

**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
Computer Vision and Learning Lab in Heidelberg
under the supervision of
Prof. Dr. Carsten Rother
and
...

Abstract

Table of Contents

Table of Contents	IV
List of Abbreviations	VII
List of Figures	VIII
List of Tables	IX
1 Introduction	1
2 Background	2
2.1 Generative Models	2
2.1.1 Variational Autoencoder	2
2.1.2 Normalizing Flow	4
2.1.3 Generative Adversarial Network	5
2.2 Diffusion and Flow Matching Models	6
2.2.1 Denoising Diffusion Probabilistic Models	6
2.2.2 Denoising Diffusion Implicit Models	11
2.2.3 Score-Based Generative Models	12
2.2.4 Stochastic Differential Equations	13
2.2.5 Flow Matching	13
2.2.6 Latent Diffusion Models	15
2.2.7 Conditioning and Classifier-Free Guidance	16
2.3 Fundamentals of Neural Network Architectures	17
2.3.1 Transformer	17
2.3.2 Attention Mechanism	18
2.3.3 Positional Encoding	20
2.3.4 Vision Transformer	21
2.3.5 Adaptive Layer Normalization	21
2.4 Model Distillation	23
2.4.1 Knowledge Distillation	23
2.5 Evaluation Metrics	25
2.5.1 Fréchet Inception Distance	25
2.5.2 CLIP-Maximum Mean Discrepancy	25
2.5.3 Human Preference Score v2 Benchmark	26
2.5.4 GenEval Benchmark	26
2.5.5 Dense Prompt Graph Benchmark	27
3 Related Work	28

TABLE OF CONTENTS

4 Methods	30
4.1 Image Generation Models	30
4.1.1 Pixart- Σ	30
4.1.2 Flux-dev	33
4.2 Block Importance Analysis	38
4.2.1 Block Selection Criterias	38
4.2.2 Block Selection Algorithms	39
4.3 Training Data and Image Synthesis	40
4.3.1 Training Datasets	40
4.3.2 Evaluation Datasets	42
4.4 Structural Compression Strategies	43
4.4.1 Block Removal	43
4.4.2 SVD Compression	43
4.4.3 Iteratively Repeated SVD Compression	44
4.5 Knowledge Distillation Loss Functions	44
4.5.1 Normalization	44
4.5.2 Feature Mapping for Loss Computation	45
4.6 Distillation Algorithms	45
4.6.1 One-Shot Model Compression	46
4.6.2 Progressive Model Compression	46
4.6.3 Linear Progressive Compression	46
5 Experiments	48
5.1 Pixart- Σ Distillation	48
5.1.1 Block Importance Analysis	48
6 Appendix	53
6.1 Appendix A	53
6.1.1 Block Analysis	53
Bibliography	54

List of Abbreviations

adaLN Adaptive Layer Normalization

ADD Adversarial Diffusion Distillation

AI Artificial Intelligence

ALD Annealed Langevin Dynamic

CFG Classifier-Free Guidance

CKA Central Kernal Alignment

CLIP Contrastive Language- Image Pre-training

CLS Classification

CMMID Fréchet Inception Distance

CNF Continous Normalizing Flow

CNN Convolutional Neural Network

DDIM Denoising Diffusion Implicit Model

DDPM Denoising Diffusion Probabilistic Model

DiT Diffusion Transformer

DM Diffusion Model

DPG Dense Prompt Graph

DSG Dynamic Scene Graph

ELBO Evidence Lower Bound

FID Fréchet Inception Distance

GAN Generative Adversarial Network

HPSv2 Human Preference Score v2

HSIC Hilbert-Schmidt Independence Criterion

iid independent and identically distributed

Abbreviations

KL Kullback-Leibler

LADD Latent Adversarial Diffusion Distillation

LDM Latent Diffusion Model

LLM Large Language Model

LoRA Low-Rank Adaptation

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 Positional Embedding

SDE Stochastic Differential Equation

SGM Score-Based Generative Model

SOTA State-Of-The-Art

SVD Singular Value Decomposition

VAE Variational Autoencoder

VI Variational Inference

ViT Vision Transformer

List of Figures

2.1	The graphical model of DDPM from [51].	6
2.2	DDPM training [12].	10
2.3	DDPM sampling [12].	11
2.4	Rectified Flow [84].	15
2.5	Structure of a Latent Diffusion Model [108].	16
2.6	Transformer architecture from [138].	17
2.7	Self-attention mechanism based on [105].	19
2.8	Cross-attention mechanism based on [105].	19
2.9	Multi-head attention from [138].	20
2.10	Vision transformer from [32].	21
2.11	Example of how image is investigated for GenEval benchmark [40].	27
4.1	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.	31
4.2	Diffusion block architecture taken from [16].	32
4.3	Overview Flux-dev architecture.	35
4.4	Single-stream block (left) and double-stream block (right) of Flux-dev. Graphics are adapted from [13].	36
4.5	Example of LAION Dataset	40
4.6	PixArt- Σ generated dataset	41
4.7	Flux-dev generated dataset	41
4.8	Examples of Mapillary Dataset	42
4.9	Examples of Cityscapes Dataset	42
4.10	Examples of MHJQ-30k Dataset	43
4.11	Example of removal (left) and compression (right) strategy based on a simplified representation of PixArt- Σ architecture. The depictions show the case that the blocks 8,11,15,17 and 20 are removed or compressed.	44
4.12	45
5.1	Qualitative analyse of the block sensitivit�� of PixArt- Σ	49
5.2	Magnitude-base pruning vs CLIP-score	50
5.3	Magnitude-base pruning vs CMMD	50
5.4	CKA transformation intensity vs CLIP-score	51
5.5	CKA transformation intensity vs CMMD	51

List of Tables

4.1	Number of parameters for the individual components of the Transformer block and the total Pixart- Σ model.	32
4.2	Number of parameters for the individual components of double-stream block from Flux-dev.	37
4.3	Inference Parameters	41
4.4	Inference Parameters	41
5.1	Description of your table data.	52

Chapter 1

Introduction

Chapter 2

Background

2.1 Generative Models

Generative models are a class of machine learning models that are trained to learn the true data distribution to generate new synthetic data elements. They are of high relevance for text and image generation tasks. In particular, famous models for text generation, also known as large language models (LLMs) include ChatGPT [1], Claude [5], Gemini [132] or Lama [136] whereas Stable Diffusion [98, 107], Flux [71] and Pixart [14] 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) [64], generative adversarial networks (GANs) [41, 101], normalizing flows (NFs) [106], and diffusion models (DMs) [118].

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

$$\mathbf{x} = f_\theta(\mathbf{z}) \sim p_{\text{model}}(\mathbf{x}) \approx p_{\text{data}}(\mathbf{x}) \quad (2.1)$$

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 generative models in chapter 2.2 that are the most promising models today.

2.1.1 Variational Autoencoder

Autoencoder

An autoencoder [50] 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 [10, 64]. The VAE [64] 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

$$\mathcal{L}_{\text{MLP}} = \prod_{i=1}^n p_{\text{model}}(\mathbf{x}_i) . \quad (2.2)$$

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

$$\mathcal{L}_{\text{NLL}} = - \sum_{i=1}^n \log p_{\text{model}}(\mathbf{x}_i) . \quad (2.3)$$

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

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} \quad (2.4)$$

leading to the so-called evidence lower bound (ELBO) loss [10] 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

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{z}) \cdot p(\mathbf{x}|\mathbf{z}) \quad (2.5)$$

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 [135], 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) [11]. In contrast to MCMC VI reformulates the problem as an optimization task

$$q^*(\mathbf{z}) = \arg \min_{q(\mathbf{z}) \in \mathcal{Q}} \text{KL}[q(\mathbf{z}) \| p(\mathbf{z}|\mathbf{x})] \quad (2.6)$$

where \mathcal{Q} is a family of distributions. The goal is to find the proposal density $q(\mathbf{z}) \in \mathcal{Q}$ 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) [70] is used having the general form of

$$\text{KL}[p_0(\mathbf{x}) \| p_1(\mathbf{x})] = \int p_0(\mathbf{x}) \log \frac{p_0(\mathbf{x})}{p_1(\mathbf{x})} d\mathbf{x} \quad (2.7)$$

comparing two distributions p_0 and p_1 . The key for a good choice of \mathcal{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

$$\text{KL} [q(\mathbf{z}) \parallel p(\mathbf{z}|\mathbf{x})] = \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z} \quad (2.8)$$

$$= \int q(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) d\mathbf{z} \quad (2.9)$$

$$= \int q(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \log \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} d\mathbf{z} \quad (2.10)$$

$$= \mathbb{E} [\log q(\mathbf{z})] - \mathbb{E} [\log p(\mathbf{z}, \mathbf{x})] + \log p(\mathbf{x}) \quad (2.11)$$

From here, the ELBO loss is obtained by

$$\log p(\mathbf{x}) \geq \log p(\mathbf{x}) - \text{KL} [q(\mathbf{z}) \parallel p(\mathbf{z}|\mathbf{x})] \quad (2.12)$$

$$= \mathbb{E} [\log p(\mathbf{z}, \mathbf{x})] - \mathbb{E} [\log q(\mathbf{z})] \quad (2.13)$$

$$= \mathbb{E} [\log p(\mathbf{x}|\mathbf{z})] - \text{KL} [q(\mathbf{z}) \parallel p(\mathbf{z})] \quad (2.14)$$

$$= \text{ELBO} \quad . \quad (2.15)$$

Maximizing the ELBO loss is equivalent to minimizing the KL divergence from 2.6 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

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [-\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} + \underbrace{\text{KL} [q_\phi(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z})]}_{\text{Regularization}} \quad . \quad (2.16)$$

In practice, the latent distribution is often set to a standard normal distribution $p_{\text{latent}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$ and the reparameterization trick [64] 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.

2.1.2 Normalizing Flow

Normalizing flows (NFs) [106] are another class of probabilistic 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 \bar{F}_θ 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 [95]

$$p_{\text{latent}}(\mathbf{z}) = p_{\text{model}}(\mathbf{x}) \left| \frac{\partial F_\theta(\mathbf{z})}{\partial \mathbf{z}} \right| = p_{\text{model}}(F_\theta(\mathbf{z})) \left| \frac{\partial F_\theta(\mathbf{z})}{\partial \mathbf{z}} \right| \quad (2.17)$$

for the latent distribution and

$$p_{\text{model}}(\mathbf{x}) = p_{\text{latent}}(\mathbf{z}) \left| \frac{\partial F_\theta(\mathbf{z})}{\partial \mathbf{z}} \right|^{-1} = p_{\text{latent}}(\bar{F}_\theta(\mathbf{x})) \left| \frac{\partial \bar{F}_\theta(\mathbf{x})}{\partial \mathbf{x}} \right| \quad . \quad (2.18)$$

It is worth noting that computing the likelihood (2.18) includes computing the determinant of a jacobian matrix. Therefore, NFs rely on carefully designed invertible layers (e.g., affine coupling layers [31]) 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.

2.1.3 Generative Adversarial Network

In contrast to VAEs, NFs or DMs, GANs [41] 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

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (2.19)$$

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 [61, 82, 113] 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 [80, 110]. 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 [134], 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.

2.2 Diffusion and Flow Matching Models

Diffusion models [51, 108, 119, 120, 122, 126] are probabilistic generative models that have achieved remarkable success in image generation tasks [2, 71], 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 [41, 64, 106], 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, [108] 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 [24]: Denoising diffusion probabilistic models (DDPMs) [51, 119] and its further development denoising diffusion implicit models (DDIMs, score-based generative models (SGMs) [122] and stochastic differential equations (SDEs) [126].

Recently, a generalization of DMs was proposed, known as flow matching or Flow-based models [81, 84]. 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 [71] or Sana [142].

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.

2.2.1 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 [128].

In DDPMs [51] 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. 2.1. Let $\mathbf{x}_0, \dots, \mathbf{x}_T$ 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. 2.1 and is given by $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ where no learned parameters are required.

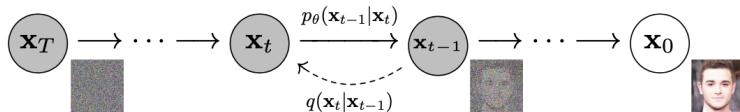


Figure 2.1: The graphical model of DDPM from [51].

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 [51], a cosine [92] or a learnable [63] 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

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) = K(\mathbf{x}_t; \mathbf{x}_{t-1}, \beta_t) \quad (2.20)$$

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

$$q(\mathbf{x}_t | \mathbf{x}_{t-1}) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \quad (2.21)$$

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

$$\mathbf{x}_t = \sqrt{\alpha_t} \mathbf{x}_{t-1} + \sqrt{1 - \alpha_t} \mathbf{z}_t \quad (2.22)$$

$$= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{\alpha_t(1 - \alpha_{t-1})} \mathbf{z}_{t-1} + \sqrt{1 - \alpha_t} \mathbf{z}_t \quad (2.23)$$

$$= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \mathbf{z}_{t-1:t} \quad (2.24)$$

$$= \dots \quad (2.25)$$

$$= \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \mathbf{z}_{0:t}. \quad (2.26)$$

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

$$q(\mathbf{x}_t | \mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}\right). \quad (2.27)$$

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

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{(1 - \bar{\alpha}_t)} \boldsymbol{\epsilon} \quad (2.28)$$

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 [37] 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

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t) = q(\mathbf{x}_t | \mathbf{x}_{t-1}) \frac{q(\mathbf{x}_{t-1})}{q(\mathbf{x}_t)}. \quad (2.29)$$

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

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad (2.30)$$

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

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \quad . \quad (2.31)$$

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. 2.1.1 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 [95] and [51]. 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}

$$-\mathbb{E}_{q(\mathbf{x}_0)} [\log p_\theta(\mathbf{x}_0)] \quad (2.32)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) \right) \right] \quad (2.33)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T p(\mathbf{x}_T) q(\mathbf{x}_1, \dots, \mathbf{x}_T | \mathbf{x}_0) \prod_{t=1}^T \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} \right) \right] \quad (2.34)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T | \mathbf{x}_0)} \left[p(\mathbf{x}_T) \prod_{t=1}^T \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} \right] \right] \quad (2.35)$$

using Jensen's inequality

$$-\mathbb{E}_{q(\mathbf{x}_0)} [\log p_\theta(\mathbf{x}_0)] \leq -\mathbb{E}_{q(\mathbf{x}_0)} \left[\mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T | \mathbf{x}_0)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^T \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} \right) \right] \right] \quad (2.36)$$

$$= -\mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^T \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} \right) \right] \quad (2.37)$$

$$= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=1}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} \right] \quad (2.38)$$

$$=: \mathcal{L}_{\text{DDPM}} \quad (2.39)$$

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

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=2}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_t | \mathbf{x}_{t-1})} - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \quad (2.40)$$

$$= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=2}^T \log \left(\frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} \cdot \frac{q(\mathbf{x}_{t-1} | \mathbf{x}_0)}{q(\mathbf{x}_t | \mathbf{x}_0)} \right) - \log \frac{p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}{q(\mathbf{x}_1 | \mathbf{x}_0)} \right] \quad (2.41)$$

$$= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log \frac{p(\mathbf{x}_T)}{q(\mathbf{x}_T | \mathbf{x}_0)} - \sum_{t=2}^T \log \frac{p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)}{q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0)} - \log p_\theta(\mathbf{x}_0 | \mathbf{x}_1) \right] \quad (2.42)$$

$$= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[\text{KL}(q(\mathbf{x}_T | \mathbf{x}_0) \| p(\mathbf{x}_T)) + \sum_{t=2}^T \text{KL}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t)) - \log p_\theta(\mathbf{x}_0 | \mathbf{x}_1) \right] . \quad (2.43)$$

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

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_0, \mathbf{x}_t)} [\text{KL}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))] - \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} [\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)] + \text{const} \quad (2.44)$$

$$\approx \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_0, \mathbf{x}_t)} [\text{KL}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))] + \text{const} \quad (2.45)$$

assuming that the term $\mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} [\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)]$ is negligible. Note that in eq. 2.45 only Gaussians, $p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$ with $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$ and $q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \hat{\mu}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$ are compared in the KL-divergence meaning there is a closed-form solution. The loss for the diffusion model boils down to

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_0, \mathbf{x}_t)} \left[\frac{1}{2\sigma^2} \|\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2 \right] . \quad (2.46)$$

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

$$\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0 + \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t \quad (2.47)$$

$$\tilde{\beta}_t := \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t . \quad (2.48)$$

However, instead of predicting the mean, diffusion models are more often trained on predicting the noise ϵ_θ because [51] empirically found that this lead to more stable training

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^T \frac{1}{2\tilde{\beta}_t} \frac{(1 - \alpha_t)^2}{\alpha_t (1 - \bar{\alpha}_t)} \cdot \mathbb{E}_{q(x_t | x_0)} [\|\epsilon_\theta(x_t, t) - \epsilon_t\|_2^2] \quad (2.49)$$

which is achieved by reparameterization of eq. 2.46 via

$$\hat{\mu}(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_t \right) . \quad (2.50)$$

In addition, they found empirically that ignoring the weighting term of eq. 2.49 worked better

$$\mathcal{L}_{\text{DDPM}} = \sum_{t=2}^T \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)} [\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \boldsymbol{\epsilon}_t\|_2^2] . \quad (2.51)$$

DDPM Sampler

Algorithm 1 and fig. 2.2 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 $[0, \dots, T]$ and Gaussian noise is drawn according to $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Using eq. 2.28 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. 2.51) 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 [128]

```

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

```

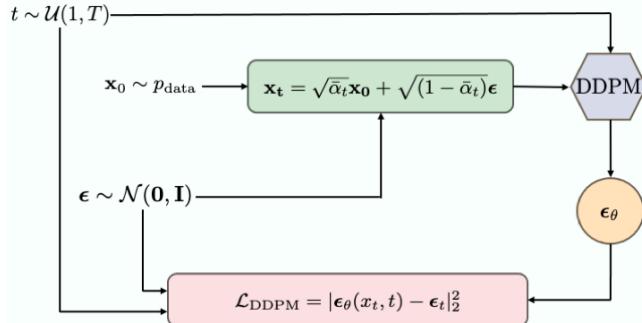


Figure 2.2: DDPM training [12].

