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

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

CMMD Fréchet Inception Distance

CNF Continous Normalizing Flow

CNN Convolutional Neural Network

DDIM Denoising Diffusion Implicit Model

DDPM Denoising Diffusion Probablistic 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

GRASP Gradient-based Retention of Adaptive Singular Parameters

HPSv2 Human Preference Score v2

HSIC Hilbert-Schmidt Independence Criterion

iid independent and identically distributed

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

PTQ Post-Training Quantization

PTQD Post-Training Quantization for Diffusion Models

QAT Quantizaiton-Aware Training

RMS Root Mean Square

RMSNorm Root Mean Square Normalization

RNN Recurrent Neural Network

RoPE Rotary Positiona Embeeding

SDE Stochastic Differential Equation

SGM Score-Based Generative Model

SOTA State-Of-The-Art

STE Straight-Through Estimator

SVD Singular Value Decomposition

VAE Variational Autoencoder

VI Variational Inerence

ViT Vision Transformer

VRAM Video Random Access Memory

List of Figures

List of Tables

Chapter 1

Introduction

Over the past few years, generative artificial intelligence (AI) has gained immense importance and relevance in society, as well as in business and politics. In addition to large language models (LLMs) [1, 5, 157, 161], which have brought generative AI into the mainstream, generative image models [9, 14, 19, 86, 120, 129] are increasingly coming into focus. On the one hand, they open up a range of new possibilities in areas such as design and marketing or gaming and entertainment. On the other hand, they enable the generation of highly realistic synthetic images and videos, often referred to as deepfakes, raising severe ethical and privacy concerns, which is why policymakers are striving to establish rule-based guidelines for their use (European AI Act).

The advent of deep learning based image generators is often marked by the introduction of variational autoencoders (VAEs) in 2013 [77], as they demonstrated that neural networks could learn complex data distribution and generate novel samples. Subsequently, generative adversarial networks (GANs) [49, 74, 137] became the dominant model type in the image generation field as they produce images of significantly higher quality than VAEs. However, GANs are notoriously difficult to train, as they often exhibit mode collapse and suffer from exploding or vanishing gradients [99, 134, 159].

In recent years, GANs were largely surpassed by diffusion models [63] deploying an iterative image generation denoising process. The breakthrough of diffusion models as the new state-of-the-art (SOTA) in image generation occured mainly in 2022 when the latent diffusion models, most notably Stable Diffusion [130], were released. At this time, they excelled in image quality and image generation diversity, but came with the cost of high inference time due to the iterative nature of their image generation process. The transition from pixel space to latent space massively reduced the computing time of the models and enabled images with higher resolution to be generated.

The first stable diffusion models [120, 129] leveraged a U-Net architecture [131] and already required a huge amount of training data and computational resources. The image generation process is typically controlled by textual prompts provided in the stable diffusion models via cross-attention mechanisms [163]. To expand these capabilities, several new control and finetuning methods like ControlNet [176] and DreamBooth [132] were developed to enable precise spatial conditioning and personalization. Since the U-Net-based SD models already included transformer layers [163] for text-conditioning, the next logical evolution was to transition to purely transformer-based models known as diffusion transformers (DiTs) [118], which also increased significantly in parameter size and required training data. In addition, a new formulation, so-called flow-matching, became predominant due to improved training stability and sampling efficiency which is leveraged by the current SOTA image generation models like Flux.1-dev [86] being a 11.9 billion parameter model.

Possible areas of application for these models include edge devices such as mobile phones [26, 96, 180]. In these cases, it is advantageous to run these models directly on the edge devices if possible in order to avoid server costs, to make the associated functionality available offline, and to protect the privacy of users so that, for example, images do not have to be uploaded to external servers for processing. However, the current flow-based

models like Flux.1-dev face two distinct challenges for real-time applications on edge devices. On the one hand, as mentioned before, they suffer from a long inference time due to the sequential application of the model. On the other hand, these models also require a substantial amount of VRAM during inference, e.g. Flux.1-dev requires about 32GB, which makes them unsuitable on edge devices like smartphones.
Beyond edge device deployment, reducing VRAM and faster execution plays a pivotal role in democratising these models and making them more environmentally friendly, as they can run on consumer-grade hardware and do not require power-intensive data centres reducing carbon footprint.
There are several approaches to speed up inference time. For this, the main research stream is to enable the model to generate high-quality images with fewer inference steps. While at the beginning, diffusion models required up to 1000 inference steps to generate one single image, using more advanced ODE-solvers for the denoising trajectory has led to significant reduction in inference steps. Another line of research is step- and guidance distillation. Distillation methods generally attempt to transfer the capabilities of a teacher model to a student model. In step distillation, the same diffusion model is often used as both the teacher and student model, and the goal is to teach the student model to generate images of the same or similar quality with as few as four or two, sometimes even only one inference step leveraging its original capabilities [107, 135, 136, 138, 145, 171]. This technique is for example applied to Flux.1-dev resulting in Flux.1-schnell, reducing the required inference steps to only four. In guidance distillation, the goal is to incorporate the concept of classifier free guidance (CFG), which normally requires the model to be applied twice per inference step, into the model itself, thereby saving half of the model evaluations [111]. This procedure was utilized to distill Flux.1-pro into Flux.1-dev.
However, while step distillation and advanced solver effectively reduce the latency (inference time), they do not necessiarily adress the second bottleneck, namely high VRAM consumption. Even a faster model cannot be employed to an edge device if its parameters exceed the available VRAM. Proposed methods for VRAM reduction range from quantization [54, 168] and more efficient implementation of model components, e.g. Flash Attention [33], to structural or unstructural model pruning [16, 32, 75, 85, 89, 108, 158, 175, 178]. While quantization compresses the model size by reducing the precision of the model parameters, pruning directly addresses the phenomenon that modern models, which have a high number of parameters, exhibit high redundancy in and between the individual transformer blocks. For this reason, identification followed by compression of redundant model parts is a promising orthogonal approach to reducing inference steps for further efficiency gains.
Therefore, the first part of this thesis systematically examines how the number of parameters in diffusion transformers can be effectively reduced while minimizing the degradation of image generation quality, using PixArt-Σ [19] as a representative baseline enabling the execution of numerous experiments due to its significantly smaller parameter count compared to Flux.1-dev. Specifically, the identification of architectural components with redundant parameters alongside the best pruning strategies, e.g., complete layer removal versus low-rank compression, and pruning schedules, e.g., one-shot versus progressive pruning are investigated. Furthermore, various post-compression retraining frameworks are evaluated to optimize performance recovery. The empirical results demonstrate that progressively compressing model blocks via low-rank approximation using a linear distribution of compression ratios meaning compressing the least important structures most aggressively and more critical structures less, leads to the best results.
In the second part of this thesis, the previously identified best pruning strategy is applied to the Flux.1-dev model and compared with other existing compression methods.
Furthermore, a training-free compression strategy originally proposed for LLMs GRASP that is also based on singular value decomposition (SVD) is transferred to Flux.1-dev and tested in comparison with current alternative approach [178].

In the following, chapter two provides a comprehensive overview of image generator models, emphasizing diffusion and flow models, alongside relevant metrics used for measuring the performance of image generators. Chapter three reviews alternative model compression methods focusing on methods applied to diffusion and

flow models. Chapter four introduces the specific methodologies investigated in this work. In chapter five, the experimental results are presented tp assess the effectiveness of these methods. Finally, chapter six concludes the thesis with a comprehensive discussion of the main findings and their implications.

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 [157] or Lama [161] whereas Stable Diffusion [120, 129], Flux [86] and Pixart [19] 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) [77], generative adversarial networks (GANs) [49, 123], normalizing flows (NFs) [128], and diffusion models (DMs) [142].
The principal idea of a generative model f_θ is to learn a mapping from a simple distribution, e.g., a normal distribution $p_{\text{latent}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, often called the latent distribution, to a highly complex data distribution $p_{\text{data}}(\mathbf{x})$. A new data element $\mathbf{x}$ is obtained by sampling from the latent distribution $\mathbf{z} \sim p(\mathbf{z})$ and applying the model

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

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

2.1.1 Variational Autoencoder

Autoencoder

An autoencoder [62] 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 [12, 77]. The VAE [77] belongs to the class of probabilistic models. A common learning objective for probabilistic models is maximum likelihood estimation (MLE). Given a set of observations $\{\mathbf{x}_i\}_{i=1}^N$ the likelihood is given under the assumption of identical and independent distributed (iid) samples as

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

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

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

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

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

leading to the the so-called evidence lower bound (ELBO) loss [12] being used as training objective for VAEs. In general, untractable probability distributions are a core challenge in modern statistics. This is especially the case for posterior distributions in Bayesian statistics. Assuming the joint probability distribution

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

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 [160], 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) [13]. In contrast to MCMC VI reformulates the problem as an optimization task

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z})\in\mathscr{Q}}{\arg\min}\ \text{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\right] \tag{2.6}$$

where $\mathscr{Q}$ is a family of distributions. The goal is to find the proposal density $q(\mathbf{z}) \in \mathscr{Q}$ that best approximates the true posterior distribution $p(\mathbf{z}|\mathbf{x})$. To compare the proposal density and the posterior the Kullback-Leibler divergence (KL) [84] is used having the general form of

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

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

$$\begin{aligned}
\mathrm{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\right] &= \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})} d\mathbf{z} && (2.8)\\
&= \int q(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \log p(\mathbf{z}|\mathbf{x}) d\mathbf{z} && (2.9)\\
&= \int q(\mathbf{z}) \log q(\mathbf{z}) d\mathbf{z} - \int q(\mathbf{z}) \log \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} d\mathbf{z} && (2.10)\\
&= \mathbb{E}\left[\log q(\mathbf{z})\right] - \mathbb{E}\left[\log p(\mathbf{z}, \mathbf{x})\right] + \log p(\mathbf{x}) && (2.11)
\end{aligned}$$

From here, the ELBO loss is obtained by

$$\begin{aligned}
\log p(\mathbf{x}) &\geq \log p(\mathbf{x}) - \mathrm{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z}|\mathbf{x})\right] && (2.12)\\
&= \mathbb{E}\left[\log p(\mathbf{z}, \mathbf{x})\right] - \mathbb{E}\left[\log q(\mathbf{z})\right] && (2.13)\\
&= \mathbb{E}\left[\log p(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left[q(\mathbf{z}) \,\|\, p(\mathbf{z})\right] && (2.14)\\
&= \mathrm{ELBO} \quad . && (2.15)
\end{aligned}$$

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

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

In practice, the latent distribution is often set to a standard normal distribution $p_{\mathrm{latent}}(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{1})$ and the reparameterization trick [77] enabling gradient computation is applied. After training, a new data point can be generated by sampling from the latent distribution $\mathbf{z} \sim p_{\mathrm{latent}}(\mathbf{z})$ and applying the decoder of the trained model.

2.1.2 Normalizing Flow

Normalizing flows (NFs) [128] are another class of propabilistic models, similar to VAEs, which learn a mapping between a latent distribution and the data distribution. However, NFs have some key differences compared to VAEs. First, NFs are fully invertible, meaning the network F_θ consists of a sequence of (simpler) bijective operations so that an inverse transformation $\overline{F_\theta}$ mapping back from data to latent distribution exists. As a consequence, the latent space and the data space need to have the same dimension.
A second decisive advantage is that NFs allow the exact and tractable computation of the likelihood using the change of variables formula [117]

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

for the latent distribution and

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

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 [38]) 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 [49] belong to the class of non-probabilistic generators which do not require an explicit likelihood function. The model, also called the generator G, is trained in an adversarial training setup that contains, on the one hand, the generator that should generate new data points being as realistic as possible and, on the other hand, a discriminator that should be able to distinguish if a data point is generated by the generator ("fake") or from the original dataset ("real"). The generator and the discriminator are trained in an alternating fashion, where the generator tries to fool the discriminator by generating more realistic data points, and the discriminator D becomes better at distinguishing "fake" from "real". This is formalized in a MinMax condition

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

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 [74, 101, 137] 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 [99, 134]. 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 [159], 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 [63, 130, 143, 144, 146, 150] are probabilistic generative models that have achieved remarkable success in image generation tasks [2, 86], 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 [49, 77, 128], 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, [130] 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 [31]: Denoising diffusion probabilistic models (DDPMs) [63, 143] and its further development denoising diffusion implicit models (DDIMs, score-based generative models (SGMs) [146] and stochastic differential equations (SDEs) [150].

