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

adaLN Adaptive Layer Normalization

ADD Adversarial Diffusion Distillation

AI Artificial Intelligence

ALD Annealed Langevin Dynamic

CFG Classifier-Free Guidance

CLIP Contrastive Language- Image Pre-training

CLS Classification

CMMD Fréchet Inception Distance

CNN Convolutional Neural Network

DDIM Denoising Diffusion Implicit Model

DDPM Denoising Diffusion Probablistic Model

DiT Diffusion Transformer

DM Diffusion Model

ELBO Evidence Lower Bound

FID Fréchet Inception Distance

GAN Generative Adversarial Network

HPSv2 Human Preference Score v2

iid independent and identically distributed

KL Kullback-Leibler

LADD Latent Adversarial Diffusion Distillation

LDM Latent Diffusion Model

LLM Large Language Model

LPIPS Learned Perceptual Image Patch Similarity

MC Markov Chain

MCMC Markov Chain Monte Carlo

MLE Maximum Likelihood Estimation

MLP Multi Layer Perceptron

MMD Maximum Mean Discrepancy

MMDiT Multi-Modal Diffusion Transformer

MSE Mean Squared Error

NCSN Noise-Conditional Score Network

NF Normalizing Flow

NNL Negative Log-Likelihood

ODE Ordinary Differential Equation

RMSNorm Root Mean Square Normalization

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

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 [2], Claude [5], Gemini [93] or Lama [96] whereas Stable Diffusion [67, 75], Flux [51] and Pixart [13] 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) [46], generative adversarial networks (GANs) [30, 69], normalizing flows (NFs) [74], and diffusion models (DMs) [80]. 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 diffusion and flow-based generativ models in chapter 1.2 that are the most promising models today.

1.1.1 Variational Autoencoder

Autoencoder

An autoencoder [36] 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)

This paragraph and its mathematical derivations are mainly based on [9, 46]. The VAE [46] 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 (iid) samples 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 [9] 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 (MCMC) simulations [95], 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) [10]. In contrast to MCMC VI reformulates the problem as an optimization task

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

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 $p(\mathbf{z}|\mathbf{x})$. To compare the proposal density and the posterior the Kullback-Leibler divergence (KL) [50] is used having the general form of

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

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

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

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 [46] 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) [74] are another class of propabilistic models, similar to VAEs, which learn a mapping between a latent distribution and 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 likelihood using the change of variables formula [64]

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

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 [24]) 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 [30] belong to the class of non-probabilistic 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 trained in an alternating fashion, 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 generation [44, 56, 79] but were overtaken by diffusion models over the last few years. The GANs training process turned out to be often unstable, and extensive hyperparameter tuning and engineering is required [54, 78]. 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 [94], 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 [6] which prevents effective learning of the generator.

1.2 Diffusion and Flow Matching Models

Diffusion models [37, 76, 81, 82, 83, 87] are probabilistic generative models that have achieved remarkable success in image generation tasks [3, 51], 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 [30, 46, 74], they have become the dominant generative modeling paradigm in the research community. The main principles of diffusion models are reflected in two processes motivated by non-equilibrium thermodynamics, namely the forward process and the reverse process. In the forward process, the original image is gradually perturbed by systematically adding Gaussian noise until only 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 with a diffusion model, the starting point is randomly sampled noise to which the diffusion model is applied iteratively. This repeated application gradually produces a clean, high-quality image. To reduce the computational load, instead of executing the diffusion process in image space, [76] proposed to first convert the image/noise into a compressed latent space using an autoencoder, where the diffusion process then took place. This class of diffusion models is therefore called latent diffusion models and are widely applied used.
Over time, several different formulations of diffusion models were proposed. However, there are three dominant formulations recognized in the literature [19]: Denoising diffusion probabilistic models (DDPMs) [37, 81] and its further development denoising diffusion implicit models (DDIMs, score-based generative models (SGMs) [83] and stochastic differential equations (SDEs) [87].

Recently, a generalization of DMs was proposed, known as flow matching or Flow-based models [55, 58]. Unlike classical diffusion, flow matching models directly learn a vector field that defines the path between the noise distribution and the data distribution. By enforcing a straight path trajectory, the sampling process becomes computationally more efficient. This formulation was successfully applied in several recent models like Flux-dev [51] or Sana [101].
In the following, the different formulations of diffusion models, DDPM, DDIM, SGM and SDE, are presented followed by an introduction to the concept of flow matching.

1.2.1 Denoising Diffusion Probabilistic Models

Unless stated otherwise, the discussion and mathematical formulations of DDPM and DDIM presented in this chapter are based on the lecture notes by Inga Strümke and Helge Langseth [89].
In DDPMs [37] the forward and reverse process are modeled as Markov Chain (MC), 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 MC 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 model being used to predict $\mathbf{x}_{t-1}$. The forward process, which progressively adds noise proceeds from right to left in fig. 1.1 and is given by $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ where no learned parameters are required.

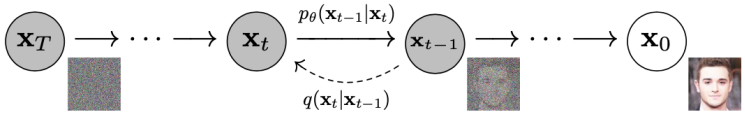


Figure 1.1: The graphical model of DDPM from [37].

Forward Process

Formally, the forward process is defined as a MC that progressively injects noise into the clean image $\mathbf{x}_0 \sim p_{\text{data}}$ which is transformed such that for a sufficiently large number of steps T the final latent variable $\mathbf{x}_T$ approximately follows a new often simpler prior distribution p_{prior}. This boundary condition is necessary so that in the reverse process, which starts with a sample of p_{prior}, the model is able to transfer it back to the image domain p_{data}. The transformation is governed by a variance schedule $\{\beta_t\}_{t=1}^T$, e.g., a linear [37], a cosine [61] or a learnable [45] schedule, to ensure smooth interpolation between p_{prior} and p_{data}. The schedule dictates the incremental increase of variance β_t.

Concretely, the intermediate state $\mathbf{x}_t$ within the MC follows the distribution

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

with the Markov kernel K, which is chosen as a Gaussian Markov diffusion kernel in DDPM leading to the explicit form

$$q\left(\mathbf{x}_t|\mathbf{x}_{t-1}\right) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}\right) \tag{1.21}$$

and the prior distribution $p_{\text{prior}}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mathbf{0}, \mathbf{I})$.

A key property of this Gaussian formulation is the existence of a closed-form expression for sampling any $\mathbf{x}_t$ directly. At every timestep t the intermediate latent variable is computed by

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{z}_t && (1.22)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\,\mathbf{z}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{z}_t && (1.23)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\,\mathbf{z}_{t-1:t} && (1.24)\\
&= \dots && (1.25)\\
&= \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\mathbf{z}_{0:t}. && (1.26)
\end{aligned}$$

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

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x_t}; \sqrt{\bar{\alpha}_t}\mathbf{x_0}, (1-\bar{\alpha}_t)\mathbf{I}\right) \quad . \tag{1.27}$$

This 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

$$\mathbf{x_t} = \sqrt{\bar{\alpha}_t}\mathbf{x_0} + \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon} \tag{1.28}$$

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

While the forward process was all about adding noise to the clean image the reverse process aims recover the clear image by iteratively removing noise. Feller [28] showed that in the limit of infinitesimal steps the reverse process retains the same distributional form as the forward process, a Gaussian. Particularly, in the reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is the quantity of interest and given by

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t) = q(\mathbf{x}_t|\mathbf{x}_{t-1})\frac{q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)} \quad . \tag{1.29}$$

However, it is not tractable because computing the marginal distributions $q(\mathbf{x}_{t-1})$ and $q(\mathbf{x}_t)$ requires the integration over the whole distribution $q(\mathbf{x}_0)$. Therefore, it is approximated 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

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{1.30}$$

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

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}\left(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)\right) \quad . \tag{1.31}$$

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 $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$. Commonly the network does not directly predict the mean $\boldsymbol{\mu}_\theta$ but the noise $\boldsymbol{\epsilon}_\theta$ from which the mean can be computed.

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 based on [64] and [37]. Starting with the expectation of the negative log-likelihood under the data distribution $q(\mathbf{x}_0)$ typically equivalent to the empirical data distribution p_{prior}

$$\begin{aligned}
&- \mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0)\right] && (1.32)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T \, p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right)\right] && (1.33)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T \, p(\mathbf{x}_T) \, q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right)\right] && (1.34)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \, \mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T | \mathbf{x}_0)} \left[p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right]\right] && (1.35)
\end{aligned}$$

using Jensen's inequality

$$\begin{aligned}
-\mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0)\right] &\leq -\mathbb{E}_{q(\mathbf{x}_0)} \left[\mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T | \mathbf{x}_0)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right)\right]\right] && (1.36)\\
&= -\mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right)\right] && (1.37)\\
&= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] && (1.38)\\
&=: \mathcal{L}_{\text{DDPM}} && (1.39)
\end{aligned}$$

the loss for training a diffusion model is obtained. This can be further transformed to

