to a target model often referred to as student model. The research landscape reveals two complementary strategies: accelerating inference by reducing the number of denoising steps, and creating more compact models through model size reduction. The first strategy, reducing number of inference step, is more dominant in the literature and several different approaches such as adversarial distillation [66, 68], progressive distillation [40, 65], distribution matching distillation [88, 89], consistency models [47, 72] and guided distillation [48]. The second strategy was explored via knowledge Distillation being very suitable for reducing the model size which was applied to stable diffusion [30] and stable diffusion XL [37, 90].

1.4.1 Distillation Methods for Few-Step Diffusion Models

In the following various approaches are presented for training few step diffusion models. Each method employs a pre-trained teacher model to guide the distillation of a student model capable of generating images in significantly fewer steps.

Adversarial Diffusion Model Distillation

Adversarial Diffusion Distillation (ADD) [68] adopts a training setup similar to GANs. A discriminator is trained to distinguish whether an image comes from the real dataset or was generated by the student model. This component uses the standard MinMax loss, as commonly used in GAN training.

In addition to the adversarial loss, a second learning signal is introduced by comparing images generated by the student model that is designed to perform generation in a few (or even a single) denoising steps, with those produced by a fully sampled teacher diffusion model. The full training procedure is illustrated in Fig. 1.11, where Stable Diffusion is used as the teacher model.

As shown in the upper diagram of fig. 1.11, the input image is first encoded into the latent

space, and noise is added. For example, if the student is trained to operate in 4 steps, one of four noise levels (e.g., steps 999, 749, 499, 249) is randomly selected. The student model then performs a single denoising step.

The student's output serves two roles. First, it is passed to the discriminator to compute the adversarial loss $\mathcal{L}_{\text{adv}}$. The discriminator consists of a frozen backbone, DinoV2 [50], and learnable heads. Second, noise is added again before feeding it into the frozen teacher model. After the teacher performs full denoising, its output is compared to the student's original prediction using an MSE loss $\mathcal{L}_{\text{dis}}$. The overall training objective combines both the adversarial and distillation losses with a weighting constant λ

$$\mathcal{L} = \mathcal{L}_{\text{adv}} + \lambda \mathcal{L}_{\text{dis}} \tag{1.88}$$

A famous model that has been obtained by employing adversarial diffusion distillation is SDXL-Turbo [68]. Despite it success, the proposed distillation method comes with two decisive shortcomings. First, the use of DinoV2 as a discriminator backbone limits the size of the generated images to 512×512, since one must not deviate too much from the image size on which DinoV2 was trained in order to retain the expressiveness of DinoV2's features. Second, the encoding and decoding of the images into the latent space and back is computationally expensive. Ideally, distillation would be directly performed in the latent space. This insight let to latent adversarial diffusion distillation (LADD) [66]. The first key difference is the replacement of real images from a dataset with synthetic images generated by the teacher model prior to training. To be more precise, only the latents not the images of the teacher model needs to be stored because the whole distillation process takes place in the latent space. Second, the role of the discriminator was taken over by the teacher model itself. Third, only the adversarial loss was employed, as adding the distillation loss does not show empirical benefits. The full training scheme is illustrated in the lower diagram of fig. 1.11. Initially, noise is added to the pre-generated latents of the teacher. Afterwards the student model is applied to predict denoised latents. To compute the MinMax objective noise is added to the student generated latents before it is given as input to the discriminator consisting of the frozen teacher model and trainable heads. As "real" latents the noised pre-generated latents of the teacher model are used.
In summary, LADD is a simplified and more compute efficient version of ADD maintaining the main idea of using adversarial distillation.

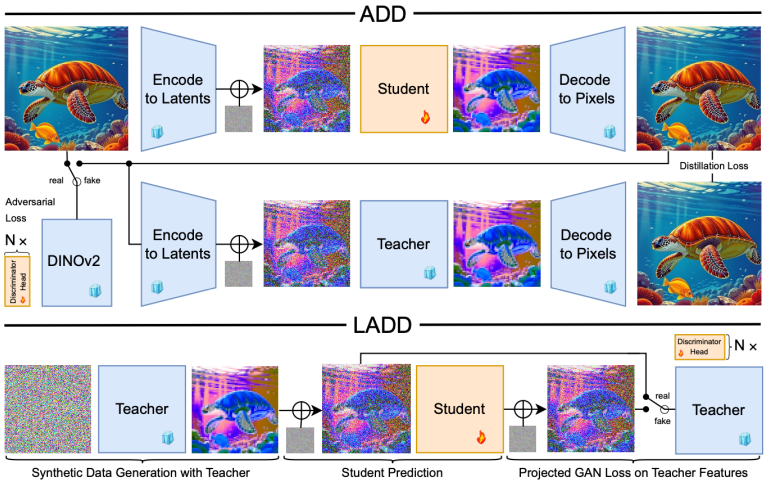

Figure 1.11: (Latent) adversarial diffusion distillation from [66].