Recently, a generalization of DMs was proposed, known as flow matching or Flow-based models [100, 104]. 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 [86] or Sana [167].
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 [152].
In DDPMs [63] 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 $\mathbf{x}_0, ..., \mathbf{x}_T$ 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.

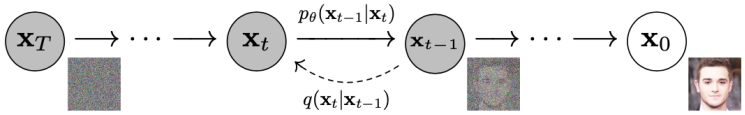


Figure 2.1: Visualization of the forward and backward process of DDPM taken from [63].

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 [63], a cosine [114] or a learnable [76] schedule, to ensure smooth interpolation between p_{prior} and p_{data}. The schedule dictates the incremental increase of variance β_t.

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

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

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

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

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

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\,\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{z}_t && (2.22)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{\alpha_t(1-\alpha_{t-1})}\,\mathbf{z}_{t-1} + \sqrt{1-\alpha_t}\,\mathbf{z}_t && (2.23)\\
&= \sqrt{\alpha_t\alpha_{t-1}}\,\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\,\mathbf{z}_{t-1:t} && (2.24)\\
&= \dots && (2.25)\\
&= \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\mathbf{z}_{0:t}. && (2.26)
\end{aligned}$$

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

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x_t}; \sqrt{\bar{\alpha}_t}\mathbf{x_0}, (1-\bar{\alpha}_t)\mathbf{I}\right) \quad . \tag{2.27}$$

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

$$\mathbf{x_t} = \sqrt{\bar{\alpha}_t}\mathbf{x_0} + \sqrt{(1-\bar{\alpha}_t)}\boldsymbol{\epsilon} \tag{2.28}$$

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 [45] showed that in the limit of infinitesimal steps the reverse process retains the same distributional form as the forward process, a Gaussian. Particularly, in the reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is the quantity of interest and given by

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

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

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

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

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

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 [117] and [63]. Starting with the expectation of the negative log-likelihood under the data distribution $q(\mathbf{x}_0)$ typically equivalent to the empirical data distribution p_{prior}

$$\begin{aligned}
&- \mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0)\right] && (2.32)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T \, p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \right)\right] && (2.33)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \left(\int d\mathbf{x}_1 \dots d\mathbf{x}_T \, p(\mathbf{x}_T) \, q(\mathbf{x}_1, \dots, \mathbf{x}_T \mid \mathbf{x}_0) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right)\right] && (2.34)\\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \left[\log \, \mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T|\mathbf{x}_0)} \left[p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right]\right] && (2.35)
\end{aligned}$$

using Jensen's inequality

$$\begin{aligned}
-\mathbb{E}_{q(\mathbf{x}_0)} \left[\log p_\theta(\mathbf{x}_0)\right] &\leq -\mathbb{E}_{q(\mathbf{x}_0)} \left[\mathbb{E}_{q(\mathbf{x}_1, \dots, \mathbf{x}_T|\mathbf{x}_0)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right)\right]\right] && (2.36)\\
&= -\mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[\log \left(p(\mathbf{x}_T) \prod_{t=1}^{T} \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right)\right] && (2.37)\\
&= \mathbb{E}_{q(\mathbf{x}_0, \dots, \mathbf{x}_T)} \left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})} \right] && (2.38)\\
&=: \mathcal{L}_{\text{DDPM}} && (2.39)
\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

However, instead of predicting the mean, diffusion models are more often trained on predicting the noise $\boldsymbol{\epsilon}_\theta$ because [63] empirically found that this lead to more stable training

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

which is achieved by reparameterization of eq. 2.46 via

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

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

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

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 $[0, ..., T]$ 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 [152]

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

until converged

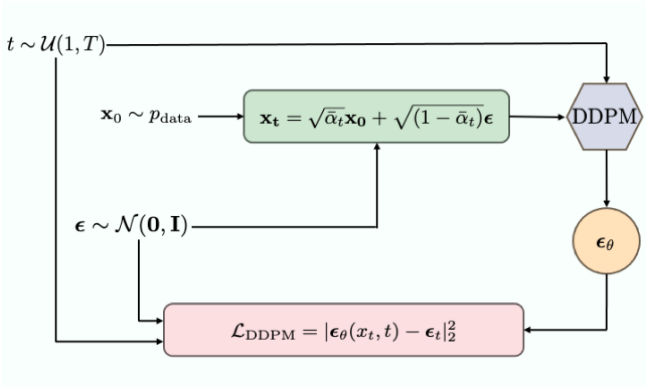


Figure 2.2: Visualization of the training procedure for DDPM. Taken from [15].

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

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

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

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

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 [152]

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

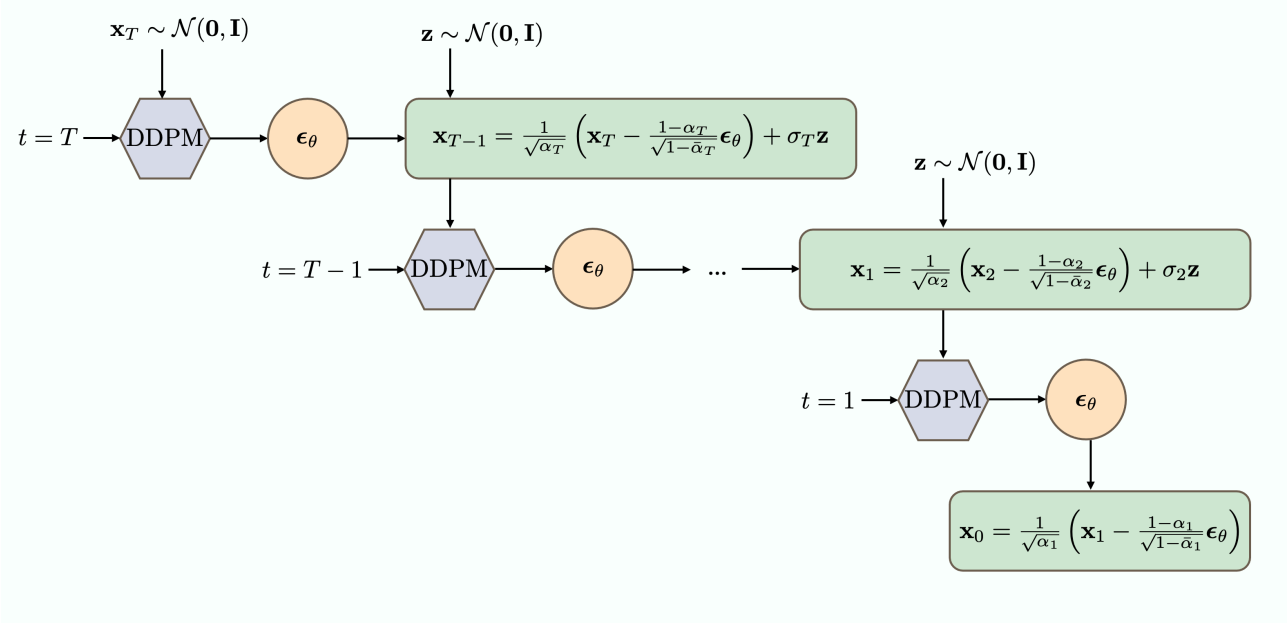


Figure 2.3: Visualization of the iterative sampling process of DDPM. Taken from [15].

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

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

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 [144] parameterize the reverse process as

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

The corresponding sampling equation becomes

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

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 $\boldsymbol{\epsilon}_\theta$. The second part is a directional term describing the transition between $\mathbf{x}_t$ and $\mathbf{x}_0$ and the last part adds Gaussian noise scaled by σ_t. The key advantage of DDIM is the flexibility in choosing σ_t, which does not need to follow a predefined variance schedule. Different values of σ_t yield different diffusion processes while using the same trained model. In particular, for the choice

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

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 [31, 169]. Score-Based Generative Models (SGMs) [147, 148] 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 [50, 149, 151]. Here, the one closest to the DDPM approach is presented [147]. During training, the data is perturbed by Gaussian noise at different noise levels and a neural network is trained to estimate the score function based on the noise level $s_\theta(\mathbf{x}_t, t) \approx \nabla_{\mathbf{x}_t} \log q(x_t)$. Perturbing the data is necessary because the score estimation is inaccurate in low-density regions and to ensure the process does not get stuck. These networks are called Noise-Conditional Score Networks (NCSNs). The learning objective boils down to

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

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

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

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

In eq. 2.61 the relation to DDPM loss becomes visible by identifying $\boldsymbol{\epsilon}(\mathbf{x}_t, t) = -\sigma_t s_\theta(\mathbf{x}_t, t)$ [169], showing that learning to predict noise is equivalent to learning the score function.
For sampling a new element from the data distribution [147] proposed Annealed Langevin Dynamics (ALD) which produces samples by only using the score function being approximated by the learned neural network

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

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 [31, 169]. Stochastic Differential Equations (SDEs) [151] are a generalization of SGMs and DDPMs to continuous time where the forward and the reverse process are modeled by the solution of stochastic differential equations. The forward process is described by the SDE

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

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

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

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

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

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 [151] 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 by the deterministic version of eq. 2.64

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

Both reversed-SDE or probability flow ODE can be used to generate new samples using SDE solvers [71, 151] or respectively ODE solvers [73, 106, 151, 177].

2.2.5 Flow Matching

The concept of flow matching [100] builds upon continuous normalizing flows (CNF)s [24]. 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

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

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

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

However, the vector field $\mathbf{u}_t(\mathbf{x})$ is unknown which is why [100] 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 sampel-based conditional probability paths $p_t(\mathbf{x}|\mathbf{x_1})$

$$p_t(\mathbf{x}) = \int p_t(\mathbf{x} \mid \mathbf{x}_1) q(\mathbf{x}_1) d\mathbf{x}_1 \tag{2.69}$$

and its corresponding marginalized vector field is defined as

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

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

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

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

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

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, [104] proposed an iterative algorithm called Reflow to untangle and straighten the trajectories. This is achieved by iterativelly 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

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

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

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

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)

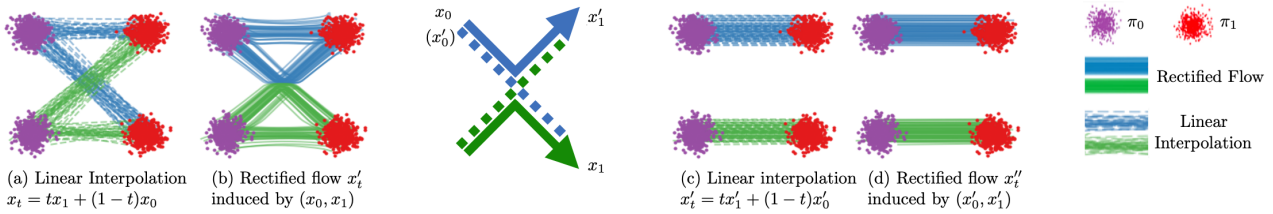


Figure 2.4: Visualization of the path unwiring and straighting of the first iteration in the Reflow algorithm. Taken from [104].

Algorithm 3 Rectified Flow: Main Algorithm [104]

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

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

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 [130] and are widely used nowadays [19, 86, 120, 129]. 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 [77] or VQ-regularization [162]. 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)

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

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 denosiding 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.