Algorithm 2 and fig. 2.3 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

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z} \quad (2.52)$$

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

$$\mathbf{x}_0 = \frac{1}{\sqrt{\alpha_1}} \left(\mathbf{x}_1 - \frac{1 - \alpha_1}{\sqrt{1 - \bar{\alpha}_1}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_1, 1) \right) + \sigma_1 \mathbf{z} \quad (2.53)$$

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 [128]

```

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

```

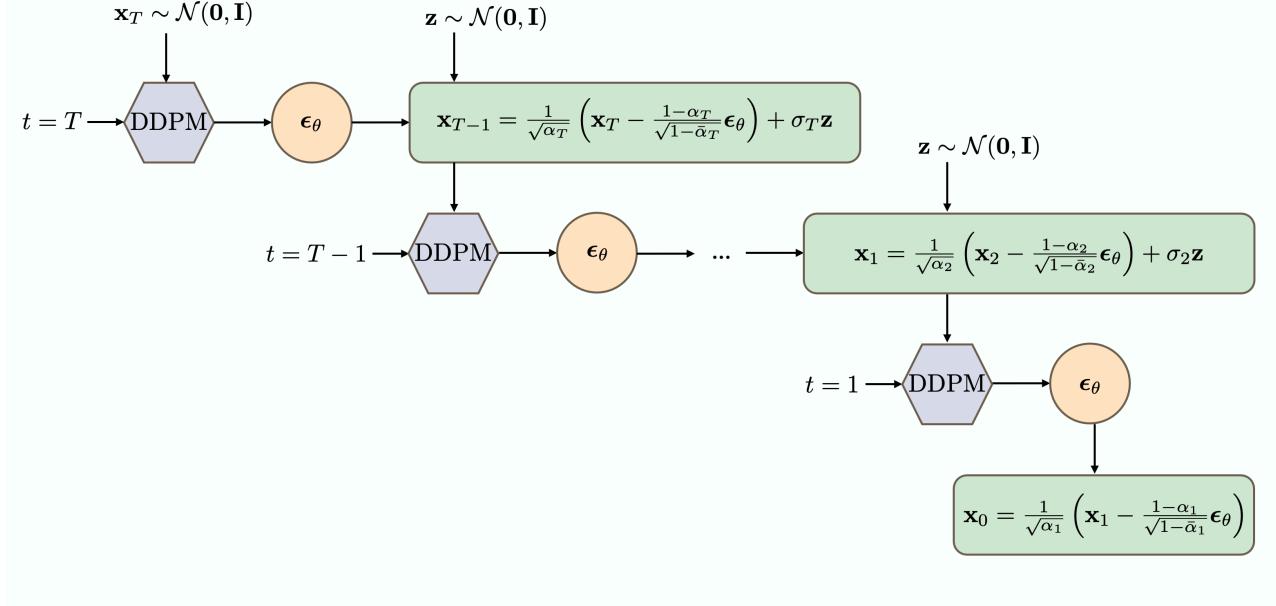


Figure 2.3: DDPM sampling [12].

2.2.2 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) [120]. Unlike DDPM, DDIM defines the joint distribution as

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = q(\mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=2}^T q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \quad . \quad (2.54)$$

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 [120] parameterize the reverse process as

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N} \left(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \frac{\mathbf{x}_t - \sqrt{\bar{\alpha}_t} \mathbf{x}_0}{\sqrt{1 - \bar{\alpha}_t}}, \sigma_t^2 \cdot \mathbf{I} \right) \quad . \quad (2.55)$$

The corresponding sampling equation becomes

$$\mathbf{x}_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}} (\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_\theta(\mathbf{x}_t, t))}_{\text{Predicted } \mathbf{x}_0} + \underbrace{\sqrt{1 - \bar{\alpha}_{t-1} - \sigma_t^2} \cdot \epsilon_\theta(\mathbf{x}_t, t)}_{\text{Direction of } \mathbf{x}_t} + \underbrace{\sigma_t \cdot \mathbf{z}}_{\text{Noise}} \quad (2.56)$$

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 ϵ_θ . 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

$$\sigma_t = \sqrt{\frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t}} \sqrt{1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}} \quad (2.57)$$

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.

2.2.3 Score-Based Generative Models

This discussion is mainly based on [24, 143]. Score-Based Generative Models (SGMs) [123, 124] 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 [42, 125, 127]. Here, the one closest to the DDPM approach is presented [123]. 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

$$\mathbb{E}_{T \sim \mathcal{U}[1, T], \mathbf{x}_0 \sim q(\mathbf{x}_0), \mathbf{x}_t \sim q(\mathbf{x}_t | \mathbf{x}_0)} [\lambda(t) \sigma_t^2 \| \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t) - s_\theta(\mathbf{x}_t, t) \|^2] \quad (2.58)$$

$$= \mathbb{E}_{T \sim \mathcal{U}[1, T], \mathbf{x}_0 \sim q(\mathbf{x}_0), \mathbf{x}_t \sim q(\mathbf{x}_t | \mathbf{x}_0)} [\lambda(t) \sigma_t^2 \| \nabla_{\mathbf{x}_t} \log q(\mathbf{x}_t | \mathbf{x}_0) - s_\theta(\mathbf{x}_t, t) \|^2] + \text{const} \quad (2.59)$$

$$= \mathbb{E}_{T \sim \mathcal{U}[1, T], \mathbf{x}_0 \sim q(\mathbf{x}_0), \mathbf{x}_t \sim q(\mathbf{x}_t | \mathbf{x}_0)} \left[\lambda(t) \left\| -\frac{\mathbf{x}_t - \mathbf{x}_0}{\sigma_t} - \sigma_t s_\theta(\mathbf{x}_t, t) \right\|^2 \right] + \text{const} \quad (2.60)$$

$$= \mathbb{E}_{T \sim \mathcal{U}[1, T], \mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} [\lambda(t) \|\epsilon + \sigma_t s_\theta(\mathbf{x}_t, t)\|^2] + \text{const} \quad . \quad (2.61)$$

In eq. 2.61 the relation to DDPM loss becomes visible by identifying $\epsilon(\mathbf{x}_t, t) = -\sigma_t s_\theta(\mathbf{x}_t, t)$ [143], showing that learning to predict noise is equivalent to learning the score function.

For sampling a new element from the data distribution [123] proposed Annealed Langevin Dynamics (ALD) which produces samples by only using the score function being approximated by the learned neural network

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \alpha \nabla_{\mathbf{x}} \log p(\mathbf{x}_{t-1}) + \sqrt{2\alpha} \mathbf{z}_t, \quad 1 \leq t \leq T \quad (2.62)$$

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

2.2.4 Stochastic Differential Equations

This discussion is mainly based on [24, 143]. Stochastic Differential Equations (SDEs) [127] 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

$$d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)dt + g(t)d\mathbf{w} \quad (2.63)$$

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. 2.63 can be reversed leading to the reversed-time SDE given by

$$d\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log q(\mathbf{x}_t)] dt + g(t)d\tilde{\mathbf{w}} \quad (2.64)$$

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. 2.58) to continuous time

$$\mathbb{E}_{t \sim \mathcal{U}[0, T], \mathbf{x}_0 \sim q(\mathbf{x}_0), \mathbf{x}_t \sim q(\mathbf{x}_t | \mathbf{x}_0)} \left[\lambda(t) \|s_\theta(\mathbf{x}_t, t) - \nabla_{\mathbf{x}_t} \log q_t(\mathbf{x}_t)\|^2 \right] \quad (2.65)$$

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. 2.64 has to be computed using the score neural network and a SDE solver. Another finding of [127] is that for eq. 2.64 an ODE exists having the same marginal as the SDE being known as probability flow ODE in the literature. It's given by the deterministic version of eq. 2.64

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - \frac{1}{2}g(t)^2 \nabla_{\mathbf{x}} \log q(\mathbf{x}_t) \right] dt \quad (2.66)$$

Both reversed-SDE or probability flow ODE can be used to generate new samples using SDE solvers [59, 127] or respectively ODE solvers [60, 86, 127, 151].

2.2.5 Flow Matching

The concept of flow matching [81] builds upon continuous normalizing flows (CNFs) [18]. The key elements of CNFs are the probability path $p_t(x)$ connecting two distributions p_0 and p_1 , its corresponding vector field $\mathbf{u}_t(\mathbf{x})$ and the flow $\phi_t(\mathbf{x})$ that is generated by the vector field and given by the ODE

$$\frac{d}{dt} \phi_t(\mathbf{x}) = \mathbf{u}_t(\phi_t(\mathbf{x})), \phi_0(\mathbf{x}) = \mathbf{x} \quad . \quad (2.67)$$

In flow matching, the vector field $\mathbf{u}_t(\mathbf{x})$ is approximated by a neural network $\mathbf{v}_{\theta,t}(\mathbf{x})$ to learn the probability path and enable mapping between both probability distributions. The corresponding flow matching loss function is

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t \sim \mathcal{U}(0, 1), \mathbf{x} \sim p_t(\mathbf{x})} \left[\|\mathbf{v}_{\theta,t}(\mathbf{x}) - \mathbf{u}_t(\mathbf{x})\|^2 \right] \quad . \quad (2.68)$$

However, the vector field $\mathbf{u}_t(\mathbf{x})$ is unknown which is why [81] constructed a sample-based vector field $\mathbf{u}_t(\mathbf{x} | \mathbf{x}_1)$ with $\mathbf{x}_1 \sim q(\mathbf{x})$, where $q(\mathbf{x})$ is the real, unknown data distribution from which only individual samples are known. They showed that this approach leads to identical gradients and is tractable.

Concretely, first the marginal probability path $p_t(\mathbf{x})$ is constructed by a mixture of simpler sample-based conditional probability paths $p_t(\mathbf{x} | \mathbf{x}_1)$

$$p_t(\mathbf{x}) = \int p_t(\mathbf{x} | \mathbf{x}_1) q(\mathbf{x}_1) d\mathbf{x}_1 \quad (2.69)$$

and its corresponding marginalized vector field is defined as

$$\mathbf{u}_t(\mathbf{x}) = \int \mathbf{u}_t(\mathbf{x} | \mathbf{x}_1) \frac{p_t(\mathbf{x} | \mathbf{x}_1) q(\mathbf{x}_1)}{p_t(\mathbf{x})} d\mathbf{x}_1 . \quad (2.70)$$

The trick is not to use the marginal distributions, because the integrals are still intractable. Instead, the loss function (eq. 2.68) is reformulated using the conditional vector field

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t \sim \mathcal{U}(0,1), q(\mathbf{x}_1), \mathbf{x} \sim p_t(\mathbf{x} | \mathbf{x}_1)} \left[\|\mathbf{v}_{\theta,t}(\mathbf{x}) - \mathbf{u}_t(\mathbf{x} | \mathbf{x}_1)\|^2 \right] . \quad (2.71)$$

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

There are many different possibilities to define the conditional vector field $\mathbf{u}_t(\mathbf{x} | \mathbf{x}_1)$. One choice which demonstrates the advantage of this approach over the diffusion process like defined in sec. 2.2.1 is to use optimal transport displacement interpolation. In this case, the mean $\mu_t(\mathbf{x}) = t\mathbf{x}_1$ and the standard deviation $\sigma_t(\mathbf{x}) = 1 - (1 - \sigma_{\min})t$ for the probability path are defined linearly in time resulting in the conditional vector field

$$\mathbf{u}_t(\mathbf{x} | \mathbf{x}_1) = \frac{\mathbf{x}_1 - (1 - \sigma_{\min})\mathbf{x}}{1 - (1 - \sigma_{\min})t} \quad (2.72)$$

which leads to more straight trajectories of the probability mass. These lines are computationally more efficient compared to the diffusion process with its complex, curved trajectories. The resulting flow can be solved with significantly fewer steps during inference while maintaining high image quality.

Rectified Flow

While standard flow matching connects noise and data via the probability path p_t , it typically relies solely on independent coupling, where noise-image pairs are matched randomly. This often leads to intersection of trajectories during the training phase. The consequence is that the neural network $\mathbf{v}_{\theta,t}(\mathbf{x})$ is presented with two different, contradictory target velocities at the specific location of the crossing. Therefore, [84] proposed an iterative algorithm called Reflow to untangle and straighten the trajectories. This is achieved by iteratively refining the learned rectified flow 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 rectified flow is defined via the ODE

$$d\mathbf{x}_t = \mathbf{v}_{\theta,t}(\mathbf{x}_t) dt \quad (2.73)$$

where $\mathbf{v}_{\theta,t}$ is the vector field approximated by a neural network. The flow should learn to follow the straight trajectory $\mathbf{x}_1 - \mathbf{x}_0$. This is enforced by the learning objective

$$\min_{\mathbf{v}} \int_0^1 \mathbb{E} \left[\|(\mathbf{x}_1 - \mathbf{x}_0) - \mathbf{v}_{\theta,t}(\mathbf{x}_t)\|^2 \right] dt \quad (2.74)$$

where $\mathbf{x}_t = t\mathbf{x}_1 + (1 - t)\mathbf{x}_0$ is the linear interpolation between \mathbf{x}_0 and \mathbf{x}_1 . As depicted in fig. 2.4 (a) crossing trajectories occur when random coupling between samples from p_0 and p_1 is applied. Fig. 2.4 (b) shows that the connections of samples from p_0 and p_1 using the trained model are unwired due to the fact that the vector field of an ODE is uniquely defined at every point. However, the trajectories are not necessarily straight yet. To obtain straight trajectories, new couples $(\mathbf{x}_0^1, \mathbf{x}_1^1)$ are generated where $\mathbf{x}_0^1 \sim p_0$ and \mathbf{x}_1^1 is the solution obtained by applying the learned ODE. These newly acquired pairs are used for retraining the model (see fig. 2.4 (c) and (d)). The advantage is that the trajectories between the new pairs $(\mathbf{x}_0^1, \mathbf{x}_1^1)$ that the network encounters during training are straighter and non-overlapping. By iteratively training and sampling new pairs $(\mathbf{x}_0^k, \mathbf{x}_1^k)$, the

learned trajectories become increasingly straight and have a lower transportation cost. This iterative approach is called Reflow (see alg. 8)

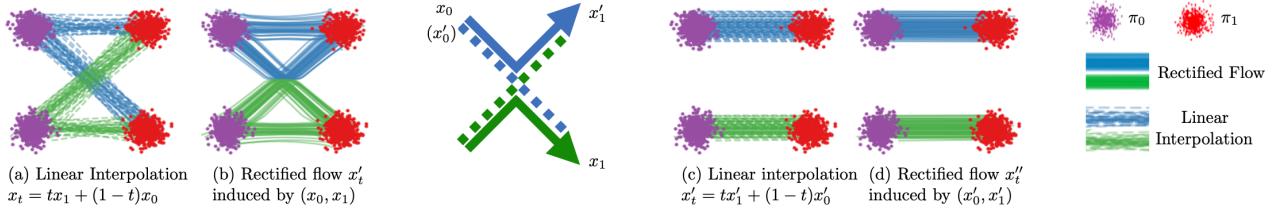


Figure 2.4: Rectified Flow [84].

Algorithm 3 Rectified Flow: Main Algorithm [84]

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 $v_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ with parameter θ .