Progressive Distillation

In contrast to ADD, where a single training run is used to create a model that requires only very few inference steps to generate a high-quality image, the idea of progressive distillation [65] is to iteratively train models that require fewer and fewer inference steps, thereby reducing the complexity of the task within one iteration. Assuming a diffusion model requires N steps during inference, then in the first iteration, a student model, initialized with the weights of the teacher model, is trained to generate images with only $\frac{N}{2}$ steps. In the second iteration, the previously trained student model is used as the teacher and another student model is trained, needing only $\frac{N}{4}$ inference steps. This procedure can be repeated until a one-step model is reached. The objective function is similar to that used in training of a diffusion model. Let $\hat{f}$ be the teacher model and f the student model. During training, a noisy input $\mathbf{z}_t$ is sampled from the training dataset. The student model is applied exactly once, $\mathbf{z}_{\text{pred}} = f(\mathbf{z}_t)$ while the teacher model is applied twice, $\hat{\mathbf{z}} = \hat{f} \circ \hat{f}\,(\mathbf{z}_t)$. In the objective function, the prediction of the teacher $\hat{\mathbf{z}}$ is compared with the prediction of the student $\mathbf{z}_{\text{pred}}$, e.g, via MSE. This core idea is shown in fig. 1.12. A possible downside of this method is that progressive distillation carries the risk of error accumulation, as in each iteration the previous student is used as a teacher, therefore especially two or one step models show a downgrade in performance. Furthermore, the iterative approach is resource intensive, as many trainings have to be performed.
Stable Diffusion XL lightning [40] is a distilled version of Stable Diffusion XL (SDXL) where, a combination of progressive and adversarial distillation is applied. First, the authors trained a version of SDXL with 32 inference steps using only progressive distillation. Afterwards, they additionally applied adversarial distillation to go further down to a few step-model.

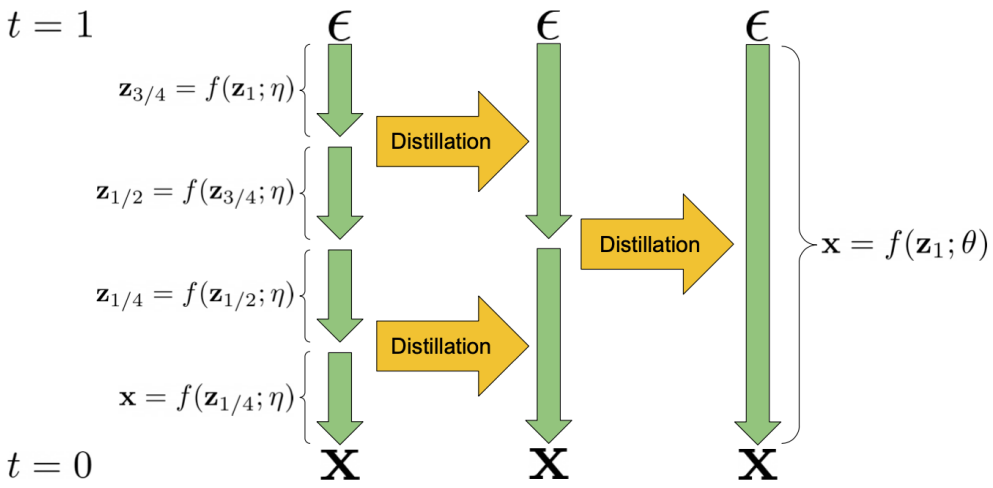

Figure 1.12: Progressive distillation from [65].

Consistency Model

Consistency models [72] have been designed to map images with different noise levels to their corresponding clean image in one step. To achieve this, during training the self-consistency property is enforced for the model meaning that two noisy images lying on the same denoising path are mapped to the same clean image. This self-consistency property is the foundation of the training procedures which enables one-step image generation.