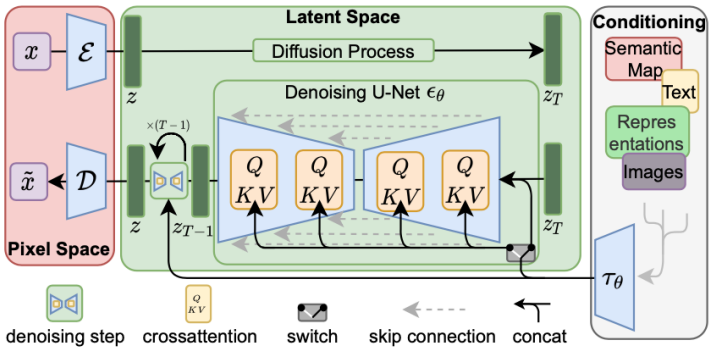


Figure 2.5: Visualizaiton of the structure of a latent diffusion model. Taken from [130].

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 [176], or images [155] users can precisly control the content of the final image. Formally, the diffusion model is trained to learn the conditioned distribution $p(\mathbf{x}|\mathbf{y})$ where $\mathbf{y}$ represents the additional conditioning signal. Consequently, the noise prediction function is modified from $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{y}, t)$. Cross-attention layers (see section 2.3.2) [163] 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 [19]. 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) [64]. This method builds upon classifier guidance [37], which was introduced for class conditioned diffusion models. In the classifier guidance framework, during training an external classifier predicts the class label and its gradients are used to steer the diffusion model to produce images that can be easily identified by the classifier. However this necessitate the training of an additional classifier which is able to operate on noisy input images as standard classifier are unsuitable for this task. The goal is to steer the diffusion through the additional learning signal to producing images with higher fidelity to the conditioning signal. To facilitate the training process and get rid of the extra classifier CFG was introduced. In the CFG framework, the diffsuion model is jointly trained on conditional $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{y})$ and unconditional $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \mathbf{y} = \varnothing)$ objectives. For that during training the condition is randomly dropped with the probability $p_{\text{unconditonal}}$. During inference, the same model is used twice to predict the conditional noise and the unconditional noise which are linearly combined to

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

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 [163] 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, 86, 157].
The transformer was introduced as an alternative to recurrent neural networks [133] 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, 161] or encoder-only transformer [36].
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 [56] 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.

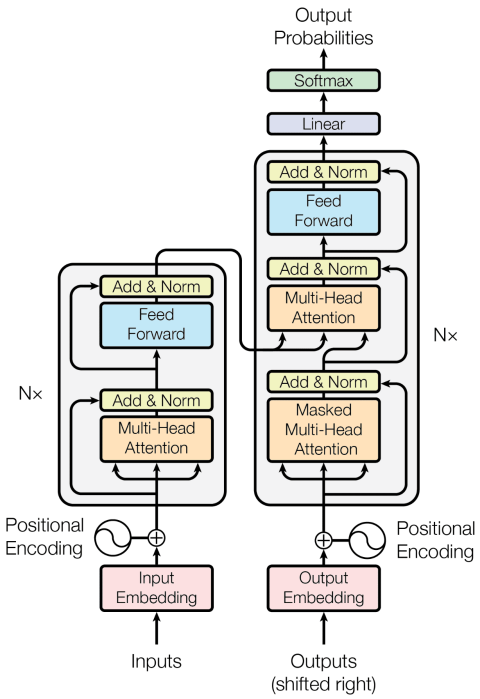


Figure 2.6: Visualization of the transformer architecture. Taken from [163].

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 [163] 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

$$Q = X \cdot W_q^T \tag{2.77}$$

$$K = X \cdot W_k^T \tag{2.78}$$

$$V = X \cdot W_v^T \tag{2.79}$$

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

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

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

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

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

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

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 [27, 29, 33, 80].

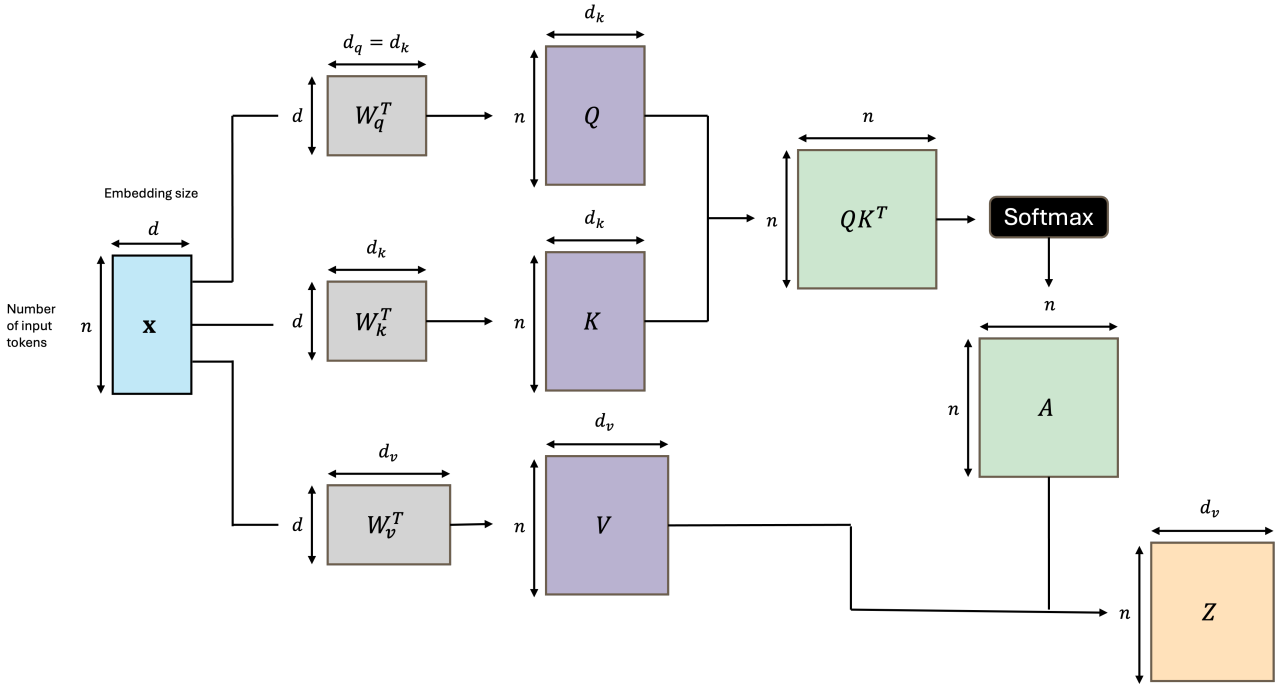


Figure 2.7: Visualization of the self-attention mechanism. Based on [127].

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.

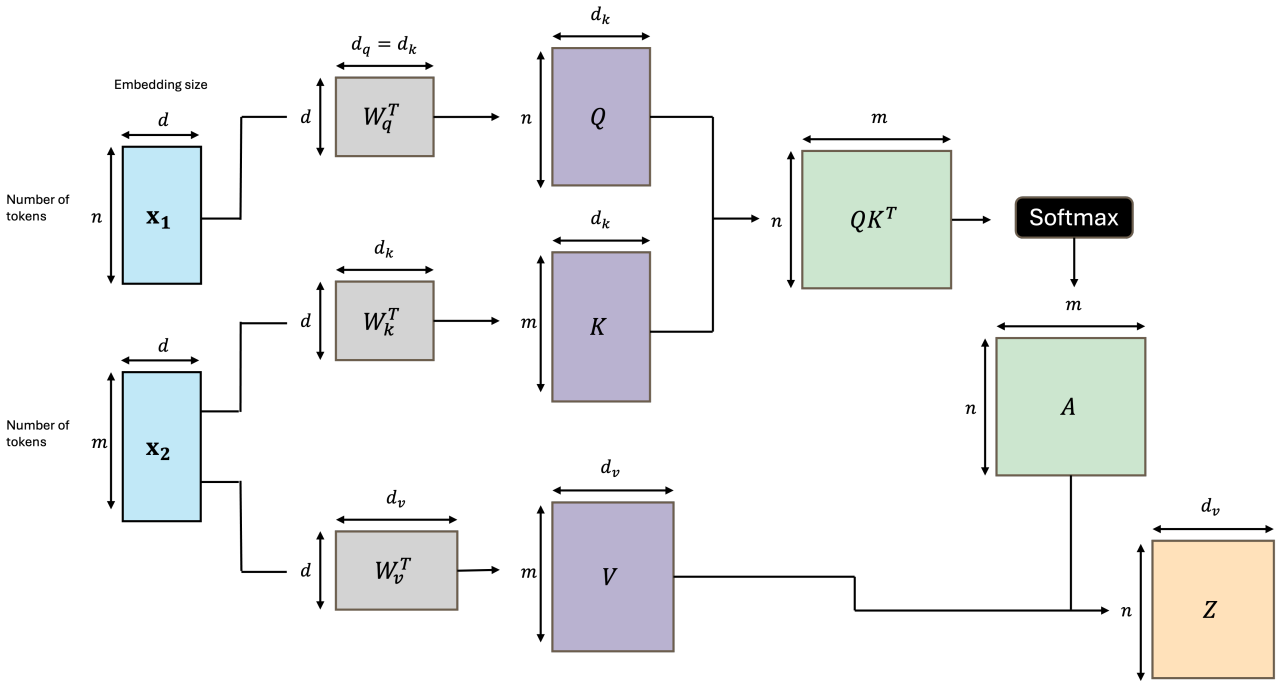


Figure 2.8: Visualization of the cross-attention mechanism. Based on [127].

Multi-Head Attention

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

$$\text{MultiHead}\left(Q, K, V\right) = \text{Concat}\left(\text{head}_1, ..., \text{head}_N\right) W^O \tag{2.83}$$

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

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.

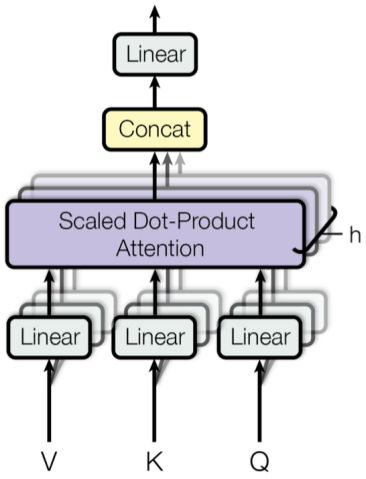


Figure 2.9: Visualization of the multi-head attention mechanism. Taken from [163].

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 [94, 115, 153] versus absolute [81, 97, 105, 163] positional embedding or deterministic [81, 97, 115, 153, 163] versus learnable [94, 105] positional embedding.

Sinusoidal Positional Encoding

Sinusoidal positional encoding is used in the original transformer paper [163] which leverages the sinus and cosine function to inject positional information into the model. It belongs to the category of absolute and deterministic positional encoding.

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

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

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 [67, 116, 125], but for a long time, convolutional neural networks (CNNs) [126] remained the dominant architecture. CNNs have an inherent advantage over transformers because they possess an image-related inductive bias. On the one hand, CNNs are translationally equivariant by construction and, on the other hand, the concept of locality is embedded in the way they are built. In contrast, these two concepts must be learned by a transformer, as they are not predefined by its architecture, which makes a sufficiently large dataset essential to achieve competitive performance.

The introduction of vision transformer (ViT) [40] 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.

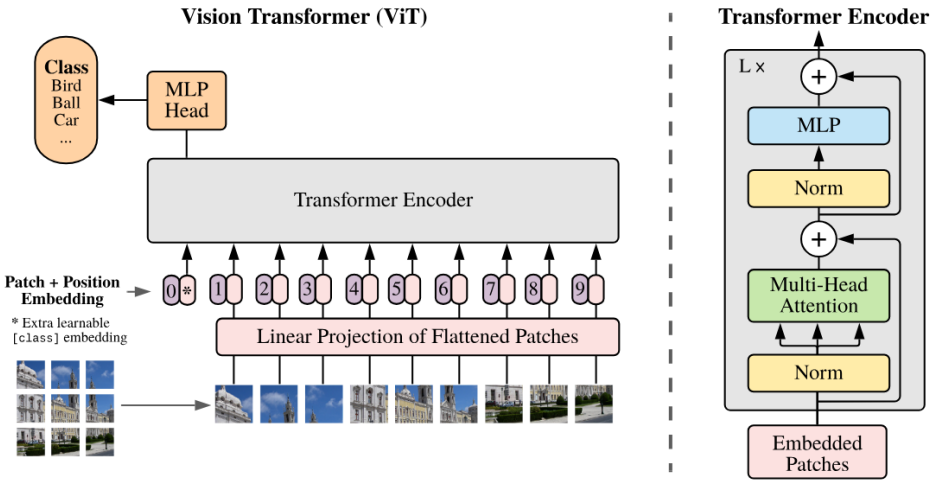


Figure 2.10: Visualization of vision transformer architecture. Taken from [40].