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{q(\mathbf{x}_0,\dots,\mathbf{x}_T)}\left[-\log p(\mathbf{x}_T) - \sum_{t=2}^{T}\log\frac{p_\theta(\mathbf{x}_{t-1}\mid\mathbf{x}_t)}{q(\mathbf{x}_t\mid\mathbf{x}_{t-1})} - \log\frac{p_\theta(\mathbf{x}_0\mid\mathbf{x}_1)}{q(\mathbf{x}_1\mid\mathbf{x}_0)}\right] \tag{1.40}$$

$$= \mathbb{E}_{q(\mathbf{x}_0,\dots,\mathbf{x}_T)}\left[-\log p(\mathbf{x}_T) - \sum_{t=2}^{T}\log\left(\frac{p_\theta(\mathbf{x}_{t-1}\mid\mathbf{x}_t)}{q(\mathbf{x}_{t-1}\mid\mathbf{x}_t,\mathbf{x}_0)}\cdot\frac{q(\mathbf{x}_{t-1}\mid\mathbf{x}_0)}{q(\mathbf{x}_t\mid\mathbf{x}_0)}\right) - \log\frac{p_\theta(\mathbf{x}_0\mid\mathbf{x}_1)}{q(\mathbf{x}_1\mid\mathbf{x}_0)}\right] \tag{1.41}$$

$$= \mathbb{E}_{q(\mathbf{x}_0,\dots,\mathbf{x}_T)}\left[-\log\frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T\mid\mathbf{x}_0)} - \sum_{t=2}^{T}\log\frac{p_\theta(\mathbf{x}_{t-1}\mid\mathbf{x}_t)}{q(\mathbf{x}_{t-1}\mid\mathbf{x}_t,\mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0\mid\mathbf{x}_1)\right] \tag{1.42}$$

$$= \mathbb{E}_{q(\mathbf{x}_0,\dots,\mathbf{x}_T)}\left[\mathrm{KL}\left(q(\mathbf{x}_T\mid\mathbf{x}_0)\,\|\,p(\mathbf{x}_T)\right) + \sum_{t=2}^{T}\mathrm{KL}\left(q(\mathbf{x}_{t-1}\mid\mathbf{x}_t,\mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}\mid\mathbf{x}_t)\right) - \log p_\theta(\mathbf{x}_0\mid\mathbf{x}_1)\right] \quad . \tag{1.43}$$

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

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_t)}\left[\mathrm{KL}\left(q(\mathbf{x}_{t-1}\mid\mathbf{x}_t,\mathbf{x}_0)\,\|\,p_\theta(\mathbf{x}_{t-1}\mid\mathbf{x}_t)\right)\right] - \mathbb{E}_{q(\mathbf{x}_0,\dots,\mathbf{x}_T)}\left[\log p_\theta(\mathbf{x}_0\mid\mathbf{x}_1)\right] + \text{const} \tag{1.44}$$

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

assuming that the term $\mathbb{E}_{q(\mathbf{x}_0,\dots,\mathbf{x}_T)}\left[\log p_\theta(\mathbf{x}_0\mid\mathbf{x}_1)\right]$ is negligible. Note that in eq. 1.45 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 $q(\mathbf{x}_{t-1}\mid\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1};\hat{\mu}(\mathbf{x}_t,\mathbf{x}_0),tilde\beta_t\mathbf{I}\right)$ are compared in the KL-divergence meaning there is a closed-form solution. The loss for the diffusion model boils down to

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{x}_0,\mathbf{x}_t)}\left[\frac{1}{2\sigma^2}||\hat{\mu}(\mathbf{x}_t,\mathbf{x}_0) - \mu_\theta(\mathbf{x}_t,t)||^2\right] \quad . \tag{1.46}$$

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

$$\hat{\mu}(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \tag{1.47}$$

$$\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \quad . \tag{1.48}$$

However, instead of predicting the mean, diffusion models are more often trained on predicting the noise $\boldsymbol{\epsilon}_\theta$ because [37] empirically found that this lead to more stable training

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T}\frac{1}{2\tilde{\beta}_t}\frac{(1-\alpha_t)^2}{\alpha_t(1-\bar{\alpha}_t)}\cdot\mathbb{E}_{q(x_t|x_0)}\left[\|\boldsymbol{\epsilon}_\theta(x_t,t) - \boldsymbol{\epsilon}_t\|_2^2\right] \tag{1.49}$$

which is achieved by reparameterization of eq. 1.46 via

$$\hat{\mu}(\mathbf{x}_t,\mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right) \quad . \tag{1.50}$$

In addition, they found empirically that ignoring the weighting term of eq. 1.49 worked better

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} \left[\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t\|_2^2 \right] \quad . \tag{1.51}$$

DDPM Sampler

Algorithm 1 and fig. 1.2 depict the training process of the DDPM approach. 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. 1.28 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. 1.51) 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 [89]

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

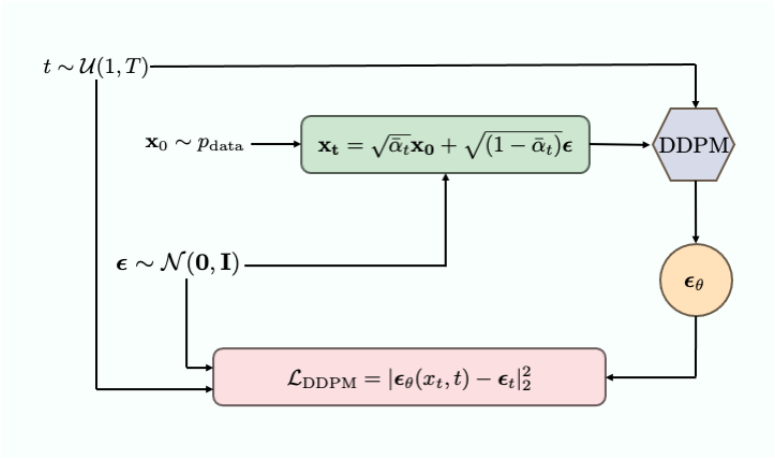


Figure 1.2: DDPM training [11].

Algorithm 2 and fig. 1.3 show schematically the sampling process which reverses the diffusion process to generate images from 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

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z} \tag{1.52}$$

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

$$\mathbf{x}_0 = \frac{1}{\sqrt{\alpha_1}} \left(\mathbf{x}_1 - \frac{1 - \alpha_1}{\sqrt{1 - \bar{\alpha}_1}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_1, 1) \right) + \sigma_1 \mathbf{z} \tag{1.53}$$

with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Here, it becomes clear that the model has to predict the noise T times which makes inference

of such a diffusion model slow.

Algorithm 2 Sampling of DDPM [89]

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
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 $\mathbf{x}_0$

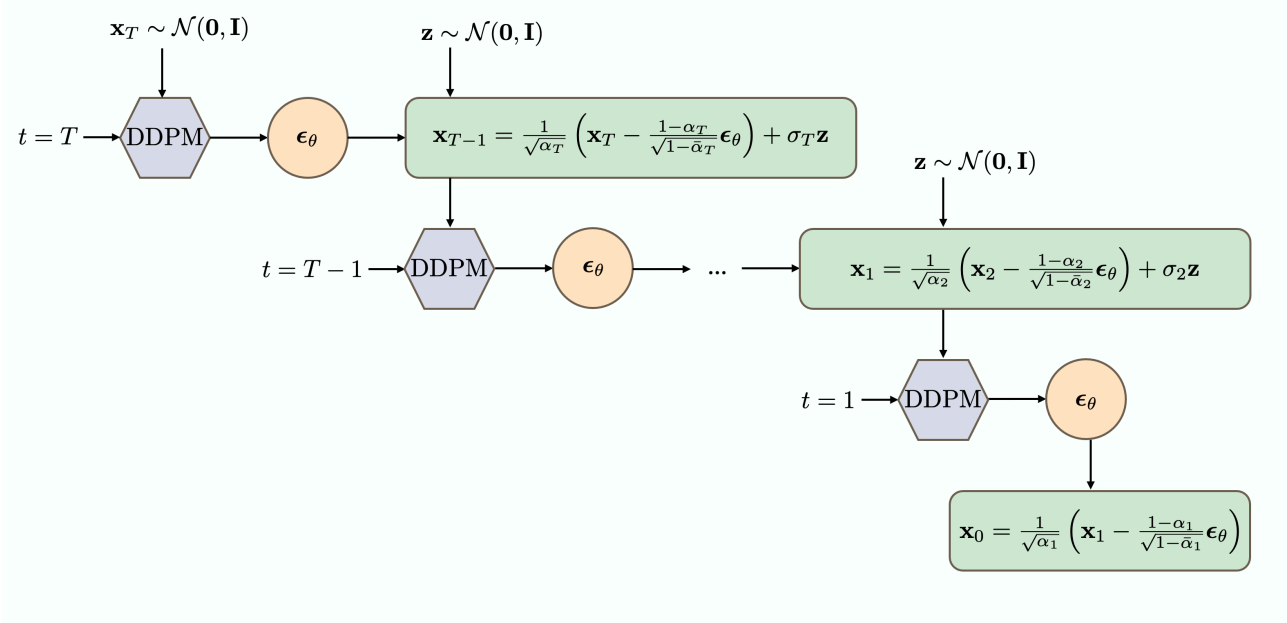


Figure 1.3: DDPM sampling [11].

1.2.2 Denoising Diffusion Implicit 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) [82]. Unlike DDPM, DDIM defines the joint distribution as

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = q(\mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=2}^{T} q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \quad . \tag{1.54}$$

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 [82] parameterize the reverse process as

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t} \mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \sigma_t^2 \cdot \mathbf{I} \right) \quad . \tag{1.55}$$

The corresponding sampling equation becomes

$$\mathbf{x}_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\cdot\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)}_{\text{Predicted } \mathbf{x}_0} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2}\cdot\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)}_{\text{Direction of } \mathbf{x}_t} + \underbrace{\sigma_t\cdot\mathbf{z}}_{\text{Noise}} \tag{1.56}$$

where σ_t is a parameter to choose. $\mathbf{x}_{t-1}$ 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 σ_t. The key advantage of DDIM is the flexibility in choosing σ_t, which does not need to follow a predefined variance schedule. Different values of σ_t yield different diffusion processes while using the same trained model. In particular, for the choice

$$\sigma_t = \sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}}\sqrt{1-\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}} \tag{1.57}$$

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 $\sigma = 0$ the whole process becomes deterministic.