Let $\{x_t\}_{t=\epsilon}^{T}$ be a denoising trajectory where x_ϵ is the clean image and x_T is pure noise. A model f_θ has the self-consistency property if for two images x_t and $x_{t'}$ with $t, t^{'} \in [\epsilon, T]$ which are on the same denoising trajectory $f_\theta(x_t, t) = f_\theta(x_{t'}, t')$ holds (see fig. 1.13). In addition, a consistency model has to fulfill the boundary condition $f_\theta(x_\epsilon, \epsilon) = x_\epsilon$. This can be enforced by parameterize the model as

$$f_\theta(x, t) = c_{\text{skip}}(t)x + c_{\text{out}}(t)F_\theta(x) \tag{1.89}$$

where F_θ is a neural network and $c_{\text{skip}}(t)$ and $c_{\text{out}}(t)$ are differentiable function such as $c_{\text{skip}}(\epsilon) = 1$ and $c_{\text{out}}(\epsilon) = 0$. During the training process, noise is first added to the image from the training dataset in a similar way to training a standard diffusion model obtaining a noisy image x_{t+1}. Next, the teacher model is applied once to obtain a slightly denoised image x_t^ϕ where ϕ denotes the fixed parameters of the teacher model. It is important to note that both x_{t+1} and x_t^ϕ lie on the same denoising path and accodingly to the self-consistency property $f_\theta(x_{t+1}, t+1) = f_\theta(x_t^\phi, t)$ should hold. Therefore, the training objective is given by

$$\mathcal{L}_{\text{CD}}\left(\theta, \theta^-; \phi\right) = \mathbb{E}\left[\lambda(t) d\left(f_\theta(x_{t+1}, t+1), f_{\theta^-}(x_t^\phi, t)\right)\right] \tag{1.90}$$

where θ denotes the parameters of the student consistency model, θ^- denotes a running exponential moving average (EMA) of the past θ values and d stands for the MSE. The EMA is used

for training stability reasons. With this training framework one step models can be obtained. But also multi-step inference can be accomplished with this model if higher image quality is needed. It is worth noting that distillation is only one way to train consistency models. Another way is to train them from scratch. How this could be achieved is described in [72] but out of scope of this short summary. Furthermore, successive works show that this concept can also be extended to latent space. [47]. However, some literature [66] claims that consistency models requires extensive hyperparameter tuning and engeneering in order to work.

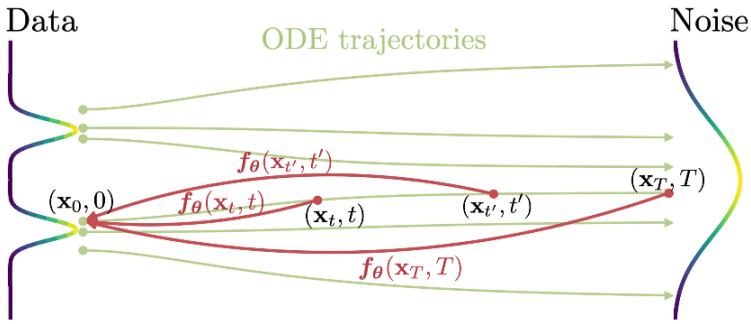

Figure 1.13: Self-consistency property taken from [72].

Distribution Matching Distillation

In contrast to other distillation approaches, distribution matching distillation [89] does not regress individual samples, but instead focuses on matching the whole image distribution. Concretely, it optimizes the student model to produce samples that collectively match the statistical porperties of real data and not the output of the teacher on specific noise-image pairs. To do so, two diffusion models and an additional noise-image dataset used for regularization is required (see fig. 1.14). The distilled student model f_θ is initialized with the weights of a pretrained teacher model $\hat{f}_\phi$. The learning objective is to minimize the KL-divergence between the *real* data distribution p_{real} and the *fake* data distribution p_{fake} produced from the distilled model f_θ

$$\mathrm{KL}[p_{\text{real}} \,\|\, p_{\text{fake}}] = \mathbb{E}_{\substack{z\sim\mathcal{N}(0;\mathbf{I}) \\ x=f_\theta(z)}} \left[-\left(\log p_{\text{real}}(x) - \log p_{\text{fake}}(x)\right)\right] \tag{1.91}$$