2.3.5 Adaptive Layer Normalization

Layer normalization [7] was introduced as an alternative to batch normalization [68] as a normalization method to reduce the internal covariate shift in deep neural networks independently of the batch size. The covariate shift describes the phenomenon whereby the range of the input to a layer which is the ouptut of the previous layer changes permanently, which hinders effective learning because the layer needs to constantly adapt to a new input distribution. To stabilize and accelerate the training process normalizations are applied in the neural

network. In layer normalization, the activations of each layer $\mathbf{z}_l$ are normalized to a zero mean and unit variance. In addition, a scale γ and a shift η parameter are learned to maintain the flexibility and expressiveness of the network and to ease the constrained range due to the normalization. The normalized layer output is given by

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

A further development of layer normalization occured when [119] 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 $\gamma \to \gamma(c)$ and shift $\eta \to \eta(c)$. This method is known as adaptive layer normalization (adaLN) and can be leveraged in diffusion models to incorporate the timestep or class conditioning [118].

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 primarly 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 [86] 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 [136, 138], progressive distillation [98, 135], distribution matching distillation [170, 171], consistency models [107, 145] and guided distillation [111] were proposed. The second strategy, focusing on structural-based knowledge distillation [61] 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 [61] was originally introduced for training a small model, referred to as the student, on classification tasks such as MNIST [87]. 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 $\mathcal{L}_{\text{KD}}$, calculated using the outputs from the teacher, and the task loss $\mathcal{L}_{\text{Task}}$, calculated with the hard labels, is advantageous

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

This idea has been adapted to train smaller versions of diffusion [75, 89, 175] and flow [16, 32, 44, 109, 158, 181] 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.

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

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

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{OutKD}} \mathcal{L}_{\text{OutKD}} + \lambda_{\text{FeatKD}} \mathcal{L}_{\text{FeatKD}} \tag{2.90}$$

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 follwing, the metrics used in this work are presented: Fréchet Inception Distance (FID) [60], CLIP-Maximum Mean Discrepancy (CMMD) [70], Human Preference Score v2 (HPSv2) [166]

2.5.1 Fréchet Inception Distance

The FID [60] 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 [154], extracts features for every image of the sets. Under the assumption that the features from both sets follow a multivariate normal distribution the mean and covariance of the generated image features ($\boldsymbol{\mu}_{\text{model}}$, Σ_{model}) and the features from the original data ($\boldsymbol{\mu}_{\text{data}}$, Σ_{data}) are computed and compared by the Fréchet Distance [41]

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

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

Although FID is widely used [70] identified several shortcomings. First, the underlying Inception-v3 model is trained only on images from 1,000 classes which leads to embeddings that often fail to capture the rich and varied content of the images generated by current diffusion models. Secondly, the normality assumption for the feature distributions is frequentyl violated in reality which can lead to FID scores which do not align to human ratings. A third drawback is that the FID score does not capture complex distortion and depends highly sensitive on the sample size and exhibit bias. For these reasons [70] proposed an alternative score which is discussed next.

2.5.2 CLIP-Maximum Mean Discrepancy

Similar to the FID, the CMMD [70] score measures the discrepancy between the data and the model distribution. It replaces the Inception-v3 embeddings with Contrastive Language- Image Pre-training (CLIP) [122] embeddings. Since CLIP is trained on more images and leverages natural langugage supervision, its embeddings have better representation able to capture more semantics details of the images. The features are compared via the maximum mean discrepancy (MMD) [51, 52]

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

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. [70] showed in several ablation studies that the CMMD 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 [166] is designed to rank generated images based on human preferences. To model the human preference score, a CLIP model[122] is finetuned on the HPDv2 dataset [166] which contains 434,000 images, generated by nine different text-to-image models or taken from the COCO dataset with over 798,000 human preference choices. A human preference choice always includes two images $\mathbf{x}_1$ and $\mathbf{x}_2$ and a binary ranking $\mathbf{y}$ indicating whether image $\mathbf{x}_1$ is preferred over image $\mathbf{x}_2$, e.g. $\mathbf{y} = [1, 0]$ means that $\mathbf{x}_1$ is preferred by the annotator. The human preference score is determined by the finetuned CLIP model via

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

where p is the text prompt, x is the image, and $\mathcal{E}_{\text{img}}$ and $\mathcal{E}_{\text{txt}}$ are the image and text encoder of the CLIP model, respectively. The HPSv2 benchmark contains four categories, namely "Animation", "Concept-Art", "Painting", and "Photo" with 800 prompts each. The prompts are derived on COCO captions [25] and DiffusionDB [165] 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 [48] 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.

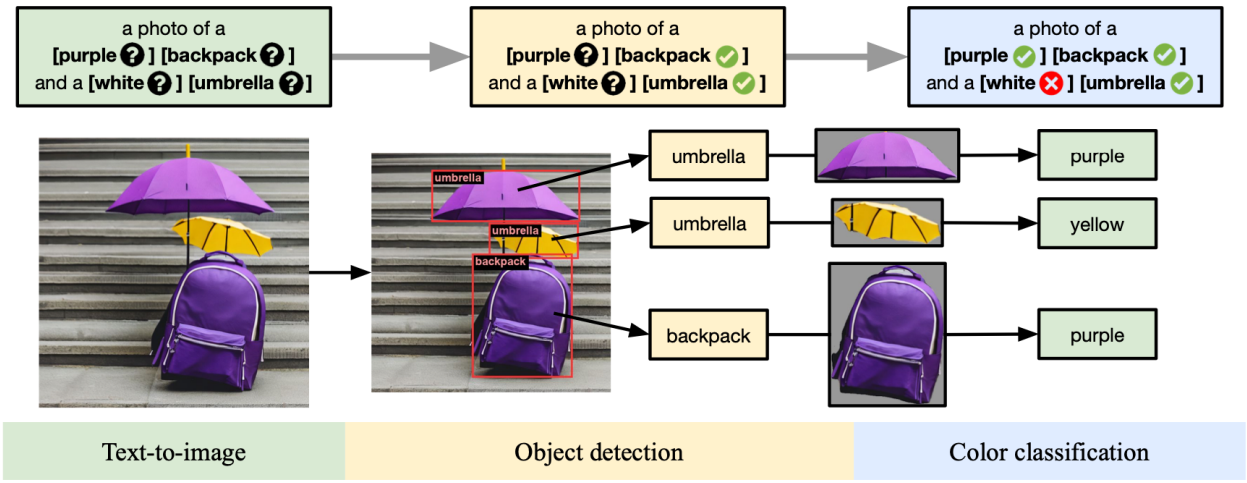


Figure 2.11: Visualization of the evaluation procedure for the GenEval benchmark. Taken from [48].

2.5.5 Dense Prompt Graph Benchmark

The goal of the Dense Prompt Graph (DPG) benchmark [66] is to evaluate the performance of image generation models on long and detailed prompts. It leverages existing datasets, such as COCO [25], PartiPrompts [172] and DSG-1k [28] and enriches their prompts with GPT-4 [1] by adding details, attributes, and spatial relations between objects. A fourth dataset, Object365 [141] is leveraged which is a dataset originally developed for object detection models. The various objects in this dataset are organized into super-cantegories. 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 [28] 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 strucure 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 obsolet. The vision language model mPLUG-large [91] 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 (strucured, unstructured, transformer pruning?)
- qunatization (SVDQuant)
- distillation
- weight sharing
- token pruning
- architecture search (NAS)

Competitor Methods Flux

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

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 Decompostion For Compressing Large Language Models" (LLMs)

- SVD-LLM V2
- (SVD-Delta, SVDQuant) -> machen etwas anderes
- "Efficient Low-Rank Diffuison Model Training for Text-to-Image Generation" (Uses exactly like us SVD for Unet diffusion model; was ist OpenReview? Keine Autoren genannt; kein richitges paper)
- "Accelerating Diffusion Transformer via Increment-Calibrated Caching with Channel-Aware Singular Value Decomposition"

Advances in model performance are achieved not only through new algorithms and architectures, but also through the continuous scaling of parameter counts. This scaling paradigm is facilitated due to the continuous advancement in computation hardware and the availability of internet scale training datasets. While massive architecture yields superior generative capabilities, there is a growing need to achieve comparable performance with smaller models for certain applications and due to environmental concerns, as the hardware required for such large models is extremely expensive and power-intensive leading to a substantial carbon footprint. Furthermore, deplyong these models is often infeasible in ressource constraint setup, e.g. on edge devices. Consequently, a diverse body of literature has emerged around model compression techniques, aiming to minimize the parameter count and the memory footprint during inference and, ideally, also reducing the inference time. This chapter provides a comprehensive overview of various model compression methods, with a particular focus on methodologies adapted for diffusion and flow models.

Quantization

Quantization is a widely adapted technique to reduce model size and speed up the inference process [17, 54, 69]. In quantization, the high-precision weight values e.g., FP32, are mapped to discrete or lower-precision values e.g., INT8, thereby significantly reducing the memory footprint required to store the weights. Quantization methods can be categorized in mainly two different approaches.
First, post-training quantization (PTQ) [58, 69, 112] describes all methods which operates on fully trained models. They only rely on a small calibration dataset to determine the scaling and zero-point parameters for mapping the high-precision values to their discrete low-precision counterparts. These methods are computationally highly efficient as no training of the complete, potentially very large model is necessary.
Second, quantization-aware training (QAT) [10, 43, 57] integrates the quantization process into the training or finetuning process of the model. During the forward pass the weights are approximated with the chosen low-precision format. However, the rounding function is not differentiable, which is why in the backward pass a straight-through estimator (STE) is employed to approximate the gradients. Although the quantized weights are simulated during the forward pass, the high-precision weights are updated during training. Upon completion of the training, the high-precision weights are permanently replaced with the quantized weights. While generally QAT leads to superior performance compared to PTQ the full retraining of the model makes it computationally demanding.
The application of quantization to diffusion and flow-matching models presents unique challenges due to the iterative denoising process, wherein activations are time dependent. Moreover, even minor quantization errors can exert a significant influence because of their accumulation over several inference steps. Consequently, diffusion-specific quantization methods have been developed to address these specifc challenges. Methods such as post-training quantization for diffusion models (PTQD) [59] or Q-Diffusion [95] leveraging a timestep aware calibration dataset sampling features across the entire reverse diffusion process to compute optimal scaling factors have been proposed. Building on this [22] successfully transfered these advanced quantization methods to DiT. A critical challenge inherited from transformer-based architectures is the emergence of extreme activation

outliers which is addressed by [93]. They absorbe these anomalous values into high-precision, low-rank components successfully compression models like Flux.1-dev to 4-bit precision. Most recently, [168] demonstrated that the weights of the Flux.1-dev can be even further compressed to 1.58-bit.
Ultimately, quantization serves as an orthogonal technique to model pruning. Once a model's architecture is structurally pruned, post-training quantization can be applied to further reduce its memory footpring.