Training: $\hat{\theta} = \arg \min_{\theta} \mathbb{E} [\|x_1 - x_0 - v(tx_1 + (1-t)x_0, t)\|^2]$, 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 $x_1^k \approx \hat{T}(x_0^k)$.

2.2.6 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 [108] and are widely used nowadays [14, 71, 98, 107]. With LDMs, the diffusion process is no longer performed in pixel space but in a compressed latent space (see fig. 2.5), which significantly reduces the computational effort. To perform the compression, an autoencoder is used, whereby additional regularizations are often applied such as KL-regularization [64] or VQ-regularization [137]. 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. 2.51)

$$\mathcal{L}_{\text{LDM}} = \sum_{t=2}^T \mathbb{E}_{q(\mathbf{z}_t | \mathbf{z}_0)} [\|\epsilon_\theta(\mathbf{z}_t, t) - \epsilon_t\|_2^2] \quad (2.75)$$

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 denoising process. The completely denoised latent is then fed to the decoder \mathcal{D} , which transforms it back to pixel space. Fig. 2.5 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 2.2.7.

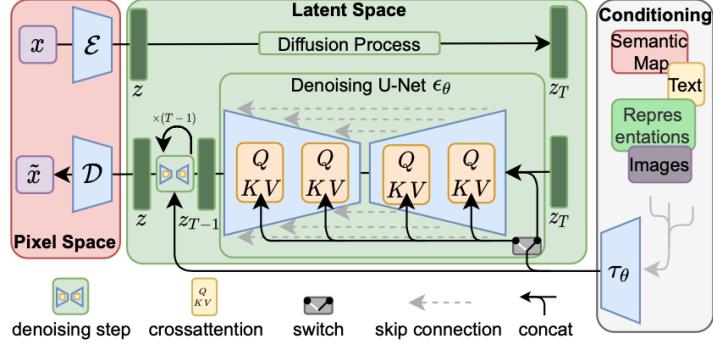


Figure 2.5: Structure of a Latent Diffusion Model [108].

2.2.7 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 [150], or images [131] 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 $\epsilon_\theta(\mathbf{x}_t, t)$ to $\epsilon_\theta(\mathbf{x}_t, \mathbf{y}, t)$. Cross-attention layers (see section 2.3.2) [138] 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. 2.5, or within the transformer blocks of modern models like Pixart [14]. 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 2.3.3) 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) [52]. This method builds upon classifier guidance [30], 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 $\epsilon_\theta(\mathbf{x}_t, t, \mathbf{y})$ and unconditional $\epsilon_\theta(\mathbf{x}_t, t, \mathbf{y} = \emptyset)$ 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

$$\tilde{\epsilon}_\theta = (1 + w)\epsilon_\theta(\mathbf{x}_t, t, \mathbf{y}) - w\epsilon_\theta(\mathbf{x}_t, t, \mathbf{y} = \emptyset) \quad . \quad (2.76)$$

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.

2.3 Fundamentals of Neural Network Architectures

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

2.3.1 Transformer

The introduction of the transformer architecture [138] was a decisive moment for the whole AI community. It significantly pushed the development in natural language and image processing forward. Therefore, it is no surprise that the latest models are based on this architecture type [2, 5, 71, 132].

The transformer was introduced as an alternative to recurrent neural networks [109] for processing sequential data like text. In the original paper, the transformer architecture leverages an encoder-decoder structure, however, there are many variants like decoder-only [1, 136] or encoder-only transformer [29].

Transformers, as introduced in the original paper (Fig. 2.6), 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 [47] 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.

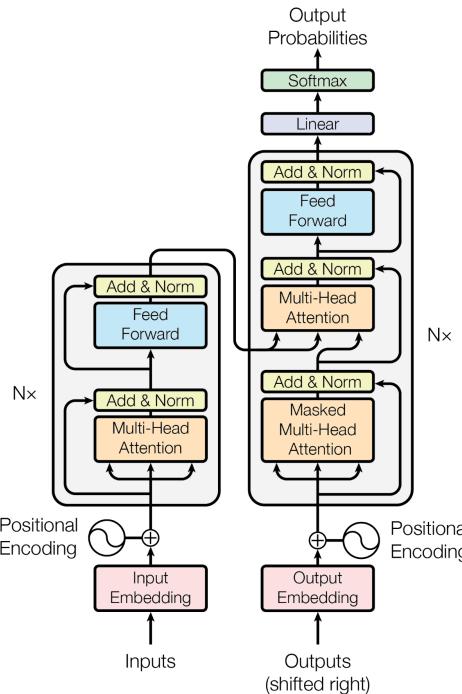


Figure 2.6: Transformer architecture from [138].

2.3.2 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 [138] 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. 2.7) and cross-attention (fig. 2.8). 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

$$Q = X \cdot W_q^T \quad (2.77)$$

$$K = X \cdot W_k^T \quad (2.78)$$

$$V = X \cdot W_v^T \quad (2.79)$$

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

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) \quad (2.80)$$

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

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (2.81)$$

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

$$Z = \text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V \quad (2.82)$$

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 [20, 22, 26, 67].

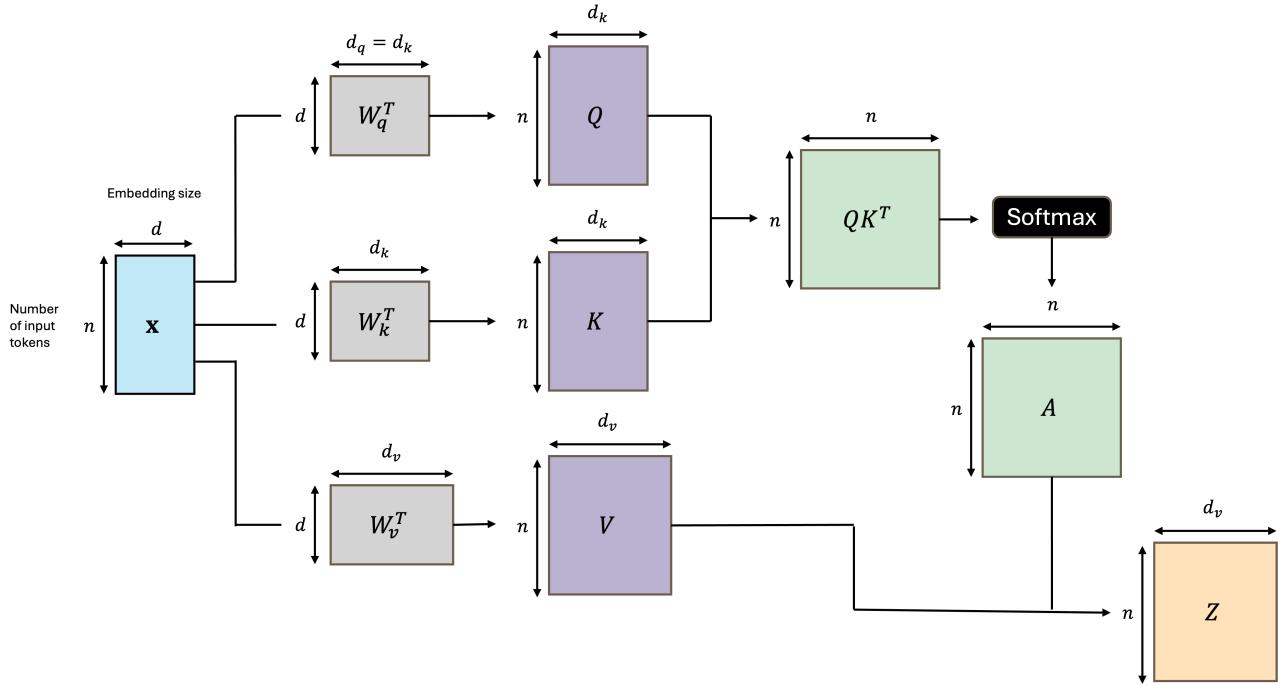


Figure 2.7: Self-attention mechanism based on [105].

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.

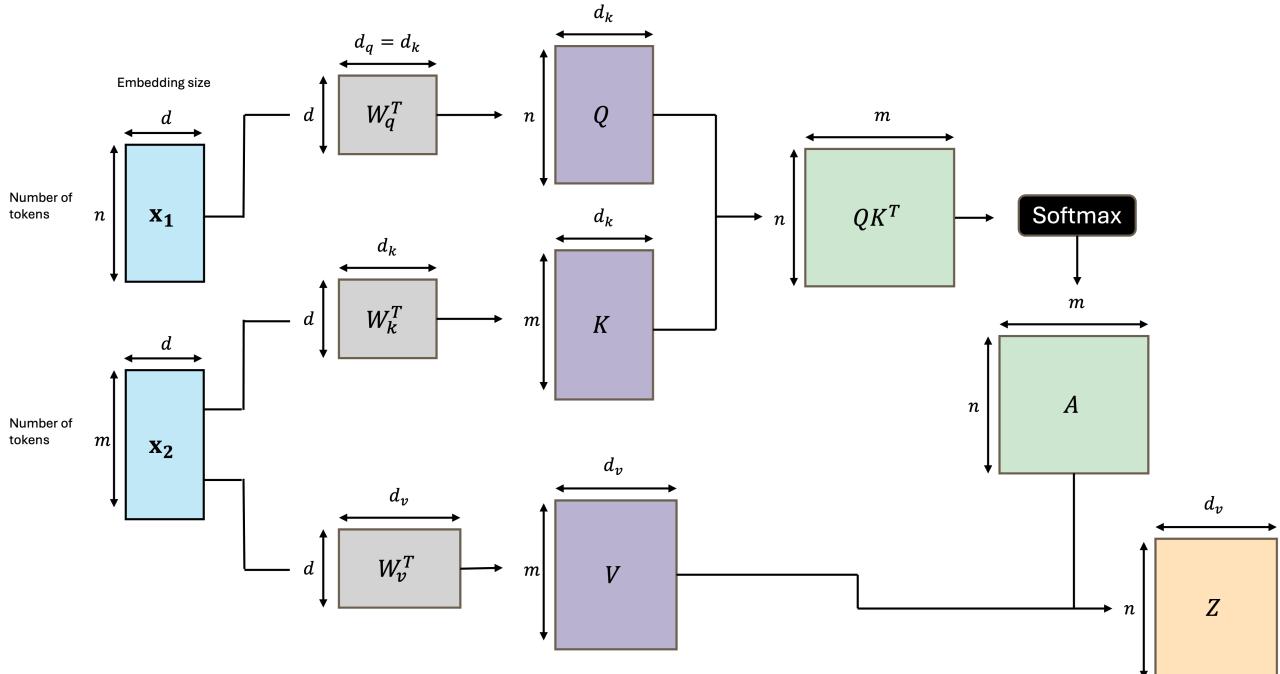


Figure 2.8: Cross-attention mechanism based on [105].

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 $i = 1, \dots, N$, and the attention mechanism is executed. Each attention that is carried out is designated as a head _{i} . The results of the N attentions are concatenated and combined with another linear projection $W^O \in \mathbb{R}^{Nd_v \times d}$.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_N) W^O \quad (2.83)$$

$$\text{head}_i = \text{Attention}(XW_{q_i}^T, XW_{k_i}^T, XW_{v_i}^T) \quad (2.84)$$

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.

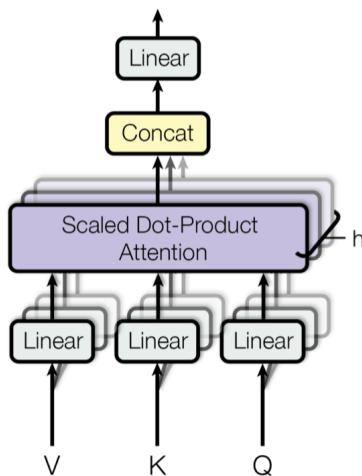


Figure 2.9: Multi-head attention from [138].

2.3.3 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 [77, 93, 129] versus absolute [68, 78, 85, 138] positional embedding or deterministic [68, 78, 93, 129, 138] versus learnable [77, 85] positional embedding.

Sinusoidal Positional Encoding

Sinusoidal positional encoding is used in the original transformer paper [138] 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.

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad (2.85)$$

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \quad (2.86)$$

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.

2.3.4 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 [55, 94, 103], but for a long time, convolutional neural networks (CNNs) [104] 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 pre-defined by its architecture, which makes a sufficiently large dataset essential to achieve competitive performance.

The introduction of vision transformer (ViT) [32] 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.

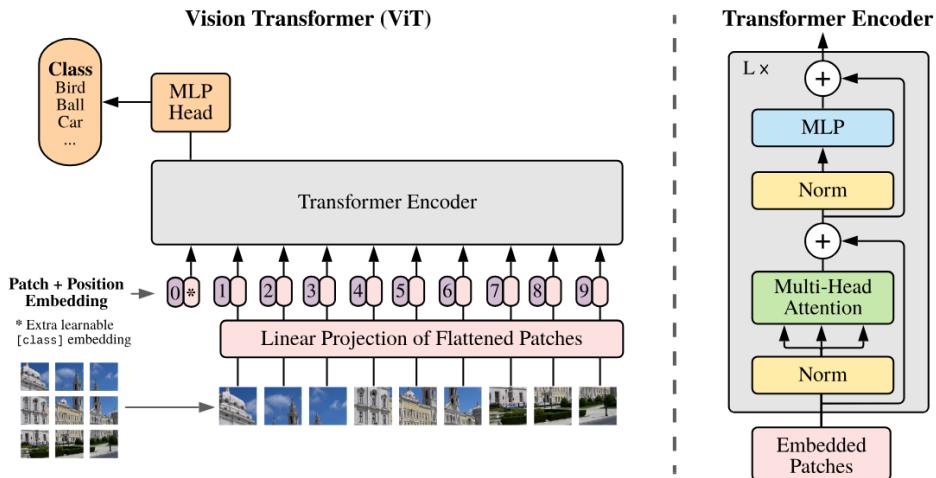


Figure 2.10: Vision transformer from [32].

2.3.5 Adaptive Layer Normalization

Layer normalization [7] was introduced as an alternative to batch normalization [56] 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 output 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

$$\hat{\mathbf{z}}_l = \gamma_l \frac{\mathbf{z}_l - \mu_l(\mathbf{z}_l)}{\sigma_l(\mathbf{z}_l)} + \eta_l \quad . \quad (2.87)$$

A further development of layer normalization occurred when [97] 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 [96].

2.4 Model Distillation

Diffusion models excel in generating high-quality and diverse images, showcasing remarkable capabilities in image synthesis. However, their main disadvantage is their prolonged inference time which stems primarily from two factors. First, diffusion models rely on an iterative generation process, removing noise step-by-step. To obtain high-quality images often 50-100 steps are required. Second, recent diffusion models like Flux-dev [71] are remarkably large, with billions of parameters, making each step computationally and memory-intensive. These issues limit their use in real-time applications or in memory-constraint environments, such as edge devices.

Diffusion model distillation addresses these challenges. The principal idea of most distillation methods is to transfer the knowledge and image generation capacity from a fully trained diffusion model, often referred to as teacher model, to a target model, often referred to as student model. The research landscape reveals two complementary strategies. First, accelerating inference by reducing the number of denoising steps, and second, creating more compact models through model size reduction. The first strategy, reducing the number of inference steps, is predominant in the literature and several different approaches such as adversarial distillation [112, 114], progressive distillation [79, 111], distribution matching distillation [144, 145], consistency models [87, 121] and guided distillation [90] were proposed. The second strategy, focusing on structural-based knowledge distillation [49] is very suitable for reducing the model complexity and parameter count.

All these distillation methods have in common that there is an efficiency-quality trade-off. The smaller the student model gets and the less inference steps are used the lower is the image quality.

This thesis focuses on reducing the model size of the models of Pixart- Σ and Flux-dev using the concept of knowledge distillation which is presented in the following section in detail.