Although this KL-divergence is not tractable, it can be used for optimizing the student model because the goal is not to evaluate the KL divergence itself, but to compute its gradient, which can be expressed as

$$\nabla_\theta \mathrm{KL}[p_{\text{real}} \,\|\, p_{\text{fake}}] = \mathbb{E}_{\substack{z\sim\mathcal{N}(0;\mathbf{I}) \\ x=f_\theta(z)}} \left[-\left(s_{\text{real}}(x) - s_{\text{fake}}(x)\right) \frac{df}{d\theta} \right] \tag{1.92}$$

with $s_{\text{real}}(x) = \nabla_x \log p_{\text{real}}(x)$ and $s_{\text{fake}}(x) = \nabla_x \log p_{\text{fake}}(x)$ being the scores of the real and fake distributions. To prevent diverging scores, the distributions are perturbed with Gaussian noise, which ensures that the gradients are well-defined. The score functions $s_{\text{real}}(x)$ and $s_{\text{fake}}(x)$

are modeled by two diffusion models $\hat{f}_{\text{fake}}$ and $\hat{f}_{\text{real}}$, both initialized with pretrained weights of the teacher model. The weight of $\hat{f}_{\text{real}}$ are kept fixed throughout the whole training process, whereas $\hat{f}_{\text{fake}}$ is trained jointly with the distilled student model which is necessary because the fake distribution evolves over the course of training. The score and the diffusion models are connected via [79]

$$s_{\text{real}}(x_t, t) = -\frac{x_t - \alpha_t \hat{f}_{\text{real}}(x_t, t)}{\sigma_t^2} \tag{1.93}$$

with α_t being the scaling factor of the diffusion process, and similarly for s_{fake}. In addition, to stabilize training and prevent mode collapse, an additional regression loss $\mathcal{L}_{\text{reg}}$ is applied, in which the generated image of the student model based on the noise from a noise-image pair is compared via Learned Perceptual Image Patch Similarity (LPIPS) to the corresponding reference image. [92] such that the total loss is given by

$$\mathcal{L} = \text{KL} + \lambda_{\text{reg}} \mathcal{L}_{\text{reg}} \tag{1.94}$$

In total, this method provide a good training framework for one-step diffusion models. However, one disadvantage is that the method is very computationally intensive due to three networks being involved during the training process and the pregeneration of the noise-image pairs. [88] removes the necessity of the noise-image pair dataset and introduces additionally a GAN loss for stabilizing the training.

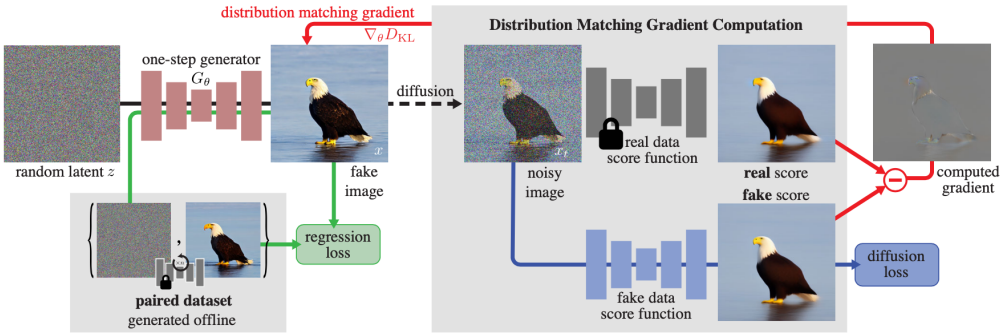

Figure 1.14: First, the regression loss for training the distilled model is computed using noise-image image pairs being computed in advance. Second, the two diffusion models are used to approximate the score functions for the *real* and *fake* distribution. The diffuison model used to approximate the fake score is updated too during the training process. Figure taken from [89].

1.4.2 Distillation-Based Model Compression

The second goal of distillation, besides reducing the number of inference steps, is to distill the knowledge and capabilities of large models into smaller ones. Hereby, knowledge distillation [23] is widely used [30, 37, 90].