VRAM Reduction Only

Alongside methods that reduce the model parameter count, several strategies have been developed to minimize VRAM consumption during inference.
An integral part of modern model architectures is the attention mechanism, which scales quadratically with the sequence length, leading to severe memory bottlenecks. Consequently, hardware-aware algorithms such as FlashAttention [33] or xFormers [90] were introduced to optimize memory access patterns and realize more efficient computation for large attention matrices.
Furthermore, modern models such as Flux.1-dev are standardly deployed using 16-bit half-precision foramts. A primary limitation of FP16 is its restricted dynamic range such that gradient or activation outliers can easily exceed the maximum representable value, leading to overflow that crashes both inference and training. For this reason, modern GPUs, e.g. the NVIDIA A100, H100, and H200, natively support the BFloat16 format (BF16). BF16 utilizes the same 16-bit memory footprint as FP16 but preserves the dynamic range of FP32 by sacrificing decimal precision [72]. Fortunately, neural networks have proven to be highly robust against slight precision loss.
A third option for reducing VRAM consumption is to load only the currently active layers into the VRAM (GPU), leaving the rest of the model in the RAM (CPU). This results in slower inference and training, as there is overhead involved in moving the model components to the VRAM and back to the RAM.

Model Pruning

Model pruning focuses on reducing the actual parameter count of models. It can be disected in mainly two categories: structural and unstrcutural model pruning. In unstructural model pruning individual, less important neurons across the model's architecture are identified, e.g. magnitude-based selection, and are masked out and do not contribute anymore to the inference process. The process to identify neurons to prune can be based on a metric, e.g. magnitude based pruning, or they can be chosen at random. While this allows to remove many neurons by maintaining high performance, it does not necessarily reduces the VRAM consumption during inference as the weight matrices are sparsified but keep their original physical shape and dimensions. To actually achieve a reduction in VRAM consumption the pruned, sparse matrices needs to be converted into a sparse tensor format, e.g. compressed sparse row (CSR) which stores the non-zero entries and its coordinates which lead to a memory overhead contradicting the efforts to reduce VRAM. Therefore, a high degree of sparsity is required to compensate for the overhead and ensure low memory requirements. Moreover, modern GPUs are highly optimized for dense matrix mutliplication which is why running inference with sparse tensor on standard hardware is slower than using dense matrix multiplication. Nevertheless there are emerging custom kernels and specialized hardware to accelerate unstructured sparsity.
This master thesis focuses on the second pruning method, namely structured pruning for transformer-based diffusion and flow models. Unlike unstructured pruning, this method targets coherent model components, such as attention heads, channels, or entire transformer blocks. To recover degraded performance and lost generation capabilities, most methods subsequently apply retraining, specifically knowledge distillation.
Based on the modification strategy the literature can be divided into two main categories, namely approaches that remove model components entirely and those which compresses them.
Most research on compressing Unet-based diffusion models, such as SDXL [120], adopts the approach of structual

component removal [75, 89, 135, 156, 175], yet differs significantly in pruning granularity. For instance, [75] removed predominantly entire middle transformer blocks from the SDXL architecture, identifying them as highly redundant. In contrast, [175] focused on pruning individual transformer layers within the blocks, whereas [89] employed a hybrid strategy that combines block removal with layer-wise pruning for higher compression ratios. Notably, [89] observed that the decoder plays a more critical role than the encoder of the U-Net for the generation process and should therefore be subject to less aggressive compression. To recover the model's fidelity after pruning, most approaches leverage knowledge distillation [61] to retrain their pruned models. They mainly employ the unpruned model as a teacher to guide the recovery process of the pruned student model. Adressing optimization challenges in this phase, [175] demonstrated that using normalized features loss terms is advantageous as the magnitude of individual features from different layers may vary significantly. Furthermore, [135] demonstrated that a progressive pruning schedule for SDXL leads to superior results compared to one-shot pruning.

Recently, the landscape of diffusion models has shifted from U-Net to purely transformer-based architectures where compression methods become increasingly critical due to the steep increase in model size. The primary advantage of these architectures is their dimensional homogeneity. Unlike U-Net architectures which may require down- or upsampling and channel adjustements, entire transformer blocks can be removed without adjusting the feature dimension. Besides straightforward transformer block removal strategies [32, 44, 158], several compression techniques have been proposed using a hybrid depth- and width-pruning paradigma [85, 108]. Regarding depth reduction, complete transformer blocks are removed. To simultaneously address width [108] identified that especially the text-stream and the MLPs of both streams exhibit high redundancy and replaced them with lightweight linear layers. Similarly, [16] proposes replacing blocks with linear layers and instead of performing a full model fine-tuning only LoRAs on the preceding and subsequent blocks are trained to mitigate the performance loss.

In a distinct approach, [181] leverages a mixture-of-expert (MoE) framework to dynamically activate only a small fraction of the model's parameters during each inference step. The core advantage of this approach is that it exploits the temporal variability of feature importance, recognizing that different transformer blocks are citical at different timesteps. Their routing mechanism adaptively selects the most important blocks and experts for the current time step, making the model more expressive than standard static pruning methods. However, while the reduction in activated parameters per inference step significantly accelerates inference, it does not necessarily alleviate the VRAM consumption bottleneck. Crucially, since the complete model (including both activated and inactivated parameters) typically remains in memory, the total memory footprint remains closte to the original model size, which distinguishes this method significantly from memory-focused pruning methods.

Finally, a distinct category of component compression is low-rank factorizaiton, commonly implemented via singular value decomposition (SVD). This method approximates a dense weight matrix by decomposing it into products of two low-rank matrices, effectively reducing the parameter count while preserving the original architectural topology.

In the domain of LLM SVD has been extensively investigated as static post-training compression strategy [89**? ?**]. However, the straightforward application of SVD has proven to lead to significant performance drops even at low compression ratios, since LLMs exhibit massive activation outliers and SVD removes components critical for these activations as the singular value selection is solely based on their magnitude [**? ?**].

In this work, SVD is revisited within the context of diffusion and flow models. Unlike the zero-shot compression often applied in LLMs, SVD is identified as a highly-effective initialization strategy for compressed model weight matrices before retraining. This work demonstrates that a progressive linear compression scheme utilizing SVD can effectively compress Flux.1-dev, outperforming SOTA pruning approaches on several benchmarks. In addition, it is demonstrated that SVD can be effectively deployed as a training-free compression approach for moderate compression ratios while maintaining acceptable performance and that [**?**] can be transfered from LLMs to flow models like Flux.1-dev.

Making Faster

Caching

Chapter 4

Methods

4.1 Image Generation Models

In the following section, the models, namely Pixart-Σ [19] and Flux.1-dev [86], which were used for the experiments in this work are described in detail.

4.1.1 Pixart-Σ

PixArt-Σ [19], the latest addition to the PixArt family [20, 21], 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 that are also of relevance for PixArt-Σ, followed by the specific enhancements implemented in PixArt-Σ.

Architecture

The shared architecture of PixArt-Σ and PixArt-α is based on the class-conditional diffusion transformer DiT-XL/2 [118] 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. The global scale and shift parameters are used as input for the customized adaptive layer normalization called adaLN-single layer, which modulates the hidden states based on the time embedding.
The second input is the **text prompt** that describes the content of the image. It is tokenized into 120 tokens in Pixart-α, repsectively 300 tokens in Pixart-Σ, and encoded by the T5-XXL text encoder [124]. 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 [120] 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.

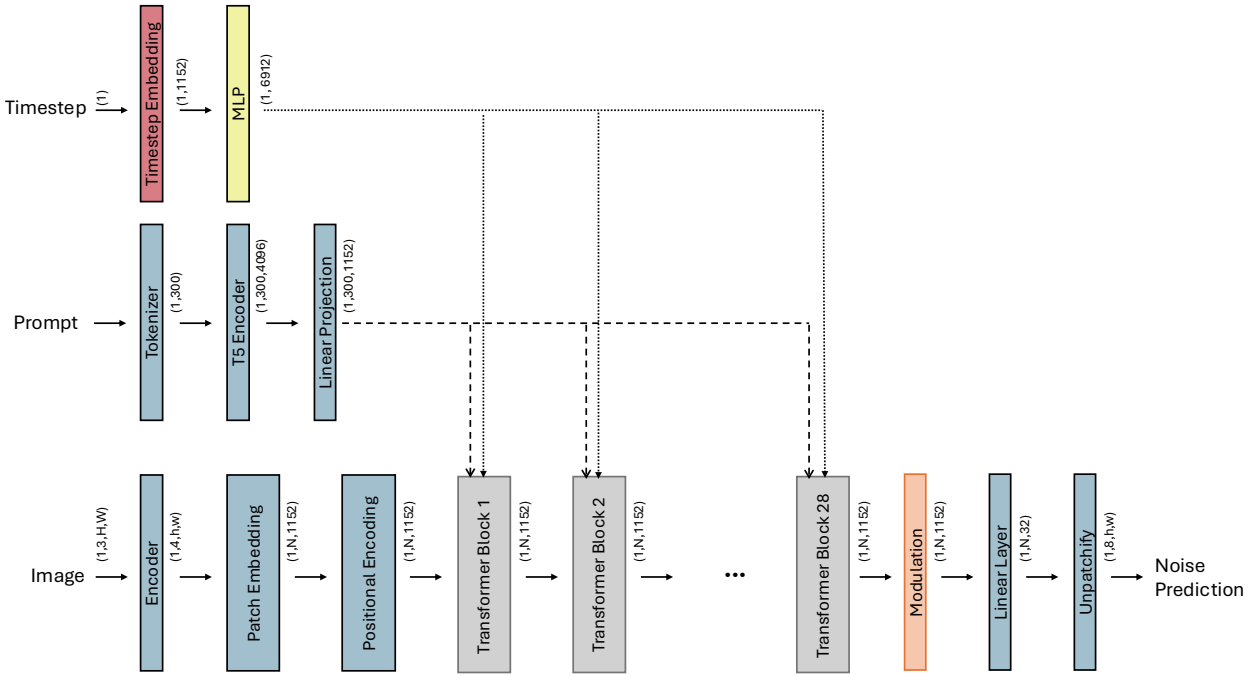


Figure 4.1: Overview of the Pixart-Σ architecture. H and W are the height and width of the image. $h = \frac{H}{8}$ and $w = \frac{W}{8}$ represent the dimensions in latent space and $N = \frac{H}{8 \cdot p} \frac{W}{8 \cdot p}$ with p being the patch size.

The main components of the diffusion model are the transformer blocks. They consist of four main parts, namely the multi-head self-attention layer, the multi-head 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 [119] 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 [118] 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.

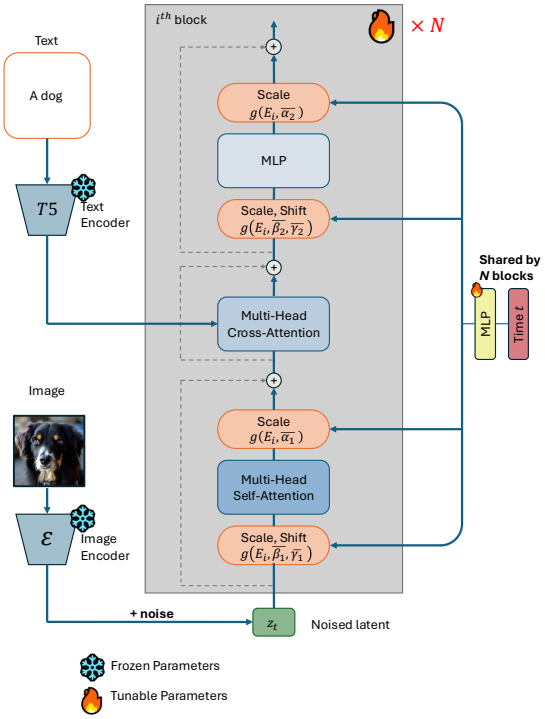


Figure 4.2: Visualization of the transformer block within PixArt-Σ. Taken from [21].

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 [35] 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 reflect the text prompt. For this purpose, the SAM [78] dataset was utilized with corresponding high-density prompts generated by the vision-language model LLaVA [102].

Finally, the model undergoes finetuning on high-resolution and high-aesthetic qualtity images to refine visual details which was done on a data set that is not publicly accessible.

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 [120], 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 [23], 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

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

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

4.1.2 Flux.1-dev

Flux.1-dev [86] is a publicly available, guidance-distilled rectified flow model based on the commercial model Flux.1 [86]. 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.1-dev is based on a multi-modal diffusion transformer (MMDiT) [42] 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 [122] and T5-XXL [124]. 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 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 are 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.1-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.1-dev uses rotary positional embedding (RoPE) [153]. 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.1-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.