2.4.1 Knowledge Distillation

The concept of knowledge distillation [49] was originally introduced for training a small model, referred to as the student, on classification tasks such as MNIST [72]. In this framework, a larger model with strong generalization capabilities, referred to as the teacher, serves as guidance. It was shown, that computing the loss between the probability outputs of the teacher and the student enables much more efficient training than relying solely on the original class labels. In such a setting, the small model can be trained on less data or with a higher learning rate. As a possible explanation, the authors hypothesized that even the very small probabilities the teacher assigns to incorrect classes, and particularly their relative magnitudes, carry valuable information for the student. For example, if a "2" from the MNIST dataset is to be classified, it is valuable information to know whether it resembles a "3" or a "5". When only hard targets, e.g., one-hot labels, are used, this relational information is lost. To make this information accessible to the student, one must deviate from the standard temperature $T = 1$ of the softmax function to soften the probabilities. Specifically, both teacher and student use temperature-scaling softmax($\frac{z_i}{T}$) where $T > 1$ making the probability distribution flatter and revealing more information about class similarities. In general, it has been shown that a combination of the distillation loss \mathcal{L}_{KD} , calculated using the outputs from the teacher, and the task loss $\mathcal{L}_{\text{Task}}$, calculated with the hard labels, is advantageous

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{KD}} \mathcal{L}_{\text{KD}} \quad . \quad (2.88)$$

This idea has been adapted to train smaller versions of diffusion [62, 74, 149] and flow [13, 25, 35, 88, 133, 153] models. Here, the original model with its pretrained weights acts as the teacher, while the student model is obtained by removing individual parts of the teacher's architecture. Often the student is initialized with the weights of the teacher. The general idea in these approaches is to use the intermediate features as well as the final output as learning signal from the teacher to the student via MSE loss. The mapping between teacher and student layers is established by selecting a subset of the teacher's blocks that corresponds to the reduced depth of the student architecture. If the architectures and therefore the dimensions of the intermediate features differ

between teacher and student model, linear projection matrices ϕ_i are used to project the student's features to the same dimension as the teacher's features. Furthermore, due to different magnitudes in different layers, a weighting factor w_i for each layer i is applied such that the contribution of the different layers are in the same range.

$$\mathcal{L}_{\text{FeatKD}} = \sum_{i \in \text{Layers}} w_i \|F_i^T - \phi_i(F_i^S)\|^2 . \quad (2.89)$$

Additionally, the original diffusion loss $\mathcal{L}_{\text{Task}}$ is taken into account too such that the final loss is given by

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{OutKD}} \mathcal{L}_{\text{OutKD}} + \lambda_{\text{FeatKD}} \mathcal{L}_{\text{FeatKD}} \quad (2.90)$$

where $\mathcal{L}_{\text{Task}}$ is the standard diffusion loss, $\mathcal{L}_{\text{FeatKD}}$ is the loss computed between the intermediate features and $\mathcal{L}_{\text{OutKD}}$ is the loss computed between the final predictions of the student and the teacher model.

2.5 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 following, the metrics used in this work are presented: Fréchet Inception Distance (FID) [48], CLIP-Maximum Mean Discrepancy (CMMMD) [58], Human Preference Score v2 (HPSv2) [141]

2.5.1 Fréchet Inception Distance

The FID [48] 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 [130], 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 ($\mu_{\text{model}}, \Sigma_{\text{model}}$) and the features from the original data ($\mu_{\text{data}}, \Sigma_{\text{data}}$) are computed and compared by the Fréchet Distance [33]

$$\text{FID} = \|\mu_{\text{data}} - \mu_{\text{model}}\|_2^2 + \text{Tr} \left(\Sigma_{\text{model}} + \Sigma_{\text{data}} - 2 (\Sigma_{\text{model}} \Sigma_{\text{data}})^{\frac{1}{2}} \right) \quad (2.91)$$

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

Although FID is widely used [58] 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 frequently 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 [58] proposed an alternative score which is discussed next.

2.5.2 CLIP-Maximum Mean Discrepancy

Similar to the FID, the CMMMD [58] score measures the discrepancy between the data and the model distribution. It replaces the Inception-v3 embeddings with Contrastive Language- Image Pre-training (CLIP) [100] embeddings. Since CLIP is trained on more images and leverages natural language 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) [43, 44]

$$\begin{aligned} \text{MMD}^2 &= \frac{1}{m(m-1)} \sum_{i=1}^m \sum_{j \neq i}^m k(\mathbf{x}_{\text{data},i}, \mathbf{x}_{\text{data},j}) \\ &\quad + \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j \neq i}^n k(\mathbf{x}_{\text{model},i}, \mathbf{x}_{\text{model},j}) \\ &\quad - \frac{2}{mn} \sum_{i=1}^m \sum_{j=1}^n k(\mathbf{x}_{\text{data},i}, \mathbf{x}_{\text{model},j}) . \end{aligned} \quad (2.92)$$

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. [58] showed in several ablation studies that the CMMMD score aligns more closely with human preferences and can detect distortion that FID struggles to identify.

2.5.3 Human Preference Score v2 Benchmark

The HPSv2 benchmark [141] is designed to rank generated images based on human preferences. To model the human preference score, a CLIP model[100] is finetuned on the HPDv2 dataset [141] 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

$$s_\theta(p, \mathbf{x}) = \mathcal{E}_{\text{txt}}(p) \cdot \mathcal{E}_{\text{img}}(\mathbf{x}) \cdot 100 \quad (2.93)$$

where p is the text prompt, \mathbf{x} is the image, and \mathcal{E}_{img} and \mathcal{E}_{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 [19] and DiffusionDB [140] but cleaned and modified by ChatGPT. A higher score computed with eq. 2.93 correspond to better image quality.

2.5.4 GenEval Benchmark

Unlike global metrics such as CLIP and FID, which struggle to capture the compositional correctness of images, the GenEval Benchmark [40] 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. 2.11). This returns a ratio representing the proportion of images that contain all described aspects of the prompt.

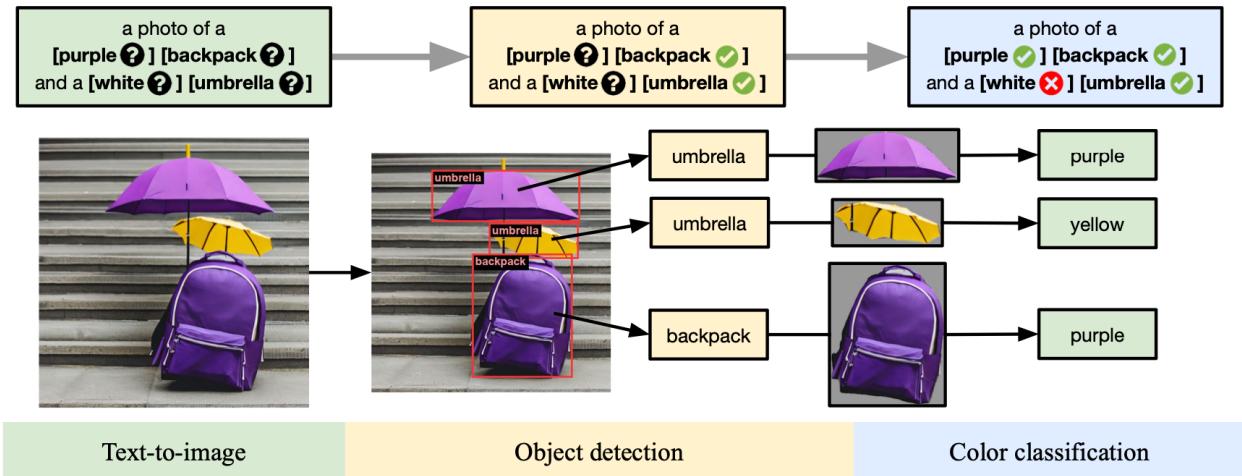


Figure 2.11: Example of how image is investigated for GenEval benchmark [40].

2.5.5 Dense Prompt Graph Benchmark

The goal of the Dense Prompt Graph (DPG) benchmark [54] is to evaluate the performance of image generation models on long and detailed prompts. It leverages existing datasets, such as COCO [19], PartiPrompts [146] and DSG-1k [21] and enriches their prompts with GPT-4 [1] by adding details, attributes, and spatial relations between objects. A fourth dataset, Object365 [117] is leveraged which is a dataset originally developed for object detection models. The various objects in this dataset are organized into super-categories. Typically, one to four objects from the same super-category are randomly sampled, after which GPT-4 generates a dense prompt based on these objects. In total, 1,065 different prompts were collected.

For evaluation, GPT-4 generates binary questions following the Dynamic Scene Graph (DSG) framework [21] which organizes the questions in a hierarchical tree where questions in child nodes can only be marked as correct if the questions in their parent nodes are also correct. This structure ensures that the answers follow the laws of logic. For example, the parent node normally contains an existence question, e.g. "Is there a motorcycle?" while the child nodes contain questions about its attributes, e.g. "Is the motorcycle blue". Therefore, if there is no motorcycle also the question regarding the color is rendered obsolete. The vision language model mPLUG-large [75] analyzes the generated images with respect to the questions. The evaluation follows a hierarchical aggregation process where the prompt score is defined as the arithmetic mean of all individual question results associated with a specific prompt, while the final DPG-Bench score is calculated by averaging these individual prompt scores across the entire dataset.

Chapter 3

Related Work

Methods to reduce model size

- pruning (structured, unstructured, transformer pruning?)
- quantization (SVDQuant)
- distillation
- weight sharing
- token pruning
- architecture search (NAS)

Competitor Methods Flux

- Plug and Play
- Flux Mini
- Flux Light
- Fast Flux
- Dense2MoE

Knowledge Distillation Method

- Koala
- Laptop
- BK-SDM
- Progressive Knowledge Distillation -> iteratively remove layers
- Tiny Fusion -> learn which block to remove (on DiT)

Using SVD

- GRASP (for LLMs)
- "ASVD: Activation Aware Singular value Decomposition For Compressing Large Language Models" (LLMs)
- SVD-LLM V2

- (SVD-Delta, SVDQuant) -> machen etwas anderes
- "Efficient Low-Rank Diffusion Model Training for Text-to-Image Generation" (Uses exactly like us SVD for Unet diffusion model; was ist OpenReview? Keine Autoren genannt; kein richtiges paper)
- "Accelerating Diffusion Transformer via Increment-Calibrated Caching with Channel-Aware Singular Value Decomposition"

Chapter 4

Methods

4.1 Image Generation Models

In the following section, the models, namely Pixart- Σ [14] and Flux-dev [71], which were used for the experiments in this work are described in detail.

4.1.1 Pixart- Σ

PixArt- Σ [14], the latest addition to the PixArt family [15, 16], 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 .

Architecture

The architecture of Pixart- Σ is based on the class-conditional diffusion transformer DiT-XL/2 [96] which contains 28 transformer blocks. The transformer blocks have primarily three inputs (see fig. 4.1) 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 , respectively 300 tokens in Pixart- Σ , and encoded by the T5-XXL text encoder [102]. 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 [98] 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.

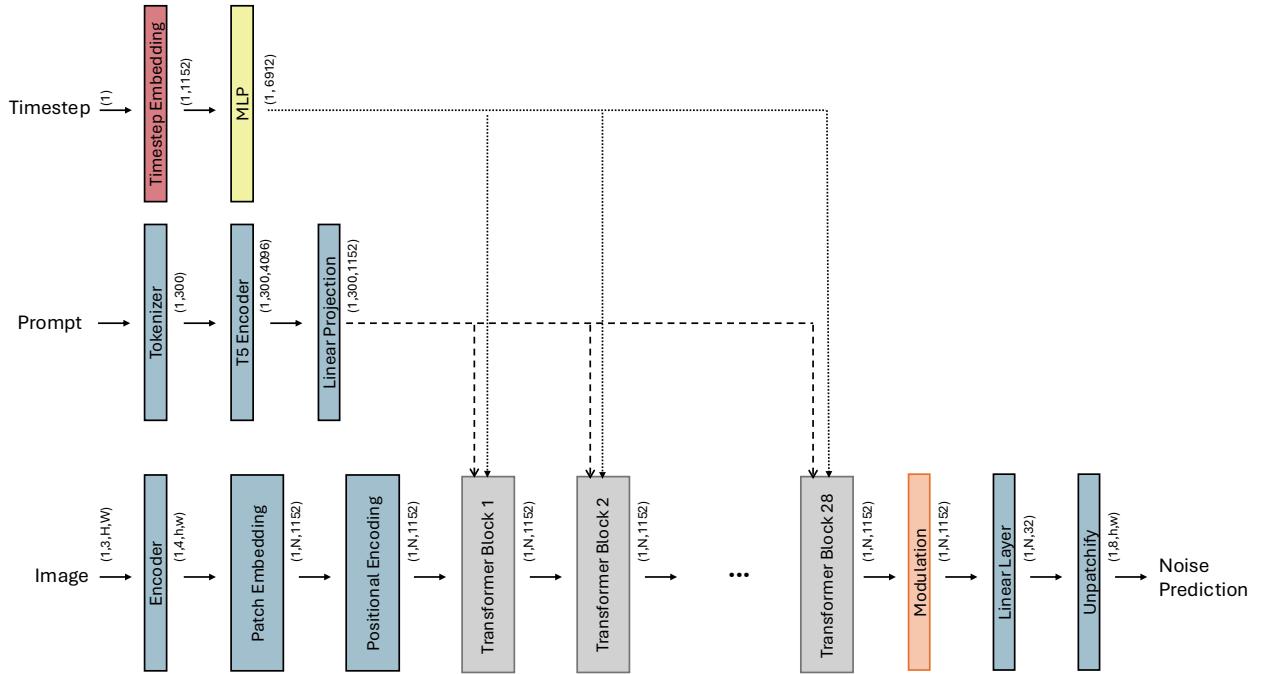


Figure 4.1: Overview architecture of Pixart- Σ . H and W are the height and width of the image. $h = \frac{H}{8}$ and $w = \frac{W}{8}$ represent the dimensions in latent space and $N = \frac{H}{8 \cdot p} \frac{W}{8 \cdot p}$ with p being the patch size.

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. 4.2).

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 2.3.5) method [97] 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 [96] 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.

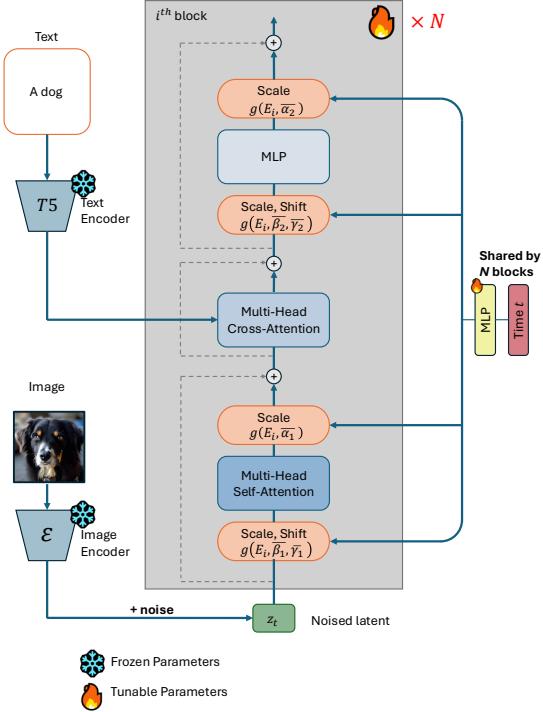


Figure 4.2: Diffusion block architecture taken from [16].

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. 4.1.

Table 4.1: 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

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 [28] 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 [65] dataset was utilized with corresponding high-density prompts generated by the vision-language model LLaVA [83]. Finally, the model undergoes finetuning on high-resolution and high-aesthetic quality images to refine visual details

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

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 [98], 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 [17], 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. 2.82) changes to

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{Q f_c(K)^T}{\sqrt{d_k}} \right) f_c(V) \quad (4.1)$$

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.

4.1.2 Flux-dev

Flux-dev [71] is a publicly available, guidance-distilled rectified flow model based on the commercial model Flux.1 [71]. 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.

Architecture

Flux-dev is based on a multi-modal diffusion transformer (MMDiT) [34] architecture containing 19 double-stream blocks followed by 38 single-stream blocks (see fig. 4.3) adding up to a total of 11.9 billion parameters (see tab. 4.2). For prompt processing, it leverages two distinct text encoders, namely CLIP ViT-L/14 [100] and T5-XXL [102]. 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. Text position IDs are included for architectural consistency but are initialized to zero, as text tokens lack meaningful 2D spatial coordinates in a latent image grid. In contrast to Pixart models, Flux-dev uses rotary positional embedding (RoPE) [129]. 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.