1.2.3 Score-Based Generative Models

This discussion is mainly based on [19, 102]. Score-Based Generative Models (SGMs) [84, 85] 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 [31, 86, 88]. Here, the one closest to the DDPM approach is presented [84]. 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 $s_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q(x_t)$. Perturbing the data is necessary because the score estimation is inaccurate in low-density regions and to ensure the process does not get stuck. These networks are called Noise-Conditional Score Networks (NCSNs). The learning objective boils down to

$$\mathbb{E}_{T\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)}\left[\lambda(t)\sigma_t^2\left\|\nabla_{\mathbf{x}_t}\log q(\mathbf{x}_t) - s_\theta(\mathbf{x}_t,t)\right\|^2\right] \tag{1.58}$$

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

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

$$= \mathbb{E}_{T\sim\mathcal{U}[1,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\lambda(t)\left\|\boldsymbol{\epsilon} + \sigma_t s_\theta(\mathbf{x}_t,t)\right\|^2\right] + \text{const} \quad . \tag{1.61}$$

In eq. 1.61 the relation to DDPM loss becomes visible by identifying $\boldsymbol{\epsilon}(\mathbf{x}_t, t) = -\sigma_t s_\theta(\mathbf{x}_t, t)$ [102], showing that learning to predict noise is equivalent to learning the score function.
For sampling a new element from the data distribution [84] proposed Annealed Langevin Dynamics (ALD) which produces samples by only using the score function being approximated by the learned neural network

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \alpha\nabla_{\mathbf{x}}\log p(\mathbf{x}_{t-1}) + \sqrt{2\alpha}\mathbf{z}_t, \quad 1 \le t \le T \tag{1.62}$$

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. 1.62 iteratively to initial Gaussian noise, a new element of the data distribution is generated.

1.2.4 Stochastic Differential Equations

This discussion is mainly based on [19, 102]. Stochastic Differential Equations (SDEs) [88] 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

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \tag{1.63}$$

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). [4] proved that the diffusion process in eq. 1.63 can be reversed leading to the reversed-time SDE given by

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_x \log q(\mathbf{x}_t)\right] dt + g(t)d\tilde{\mathbf{w}} \tag{1.64}$$

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. 1.58) to continuous time

$$\mathbb{E}_{t\sim\mathcal{U}[0,T],\mathbf{x}_0\sim q(\mathbf{x}_0),\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)} \left[\lambda(t) \left\| s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)\right\|^2\right] \tag{1.65}$$

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. 1.64 has to be computed using the score neural network and a SDE solver. Another finding of [88] is that for eq. 1.64 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. 1.64

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log q(\mathbf{x}_t)\right] dt \tag{1.66}$$

Both reversed-SDE or probability flow ODE can be used to generate new samples using SDE solvers [42, 88] or respectively ODE solvers [43, 60, 88, 105].

1.2.5 Flow Matching

The concept of flow matching was introduced for normalizing flows [55] 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

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t\sim\mathcal{U}(0,1=),x\sim p(t,x)} \left[\left\| v_\theta(t, x) - u(t, x)\right\|^2\right] \tag{1.67}$$

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

$$p(x, t) = \int p(t, x \mid x_1) q(x_1) dx_1 \tag{1.68}$$

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

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

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. 1.67) using the conditional vector field.

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}(0,1=),x\sim p(t,x)}\left[\|v_\theta(t,x) - u(t,x \mid x_1)\|^2\right] \tag{1.70}$$

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

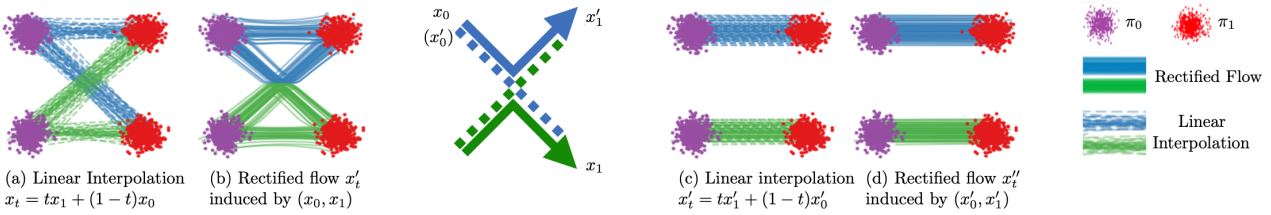
[58] proposed rectified flow, a specialized variant of flow matching [55] that aims to straighten the trajectories between source and target distributions π_0 and π_1 by iteratively refining the learned ODE. This approach offers significant computational advantages, specifically, the straightened paths 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

$$dx_t = v_\theta(x_t,t)dt \tag{1.71}$$

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

$$\min_v \int_0^1 \mathbb{E}\left[\|(x_1 - x_0) - v_\theta(x_t,t)\|^2\right] dt \tag{1.72}$$

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. 1.71 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. It is important to distinguish distillation from the rectification process itself. While rectification creates a new transport map (x_0^{k+1}, x_1^{k+1}) that is straighter and has lower transport costs, distillation merely trains a student network to mimic the existing coupling (x_0^k, x_1^k) for faster execution.



(a) Linear Interpolation $x_t = tx_1 + (1-t)x_0$ (b) Rectified flow x_t' induced by (x_0, x_1) (c) Linear interpolation $x_t' = tx_1' + (1-t)x_0'$ (d) Rectified flow x_t'' induced by (x_0', x_1')

Figure 1.4: Rectified Flow [58].

Algorithm 3 Rectified Flow: Main Algorithm [58]

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: $x = \{x_t : t \in [0,1]\}$.

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 $x_1^k \approx \hat{T}(x_0^k)$.

1.2.6 Latent Diffusion Models

Diffusion models operating in pixel space produce images of high quality and diversity. However, a major problem is the high computational effort required during training and inference. This becomes particularly relevant for the generation of high-resolution images. For this reason, latent diffusion models (LDMs) were introduced by [76] and are widely used nowadays [13, 51, 67, 75]. With LDMs, the diffusion process is no longer performed in pixel space but in a compressed latent space (see fig. 1.5), which significantly reduces the computational effort. To perform the compression, an autoencoder is used, whereby additional regularizations are often applied such as KL-regularization [46] or VQ-regularization [97]. The training process of LDMs is divided into two steps. First, the autoencoder $(\mathcal{E}, \mathcal{D})$ is trained to reconstruct images learning to preserve local details and pixel-level information. In a second step, the diffusion model is trained on the compressed latent representations to learn structure and semantics. This procedure has the advantage that the perceptual task handled by the autoencoder and the generative task handled by the DM are learned separately, in contrast to DMs in pixel space where both tasks must be learned simultaneously. The training loss for a LDM is similar to the loss for DMs in pixel space. Following the DDPM formulation the loss for LDMs has the same form like for DMs (see eq. 1.51)

$$\mathcal{L}_{\text{LDM}} = \sum_{t=2}^{T} \mathbb{E}_{q(\mathbf{z}_t|\mathbf{z}_0)} \left[\|\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t) - \boldsymbol{\epsilon}_t\|_2^2\right] \tag{1.73}$$

with $\mathbf{z}$ denoting the latents, specifically, $\mathbf{z}_0$ being the latent representation of the clean image obtained by $\mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0)$, where $\mathcal{E}$ is the encoder of the autoencoder. During inference, random noise is sampled in the latent space and serves as input to the LDM during the iterative denosiding process. The completely denoised latent is then fed to the decoder $\mathcal{D}$, which transforms it back to pixel space. Fig. 1.5 also shows the conditioning mechanism of the LDM via text, semantic maps, etc.. This important aspect based on cross-attention will be discussed in the following section **??**.

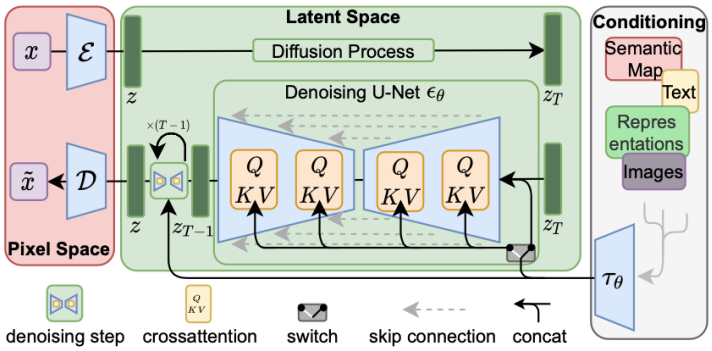


Figure 1.5: Structure of a Latent Diffusion Model [76].

1.2.7 Conditioning and Classifier-Free Guidance

In DMs, conditioning plays a pivot role in steering the diffusion process through incorporating extra inputs such as text, semantic maps [104], or images [92] users can precisely control the content of the final image. Formally, the diffusion model is trained to learn the conditioned distribution $p(\mathbf{x}|\mathbf{y})$ where $\mathbf{y}$ represents the additional conditioning signal. Consequently, the noise prediction function is modified from $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t)$. Cross-attention layers (see section 1.3.2) [98] are an effective mechanism for integrating the additional guidance signal into the model. These layers are embedded at various resolutions within the U-net architecture, as shown in fig. 1.5, or within the transformer blocks of modern models like Pixart [13]. To process text prompts, a tokenizer first breaks down the text into discrete subword units, known as tokens, and assigns them a unique numerical ID. Next, the tokens are embedded in a higher-dimensional vector space by a text encoder, and positional encoding (see section 1.3.3) is applied to preserve the sequence order. These embeddings are then used as input for the cross-attention mechanism.

Classifier-Free Guidance

A frequently used method to improve the alignment between the generated image and the text prompt is classifier-free guidance (CFG) [38]. This method builds upon classifier guidance [23], which was introduced for class conditioned diffusion models. In the classifier guidance framework, during training an external classifier predicts the class label and its gradients are used to steer the diffusion model to produce images that can be easily identified by the classifier. However this necessitate the training of an additional classifier which is able to operate on noisy input images as standard classifier are unsuitable for this task. The goal is to steer the diffusion through the additional learning signal to producing images with higher fidelity to the conditioning signal. To facilitate the training process and get rid of the extra classifier CFG was introduced. In the CFG framework, the diffsuion model is jointly trained on conditional $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{y})$ and unconditional $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{y} = \varnothing)$ objectives. For that during training the condition is randomly dropped with the probability $p_{\text{unconditonal}}$. During inference, the same model is used twice to predict the conditional noise and the unconditional noise which are linearly combined to

$$\tilde{\boldsymbol{\epsilon}}_\theta = (1 + w)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{y}) - w\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{y} = \varnothing) \quad . \tag{1.74}$$

with w being a hyperparameter called guidance scale.