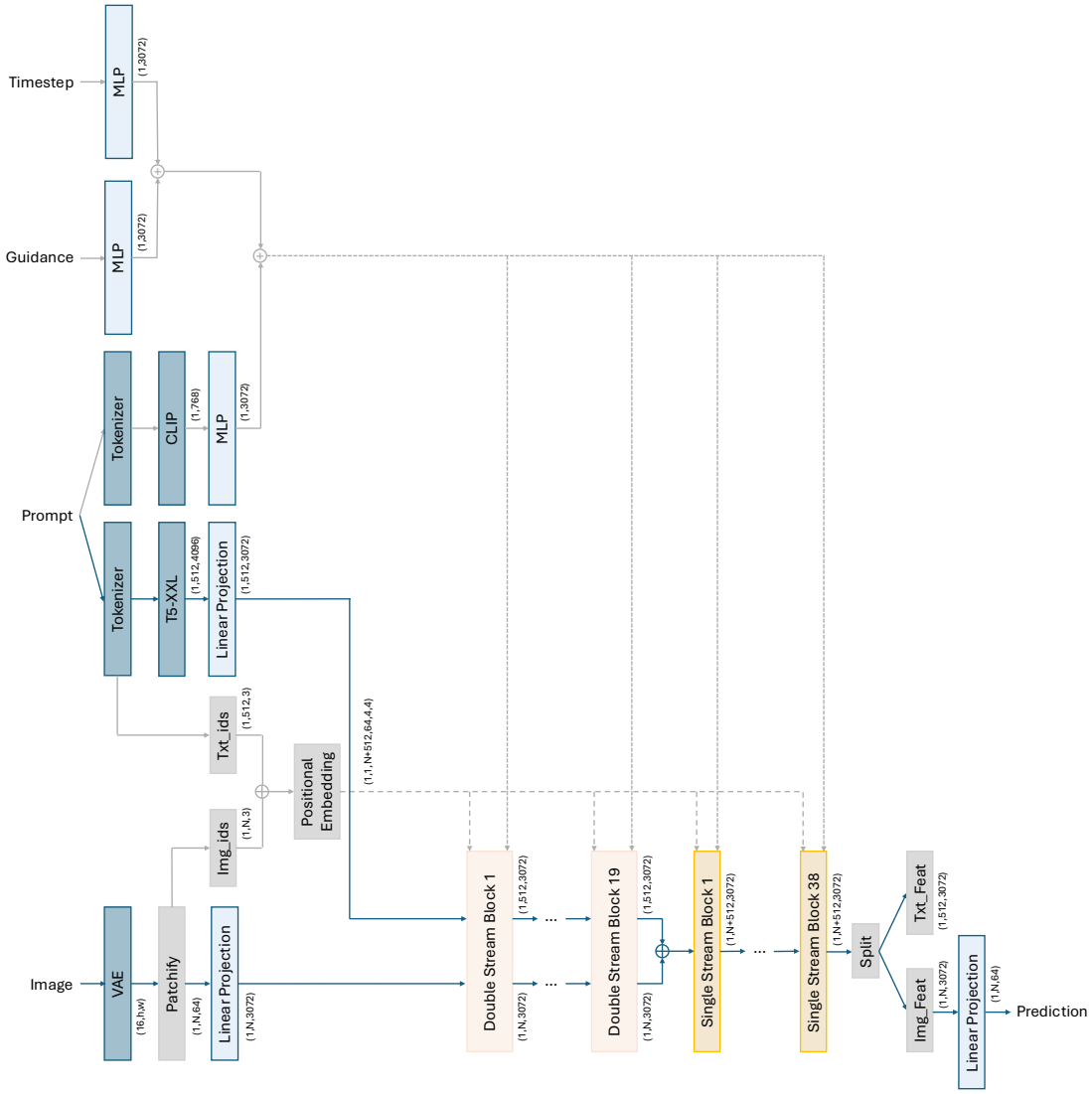


Figure 4.3: Overview of the Flux.1-dev architecture.

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.1-dev architecture. In contrast to other diffusion-based models [19, 120, 129] 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 paramerters for adaLN as well as two gating parameters. The scaling and shifting of the features to incorporate the time, guidance and pooled text conditioning is applied twice, once before the attention mechanism and once before the MLP. Subsequently, the gating parameters scale the updated features before they are added to the residual stream. This technique is often applied in very deep neural networks because the linear projection layers producing the gating values are initialized with zeros. Consequently, the gating paramters start at zero, such

that the block initially acts as an identity function, which stabilizes the training.
The **attention** mechanism enables the exchange of information between the text and the image features. First, the text and image features are processed by seperate linear layers which project them into query, key and value representations and are then normalized via RMSNorm [174]. 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.1-dev architecture. Their primary components are the self-attention mechanism and the modulation layers. First, the concatenated image and text features X are modulated using adaLN where the shift and scale parameters are obtained by processing the guidance vector through a linear layer. Similar to the double-stream architecure, a gating mechanism is applied after the attention mechnism is executed. To optimize throughput, the model uses a fused linear projection where the hidden states for the attention mechanism and the parallel feed-forward path are computed simultaneously. The outputs of the attention mechanism and the feed-forward path are concatenated together and processed by a subsequent linear projection before the gating mechanism is applied. Finally, the updated features are combined with the residual path to form the final output of the block.

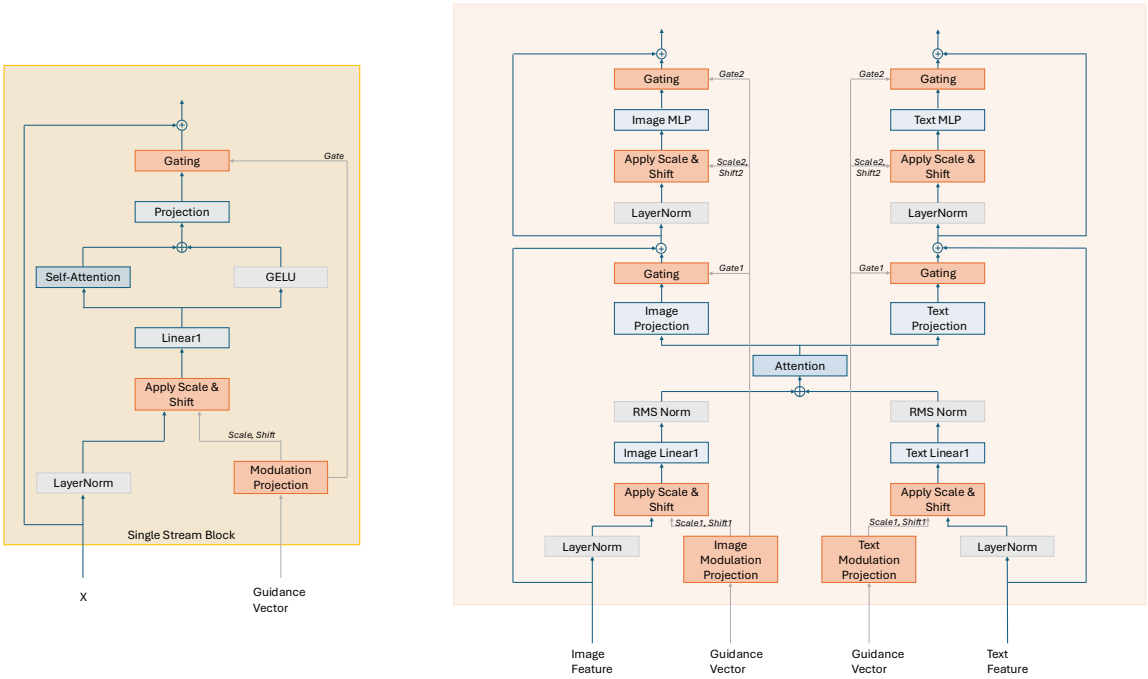


Figure 4.4: Visualization of single-stream block (left) and double-stream block (right) of Flux.1-dev. Graphics are adapted from [16].

Table 4.2: Number of parameters for the individual components of double-stream block from Flux.1-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.1-dev)	11,901.4

4.2 Block Importance Analysis

Identifying the least important parts of the model which can be compressed or removed with the least impact of the final model performance is crucial. In the following, first different block selection criterias are presented followed by block selection algorithms tackling the challenge of identifying a subset of several blocks being compressed simultanously.

4.2.1 Block Selction Criterias

Magnitude-Pruning

In magnitude pruning [46, 55, 88], 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 compressed 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

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

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

Representational Similarity Analysis

Another approach to identify redundant model components is to compare their input representations with their corresponding outputs representations [110]. 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 [83, 121] 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 $H = I - \frac{1}{n}11^T$ is applied. The final CKA score is given by

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

This calculation utilizes the Hilbert-Schmidt Independence Criterion (HSIC) [53] $\text{HSIC}(K, L) = \frac{1}{(n-1)^2}\text{tr}(KHLH)$ [34].

Metric Based Selection

A third approach to identify the least important model components is to compress all components individually, e.g. each transformer block, and compute for each configuration image quality metrics on a small reference dataset. Specifically, let $\{B_1, \ldots, B_L\}$ denote the individual blocks of a transformer based model architecture. Each block is modified seperately leading to L different model configuration $\{\{\hat{B}_1, \ldots, B_L\}, \ldots, \{B_1, \ldots, \hat{B}_i, \ldots, B_L\}, \ldots, \{B_1, \ldots, \hat{B}_L\}\}$ 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 the CLIP-score 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 maximizaiton), 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

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

In the 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

The need for the use of block selection algorithms results from a specific challenge in model pruning. The challenge 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 remaing 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 for compression are described and investigated in sec. 5.1.2.

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 $B = \{B_1, \ldots, B_L\}$. 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 remain. 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 compresssed has been identified.

Black-Box Optimization with Optuna

Optuna [3] is an open-source hyperparameter optimization framework. It primarily leverages a tree-strucured parzen estimator [11], a bayesian optimization algorithm, to search the space of all possible hyperparmeter 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 descritption of the Optuna algorithm the reader is refered to the original paper [3] because this would go beyond the scope of this work.

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

where $\Delta\Phi$ represents the updates for the model parameters Φ. The learned mask provides the infromation 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) [65] 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

In this section, the datasets leveraged for finetuning PixArt-Σ and Flux.1-dev are briefly described. Both synthetic, self-generated data sets and pre-existing data sets were used.

LAION Aesthetic

LAION-5B [139] 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 [140] 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 aesthtic subset are utilized as the training dataset. Due to poor quality of the web-scrabed captions which come along with the LAION-5B dataset, new synthetic prompts were generated using the visual language model JoyCaption [47]. 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: Generation Parameters for LAION-PixArt-Σ dataset.

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.1-dev Dataset

Following the same methodology as for the LAION-PixArt-Σ 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.1-dev model. The configuration used during the image generation process is detailed in tab. **??**. 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.

[Parameter LAION-Flux.1-dev Dataset]Generation Parameters for LAION-Flux.1-dev dataset.

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.1-dev generated dataset based on JoyCaptions prompts from LAION aesthetic dataset.

4.3.2 Evaluation Datasets

This section presents data sets on which the compressed models were evaluated.

Mapillary

The Mapillary [113] 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: Example images of Mapillary dataset.

Cityscapes

The Cityscapes [30] 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, e.g., for the automotive sector, because of autonomous driving. However, due to its high degree of standardization it is less diverse than the Mapillary dataset.

Figure 4.9: Example images of Cityscapes dataset.

MJHQ-30k

The MJHQ-30k [92] 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 [92] and selected based on their text-alignmen (CLIP) and their aesthetic [79] scores.

Figure 4.10: Example images of MHJQ-30k dataset.

4.4 Model Compression Strategies

There are two primary strategies in 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.1-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 and variants of them are described in the following.