Knowledge Distillation

The concept of knowledge distillation was introduced for training a small model referred to as the student - on classification tasks, using a larger model - referred to as the teacher with strong generalization capabilities as guidance. It was shown, that computing the loss between the probability outputs of the teacher and the student enables much more efficient training than using the original class labels. In such a setting the small model could be trained on less data and with a higher learning rate. The authors hypothesized that even the very small probabilities the teacher assigns to incorrect classes, and particularly their relative magnitudes, carry valuable information for the student For example if a 2 of the MNIST dataset [] should be classified it is valuable information for the model whether it resembles a 3 or a 5. When only hard targets, e.g., one-hot labels, are used, this relational information is lost. Here, it is important to make this information accessible to the student meaning that e.g. in a classification problem one might have to deviate from the standard temperature T of the softmax function to soften the probabilities even more. Specifically, both teacher and student use temperature scaling $\text{softmax}(\frac{z_i}{T})$ where $T > 1$ makes the probability distribution softer, revealing more information about class similarities. In general, it has been shown that a combination of the loss $\mathcal{L}_{\text{KD}}$, calculated with the outputs from the teacher, and the original loss $\mathcal{L}_{\text{Task}}$, calculated with the hard labels, is advantageous

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{KD}}\mathcal{L}_{\text{KD}} \tag{1.95}$$

This idea was used to train a smaller version of Stable Diffusion [16] or Stable Diffusion XL [37, 90]. Hereby, the original model with its pretrained weights is the teacher and the student model was obtained by removing individual parts of the Unet [62] architecture. The general idea in all approaches is to use the intermediate features from the Unet architecture as well as the final output as learning signal from the teacher to the student. Additionally, the original diffusion loss $\mathcal{L}_{\text{Task}}$ taken into account too such that the final loss is given by

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{OutKD}}\mathcal{L}_{\text{OutKD}} + \lambda_{\text{FeatKD}}\mathcal{L}_{\text{FeatKD}} \tag{1.96}$$

where $\mathcal{L}_{\text{Task}}$ is the loss used for regular training of diffusion models, $\mathcal{L}_{\text{FeatKD}}$ is the loss computed between the intermediate features and $\mathcal{L}_{\text{OutKD}}$ is the loss computed between the final output of the student and the teacher model.

Chapter 2

Results

sdfsdfsd

2.1 Result Distillation

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, and Shyamal Anadkat. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Vivago AI. Hidream-l1. 2025. URL: `https://vivago.ai/home`.

[3] Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

[4] Anthropic. Claude 3 haiku: our fastest model yet. 2024. URL: `https://www.anthropic.com/news/claude-3-haiku`.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[8] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[9] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[10] Anja Butter, Nathan Huetsch, Sofia Palacios Schweitzer, Tilman Plehn, Peter Sorrenson, and Jonas Spinner. Jet diffusion versus jetgpt–modern networks for the lhc. *SciPost Physics Core*, 8(1):026, 2025.

[11] Junsong Chen, Chongjian Ge, Enze Xie, Yue Wu, Lewei Yao, Xiaozhe Ren, Zhongdao Wang, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *European Conference on Computer Vision*, pages 74–91. Springer.

[12] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[13] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, and Lukasz Kaiser. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[14] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(9):10850–10869, 2023.

[15] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

[17] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[18] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, and Sylvain Gelly. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[19] William Feller. On the theory of stochastic processes, with particular reference to applications. In *Proceedings of the First Berkeley Symposium on Mathematical Statistics and Probability*, pages 403–432, Berkeley, CA, 1949. The Regents of the University of California.

[20] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[21] Jackson Gorham and Lester Mackey. Measuring sample quality with kernels. In *International Conference on Machine Learning*, pages 1292–1301. PMLR.

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

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[25] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, June 01, 2020 2020. URL: `https://ui.adsabs.harvard.edu/abs/2020arXiv200611239H`, `doi:10.48550/arXiv.2006.11239`.

[26] Zilong Huang, Xinggang Wang, Lichao Huang, Chang Huang, Yunchao Wei, and Wenyu Liu. Ccnet: Criss-cross attention for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 603–612.

[27] Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *arXiv preprint arXiv:2105.14080*, 2021.

[28] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.

[29] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in neural information processing systems*, 34:852–863, 2021.

[30] Bo-Kyeong Kim, Hyoung-Kyu Song, Thibault Castells, and Shinkook Choi. Bk-sdm: A lightweight, fast, and cheap version of stable diffusion. In *European Conference on Computer Vision*, pages 381–399. Springer.