A high value for w pushes the generated images away from the unconditional distribution and towards the conditional distribution. However, this comes at the cost of higher computational inference time because the model has to be executed twice per timestep. Nevertheless, CFG lead to higher text coherencen and yields higher-contrast and sharper images.

1.3 Fundamentals of Neural Network Architectures

In this section, the focus lies on the key fundamental building blocks of the neural networks.

1.3.1 Transformer

The introduction of the transformer architecture [98] 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 [3, 5, 51, 93].
The transformer was introduced as an alternative to recurrent neural networks [77] 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 [2, 96] or encoder-only transformer [22].
Transformers, as introduced in the original paper (Fig. 1.6), 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. First, a multi-head self-attention mechanism is followed by a residual connection [34] and layer normalization [7]. Second, a feedforward network processes the output further. The decoder blocks contain three sub-layers. First, a masked multi-head self-attention prevents the model from attending to future tokens during the training process, second, a cross-attention mechanism introduces the output of the encoder in the decoder and third an additional feedforward network processes the output of the cross-attention layer. 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.

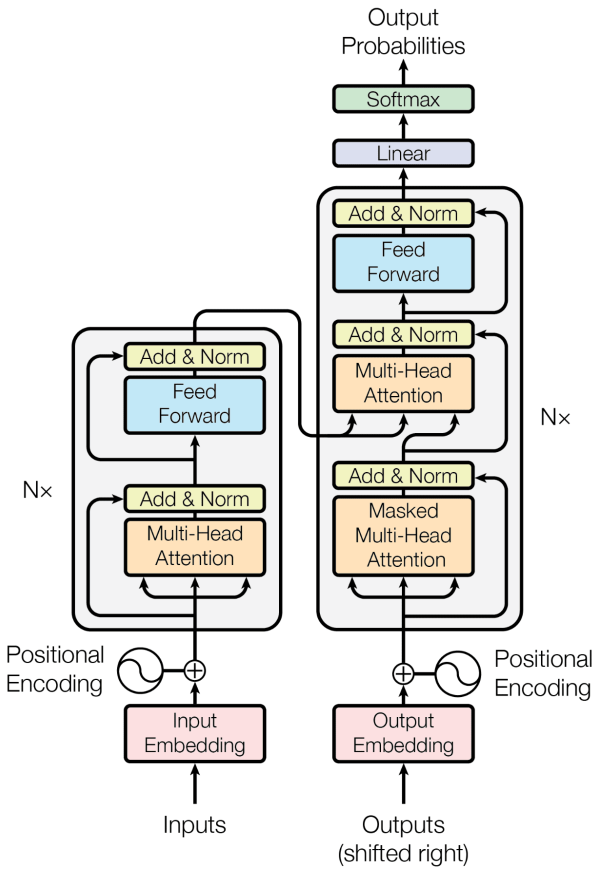


Figure 1.6: Transformer architecture from [98].

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 [8] to overcome the information bottleneck, as traditionally in RNNs only the last hidden state is given as input for the next prediction. In [8], 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 [98] 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. 1.7) and cross-attention (fig. 1.8). 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 matrix of the tokens. Then the queries Q, keys K and values V are computed by linear projection of the embedding matrix

$$Q = X \cdot W_q^T \tag{1.75}$$
$$K = X \cdot W_k^T \tag{1.76}$$
$$V = X \cdot W_v^T \tag{1.77}$$

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

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{1.78}$$

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

$$\text{softmax}\,(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1.79}$$

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 dot product results in very large values. Finally, the attention matrix is applied with the values. This can be summarized in the following formulation

$$Z = \text{Attention}\,(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \tag{1.80}$$

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 [17, 18, 20, 48].

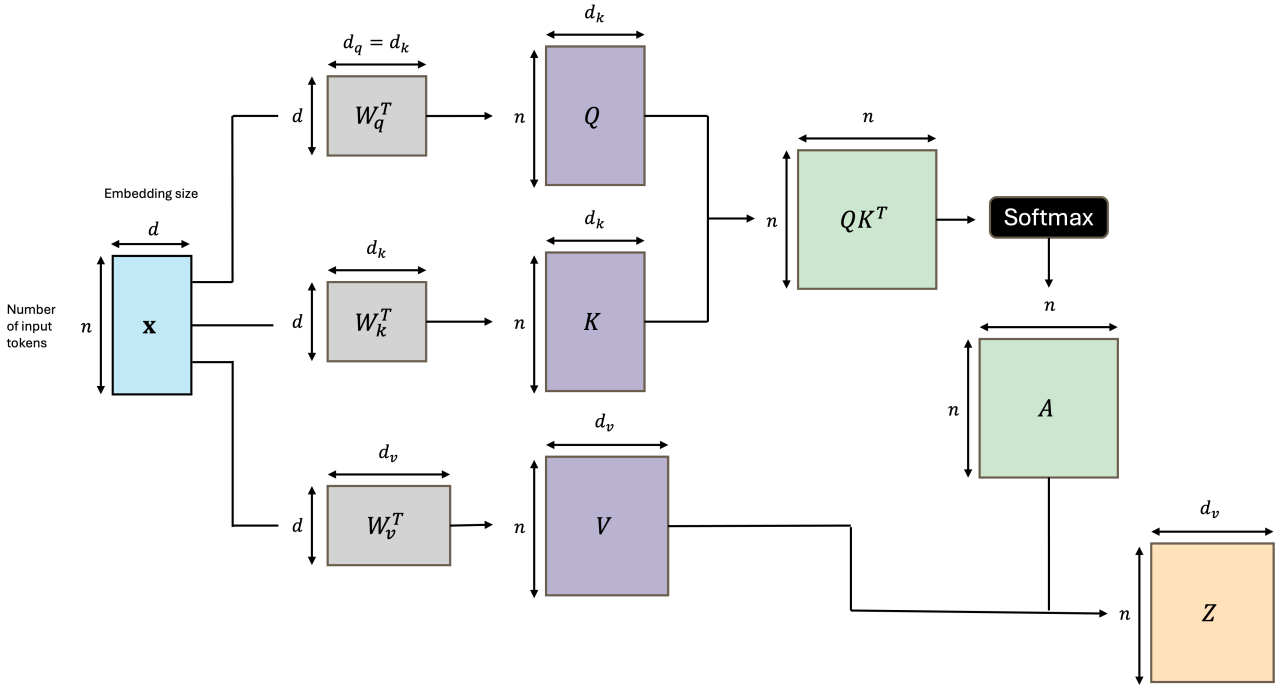


Figure 1.7: Self-attention mechanism based on [73].

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., the decoder of the transformer, while the keys and values come from the source sequence, e.g., encoder output of transformer. Apart from this difference, the computation is identical to self-attention.

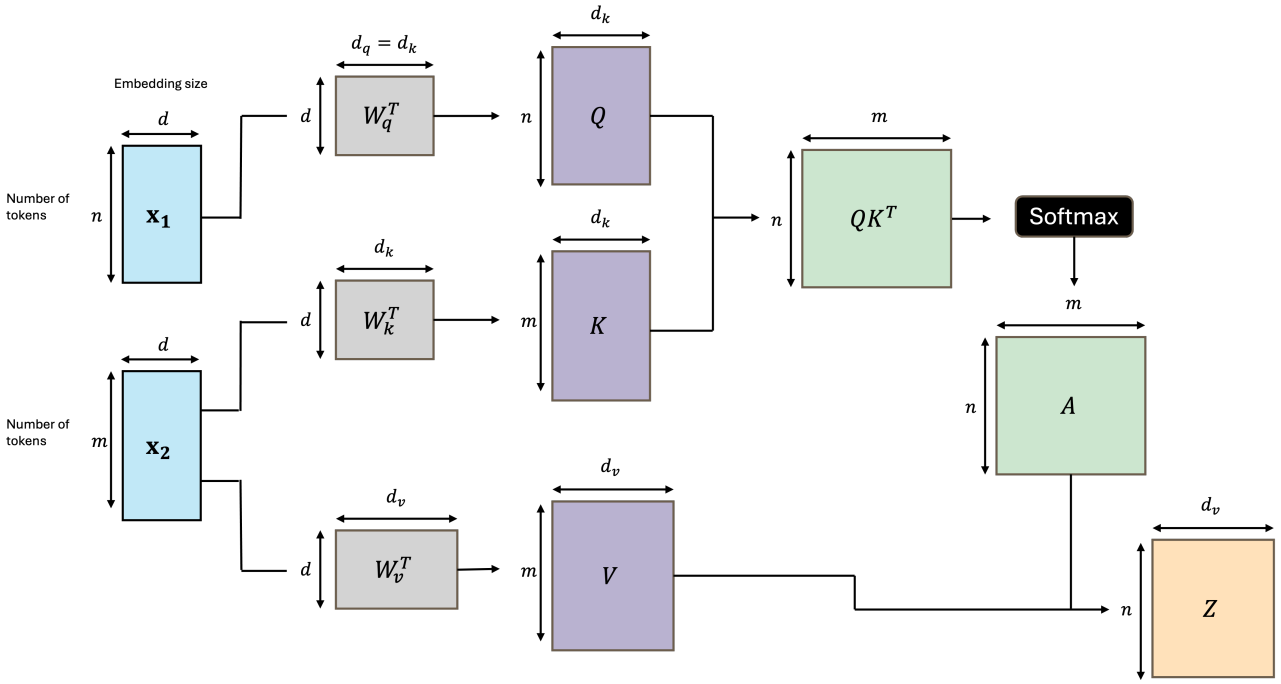


Figure 1.8: Cross-attention mechanism based on [73].