4.4.1 Block Removal

In structural model pruning, entire functional components of a model are often removed completely [75, 89, 175] 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}

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

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) [179] factorizes a matrix $W \in \mathbb{R}^{n \times m}$ into three matrices U, Σ, V such that

$$W = U\Sigma V^T \tag{4.7}$$

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 $\sigma_1 \geq \sigma_2 \geq \sigma_3 \ldots \sigma_p$ 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 [164, 173], SVD can be leveraged to approximate the weight matrics 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

$$\mathbf{x}_{\text{next}} = AB\mathbf{x} + \mathbf{b} \tag{4.8}$$

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

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

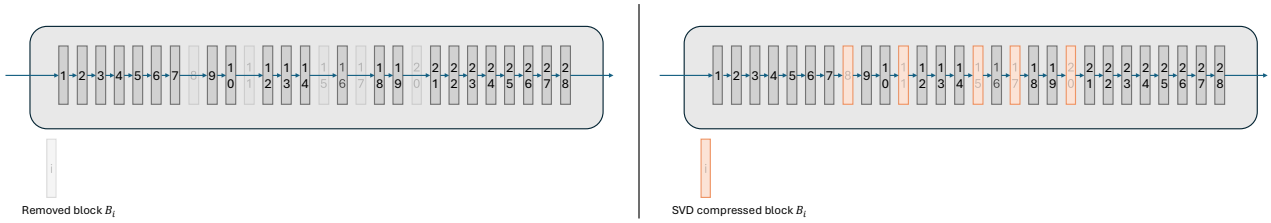


Figure 4.11: Example of removal (left) and compression (right) strategy based on a simplified representation of PixArt-Σ architecture. The depictions show the case that the blocks 8,11,15,17 and 20 are removed or compressed.

4.4.3 Iteratively Repeated SVD Compression

In the 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 full weight matrix $W_t = A_t B_t$ is reconstructed and afterward the SVD algorithm is performed on it $W_t = U^{(2)}\Sigma^{(2)}V^{(2)^T}$. Similar to the initial step, after choosing a further reduced rank, the new smaller matrices $A^{(2)}$ and $B^{(2)}$ are constructed via $A^{(2)} = U^{(2)}_{\text{red}}\sqrt{\Sigma^{(2)}_{\text{red}}}$ and $B^{(2)} = \sqrt{\Sigma^{(2)}_{\text{red}}}V^{(2)^T}_{\text{red}}$. By following this procedure, reconstrucing $W^{(j)}_t = A^{(j)}_t B^{(j)}_t$, chosing a new smaller rank and obtaining the new $A^{(j+1)} = U^{(j+1)}_{\text{red}}\sqrt{\Sigma^{(j+1)}_{\text{red}}}$ and $B^{(j+1)} = \sqrt{\Sigma^{(j+1)}_{\text{red}}}V^{(j+1)^T}_{\text{red}}$, a transformer block can be gradually compressed across multiple iterations.

4.4.4 GRASP

The method gradient-based retention of adaptive singular parameters GRASP was originally proposed for compressing LLMs [103]. It is based on the idea of exploiting the low-rank structure of redundant layers in LLMs. Moreover, the authors hypothesized that the standard SVD approach where the singular values are selected solely based on their magnitude is not optimal because even small singular values may be important for the final task of the model and should be preserved. Therefore, they proposed a gradient-based selection instead of a magnitude-based selection for the singular values. In this work, the GRASP method is transfered to image generation models based on the example of Flux.1-dev. First, the candidate blocks for pruning are identified with a method from sec. 4.2. Next, the identified blocks are compressed via SVD. Afterwards, all parameters are frozen and only the singular values are treated as the parameters for which the gradients are calculated. For computing the gradients a calibration dataset is utilized consisting of 200 images. For each image, the gradients of each singular value based on all timesteps 1 to 1000 were computed and summed together. Finally, the gradients for all 200 images were averaged to obtain a final estimate of the importance of each singular value. Instead of choosing the k largest singular values, the k singular values with the largest gradients are chosen for the low-rank matrix approximating the original weight matrix.

4.5 Knowledge Distillation Loss Functions

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

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

In the experiment section 5.1.5 the importance of the individual components of this loss function is investigated.

4.5.1 Normalization

An important point to note is that the features of different blocks may have varying magnitudes wherefore [175] found that applying a normalization to the features is beneficial. Therefore, after computing the difference between the teacher and the studen features a normalization is applied such as the individual blocks contribute equally to $\mathcal{L}_{\text{FeatKD}}$. The normalization which is utilized for this work is taken from [181]. Let T_l and S_l denote the features of the teacher and the student from transformer block $l \in \{1, \ldots, L\}$. 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 featurs

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

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

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

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

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

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

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

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 studen 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 $l \in \{1, \ldots, L\}$ 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 features 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.

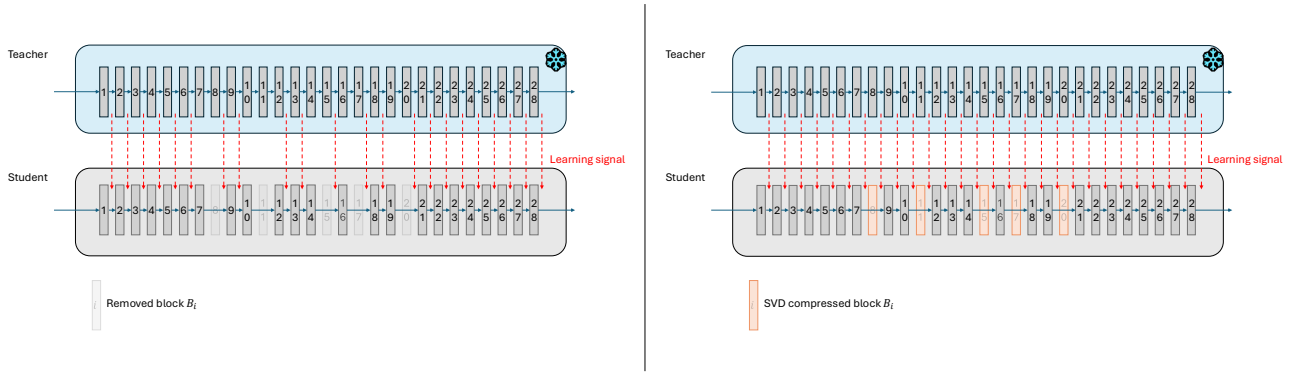


Figure 4.12: Visualization of the feature mapping between student and teacher model for loss computation.

4.6 Pruning Schedule

Two different approaches are investigated for model pruning and presented in the following section.

4.6.1 One-Shot Model Compression

The standard approach of compressing transformer based diffusion and flow models [32, 89, 158, 175] involves three main steps. First, the transformer blocks $\{B_1, \ldots, B_L\}$ are sorted according to their importance, e.g. using CLIP-score and CMMD as importance proxy. Let $\{B_1^*, \ldots, B_L^*\}$ 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 $\{B_1^*, \ldots, B_k^*\}$ 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 introduces 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. [135], 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 recovarable. In the subsequent iteration, the block importance is re-assessed for all remaining unmodified blocks $\{B_1, \ldots, B_L\} \setminus \{B_1^*, \ldots, B_k^*\}$, as the relative block ranking may have changed as a result of the retraining. Another design choice of the progressive model compression 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 CMMD 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

In this work a new linear progressive compression being an advancement development of progressive model compression is investigated. 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 like progressive model compression. First, the block importance is determined by leveraging the CMMD 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 CMMD 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

$$s_i = \frac{\text{CMMD}_i}{P_i \cdot c_{\text{fixed}}} \tag{4.15}$$

where CMMD_i is the CMMD 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 a linear compression schedule is leveraged 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

$$c_i = S - ((i-1) \cdot x) \tag{4.16}$$

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

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

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

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

ending in the final equation for the slope

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

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 leads to the following condition

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

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

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

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 investigates model pruning techniques, specifically focusing on the removal and compression of transformer blocks within the PixArt-Σ and Flux.1-dev architectures. PixArt-Σ is employed as an experimental baseline to evaluate various design choices in the pruning framework due to its relatively low parameter count enabling rapid iteration and extensive ablation studies. Subsequently, the most effective strategies are applied to Flux.1-dev to assess their efficacy in high-parameter regimes.

5.1 PixArt-Σ Pruning

5.1.1 Experimental Default Setup

In the following section, individual components of the compression pipeline such as loss function, block selection algorithm, compression strategy, ..., are systematically investigated. To isolate the impact of each component, a default experimental setup is defined below. Unless stated otherwise, all subsequent experiments utilize the baseline configuration. It is important to note that this baseline configuration serves as consistent starting point for the experiments and does not necessarily represent the final, best setting which is derived later in this work. The following elements are constituents of the baseline configurations:

- **Loss**: The learning objectiv is the sum of the unormalized feature and output distillation loss $\mathcal{L}_{\text{OutKD}} + \mathcal{L}_{\text{FeatKD}}$.
- **Pruning Schedule**: The progressive compression strategy was applied. As soon as the CMMD value became larger than 0.1 in the block selection algorithm, the model was retrained to recover its generation capbilities.
- **Structural Compression Strategy**: Complete block removal was used to reduce the model size.
- **Block Selection Algorithm**: The greedy algorithm based on CMMD and CLIP-score was utilized, computed on 100 images.
- **Training Datasets**: Training was performed sequentially starting on the LAION dataset followed by further training on the LAION-PixArt dataset. For each iteration, the models were trained for two epochs on each datasets.
- **Finetuning Protocol**: The recovery of the pruned model followed a structured three stage protocol. First, for local stabilization, optimization was restricted to the block B_{l-1} immediately preceding the removed block B_l for one epoch (38k steps) on the LAION dataset. In case where several blocks were removed simultanously, all corresponding preceding blocks were updated while the rest of the network

was frozen. Second, the complete model was optimized for an additional epoch on the LAION dataset. Third, for domain-specific refinement, the complete model was finetuned for two epochs (12.5k steps) on the LAION-PixArt dataset.

- **Training Hyperparameters**: All other training hyperparameters are summarized in tab. 5.1.

The implementation of all experiments build up on the official git repository for PixArt-Σ [18]. For evaluation of the model, the HPSv2 and the GenEval benchmarks were utilized in this section.

Table 5.1: Default hyperparameter settings for PixArt-Σ finetuning after compression.

Hyperparameter	**Value**
Optimization	
Optimizer Type	CAME
Learning Rate	2×10^{-5} (Block Removal)/ 2×10^{-6} (SVD Compression)
Weight Decay	0.03
Betas	$(0.9, 0.999, 0.9999)$
Epsilon	$(10^{-30}, 10^{-16})$
Training Schedule	
Warmup Steps	1000
Batch Size	16
Gradient Clipping	0.001
Model & Data	
Precision	bf16
Resolution	512×512
Class Dropout Prob.	0.1
Max Prompt Length	300

5.1.2 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 [110] that redundancies are primarily find in the middle part of transformer based models. To identify which blocks are best to remove meaning which blocks have a minimal contribution on the total image generation process, several different metrics can be employed. In the following section, different approaches are investigated to determine the importance of the individual transformer blocks based on completely removed blocks. First, the traditional magnitude-based pruning and represential dissimilarity by computing the central kernal alignment score (CKA) (see sec. 4.2.1) between the input and output of each transformer block are evaluated. Moreover, they are verified against to standard performance metrics such as the CLIP-score and CMMD (see sec. 4.2.1).

Normally, several transformer blocks are removed or compressed simultaneously. Due to block interdependencies, two different options to identify the best grouping for blocks to remove are investigated. 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.

Qualitative Block Analysis

Fig. 5.1 displays an example for the qualitative influence of the removal of individual transformer blocks from the PixArt-Σ architecture. For each image a different block was completely removed. The removal of the first four blocks and the last two blocks exhibits a strong degeneration in image quality which confirms the finding that the first and last layers are more critical for transformer-based models from [110]. In diffusion or flow models the first layers aggregate global context while the middle layers refine details. The last layers are important for high frequency details as seen in fig. 5.1 where for removed block 27 or 28 the concept of a man is present in the image but very blurry.