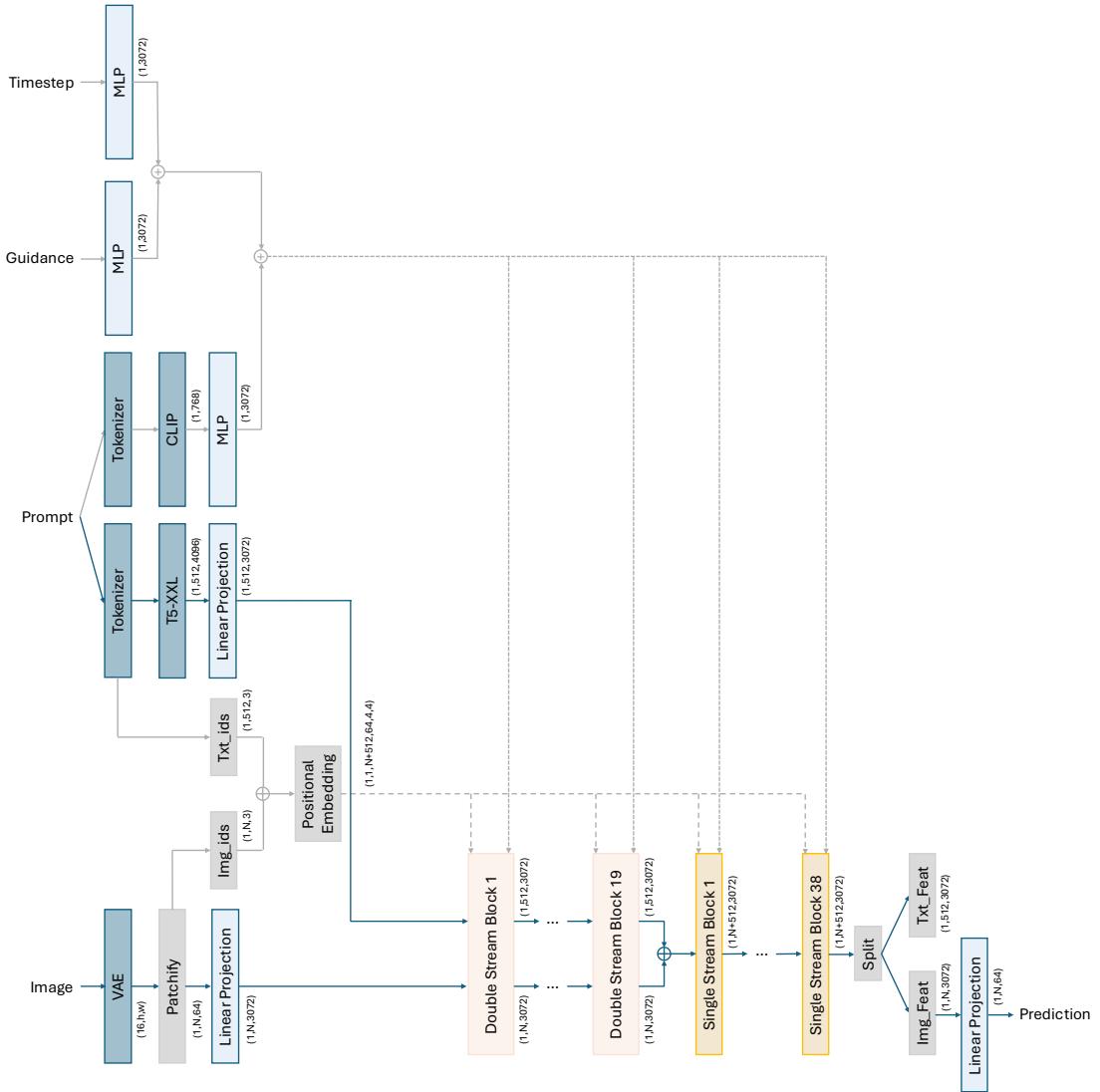


Figure 4.3: Overview Flux-dev architecture.

Double-Stream Blocks

The double-stream blocks (see fig. 4.4 on the right) are the fundamental building blocks of the first stage of the Flux-dev architecture. In contrast to other diffusion-based models [14, 98, 107] 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. 4.4, the guidance vector passes through two separate projection layers, which generate each two sets of scale and shift parameters 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 parameters 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 separate linear layers which project them into query, key and value representations and are then normalized via RMSNorm [148]. 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. 4.4 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 architecture, a gating mechanism is applied after the attention mechanism 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.

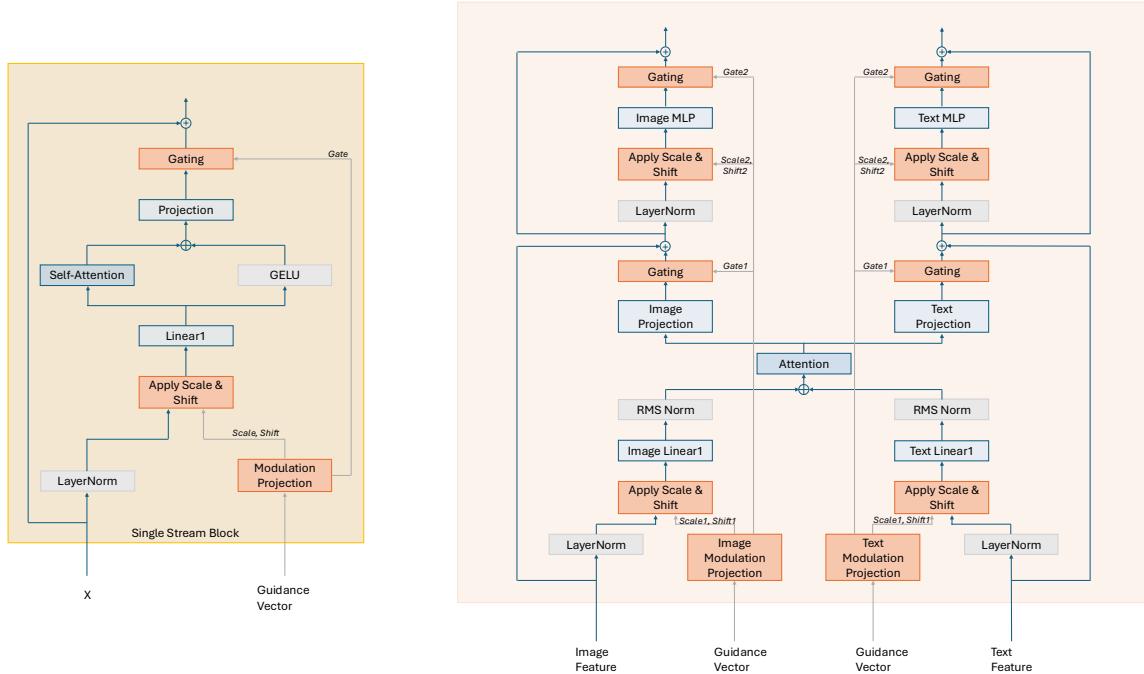


Figure 4.4: Single-stream block (left) and double-stream block (right) of Flux-dev. Graphics are adapted from [13].

Table 4.2: 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

4.2 Block Importance Analysis

4.2.1 Block Selection Criterias

Magnitude-Pruning

In magnitude pruning [38, 46, 73], the magnitudes of the weights of the individual components of a model are computed to serve as a proxy for their importance. The underlying assumption is that small weight magnitudes correspond to a low overall impact on the model performance, implying that model parts with lower weight magnitude should be removed first. The aggregated magnitude M_i of a transformer block B_i is defined as the total Frobenius norm of its constituent weight matrices $W \in B_i$. To account for the multiple sub-layers within a block, we compute the block-wise magnitude as

$$M_i = \sqrt{\sum_{W \in B_i} |W|_F^2} \quad \text{with} \quad |W|_F = \sqrt{\sum_j \sum_k |w_{jk}|^2} . \quad (4.2)$$

The block with the smallest magnitude $M_s = \min_i(M_i)$ is then identified as the primary candidate for removal or compression.

Representational Similarity Analysis

Another approach to identify redundant model components is to compare their input representations with their corresponding outputs representations [89]. If the transformation between the input and output is marginal, indicating that the specific component performs an approximate identity mapping, the model part is considered a candidate for removal. To quantify the similarity between the input and output features of a transformer block, the central kernel alignment (CKA) metric [69, 99] is employed. Let $X \in \mathbb{R}^{n \times d}$ denote the input and $Y \in \mathbb{R}^{n \times d}$ the output features which should be compared. First, the kernels K and L need to be chosen which are the inner product of the input $K = XX^T$ and the output $L = YY^T$ in the linear case. To ensure invariance to mean shifts, the centering matrix $H = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T$ is applied. The final CKA score is given by

$$\text{CKA}(K, L) = \frac{\text{HSIC}(K, L)}{\sqrt{\text{HSIC}(K, K)\text{HSIC}(L, L)}} . \quad (4.3)$$

This calculation utilizes the Hilbert-Schmidt Independence Criterion (HSIC) [45] $\text{HSIC}(K, L) = \frac{1}{(n-1)^2} \text{tr}(KHLH)$ [27].

Metric Based Selection

A third approach to identify the least important model components is to remove or compress all components individually, e.g. each transformer block, and compute for each configuration image quality metrics on a small reference dataset. Specifically, let $\{B_1, \dots, B_L\}$ denote the individual blocks of a transformer based model architecture. Each block is modified separately leading to L different model configurations $\{\hat{B}_1, \dots, B_L\}, \dots, \{B_1, \dots, \hat{B}_i, \dots, B_L\}, \dots, \{B_1, \dots, \hat{B}_L\}\}$ where \hat{B}_i indicates that the i^{th} block is modified. For each model configuration the metrics are computed. In this work, we employ CMMD and/or CLIP-score to rank the importance of the individual model components. While CLIP is used to evaluate the semantic prompt-image coherence, CMMD serves as a measure for the general image quality and distributional fidelity. Since CMMD and CLIP-score produce values in different ranges and have an inverse optimization direction (minimization vs maximization), for both metrics a separate ranking of the blocks is created where $r_{\text{CMMD},i}$ and $r_{\text{CLIP},i}$ are the corresponding ranks of block B_i . The final importance score is obtained by computing the aggregated rank $r_{\text{final},i}$ from both metrics

$$r_{\text{final},i} = r_{\text{CMMD},i} + r_{\text{CLIP},i} . \quad (4.4)$$

In our linear progressive compression approach (see sec. 4.6.3) the compression-scheme of the individual model parts leads to different numbers of removed parameters due to repeated compression of individual blocks and different compression ratios per block. Therefore, only the CMMD value per removed parameters $\frac{\text{CMMD}_i}{\text{Removed Parameters}}$ is leveraged to quantify the degradation of image quality per removed parameter and to ensure fair comparison.

4.2.2 Block Selection Algorithms

One challenge in structural model compression using metric based selection is that the importance of individual model components, e.g. transformer blocks, are normally determined in isolation meaning only one block is modified and the remaining parts of the model are fixed for computing the metrics. However, in most cases, several components are compressed simultaneously. It cannot necessarily be assumed that removing the model components that individually had the least impact on model performance will also result in the best combination with the least reduction in image quality. In the following, methods to select the best subset of the model components are described and investigated in sec. ??.

Greedy Block Selection

The greedy algorithm is applied alongside the metric-based selection criteria. It is an iterative procedure where the importance of all uncompressed blocks is determined to identify and compress the least critical one. Following this step, the block importance is re-evaluated for all remaining uncompressed blocks. Concretely, let the model consist of L transformer blocks $B = \{B_1, \dots, B_L\}$. In the first iteration, for all L combinations of model configurations $\mathcal{M}^{(1)} = \{(B \setminus \{B_i\}) \cup \{\hat{B}_i\} \mid 1 \leq i \leq L\}$ the metrics are computed and the best block B_j is selected according to the ranking. Let \bar{B}_j denote the compressed version of block B_j . In the second iteration, the metrics are re-computed for all $L - 1$ uncompressed blocks $\mathcal{M}^{(2)} = \{(B \setminus \{B_i, B_j\}) \cup \{\hat{B}_i, \bar{B}_j\} \mid 1 \leq i \leq L, i \neq j\}$. For the n^{th} iteration $L - (n - l)$ uncompressed blocks remains. Let B_{comp} denote the set of all previously compressed blocks then the potential candidates for compression are given by $\mathcal{M}^{(n)} = \{(B \setminus (B_{\text{comp}} \cup B_i)) \cup B_{\text{comp}} \cup \hat{B}_i \mid B_i \in B \setminus B_{\text{comp}}\}$. This process is repeated until the desired number of blocks to be compressed has been identified.

Black-Box Optimization with Optuna

Optuna [3] is an open-source hyperparameter optimization framework. It primarily leverages a tree-structured parzen estimator [9], a bayesian optimization algorithms, to search the space of all possible hyperparameter settings. It partitions the observed hyperparameter configurations into promising and non-promising candidates based on a user defined objective function. By modeling these two groups using probability density functions it draws a new set of hyperparameters that belongs to the promising group with high probability. Through this informed search the global optimum can be found much faster than with non-adaptive methods like random or grid search. For an in-depth description of the Optuna algorithm the reader is referred to the original paper [3] because this would go beyond the scope of this work.

TinyFusion

The authors of TinyFusion [36] propose a learning based block selection method for pruning transformer based image generation models. They hypothesize that instead of focusing on the error/loss in image quality resulting from the model compression it is advantageous to focus on the recoverability capabilities of the compressed model meaning on the post-finetuning performance. Initially, the model is divided into K subparts $\phi = [\phi_1, \dots, \phi_K]$. Each subpart consists of M transformer blocks $\phi_k = [\phi_{k1}, \dots, \phi_{kM}]$. Specifying the number N of blocks retained in each subpart a local binary mask $\mathbf{m}_k \in \{0, 1\}^M$ is introduced to determine the inclusion or exclusion of the

individual blocks. A categorical distribution $p(\mathbf{m}_k)$ is assigned to each mask to model the probability for a specific combination of blocks to be removed. The goal is to jointly optimize the model and the categorical mask distributions. To ensure that the sampling process of the mask remains differentiable during training the Gumbel-Softmax [57] is utilized. The objective function for jointly optimizing weights and masks is given by

$$\min_{\{p(\mathbf{m}_k)\}} \underbrace{\min_{\Delta\Phi} \mathbb{E}_{x, \{\mathbf{m}_k \sim p(\mathbf{m}_k)\}} [\mathcal{L}(x, \Phi + \Delta\Phi, \{\mathbf{m}_k\})]}_{\text{Recoverability: Post-Fine-Tuning Performance}} \quad (4.5)$$

where $\Delta\Phi$ represents the updates for the model parameters Φ . The learned mask provides the information which transformer blocks are essential for the recoverability of the network. Due to the potential high computational effort of a complete finetuning of a large model, the authors showed that low-rank adaptation (LoRA) [53] finetuning is sufficient. The joint optimization results in a mask distribution taking into account how removing blocks would influence the post-finetuned model.

4.3 Training Data and Image Synthesis

4.3.1 Training Datasets

LAION Aesthetic

LAION-5B [115] is a dataset containing 5.85 billion web images of varying sizes, of which 2.32 billion have English captions. All images are filtered using CLIP embeddings to prevent that harmful and inappropriate images appear in the dataset. LAION-5B is partitioned into several subsets. Specifically, an aesthetic subset [116] comprising 120 million images was created by training a linear model on top of the CLIP embeddings to predict an aesthetic score. Only images exceeding a certain aesthetic score were added to this subset. In this work, 608k images from the aesthetic subset are utilized as the training dataset. Due to poor quality of the web-scraped captions which come along with the LAION-5B dataset, new synthetic prompts were generated using the visual language model JoyCaption [39]. Five example images of the dataset are displayed in fig. 4.5.



Figure 4.5: Examples of LAION dataset.

LAION Pixart- Σ Dataset

The pre-trained Pixart- Σ model is utilized to generate a synthetic dataset of 100k images conditioned on prompts generated by the JoyCaptions model from the aesthetic LAION subset. Tab. 4.3 presents the configuration parameters used in the generation process. Fig. 4.6 shows example images based on the JoyCaption prompts providing a direct visual comparison to the original LAION samples presented in fig. 4.5.

Table 4.3: Inference Parameters

Parameters	Value
Image Height	512
Image Width	512
Guidance Scale	3.5
No. Inference Steps	20
Max Sequence Length	300

Figure 4.6: Examples of PixArt- Σ generated dataset based on JoyCaptions prompts from LAION aesthetic dataset.

LAION-Flux Dataset

Following the same methodology as for the LAION-PixArt dataset a LAION-Flux dataset was created. Specifically, 190k prompts generated with JoyCaptions based on the aesthetic subset of LAION-5B were utilized to generate synthetic images using the Flux-dev model. The configuration used during the image generation process are detailed in tab. 4.4. In addition, example images based on the same prompts as those used for the LAION- PixArt dataset (see 4.6) are presented in fig. 4.7.

Table 4.4: Inference Parameters

Parameters	Value
Image Height	1024
Image Width	1024
Guidance Scale	3.5
No. Inference Steps	50
Max Sequence Length	512



Figure 4.7: Examples of Flux-dev generated dataset based on JoyCaptions prompts from LAION aesthetic dataset.

4.3.2 Evaluation Datasets

Mapillary

The Mapillary [91] dataset consists of 25,000 street level images. Additionally, it contains fine-grained semantic masks for 66 different object classes. In order to achieve high diversity, the images originate from various geographical regions, e.g., Europe, North and South America, Africa, Wider Geographic Oceania, Asia, and were captured during different seasons and at varying times of day with diverse lighting conditions. In addition, no image sequences, which are often recorded by car cameras, were integrated, only individual and unique images to avoid redundancy. Most of the images were taken from the street or sidewalk. These were supplemented by images taken on highways, in rural areas, and off-road environments. The use of different camera sensors and perspectives contribute to the diversity of the dataset. For quality assurance, only images with at least full HD resolution (1920×1080) were selected that contain only a small amount of motion blurr. Due to privacy protection faces and number plates are blurred.