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 $W^O \in \mathbb{R}^{N d_v \times d}$.

$$\text{MultiHead}\left(Q, K, V\right) = \text{Concat}\left(\text{head}_1, ..., \text{head}_N\right) W^O \tag{1.81}$$

$$\text{head}_i = \text{Attention}\left(XW_{q_i}^T, XW_{k_i}^T, XW_{v_i}^T\right) \tag{1.82}$$

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.

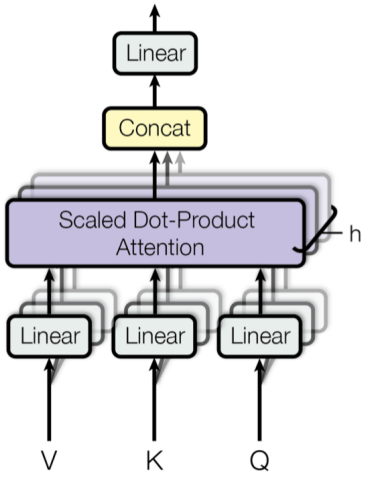


Figure 1.9: Multi-head attention from [98].

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 [52, 62, 90] versus absolute [49, 53, 59, 98] positional embedding or deterministic [49, 53, 62, 90, 98] versus learnable [52, 59] positional embedding.

Sinusoidal Positional Encoding

Sinusoidal positional encoding is used in the original transformer paper [98] which leverages the sinus and cosine function to inject positional information into the model. It belongs to the category of absolute and deterministic positional encoding.

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

$$PE\left(pos, 2i+1\right) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \tag{1.84}$$

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 added to it.

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 [39, 63, 71], but for a long time, convolutional neural networks (CNNs) [72] remained the dominant architecture. CNNs have an inherent advantage over transformers because they possess an image-related inductive bias. On the one hand, CNNs are translationally equivariant by construction 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 by a transformer, as they are not predefined by its architecture, which makes a sufficiently large dataset essential to achieve competitive performance.

The introduction of vision transformer (ViT) [25] for a classification task 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 $\mathbf{x} \in R^{H \times W \times C}$ 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.

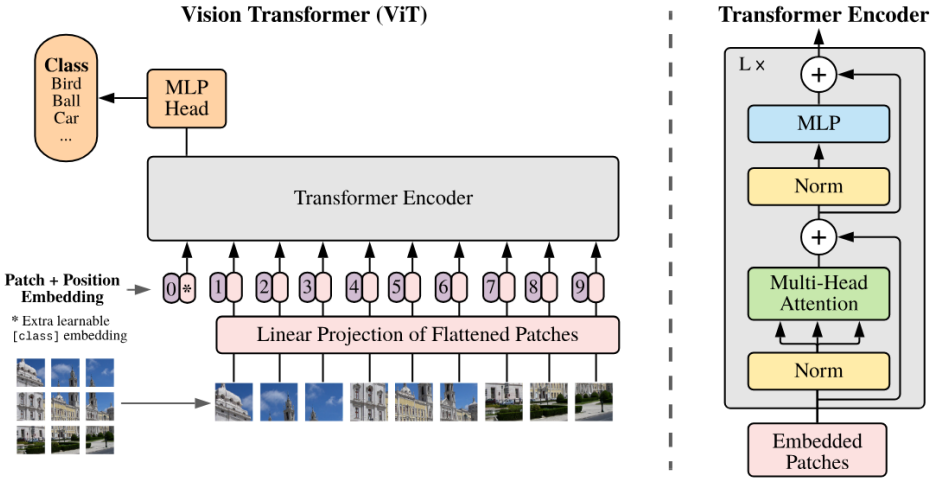


Figure 1.10: Vision transformer from [25].

1.3.5 Adaptive Layer Normalization

Layer normalization [7] was introduced as an alternative to batch normalization [40] as a normalization method to reduce the internal covariate shift in deep neural networks independently of the batch size. The covariate shift describes the phenomenon whereby the range of the input to a layer which is the ouptut of the previous layer changes permanently, which hinders effective learning because the layer needs to constantly adapt to a new input distribution. To stabilize and accelerate the training process normalizations are applied in the neural network. In layer normalization, the activations of each layer $\mathbf{z}_l$ are normalized to a zero mean and unit variance. In addition, a scale γ and a shift η parameter are learned to maintain the flexibility and expressiveness of the network and to ease the constrained range due to the normalization. The normalized layer output is given by

$$\hat{\mathbf{z}}_l = \gamma_l \frac{\mathbf{z}_l - \mu_l(\mathbf{z}_l)}{\sigma_l(\mathbf{z}_l)} + \eta_l \quad . \tag{1.85}$$

A further development of layer normalization occured when [66] showed that the scale and shift parameters could be used to incorporate additional information into the model by parameterizing them as functions of an additional input c such that $\gamma \rightarrow \gamma(c)$ and shift $\eta \rightarrow \eta(c)$. This method is known as adaptive layer normalization (adaLN) and can be leveraged in diffusion models to incorporate the timestep or class conditioning [65].

1.4 Evaluation Metrics

For DMs, several different metrics were introduced to measure the overall image quality, the distance of the generated distribution with the true distribution or the image prompt alignment. In the follwing, the metrics used in this work are presented: Fréchet Inception Distance (FID) [35], CLIP-Maximum Mean Discrepancy (CMMD) [41], Human Preference Score v2 (HPSv2) [100]

1.4.1 Fréchet Inception Distance

The FID [35] measures the similarity between the model distribution p_{model} with the true data distribution p_{data}. To compare the distributions a set of images is generated by the diffusion model and another set of images from the original data is needed. An inception model, e.g., Inception-v3 [91], extracts features for every image of the sets. Under the assumption that the features from both sets follow a multivariate normal distribution the mean and covariance of the generated image features ($\boldsymbol{\mu}_{\text{model}}$, Σ_{model}) and the features from the original data ($\boldsymbol{\mu}_{\text{data}}$, Σ_{data}) are computed and compared by the Fréchet Distance [26]

$$\text{FID} = \|\boldsymbol{\mu}_{\text{data}} - \boldsymbol{\mu}_{\text{model}}\|_2^2 + \text{Tr}\left(\Sigma_{\text{model}} + \Sigma_{\text{data}} - 2\left(\Sigma_{\text{model}}\Sigma_{\text{data}}\right)^{\frac{1}{2}}\right) \tag{1.86}$$

meaning lower FID values indicate a higher degree of similarity between the generated images and the original images.

Although FID is widely used [41] identified several shortcomings. First, the underlying Inception-v3 model is trained only on images from 1,000 classes which leads to embeddings that often fail to capture the rich and varied content of the images generated by current diffusion models. Secondly, the normality assumption for the feature distributions is frequentyl violated in reality which can lead to FID scores which do not align to human ratings. A third drawback is that the FID score does not capture complex distortion and depends highly sensitive on the sample size and exhibit bias. For these reasons [41] proposed an alternative score which is discussed next.

1.4.2 CLIP-Maximum Mean Discrepancy

Similar to the FID, the CMMD [41] score measures the discrepancy between the data and the model distribution. It replaces the Inception-v3 embeddings with Contrastive Language- Image Pre-training (CLIP) [68] embeddings. Since CLIP is trained on more images and leverages natural langugage supervision, its embeddings have better representation able to capture more semantics details of the images. The features are compared via the maximum mean discrepancy (MMD) [32, 33]

$$\begin{aligned}\text{MMD}^2 = &\frac{1}{m(m-1)}\sum_{i=1}^{m}\sum_{j\neq i}^{m} k(\mathbf{x}_{\text{data},i}, \mathbf{x}_{\text{data},j}) \\ &+ \frac{1}{n(n-1)}\sum_{i=1}^{n}\sum_{j\neq i}^{n} k(\mathbf{x}_{\text{model},i}, \mathbf{x}_{\text{model},j}) \\ &- \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k(\mathbf{x}_{\text{data},i}, \mathbf{x}_{\text{model},j}) \quad .\end{aligned} \tag{1.87}$$

with $\mathbf{x}_{\text{data},i} \sim p_{\text{data}}$, $\mathbf{x}_{\text{model},i} \sim p_{\text{model}}$ and a positive semi-definite Gaussian RBF kernel $k = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$. Through the choice of this kernel no distributional assumption over the features is needed and in contrast to FID score it is an unbiased estimator. [41] showed in several ablation studies that the CMMD score aligns more closely with human preferences and can detect distortion that FID struggles to identify.

1.4.3 Human Preference Score v2 Benchmark

The HPSv2 benchmark [100] is designed to rank generated images based on human preferences. To model the human preference score, a CLIP model[68] is finetuned on the HPDv2 dataset [100] which contains 434,000 images, generated by nine different text-to-image models or taken from the COCO dataset with over 798,000 human preference choices. A human preference choice always includes two images $\mathbf{x}_1$ and $\mathbf{x}_2$ and a binary ranking $\mathbf{y}$ indicating whether image $\mathbf{x}_1$ is preferred over image $\mathbf{x}_2$, e.g. $\mathbf{y} = [1, 0]$ means that $\mathbf{x}_1$ is preferred by the annotator. The human preference score is determined by the finetuned CLIP model via

$$s_\theta(p, \mathbf{x}) = \mathcal{E}_{\text{txt}}(p) \cdot \mathcal{E}_{\text{img}}(\mathbf{x}) \cdot 100 \tag{1.88}$$

where p is the text prompt, x is the image, and $\mathcal{E}_{\text{img}}$ and $\mathcal{E}_{\text{txt}}$ are the image and text encoder of the CLIP model, respectively. The HPSv2 benchmark contains four categories, namely "Animation", "Concept-Art", "Painting", and "Photo" with 800 prompts each. The prompts are derived on COCO captions [16] and DiffusionDB [99] but cleaned and modified by ChatGPT. A higher score computed with eq. 1.88 correspond to better image quality.