[31] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

[32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[33] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[34] Shun Kiyono, Sosuke Kobayashi, Jun Suzuki, and Kentaro Inui. Shape: Shifted absolute position embedding for transformers. *arXiv preprint arXiv:2109.05644*, 2021.

[35] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[36] Black Forest Labs. Flux model. 2024. URL: `https://bfl.ai`.

[37] Youngwan Lee, Kwanyong Park, Yoorhim Cho, Yong-Ju Lee, and Sung Ju Hwang. Koala: Empirical lessons toward memory-efficient and fast diffusion models for text-to-image synthesis. *Advances in Neural Information Processing Systems*, 37:51597–51633, 2024.

[38] Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. Functional interpolation for relative positions improves long context transformers. *arXiv preprint arXiv:2310.04418*, 2023.

[39] Tatiana Likhomanenko, Qiantong Xu, Gabriel Synnaeve, Ronan Collobert, and Alex Rogozhnikov. Cape: Encoding relative positions with continuous augmented positional embeddings. *Advances in Neural Information Processing Systems*, 34:16079–16092, 2021.

[40] Shanchuan Lin, Anran Wang, and Xiao Yang. Sdxl-lightning: Progressive adversarial diffusion distillation. *arXiv preprint arXiv:2402.13929*, 2024.

[41] Zinan Lin, Ashish Khetan, Giulia Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. *Advances in neural information processing systems*, 31, 2018.

[42] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[43] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. Towards faster and stabilized gan training for high-fidelity few-shot image synthesis. In *iclr*.

[44] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

[45] Xuanqing Liu, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. Learning to encode position for transformer with continuous dynamical model. In *International conference on machine learning*, pages 6327–6335. PMLR.

[46] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.

[47] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

[48] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306.

[49] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR.

[50] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, and Alaaeldin El-Nouby. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[51] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

[52] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR.

[53] Jan Pawlowski and Tilman Plehn. Physics and machine learning. Lecture script, Institut für Theoretische Physik, Universität Heidelberg, January 2024. Version from January 30, 2024.

[54] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[55] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[56] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in neural information processing systems*, 32, 2019.

[57] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. *Advances in neural information processing systems*, 32, 2019.

[58] Sebastian Raschka. Understanding and coding the self-attention mechanism of large language models from scratch. 2023. URL: `https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html`.

[59] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR.

[60] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695.

[61] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, December 01, 2021 2021. CVPR 2022. URL: https://ui.adsabs.harvard.edu/abs/2021arXiv211210752R, doi:10.48550/arXiv.2112.10752.

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer.

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[64] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

[65] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[66] Axel Sauer, Frederic Boesel, Tim Dockhorn, Andreas Blattmann, Patrick Esser, and Robin Rombach. Fast high-resolution image synthesis with latent adversarial diffusion distillation. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11.

[67] Axel Sauer, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis. In *International conference on machine learning*, pages 30105–30118. PMLR.

[68] Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision*, pages 87–103. Springer.

[69] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr.

[70] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr.

[71] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, October 01, 2020 2020. ICLR 2021; updated connections with ODEs at page 6, fixed some typos in the proof. URL: https://ui.adsabs.harvard.edu/abs/2020arXiv201002502S, doi:10.48550/arXiv.2010.02502.

[72] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.

[73] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[74] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[75] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.

[76] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in artificial intelligence*, pages 574–584. PMLR.

[77] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[78] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[79] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[80] Inga Strümke and Helge Langseth. Lecture notes in probabilistic diffusion models. *arXiv preprint arXiv:2312.10393*, 2023.

[81] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[82] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, and Katie Millican. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

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

[84] Luke Tierney. Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.

[85] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, and Shruti Bhosale. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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

[87] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4):1–39, 2023.

[88] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and Bill Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024.

[89] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6613–6623.

[90] Dingkun Zhang, Sijia Li, Chen Chen, Qingsong Xie, and Haonan Lu. Laptop-diff: Layer pruning and normalized distillation for compressing diffusion models. *arXiv preprint arXiv:2404.11098*, 2024.

[91] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.

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

[93] Liang Zhao, Xiachong Feng, Xiaocheng Feng, Weihong Zhong, Dongliang Xu, Qing Yang, Hongtao Liu, Bing Qin, and Ting Liu. Length extrapolation of transformers: A survey from the perspective of positional encoding. *arXiv preprint arXiv:2312.17044*, 2023.