Figure 4.8: Examples of Mapillary dataset.

Cityscapes

The Cityscapes [23] dataset contains a total of 25,000 images (1024×2048) extracted from stereo-video-sequences of urban street scenes. The images were captured at 50 different cities, in spring, summer or fall. However, there are only images taken during daylight hours and in good to moderate weather conditions. While 20,000 images provide coarse annotations, 5,000 images of the dataset are densely annotated at pixel level covering 30 different object classes. The dataset is highly relevant for the automotive sector, e.g., autonomous driving but is due to its high degree of standardization less diverse than the Mapillary dataset.



Figure 4.9: Examples of Cityscapes dataset.

MJHQ-30k

The MJHQ-30k [76] dataset consists of 30,000 synthetic images from ten different categories (animals, art, fashion, food, indoor, landscape, logo, people, plants, vehicles) each containing 3,000 images to ensure a balanced distribution. These images were generated using Midjourney 5.2 [76] and selected based on their text-alignmen (CLIP) and their aesthetic [66] scores.



Figure 4.10: Examples of MHJQ-30k dataset.

4.4 Structural Compression Strategies

There are two primary strategies in structural model pruning. On the one hand, individual model parts can be removed entirely, or, on the other hand, they can be compressed. In transformer-based model architectures like PixArt- Σ and Flux-dev, a natural choice is to dissect the model according to the individual transformer blocks. Therefore, when referring to model parts in the following, this always refers to the individual transformer blocks. Both strategies are described in the following.

4.4.1 Block Removal

In structural model pruning, entire functional components of a model are often removed completely [62, 74, 149] while others that are considered as highly relevant for the models performance are maintained. Concretely, if the transformer block B_i is removed the output features of block B_{i-1} denoted as x_{i-1} are directly passed to block B_{i+1}

$$x_{i+1} = B_{i+1}(x_{i-1}) \quad . \quad (4.6)$$

Due to consistent dimensionality of the hidden features across the model's depth no additional dimension adaption needs to be done.

4.4.2 SVD Compression

Singular value decomposition (SVD) [152] factorizes a matrix $W \in \mathbb{R}^{n \times m}$ into three matrices U, Σ, V such that

$$W = U\Sigma V^T \quad (4.7)$$

where $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{m \times m}$ are orthonormal and $\Sigma \in \mathbb{R}^{n \times m}$ a diagonal matrix. The entries of Σ are called singular values and are sorted in descending order $\sigma_1 \geq \sigma_2 \geq \sigma_3 \dots \sigma_p$ with $p = \min(n, m)$. The magnitude of the singular value serves as proxy for the importance of the corresponding rows and columns of U and V . In model compression [139, 147], SVD can be leveraged to approximate the weight matrices by just using the $k < p$ largest singular values such that $U_{\text{red}} \in \mathbb{R}^{n \times k}$, $V_{\text{red}} \in \mathbb{R}^{m \times k}$ and $\Sigma_{\text{red}} \in \mathbb{R}^{k \times k}$ reducing the total parameter count in dependence of the choice of k . Specifically, the matrix W is approximated by two matrices $A = U_{\text{red}}\sqrt{\Sigma_{\text{red}}}$ and $B = \sqrt{\Sigma_{\text{red}}}V_{\text{red}}^T$. Let \mathbf{x} be the features then instead of $\mathbf{x}_{\text{next}} = W\mathbf{x} + \mathbf{b}$ the features are updated via

$$\mathbf{x}_{\text{next}} = AB\mathbf{x} + \mathbf{b} \quad (4.8)$$

where \mathbf{b} is the bias term which is kept unmodified. Instead of specifying a specific rank k we want to specify the percentage c of parameters to remove. Assuming the weight matrix $W \in \mathbb{R}^{n \times m}$ the total number of parameters is given by $P_{\text{orig}} = n \times m$. The target number of parameters can be computed via $P_{\text{targ}} = P_{\text{orig}} \times (1.0 - c)$. The

target rank is given by

$$k_{\text{targ}} = \max \left(1, \text{int} \left(\frac{P_{\text{targ}}}{n + m} \right) \right) . \quad (4.9)$$

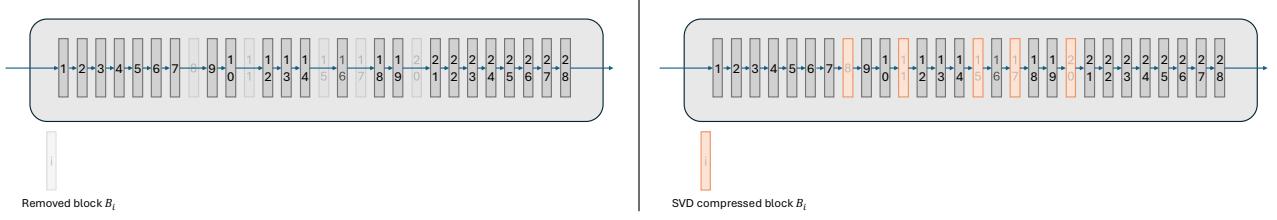


Figure 4.11: Example of removal (left) and compression (right) strategy based on a simplified representation of PixArt- Σ architecture. The depictions show the case that the blocks 8,11,15,17 and 20 are removed or compressed.

4.4.3 Iteratively Repeated SVD Compression

In our final compression setup (see sec. 4.6.3), individual transformer blocks are compressed and subsequently retrained multiple times using SVD. Initially, an original weight matrix W is decomposed into matrices A and B such as $W \approx AB$. During the subsequent retraining the matrices A and B are updated to A_t and B_t . Therefore, $W \approx A_t B_t$ does not necessarily need to hold anymore w.r.t. the initial W as the model adapts during training to the low rank constraint. Therefore, if the transformer block which originally contained the weight matrix W is to be compressed once more, first, the current matrix $W_t = A_t B_t$ is reconstructed and afterward the SVD algorithm is performed on it $W_t = U^{(2)} \Sigma^{(2)} V^{(2)\top}$. Similar to the initial step, after choosing a further reduced rank, the new smaller matrices $A^{(2)}$ and $B^{(2)}$ are constructed via $A^{(2)} = U_{\text{red}}^{(2)} \sqrt{\Sigma_{\text{red}}^{(2)}}$ and $B^{(2)} = \sqrt{\Sigma_{\text{red}}^{(2)}} V_{\text{red}}^{(2)\top}$. By following this procedure, reconstructing $W_t^{(j)} = A_t^{(j)} B_t^{(j)}$, choosing a new smaller rank and obtaining the new $A^{(j+1)} = U_{\text{red}}^{(j+1)} \sqrt{\Sigma_{\text{red}}^{(j+1)}}$ and $B^{(j+1)} = \sqrt{\Sigma_{\text{red}}^{(j+1)}} V_{\text{red}}^{(j+1)\top}$, a transformer block can be gradually compressed across multiple iterations.

4.5 Knowledge Distillation Loss Functions

The general learning objective for knowledge distillation as described in eq. 2.90 is

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{OutKD}} \mathcal{L}_{\text{OutKD}} + \lambda_{\text{FeatKD}} \mathcal{L}_{\text{FeatKD}} . \quad (4.10)$$

In the experiment section ?? we investigate the importance of the individual components.

4.5.1 Normalization

An important point to note is that the features of different blocks may have varying magnitudes wherefore [149] found that applying a normalization to the features is beneficial. Therefore, after computing the difference between the teacher and the student features a normalization is applied such as the individual blocks contribute equally to $\mathcal{L}_{\text{FeatKD}}$. The normalization which is utilized for this works is taken from [153]. Let T_l and S_l denote the features of the teacher and the student from transformer block $l \in \{1, \dots, L\}$. For a batch size of B , each feature tensor has the dimension (B, D_l) where D_l is the total numbers of flattened features. The i -th sample in the batch for layer l is denoted as $T_{i,l}$ and respectively $S_{i,l}$. First, the MSE is computed between the teacher and the student features

$$\mathcal{L}_{i,l} = \frac{1}{D_l} \|T_{i,l} - S_{i,l}\|_2^2 \quad (4.11)$$

Next, the root-mean square of the teacher features is computed acting as measure for the feature magnitude

$$\eta_{i,l} = \sqrt{\frac{1}{D_l} \|T_{i,l}\|_2^2} \quad (4.12)$$

Additionally, the loss is scaled by the magnitude of the task loss $\mathcal{L}_{i,\text{Task}}$. The normalization factor is then given by

$$w_{i,l} = \underbrace{\left(\frac{\mathcal{L}_{i,\text{Task}}}{\mathcal{L}_{i,l} + \epsilon} \right)}_{\text{Loss Balancing}} \cdot \underbrace{\left(\frac{\sum_{k=1}^L \eta_{i,k}}{L \cdot \eta_{i,l} + \epsilon} \right)}_{\text{Norm Normalization}} \quad (4.13)$$

where a small ϵ is added to the denominator to prevent devision by zero. The final feature loss is computed by

$$\mathcal{L}_{\text{FeatKD}} = \frac{1}{B} \sum_{i=1}^B \left(\frac{1}{L} \sum_{l=1}^L w_{i,l} \cdot \mathcal{L}_{i,l} \right) . \quad (4.14)$$

Please note, if the normalization is applied it is not only applied to $\mathcal{L}_{\text{FeatKD}}$ but also to $\mathcal{L}_{\text{OutKD}}$ which compares the final output of the teacher with the student model.

4.5.2 Feature Mapping for Loss Computation

The assignment of the learning signals based on the intermediate features varies between the complete block removal and the SVD block compression approach (see fig. 4.12). In the SVD compression approach, the intermediate features of every student block S_l are directly compared with the feature of the teachers' block T_l with $l \in \{1, \dots, L\}$ assuming the model consists of L transformer blocks.

In contrast, the selection of intermediate features requires more care in the block removal approach where a dynamic mapping strategy is needed. If a block B_l is removed, the feature of the preceding student block S_{l-1} is compared with the features of the teacher's block T_l . This effectively forces block B_{l-1} to approximate the removed block B_l . Consequently, if several consecutive blocks B_{l-i} through B_l (with $i \geq 1$) are removed the student feature $S_{l-(i+1)}$ is compared to T_l .

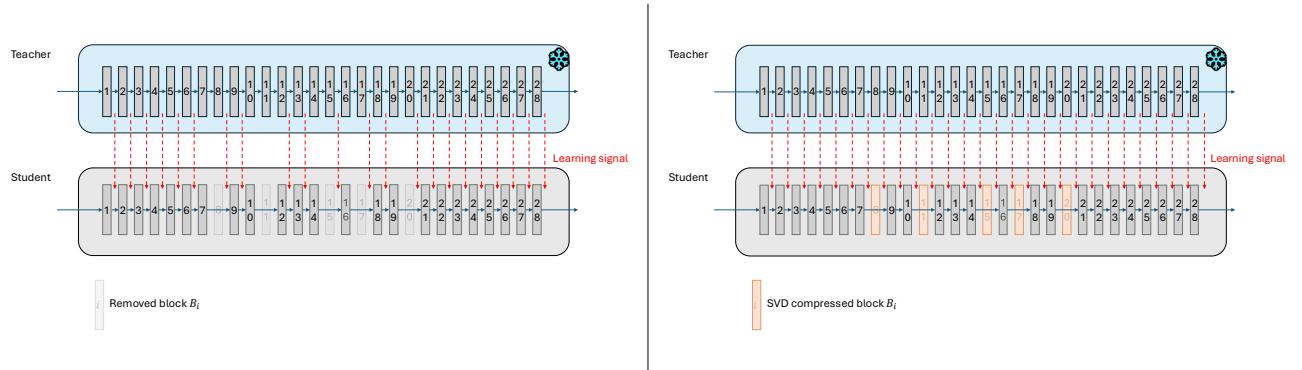


Figure 4.12

4.6 Distillation Algorithms

Various approaches can be chosen for structural model pruning.

4.6.1 One-Shot Model Compression

The standard approach of compressing transformer based diffusion and flow models [25, 74, 133, 149] involves three main steps. First, the transformer blocks $\{B_1, \dots, B_L\}$ are sorted according to their importance, e.g. using CLIP-score and CMMMD as importance proxy. Let $\{B_1^*, \dots, B_L^*\}$ denote the sorted blocks, where B_1^* represents the least important block. Next, based on the desired compression ratio of the total model, a number $k \leq L$ of blocks is determined. Consequently, the blocks $\{B_1^*, \dots, B_k^*\}$ are modified based on the chosen compression strategy (see sec. 4.4). Finally, the compressed model is re-trained, e.g. using knowledge distillation, to recover the generation capabilities. This approach introduce a significant, abrupt structural alteration of the model, leading to temporary severe degeneration in image generation capabilities.

4.6.2 Progressive Model Compression

In contrast to one-shot model compression, progressive model compression, e.g. [111], iteratively applies the three-step procedure described above. In each iteration the importance of the blocks is re-evaluated, the number of parameters to be removed specifies how many blocks need to be modified, and finally the model is retrained. The crucial difference is that the number of parameters removed in a single step is significantly smaller than in one-shot model compression. This leads to gradual and less severe performance degradation in one iteration which is easier recoverable. In the subsequent iteration, the block importance is re-assessed for all remaining unmodified blocks $\{B_1, \dots, B_L\} \setminus \{B_1^*, \dots, B_k^*\}$, as the relative block ranking may have changed as a result of the retraining. Another design choice concerns the number of parameters that are removed per iteration. On the one hand, the number of parameters can be set to a fixed value or, on the other hand, it could be chosen dynamically based on the importance metrics, e.g. by defining a threshold CMMMD value and performing retraining as if the model newly compressed model reaches it. This typically implies that in early iterations more parameters are removed than in later iterations.

4.6.3 Linear Progressive Compression

Our approach, linear progressive compression, is a further development of progressive model compression. Instead of compressing all blocks uniformly and each block only once, the compression ratio of each block is determined depending on the importance of the block and every block can be modified in several different iterations.

The approach consists of the same three main steps. First, the block importance is determined by leveraging the CMMMD value per removed parameter. Let P_i ($i \in \{1, \dots, L\}$) denote the current number of paramters of block B_i which may differ between blocks as the blocks may have been already compressed during previous iterations. For generating the images based on those the CMMMD values are computed, each block is compressed by a fixed percentage c_{fixed} . Specifically, this means that potentially different number of parameters are removed for each block. Therefore, the final importance score for each block B_i is given by

$$s_i = \frac{\text{CMMMD}_i}{P_i \cdot c_{\text{fixed}}} \quad (4.15)$$

where CMMMD_i is the CMMMD value obtained by the dataset generated with the model where block B_i is compressed leading to the sorted order of block $\{B_1^*, \dots, B_L^*\}$ where B_1^* has the smallest importance score. To determine the compression ratio for each block we propose a linear compression schedule where the least important block is compressed most, and blocks of higher importance less. Concretely, three parameters need to be specified by the user to compute the individual compression ratios c_i , namely the total number of paramters to be removed N , the number of blocks that are compressed during the current iteration k and the compression ratio of the least important block $c_1 = S$ to which we refere to as base compression ratio. To determine the

compression ratios c_i of the remaining blocks $B_i^* \in \{B_1^*, \dots, B_k^*\}$,

the base compression ratio S is decrease by a slope x between every block. Therefore, the compression ratio for every block B_i^* is given by

$$c_i = S - ((i-1) \cdot x) \quad (4.16)$$

with $i \in \{1, \dots, k\}$. The goal is to identify x based on the parameters S, N, k . The total number of parameter that are removed is computed by

$$P_{\text{total}} = \sum_{i=1}^k P_i^* \cdot c_i \quad (4.17)$$

where P_i^* is the current parameter count of block B_i^* . To solve for x , eq. 4.16 and eq. 4.17 are combined

$$\begin{aligned} & \sum_{i=1}^k P_i^*(S - (i-1) \cdot x) = N \\ \Leftrightarrow & \sum_{i=1}^k (P_i^* \cdot S) - \sum_{i=1}^k (P_i^* \cdot (i-1) \cdot x) = N \\ \Leftrightarrow & \sum_{i=1}^k (P_i^* \cdot S) - N = x \cdot \sum_{i=1}^k (P_i^* \cdot (i-1)) \end{aligned} \quad (4.18)$$

ending in the final equation for the slope

$$x = \frac{\sum_{i=1}^k (P_i^* \cdot S) - N}{\sum_{i=1}^k (P_i^* \cdot (i-1))} . \quad (4.19)$$

The input parameters S, N, k specified by the user have to meet two conditions. First, $N \leq S \cdot \sum_{i=1}^k P_i^*$ meaning the total number of parameters that should be removed during this iteration must not be larger than the number of parameters one obtain when using S as compression ratio for all blocks B_i^* . Otherwise, the compression ratio would increase for more important blocks which would contradict the ranking strategy. Second, the compression ratio of the last block c_k must not be negative which results in the following condition

$$\begin{aligned} 0 & \leq c_k = S - (k-1) \cdot x \\ \Leftrightarrow & 0 \leq S - (k-1) \frac{S \sum_{i=1}^k P_i^* - N}{\sum_{i=1}^k (P_i^* \cdot (i-1))} \\ \Leftrightarrow & N \geq S \sum_{i=1}^k P_i^* - \frac{S \sum_{i=1}^k (P_i^* \cdot (i-1))}{k-1} . \end{aligned} \quad (4.20)$$

In total we get the following condition for the total number of paramters to remove in one iteration based on the choice of S and k

$$S \sum_{i=1}^k P_i^* - \frac{S \sum_{i=1}^k (P_i^* \cdot (i-1))}{k-1} \leq N \leq S \cdot \sum_{i=1}^k P_i^* . \quad (4.21)$$

After obtaining the the compression ratios c_i the model can be compressed based on the SVD compression scheme as described in sec. 4.4.3 and then be retrained. Afterwards, the procedure is repeated until the desired model size is achieved.