1.4.4 GenEval Benchmark

Unlike global metrics such as CLIP and FID, which struggle to capture the compositional correctness of images, the GenEval Benchmark [29] is designed to evaluate text-to-image alignment based on an object-focused framework. It leverages existing models, such as object detection model, semantic model, CLIP, to analyze the content of the image and determine whether it follows the prompt closely. The benchmark is divided into six categories of varying difficulty.

1. Single object: A single object is described by the prompt.
2. Two objects: Two different objects are described by the prompt.
3. Counting: One object should appear several times.
4. Colors: An object should have a specific color.
5. Position: Two or more objects should be placed in a certain order relative to each other.
6. Attribute binding: Correctly associating attributes (e.g., colors) to specific objects (e.g., a red cube and a blue sphere).

For each category, the prompt follows a specific structure. For the benchmark, 553 different prompts are used and for each prompt four images are generated. The scoring is based on a binary classification determining whether the object, number, and color of objects, etc. are correctly present in the image (see fig. 1.11). This returns a ratio representing the proportion of images that contain all described aspects of the prompt.

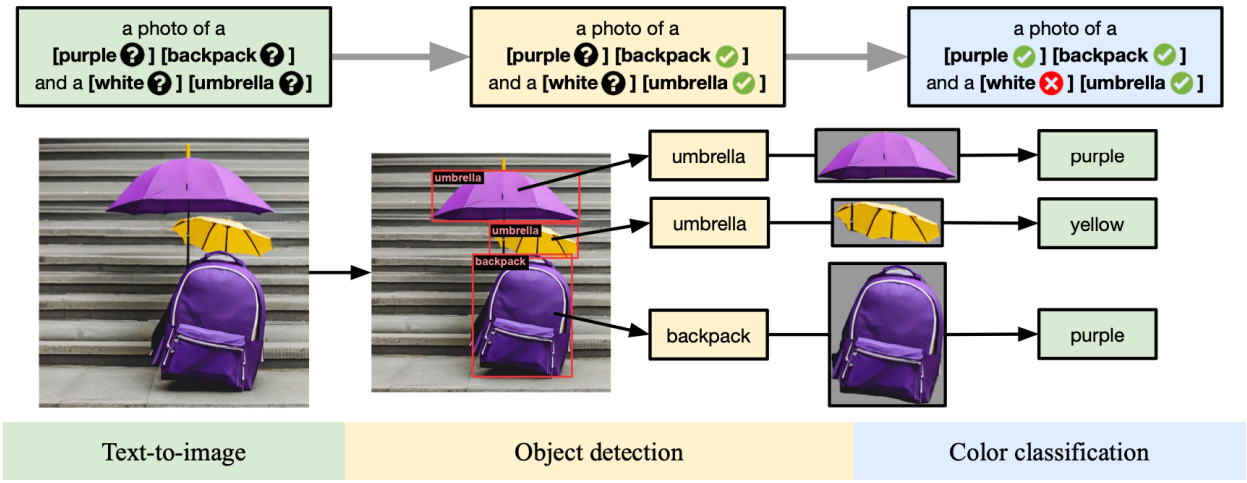


Figure 1.11: Example of how image is investigated for GenEval benchmark [29].

Chapter 2

State-of-the-Art Image Generation Models

In the following, the models, namely Pixart-Σ [13] and Flux-dev [51], which were used for the experiments in this work are described in detail.

2.1 Pixart-Σ

PixArt-Σ [13], the latest addition to the PixArt family [1, 14], was developed to demonstrate that high-fidelity image generation can be achieved with significantly reduced training costs and model parameters. Pixart-Σ is a further development of Pixart-α. Therefore, the shared architecture of Pixart-α and Pixart-Σ is introduced first. Next, the key points for efficient training of Pixart-α are described, followed by the specific enhancements implemented in PixArt-Σ.

2.1.1 Architecture

The architecture of Pixart-Σ is based on the class-conditional diffusion transformer DiT-XL/2 [65] which contains 28 transformer blocks. The transformer blocks have primarily three inputs (see fig. 2.1) which are described in the following.
The first input is the **timestep** t which is projected into a time embedding vector using a 256-frequency embedding, from which a MLP extracts the global scale and shift parameters. They are used as input for the customized adaptive layer normalization called adaLN-single layer, which modulates the hidden states based on the time embedding.
The second input is the **text prompt** that describes the content of the image. It is tokenized into 120 tokens in Pixart-α, repsectively 300 tokens in Pixart-Σ, and encoded by the T5-XXL text encoder [70]. A linear projection layer is then applied to reduce the high dimensional output from the T5-XXL encoder after which it is incorporated via cross-attention layers into the model.
The third input is the **latent image** which is encoded via a pre-trained autoencoder, e.g. SDXL-VAE [67] in Pixart-Σ, transforming the image $(3, H, W)$ into latent space $(4, H/8, W/8)$. Patches of size 2×2 are extracted from the latent representation of the image and flattened into a sequence of visual tokens. Finally, 2D sine-cosine positional encoding is applied before feeding them as input to the transformer blocks.

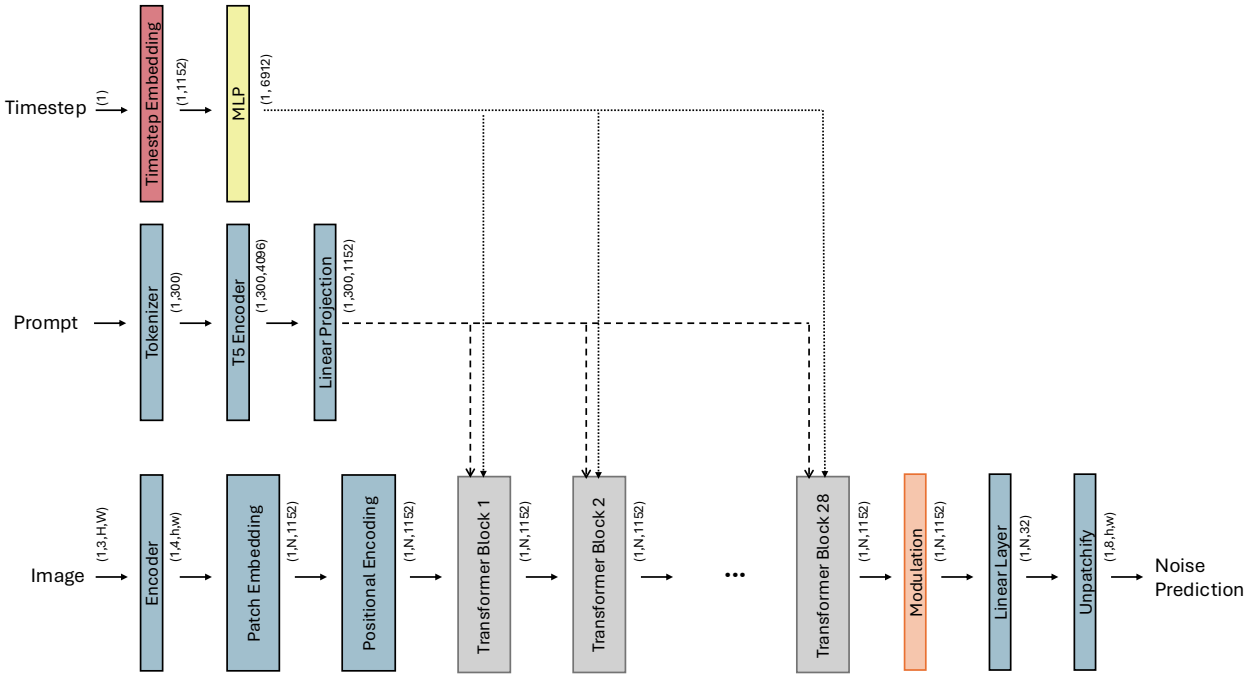


Figure 2.1: Overview architecture of Pixart-Σ. H and W are the height and width of the image. $h = \frac{H}{8}$ and $w = \frac{W}{8}$ represent the dimensions in latent space and $N = \frac{H}{8 \cdot p} \frac{W}{8 \cdot p}$ with p being the patch size.

The main components of the diffusion model are the transformer blocks. They consist of four main parts, namely the self-attention layer, the cross-attention layer, the MLP and the adaLN-single mechanism (see fig. 2.2).

The **multi-head self-attention** layer allows the tokens to attend to each other and capture spatial relationships. It is embedded between time-dependent modulation layers at the beginning of the transformer block.

The **multi-head cross-attention** layer incorporates text conditioning into the model to align the generated image with the text prompt and is placed between the self-attention and the MLP layer.

The **MLP** processes each token individually to refine the features. Similar to the self-attention layer, it is positioned between modulation layers at the end of the transformer block.

adaLN-single is a modification of adaLN (see chapter 1.3.5) method [66] replacing the static parameters of the normalization layer with dynamic values that are predicted directly from conditioning signals such as the timestep. This enables the model to efficiently modulate the feature distribution in each block, thereby controlling the generation process globally. In [65] both class and time conditioning were handled via adaLN. In contrast, Pixart-Σ only uses it for incorporating the time embedding. To reduce the number of parameters global scale and shift parameters $\bar{S}$ are extracted from one shared MLP and each block has additional learnable parameters E_i which are added to the global parameters via a summation function g $S_i = g(\bar{S}, E_i)$ to provide flexibility for every transformer block.

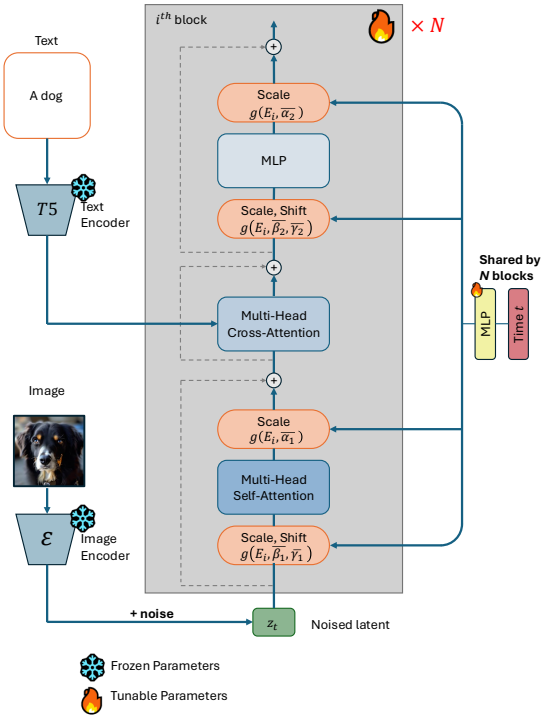


Figure 2.2: Diffusion block architecture taken from [14].

The output of the final transformer block is modulated and processed by a linear layer before the token sequence is reshaped back into spatial dimensions. The final output comprises eight channels as not only the noise but also the variance is predicted. The total number of parameters of individual parts of the transformer blocks and the total model can be found in tab. 2.1.

Table 2.1: Number of parameters for the individual components of the Transformer block and the total Pixart-Σ model.

Component	**Parameters (Mio)**
Multi-Head Self-Attention	5.3
Multi-Head Cross-Attention	5.3
Feed-Forward Network (MLP)	10.6
Total (Single Block)	21.3
Total Model (PixArt-Σ)	610.9

2.1.2 Training Strategy

Pixart training proceeds in three distinct stages, explicitly decoupling the learning of pixel dependencies from text-concept alignment to improve training efficiency. First, the focus is on learning pixel dependencies to generate semantically meaningful images. To achieve this, the model is initialized with weights pre-trained on ImageNet [21] using class-conditional generation. This pre-training provides a strong initialization for the visual distribution at a low computational cost. In a second step, the focus is on text alignment so that the model generates images that accurately reflects the text prompt. For this purpose, the SAM [47] dataset was utilized with corresponding high-density prompts generated by the vision-language model LLaVA [57]. Finally, the model undergoes finetuning on high-resolution and high-aesthetic qualtity images to refine visual details

which was done on a data set that is not publicly accessible.

2.1.3 PixArt-Σ Enhancements

Pixart-Σ is a further development of Pixart-α which is able to generate images at higher resolution, up to 4k, and of higher quality. In this process, the weights from Pixart-α are used as initialization for Pixart-Σ introducing three main changes, namely a more powerful VAE, an improved dataset and a more efficient self-attention mechanism leveraging KV-compression.
First, Pixart-Σ utilizes a more powerful VAE encoder, specifically the VAE from SDXL [67], to obtain more expressive image features.
Second, the image dataset was extended with high resolution images up to 4k. Moreover, a more powerful model, Share-Captioner [15], was leveraged to create captions of higher quality and length. To accomodate this the number of tokens for the text embedding is increased from 120 to 300.
Third, a compression technique for the KV tokens in the self-attention layers is introduced to reduce the computational cost for high resolution images which otherwise scales quadratically with the number of tokens $O(N^2)$. The KV tokens are compressed via a convolutional 2x2 kernel operating in spatial space exploiting the redundancy of feature semantics within a $R \times R$ window. However, the Q tokens are kept uncompressed to mitigate the information loss. Consequently, the computational cost reduces from $O(N^2)$ to $O(\frac{N^2}{R^2})$. The attention operation (compare to eq. 1.80) changes to

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q f_c(K)^T}{\sqrt{d_k}}\right) f_c(V) \tag{2.1}$$

with f_c representing the compression operation.
To conclude, Pixart-Σ shows that leveraging a weaker model as a base for introduce improvements helps to accelerate the training process enormously.

2.2 Flux-dev

Flux-dev [51] is a publicly available, guidance-distilled rectified flow model based on the commercial model Flux.1 [51]. Due to the absence of a publication/paper describing the exact training process, the data set used and the specific design choices, the following section focuses on presenting the models's architecture.

2.2.1 Architecture

Flux-dev is based on a multi-modal diffusion transformer (MMDiT) [27] architecture containing 19 double-stream blocks followed by 38 single-stream blocks (see fig. 2.3) adding up to a total of 11.9 billion parameters (see tab. 2.2). For prompt processing, it leverages two distinct text encoders, namely CLIP ViT-L/14 [68] and T5-XXL [70]. For image encoding a VAE is used compressing the height H and width W of the image by a factor of eight. Each double-stream block has mainly four inputs: a guidance vector, the positional information of the image patches and the text tokens as well as the text and the image hidden states. Hereby, the text and the image are processed seperately in two distinct streams allwoing the model to apply modality-specific weights. This dual-path approach is the defining characteristic of a MMDiT. In contrast, the single-stream blocks have only three inputs as they combine the text and image hidden states and process them together.

The **guidance vector** is a combination of the timestep, a guidance parameter and the pooled vector of the clip embedding. In guidance distillation, the model is trained to simulate classifier free guidance internally. This speeds up the inference process because in contrast to classifier free guidance where the model is applied

twice per inference step only one inference pass per step is needed. The guidance is a scalar which corresponds to the guidance scale in standard classifier free guidance. First, the guidance parameter and the timestep are embedded via MLPs into high dimensional tensors which are summed together. Then the pooled clip vector for the text prompt is embedded via another MLP and added. The resulting vector, called guidance vector, is injected via modulation in every double-stream and every single-stream block.

Another input to both block types is the concatenated **positional encoding** for the text prompt and the image. First, based on the latent representation of the image, 3D coordinates for the position of every patch in the image is created. The 3D vector for the image represents the 2D positional IDs (x, y) of the patch in the image and a time dimension t for the possibility to adapt Flux-dev for videos. However, for image generation the time dimension is always set to zero. Positional IDs are also assigned to text tokens representing their sequential order and concatenated with the image IDs to jointly apply a positional embedding. In contrast to Pixart models, Flux-dev uses rotary positional embedding (RoPE) [90]. The embedded vector is then used in the attention mechanism to rotate the key and query vectors such as the spatial relationships within the image and the sequential structure of the text are preserved.

For the text embedding, the **text prompt** is tokenized into 512 tokens such that the model is capable of processing long and detailed prompts. The T5-XXL encoder embeds the tokens into high-dimensional vectors $(1, 512, 4096)$ and a linear projection is applied before it is fed into the first double-stream block. The following double-stream blocks always receive the hidden states of the text from the previous block.

The **image** $(3, H, W)$ is encoded via VAE to a latent representation $(16, h = \frac{H}{8}, w = \frac{W}{8})$. To prepare the latents for the transformer, Flux-dev packs 2×2 neighboring latent pixels into a single patch and therefore increasing the dimension from 16 to 64 during the patchify operation while simultaneously reducing the sequence length N to $\frac{h}{2} \cdot \frac{w}{2}$ $(1, N = \frac{h}{2} \cdot \frac{w}{2}, 64)$. These patches are then linearly projected into a higher dimensional embedding space before being fed into the first double block.

The output, sepcifically the text and image hidden states, of the 19th double-stream block are concatenated and used as single input to the first single-stream block.

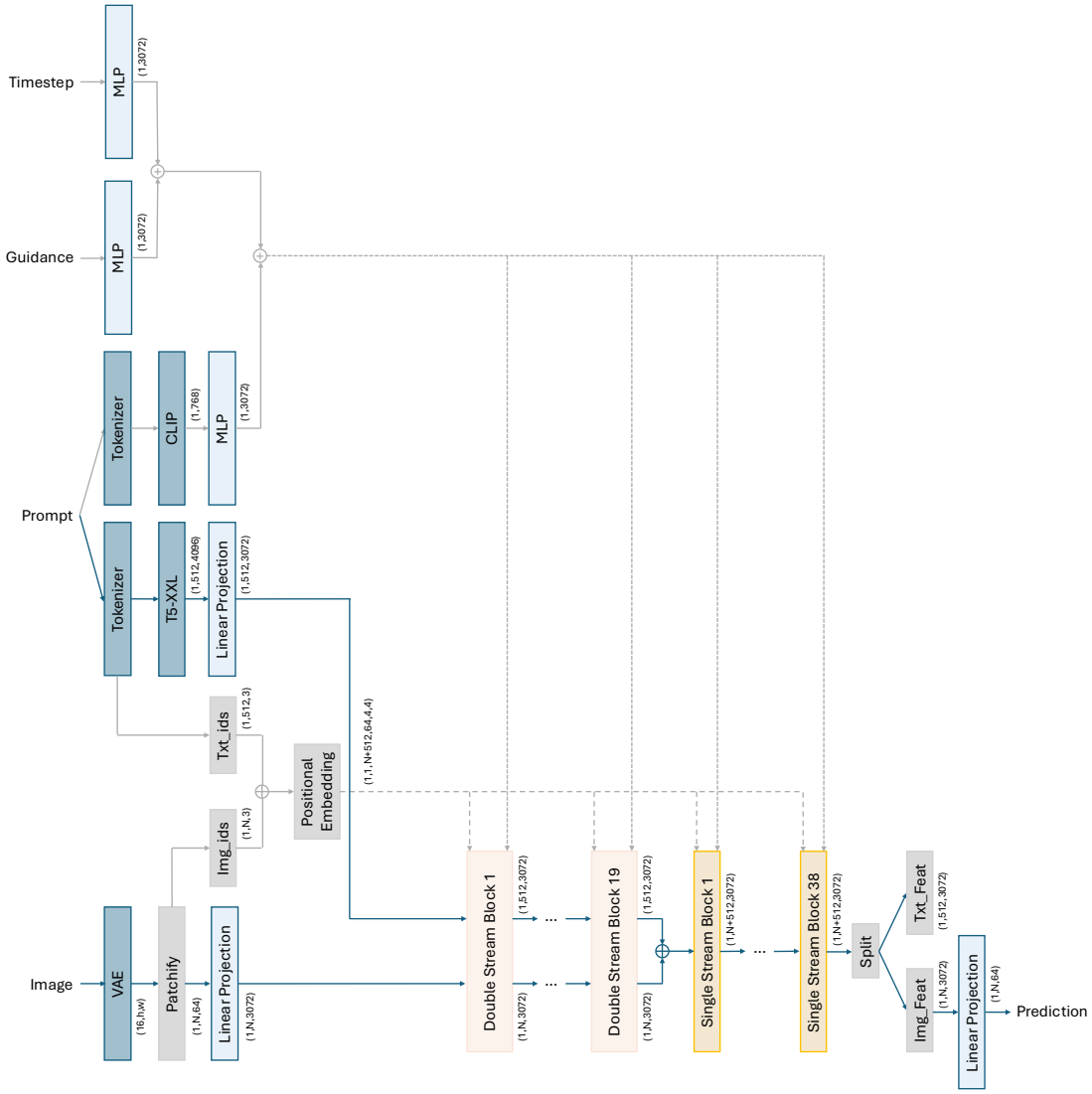


Figure 2.3: Overview Flux-dev architecture.