Chapter 5

Experiments

This master thesis investigate structural model distillation techniques, specifically focusing on the removal and compression of transformer blocks within the PixArt- Σ and Flux-dev architectures. PixArt- Σ is employed as an experimental baseline to evaluate various design choices in the distillation frameworks due to its relatively low parameter count enabling rapid iteration and extensive ablation studies. Subsequently, the most effective strategies are applied to Flux-dev to assess their efficacy in high-parameter regimes.

5.1 Pixart- Σ Distillation

5.1.1 Block Importance Analysis

In structural pruning, the selection of components to be removed or compressed is essential to ensure that the compressed model retains the highest possible image generation quality. Qualitatively, fig. 5.1 shows the effect of removing every transformer block from PixArt- Σ individually on its image generation capabilities without subsequent retraining. It is clearly visible that removing the first and last transformer blocks (blocks 1-4 and blocks 27-28) exerts the most significant influence on the image quality. This confirms the observation from previous work [89] that redundancies are primarily find in the middle part of transformer based model. To identify which blocks are best to remove meaning which blocks have a minimal contribution on the generation process of the final image, several different metrics can be employed. In the following section, different approaches are investigated to determine the importance of the individual transformer blocks. First, the traditional magnitude-based [46] pruning and representational dissimilarity by computing the central kernal alignment score (CKA) [69] between the input and output of each transformer block are evaluated. Moreover, they are compared to standard performance metrics such as the CLIP-score and CMMD.

Normally, several transformer blocks are removed or compressed simultaneously. Due to block interdependencies, we investigate three different options to identify the best grouping for blocks to remove. First, a greedy-algorithm is applied, iteratively searching for the best next block to compress based on the already compressed model from previous iterations. Second, the block selection process is formulated as hyperparameter search leveraging the Optuna framework. Last, a procedure of learning a mask indicating the blocks with least impact on the model performance is applied.

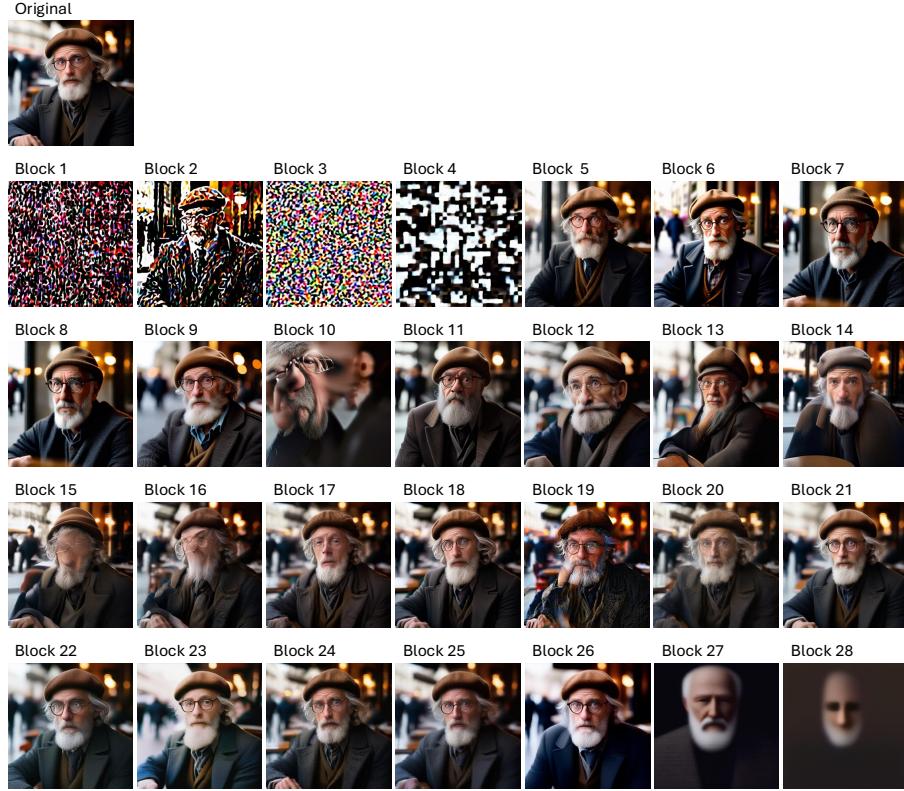


Figure 5.1: The figure shows the influence of selectively removing individual transformer blocks (1–28) on the final generation result. While the removal of middle blocks causes only minor visual deviations, interventions in early, structure-forming layers (blocks 1–4) and in the final layers (blocks 27–28) lead to a significant loss of image coherence and massive compression artifacts, respectively.

Compression Criteria

In magnitude based pruning strategies the model parts with the lowest averaged weight magnitude are considered to be less important for the overall model performance. Interestingly, in Pixart- Σ the weight magnitudes of the initial layers, which have a substantial influence on the final image synthesis (see fig. 5.1), have the lowest weight magnitudes across the architecture (see fig. 5.2 and 5.3 which both show the weight magnitudes for every block compared to the CLIP-score or CMMMD). This observation contradicts with the assumption of low impact by low weights magnitudes. Furthermore, the weight magnitudes of the final layers are only marginally higher than those of the middle layers, while the middle layers' weight magnitudes exhibit a high degree of similarity. This would make a precise selection of blocks for removal based solely on this heuristic difficult. Consequently, using the weight magnitude to identify redundant blocks within PixArt- Σ is not a suitable criterion for structural distillation in this context.

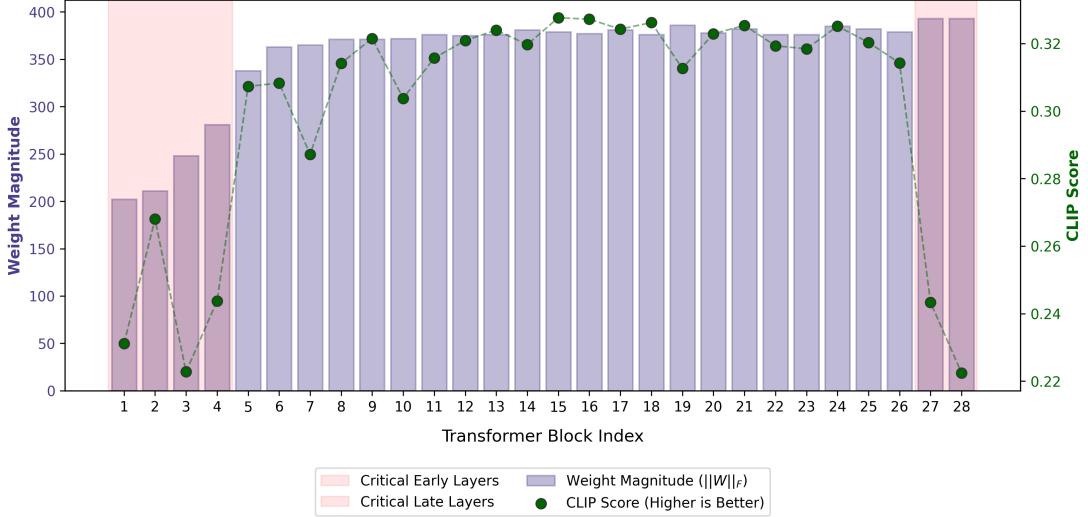


Figure 5.2: The histogram shows the magnitudes of the weights of the individual transformer blocks (1-28). The blocks which have shown qualitatively (see fig. 5.1) to be very important for the image generation capabilities of PixArt- Σ are marked with red. In addition, the CLIP-scores for each block is displayed. The CLIP-scores follow the qualitative observations in contrast to the magnitude-based analysis.

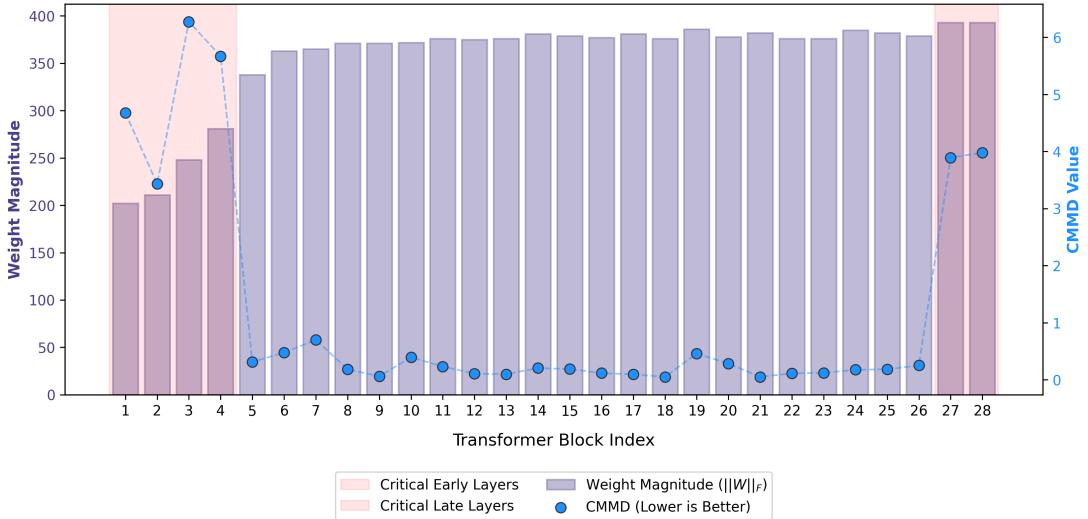


Figure 5.3: The histogram shows the magnitudes of the weights of the individual transformer blocks (1-28). The blocks which have shown qualitatively (see fig. 5.1) to be very important for the image generation capabilities of PixArt- Σ are marked with red. In addition, the CMMMD for each block is displayed. The CMMMD follow the qualitative observations in contrast to the magnitude-based analysis.

The CKA score is computed for every block and every sampling step across 100 example prompts taken from the LAION dataset. Subsequently, these scores are averaged across all samples and time steps resulting in a mean similarity score $\mu_{CKA,i}$ for each individual Pixart- Σ block. Averaging over the time steps should provide an estimate of the general importance of the block. However, the importance of a block at different time steps can vary drastically (see Appendix A). A low value indicates that the input and output of a specific block differs significantly, suggesting higher importance of the block.

Both fig. 5.4 and 5.5 show the transformation intensity $1 - \mu_{CKA,i}$ for every block compared with CMMD or CLIP score, a greater value signifies a more substantial feature transformation. While blocks 17 and 22 stand out with their high values, the implied importance of these blocks cannot be confirmed by qualitative analysis (see fig. 5.1). Furthermore, blocks 1, 24, 27, and 28 are not uniquely identified as critical blocks for the performance

of the model, as the values for blocks 19, 20, and 21 are higher, which would mark them as more relevant. As this ranking also fails to align with the qualitative review of block influence on the final image, the use of representational similarity for block selection in PixArt- Σ must be viewed critically too.

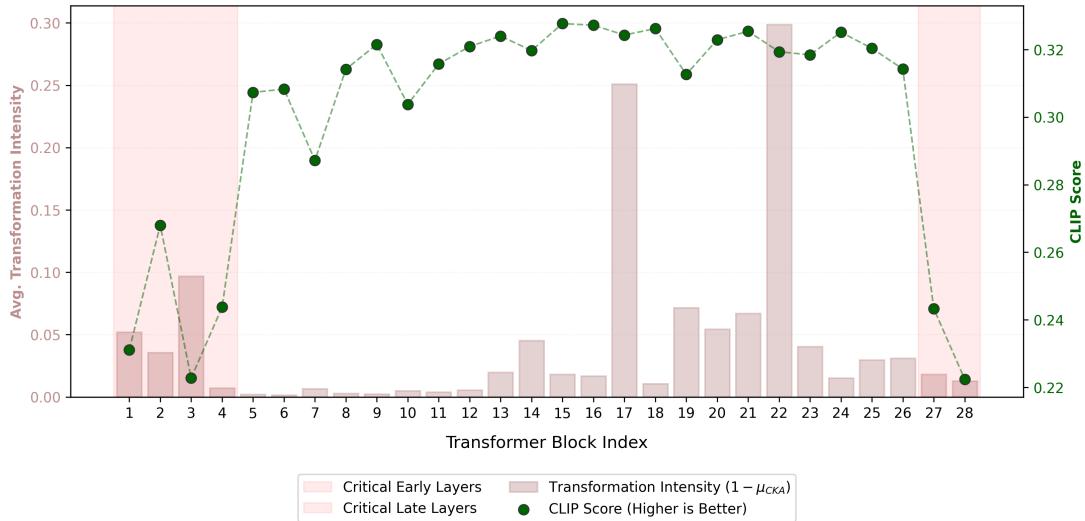


Figure 5.4: The histogram shows the transformation intensity based on CKA comparing the input and output features of the individual transformer blocks (1-28). The blocks which have shown qualitatively (see fig. 5.1) to be very important for the image generation capabilities of PixArt- Σ are marked with red. In addition, the CLIP-scores for each block is displayed. The CLIP-scores follow the qualitative observations in contrast to the transformation intensity analysis.

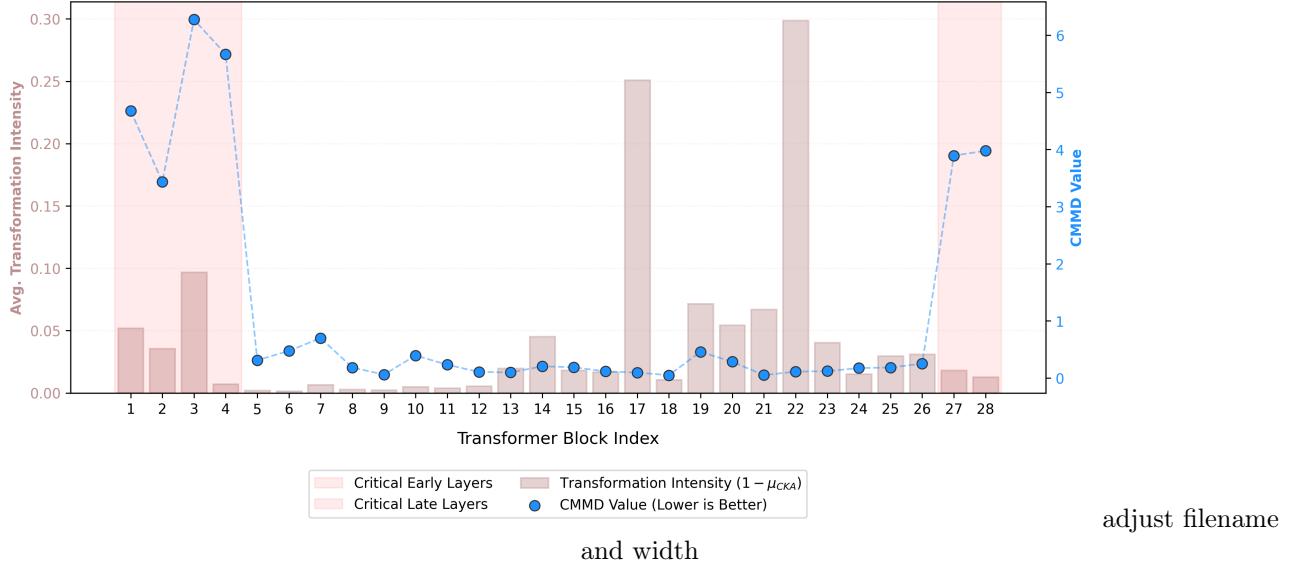


Figure 5.5: The histogram shows the transformation intensity based on CKA comparing the input and output features of the individual transformer blocks (1-28). The blocks which have shown qualitatively (see fig. 5.1) to be very important for the image generation capabilities of PixArt- Σ are marked with red. In addition, the CMMMD for each block is displayed. The CMMMD follow the qualitative observations in contrast to the transformation intensity analysis.