Double-Stream Blocks

The double-stream blocks (see fig. 2.4 on the right) are the fundamental building blocks of the first stage of the Flux-dev architecture. In contrast to other diffusion-based models [13, 67, 75] the block consists of two separate streams, which process image and text information separately and only exchange information during the attention process. The blocks consist of three main components: the modulation, which incorporates the guidance vector encoding information about the timestep, the guidance, and the pooled CLIP vector, the attention mechanism, which bidirectionally transfers information from text to image features and vice versa and the MLPs, which further process the features.

The **modulation** of the text and image features is implemented via adaLN. As shown in fig. 2.4, the guidance vector passes through two separate projection layers, which generate each two sets of scale and shift paramerters for adaLN as well as two gating parameters. The scaling and shifting of the features to incorporate the time, guidance and pooled text conditioning is applied twice, once before the attention mechanism and once before the MLP. Subsequently, the gating parameters scale the updated features before they are added to the residual stream. This technique is often applied in very deep neural networks because the linear projection layers producing the gating values are initialized with zeros. Consequently, the gating paramters start at zero, such

that the block initially acts as an identity function, which stabilizes the training.
The **attention** mechanism enables the exchange of information between the text and the image features. First, the text and image features are processed by seperate linear layers which project them into query, key and value representations and are then normalized via RMSNorm [103]. Subsequently, they are concatenated into a joint sequence. At this stage the RoPEs are integrated into the model by rotating the joint query and key vectors before the attention mechanism is applied such that self- and cross-attention take place simultaneously. Afterwards, the updated text and image features are separated and processed by individual projection layers.
The third main parts are the **MLPs**, in each stream which process the updated feature vectors further to increase the representational capacity further. Like the attention mechanism, the MLP is also enclosed by a modulation and connected via a gated residual connection.

Single-Stream Blocks

The single-stream blocks (see fig. 2.4 on the left) are the fundamental building blocks of the second stage of the Flux-dev architecture. Their primary components are the self-attention mechanism and the modulation layers. First, the concatenated image and text features X are modulated using adaLN where the shift and scale parameters are obtained by processing the guidance vector through a linear layer. Similar to the double-stream architecure, a gating mechanism is applied after the attention mechnism is executed. To optimize throughput, the model uses a fused linear projection where the hidden states for the attention mechanism and the parallel feed-forward path are computed simultaneously. The outputs of the attention mechanism and the feed-forward path are concatenated together and processed by a subsequent linear projection before the gating mechanism is applied. Finally, the updated features are combined with the residual path to form the final output of the block.

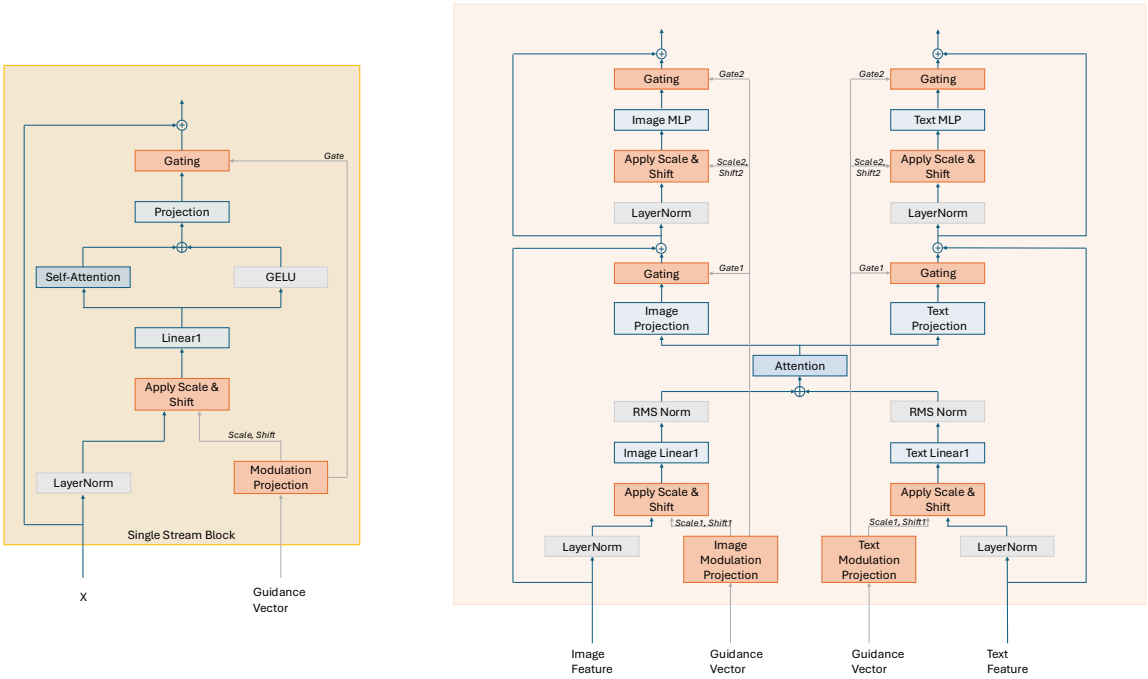


Figure 2.4: Single-stream block (left) and double-stream block (right) of Flux-dev. Graphics are adapted from [12].

Table 2.2: Number of parameters for the individual components of double-stream block from Flux-dev.

Component	**Parameters (Mio)**
Double-Stream Image Linear Layer (Attention)	28.3
Double-Stream Image MLP	75.5
Double-Stream Image Modulation	56.6
Double-Stream Image Projection	9.4
Double-Stream Text Linear Layer (Attention)	28.3
Double-Stream Text MLP	75.5
Double-Stream Text Modulation	56.6
Double-Stream Text Projection	9.4
Total Double-Stream Block	339.8
Single-Stream Fused Linear Layer (Linear1)	66.1
Single-Stream Projection	47.2
Single-Stream Modulation	28.3
Total Single-Stream Block	141.6
Total Model (Flux-dev)	11,901.4

Chapter 3

Results

sdfsdfsd

3.1 Result Distillation

Bibliography

[1] Pixart-δ.

[2] 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.

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

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

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

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

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

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

[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] 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.

[11] 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.

[12] Fuhan Cai, Yong Guo, Jie Li, Wenbo Li, Xiangzhong Fang, and Jian Chen. Fastflux: Pruning flux with block-wise replacement and sandwich training. *arXiv preprint arXiv:2506.10035*, 2025.

[13] 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.

[14] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, et al. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. *arXiv preprint arXiv:2310.00426*, 2023.

[15] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In *European Conference on Computer Vision*, pages 370–387. Springer, 2024.

[16] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

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

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

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

[25] 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.

[26] DC Dowson and BV666017 Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

[27] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first international conference on machine learning*, 2024.

[28] 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.

[29] Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: An object-focused framework for evaluating text-to-image alignment. *Advances in Neural Information Processing Systems*, 36:52132–52152, 2023.

[30] 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.

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

[32] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. A kernel method for the two-sample-problem. *Advances in neural information processing systems*, 19, 2006.

[33] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The journal of machine learning research*, 13(1):723–773, 2012.

[34] 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.

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

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

[37] 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`.

[38] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[39] 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.

[40] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[41] Sadeep Jayasumana, Srikumar Ramalingam, Andreas Veit, Daniel Glasner, Ayan Chakrabarti, and Sanjiv Kumar. Rethinking fid: Towards a better evaluation metric for image generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9307–9315, 2024.

[42] 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.

[43] 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.

[44] 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.

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

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

[47] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.

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

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

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

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

[52] 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.

[53] 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.

[54] 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.

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

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

[57] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

[58] 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.

[59] 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.

[60] 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.

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

[62] 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.

[63] 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.

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

[65] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

[66] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[67] 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.

[68] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

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

[70] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

[71] 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.

[72] 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.

[73] 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`.

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

[75] 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.

[76] 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`.

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

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] 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`.

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

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

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

[86] 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.

[87] 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.

[88] 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.

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

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

[91] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[92] Zhenxiong Tan, Songhua Liu, Xingyi Yang, Qiaochu Xue, and Xinchao Wang. Ominicontrol: Minimal and universal control for diffusion transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14940–14950, 2025.

[93] 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.

[94] 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.

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

[96] 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.

[97] Aaron Van Den Oord and Oriol Vinyals. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[98] 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.

[99] Zijie J Wang, Evan Montoya, David Munechika, Haoyang Yang, Benjamin Hoover, and Duen Horng Chau. Diffusiondb: A large-scale prompt gallery dataset for text-to-image generative models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 893–911, 2023.

[100] Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023.

[101] Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, and Yao Lu. Sana: Efficient high-resolution image synthesis with linear diffusion transformers. *arXiv preprint arXiv:2410.10629*, 2024.

[102] 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.

[103] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[104] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 3836–3847, 2023.

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