Due to the contradictory results of magnitude-based pruning and CKA-scores when compared with the qualitative observations, a third methodology was investigated, namely quantifying the importance of individual blocks based on standard performance metrics [13]. Specifically, the CLIP-score and CMMMD are computed on a small reference dataset consisting of 100 images from the LAION dataset. It should be noted that the

standard protocols for computing statistically robust CMMD and CLIP scores typically require larger datasets, e.g. upwards of 10,000. However, given the iterative nature of the greedy search (see sec. ??) and the high computational overhead associated with repeated evaluations, using such a large reference set was prohibitively expensive. Nevertheless, a sample size of 100 images serves as a sufficient proxy to capture the relative importance ranking of the transformer blocks, providing the necessary directional guidance for the distillation process. As illustrated in the fig. 5.4 and 5.5 (and likewise in fig. 5.2 and 5.3) both metrics closely align with the qualitative evaluation, confirming that the first four and the last two blocks exert the greatest impact on the final image quality. Consequently, these metrics are selected as primary criteria for identifying redundant blocks in PixArt- Σ . This ranking serves as foundation for the subsequent, more sophisticated block selection algorithms.

Blocks Selection Algorithms

A further key challenge after establishing the pruning criteria as the combination of CLIP-score and CMMD in structural model distillation is identifying the optimal combinations of blocks to remove simultaneously. When removing only a single block, there are merely 28 possibilities for Pixart- Σ . However, if the goal is to reduce the model size by 50% necessitating the removal of 14 blocks, the number of possible block combinations escalates to $\binom{28}{14}$. Therefore, it is computationally infeasible to evaluate the CLIP-score and CMMD for every possible combination.

A natural alternative is to perform a greedy search algorithm. In this approach, the importance ranking for all blocks is initially computed, after which the least important block is identified and compressed. Next, the CLIP-scores and CMMD are re-evaluated for the remaining blocks within the newly reduced architecture. By iteratively evaluating and compressing the least important block the model architecture is progressively reduced while accounting for the interdependencies between blocks.

Another approach is to reformulate the block selection problem as hyperparameter search using the hyperparameter framework Optuna. The number of blocks to compress N_c , the blocks that are compressed $\{B_i\}_{i=1,\dots,N_c}$ and the compression ratios of each block $\{\pi_i\}_{i=1,\dots,N_c}$ can be potential hyperparameters to optimize. We focused on identifying the blocks $\{B_i\}_{i=1,\dots,N_c}$ given N_c and $\{\pi_i\}_{i=1,\dots,N_c}$ to reduce the search space. As objective function the CMMD metric is used computed similarly like for the greedy algorithm on 100 reference images from LAION dataset. To further speed up the search process, the results of the greedy-algorithm are given as prior to Optuna such that the first trials are not picked randomly which would increase the search time drastically until a good setting is found.

The Optuna hyperparameter search was tested for the iterative SVD distillation framework with fixed compression ratios

Table 5.1: Description of your table data.

Iteration (Parameters)	Number Removed Blocks N_c	Greedy Algo $\{B_i\}_{i=1,\dots,N_c}$	Optuna $\{B_i\}_{i=1,\dots,N_c}$	Greedy CMMD	Algo	Optuna CMMD
1	11	[0,1,2,3,4,5,6,16, 21,23,27]	[0,1,2,3,4,6,7,16, 17,25,27]	0.122		0.086
2	4	[5,8,18,20]	[5,8,19,20]	0.128		0.097
3	3	[9,11,18]	[9,11,21]	0.188		0.171
4	3	[10,18,23]	[14,18,22]	0.551		0.292

Chapter 6

Appendix

6.1 Appendix A

6.1.1 Block Analysis

Heatmap for cka: Block importance for different time steps

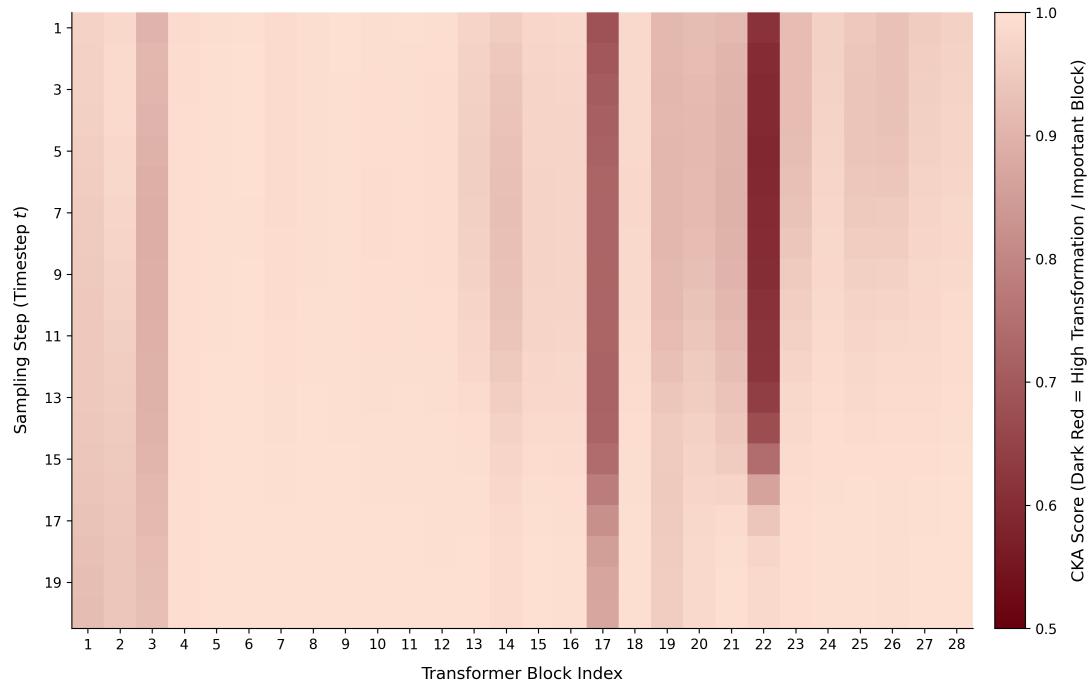


Figure 6.1: CKA-Kernel Clip

Bibliography

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, and Shyamal Anadkat. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Vivago AI. Hidream-l1. 2025. URL: <https://vivago.ai/home>.
- [3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.
- [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/clause-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] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.
- [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] 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.
- [12] 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.
- [13] 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.
- [14] 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.

BIBLIOGRAPHY

- [15] Junsong Chen, Yue Wu, Simian Luo, Enze Xie, Sayak Paul, Ping Luo, Hang Zhao, and Zhenguo Li. Pixart- δ : Fast and controllable image generation with latent consistency models. *arXiv preprint arXiv:2401.05252*, 2024.
- [16] 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.
- [17] 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.
- [18] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.
- [19] 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.
- [20] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [21] Jaemin Cho, Yushi Hu, Roopal Garg, Peter Anderson, Ranjay Krishna, Jason Baldridge, Mohit Bansal, Jordi Pont-Tuset, and Su Wang. Davidsonian scene graph: Improving reliability in fine-grained evaluation for text-to-image generation. *arXiv preprint arXiv:2310.18235*, 2023.
- [22] Krzysztof Choromanski, Valerii Likhoshesterov, 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.
- [23] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.
- [24] 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.
- [25] Javier Martín Daniel Verdú. Flux.1 lite: Distilling flux1.dev for efficient text-to-image generation. 2024.
- [26] 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.
- [27] MohammadReza Davari, Stefan Horoi, Amine Natik, Guillaume Lajoie, Guy Wolf, and Eugene Belilovsky. Reliability of cka as a similarity measure in deep learning. *arXiv preprint arXiv:2210.16156*, 2022.
- [28] 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.
- [29] 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.

BIBLIOGRAPHY

- [30] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.
- [31] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.
- [32] 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.
- [33] DC Dowson and BV666017 Landau. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.
- [34] 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.
- [35] Gongfan Fang, Kunjun Li, Xinyin Ma, and Xinchao Wang. Tinyfusion: Diffusion transformers learned shallow. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18144–18154, 2025.
- [36] Gongfan Fang, Kunjun Li, Xinyin Ma, and Xinchao Wang. Tinyfusion: Diffusion transformers learned shallow. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18144–18154, 2025.
- [37] 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.
- [38] Determine Filters’Importance. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.
- [39] fpgaminer. JoyCaption: An open, uncensored image captioning VLM. <https://github.com/fpgaminer/joycaption>, 2024. Accessed: 2026-01-22.
- [40] 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.
- [41] 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.
- [42] Jackson Gorham and Lester Mackey. Measuring sample quality with kernels. In *International Conference on Machine Learning*, pages 1292–1301. PMLR.
- [43] 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.
- [44] 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.
- [45] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *International conference on algorithmic learning theory*, pages 63–77. Springer, 2005.

BIBLIOGRAPHY

- [46] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
- [47] 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.
- [48] 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.
- [49] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [50] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [51] 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.
- [52] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.
- [53] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [54] Xiwei Hu, Rui Wang, Yixiao Fang, Bin Fu, Pei Cheng, and Gang Yu. Ella: Equip diffusion models with llm for enhanced semantic alignment. *arXiv preprint arXiv:2403.05135*, 2024.
- [55] 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.
- [56] Sergey Ioffe. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [57] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [58] Sadeep Jayasumana, Sri Kumar 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.
- [59] 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.
- [60] 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.
- [61] Tero Karras, Miika Aittala, Samuli Laine, Erik Häkkinen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in neural information processing systems*, 34:852–863, 2021.
- [62] Bo-Kyeong Kim, Hyoung-Kyu Song, Thibault Castells, and Shinkook Choi. Bk-sdm: A lightweight, fast, and cheap version of stable diffusion. In *European Conference on Computer Vision*, pages 381–399. Springer.

BIBLIOGRAPHY

- [63] Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.
- [64] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.
- [65] 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.
- [66] Yuval Kirstain, Adam Polyak, Uriel Singer, Shahbuland Matiana, Joe Penna, and Omer Levy. Pick-a-pic: An open dataset of user preferences for text-to-image generation. *Advances in neural information processing systems*, 36:36652–36663, 2023.
- [67] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.
- [68] Shun Kiyono, Sosuke Kobayashi, Jun Suzuki, and Kentaro Inui. Shape: Shifted absolute position embedding for transformers. *arXiv preprint arXiv:2109.05644*, 2021.
- [69] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pages 3519–3529. PMLR, 2019.
- [70] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.
- [71] Black Forest Labs. Flux model. 2024. URL: <https://bfl.ai>.
- [72] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 2002.
- [73] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning. *arXiv preprint arXiv:2010.07611*, 2020.
- [74] Youngwan Lee, Kwanyong Park, Yoorhim Cho, Yong-Ju Lee, and Sung Ju Hwang. Koala: Empirical lessons toward memory-efficient and fast diffusion models for text-to-image synthesis. *Advances in Neural Information Processing Systems*, 37:51597–51633, 2024.
- [75] C Li, H Xu, J Tian, W Wang, M Yan, B Bi, J Ye, H Chen, G Xu, Z Cao, et al. mplug: Effective and efficient vision-language learning by cross-modal skip-connections. arxiv 2022. *arXiv preprint arXiv:2205.12005*, 1, 2022.
- [76] Daiqing Li, Aleks Kamko, Ehsan Akhgari, Ali Sabet, Linmiao Xu, and Suhail Doshi. Playground v2.5: Three insights towards enhancing aesthetic quality in text-to-image generation, 2024. [arXiv:2402.17245](https://arxiv.org/abs/2402.17245).
- [77] 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.
- [78] 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.
- [79] Shanchuan Lin, Anran Wang, and Xiao Yang. Sdxl-lightning: Progressive adversarial diffusion distillation. *arXiv preprint arXiv:2402.13929*, 2024.

BIBLIOGRAPHY

- [80] 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.
- [81] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.
- [82] Bingchen Liu, Yizhe Zhu, Kunpeng Song, and Ahmed Elgammal. Towards faster and stabilized gan training for high-fidelity few-shot image synthesis. In *iclr*.
- [83] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.
- [84] 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.
- [85] 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.
- [86] 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.
- [87] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.
- [88] Jian Ma, Qirong Peng, Xujie Zhu, Peixing Xie, Chen Chen, and Haonan Lu. Pluggable pruning with contiguous layer distillation for diffusion transformers. *arXiv preprint arXiv:2511.16156*, 2025.
- [89] Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect, 2024. URL <https://arxiv.org/abs/2403.03853>, 2(3):4, 2024.
- [90] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306.
- [91] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kortscheder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 4990–4999, 2017.
- [92] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR.
- [93] 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.
- [94] 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.
- [95] Jan Pawłowski and Tilman Plehn. Physics and machine learning. Lecture script, Institut für Theoretische Physik, Universität Heidelberg, January 2024. Version from January 30, 2024.

BIBLIOGRAPHY

- [96] 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.
- [97] 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.
- [98] 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.
- [99] Ian Pons, Bruno Yamamoto, Anna H Reali Costa, and Artur Jordao. Effective layer pruning through similarity metric perspective. In *International Conference on Pattern Recognition*, pages 423–438. Springer, 2024.
- [100] 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.
- [101] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
- [102] 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.
- [103] 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.
- [104] 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.
- [105] 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>.
- [106] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR.
- [107] 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.
- [108] 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.
- [109] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [110] 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.

BIBLIOGRAPHY

- [111] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.
- [112] Axel Sauer, Frederic Boesel, Tim Dockhorn, Andreas Blattmann, Patrick Esser, and Robin Rombach. Fast high-resolution image synthesis with latent adversarial diffusion distillation. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11.
- [113] 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.
- [114] Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision*, pages 87–103. Springer.
- [115] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems*, 35:25278–25294, 2022.
- [116] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. LAION-Aesthetic v1. <https://github.com/LAION-AI/laion-datasets/blob/main/laion-aesthetic.md>, 2022. Accessed: 2026-01-22.
- [117] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 8430–8439, 2019.
- [118] 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.
- [119] 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.
- [120] 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.
- [121] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. 2023.
- [122] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.
- [123] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.
- [124] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. *Advances in neural information processing systems*, 33:12438–12448, 2020.
- [125] 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.

BIBLIOGRAPHY

- [126] 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.
- [127] 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.
- [128] Inga Strümke and Helge Langseth. Lecture notes in probabilistic diffusion models. *arXiv preprint arXiv:2312.10393*, 2023.
- [129] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.
- [130] 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.
- [131] 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.
- [132] 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.
- [133] TencentARC. flux-mini. <https://huggingface.co/TencentARC/flux-mini>, 2024. Hugging Face Model Hub.
- [134] 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.
- [135] Luke Tierney. Markov chains for exploring posterior distributions. *the Annals of Statistics*, pages 1701–1728, 1994.
- [136] 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.
- [137] Aaron Van Den Oord and Oriol Vinyals. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.
- [138] 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.
- [139] Xin Wang, Samiul Alam, Zhongwei Wan, Hui Shen, and Mi Zhang. Svd-llm v2: Optimizing singular value truncation for large language model compression. *arXiv preprint arXiv:2503.12340*, 2025.
- [140] 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.

BIBLIOGRAPHY

- [141] 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.
- [142] 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.
- [143] 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.
- [144] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and Bill Freeman. Improved distribution matching distillation for fast image synthesis. *Advances in neural information processing systems*, 37:47455–47487, 2024.
- [145] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6613–6623.
- [146] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv preprint arXiv:2206.10789*, 2(3):5, 2022.
- [147] Zhihang Yuan, Yuzhang Shang, Yue Song, Dawei Yang, Qiang Wu, Yan Yan, and Guangyu Sun. Asvd: Activation-aware singular value decomposition for compressing large language models. *arXiv preprint arXiv:2312.05821*, 2023.
- [148] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.
- [149] Dingkun Zhang, Sijia Li, Chen Chen, Qingsong Xie, and Haonan Lu. Laptop-diff: Layer pruning and normalized distillation for compressing diffusion models. *arXiv preprint arXiv:2404.11098*, 2024.
- [150] 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.
- [151] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.
- [152] Zhihua Zhang. The singular value decomposition, applications and beyond. *arXiv preprint arXiv:1510.08532*, 2015.
- [153] Youwei Zheng, Yuxi Ren, Xin Xia, Xuefeng Xiao, and Xiaohua Xie. Dense2moe: Restructuring diffusion transformer to moe for efficient text-to-image generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18661–18670, 2025.