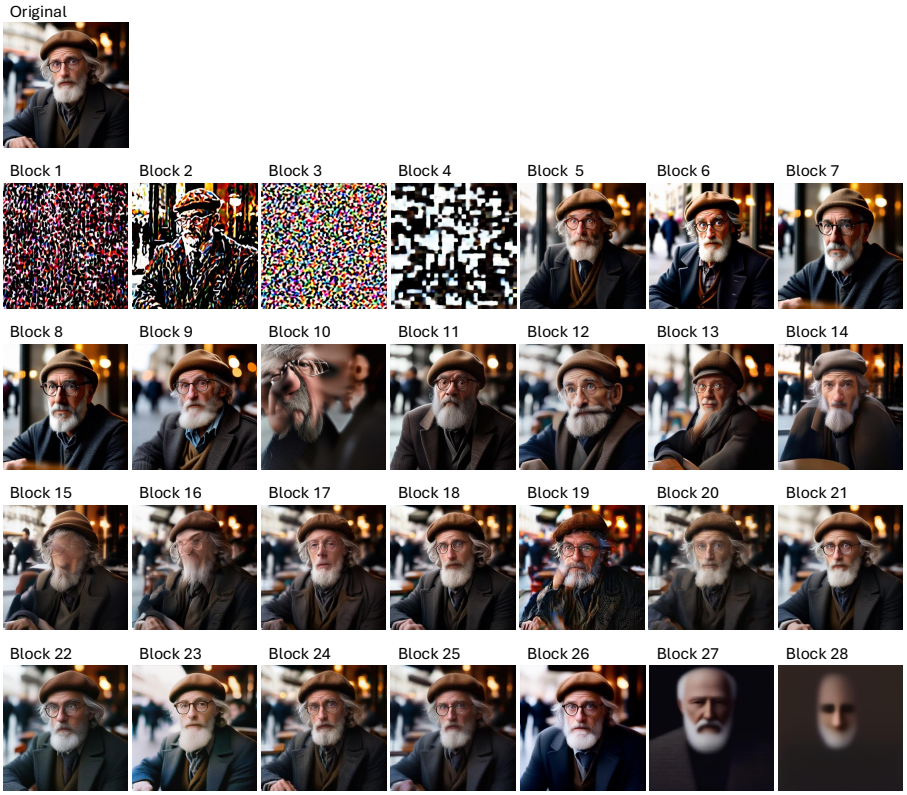


Figure 5.1: The figure shows the influence of selectively removing individual transformer blocks (1–28) on the final generation result.

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, 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 CMMD). This observation contradicts with the assumption of low impact by low weights magnitudes. It seems that despite the small weight magnitude the first layers act as important initial feature translator having a massive influence on model performance. 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 makes 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 strucural pruning in this context.

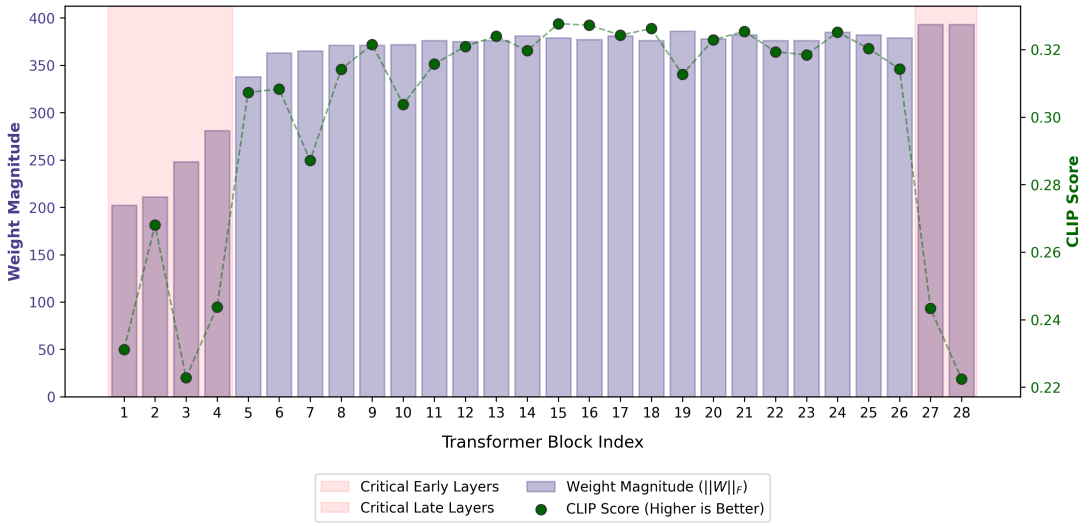


Figure 5.2: The histogram shows the magnitudes of the weights of the individual transformer blocks (1-28). The blocks which have shown qualitatively (see fig. 5.1) to be very important for the image generation capabilities of PixArt-Σ (see fig. 5.1) are marked with red. In addition, the CLIP-scores for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

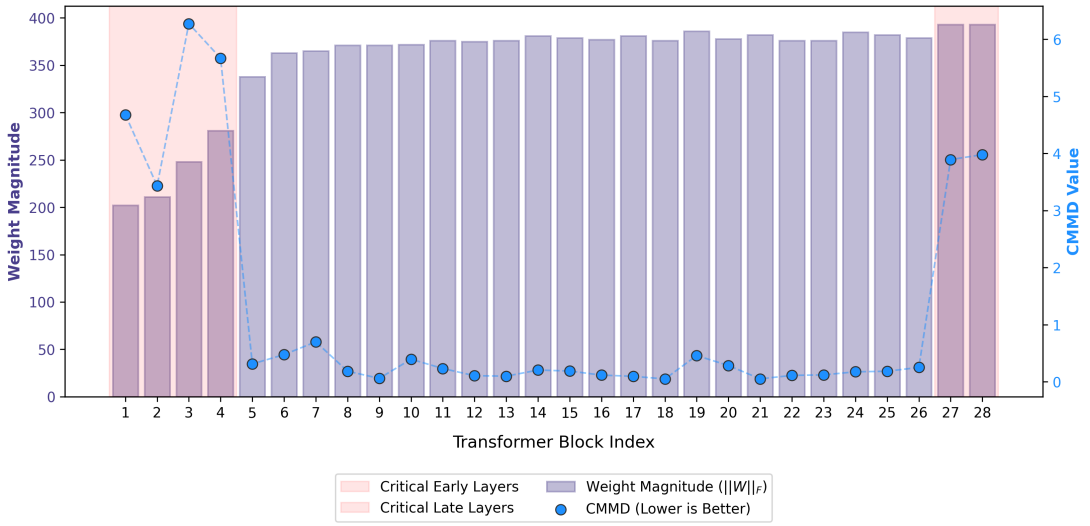


Figure 5.3: The histogram shows the magnitudes of the weights of the individual transformer blocks (1-28). The blocks which have shown qualitatively to be very important for the image generation capabilities of PixArt-Σ (see fig. 5.1) are marked with red. In addition, the CMMD for each block is displayed. The CMMD 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 greate value signifies a more substantial feature transformaton. 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.

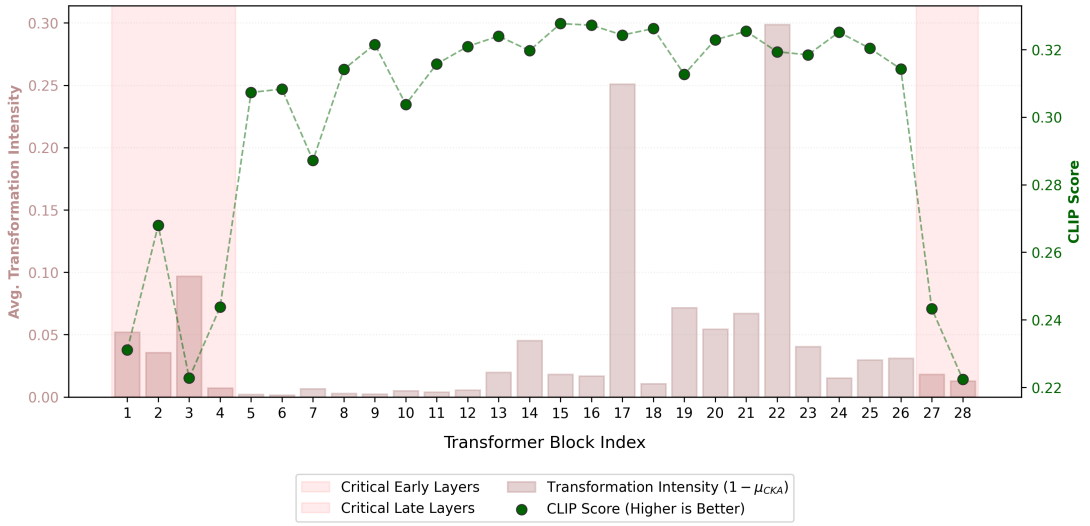
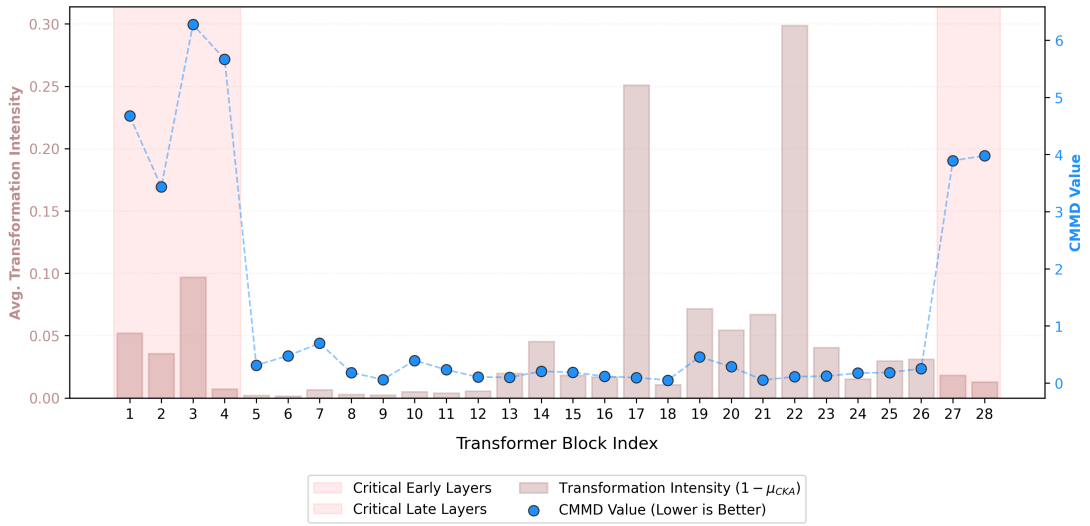


Figure 5.4: The histogram shows the transformation intensity based on CKA comparing the input and output features of the individual transformer blocks (1-28). The blocks which have shown qualitatively (see fig. 5.1) to be very important for the image generation capabilities of PixArt-Σ are marked with 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.



adjust filename and width

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 CMMD for each block is displayed. The CMMD 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 [16]. Specifically, the CLIP-score and CMMD 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. 4.2.2) and the high computational overhead associated with repeated evaluations, using such a large reference set was prohibitively expensive. Therefore, 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 algorithm and also for most experiments regarding block compression instead of block removal.

Block Analysis for Compression Instead of Removal

In the experiments in which the blocks were compressed via SVD, the scores for block selection were computed based on compressed blocks. In fig. 5.6 the results of compressing every transformer block individually by 60% are displayed. One important point to note is that the compressing the first blocks (1-4) which have a large impact when the blocks are completely removed (see fig. 5.1) resulting only in minimal image changes compared to the original image generated with the uncompressed model. Also the compression of block 28 results in less severe image quality degeneration than completely removing it. This demonstrates that these blocks are critical for the overall performans but contain high degree of redundancy. Notably, the compression of block 27 also exhibits a strong smoothing of details like it was already observed by removing it. This indicates that block 27 is crucial and does not contain a high degree of redundancies. Compressing the middle blocks does not lead to a big drop of image quality but they differ even more from the original image compared to the first blocks. A possible explanation is that for setting the global structure which is done by the first blocks a low rank structure is sufficient. The middle blocks which are responsible for the details, e.g. direction in which the eagle looks, are more sensitive for compression.

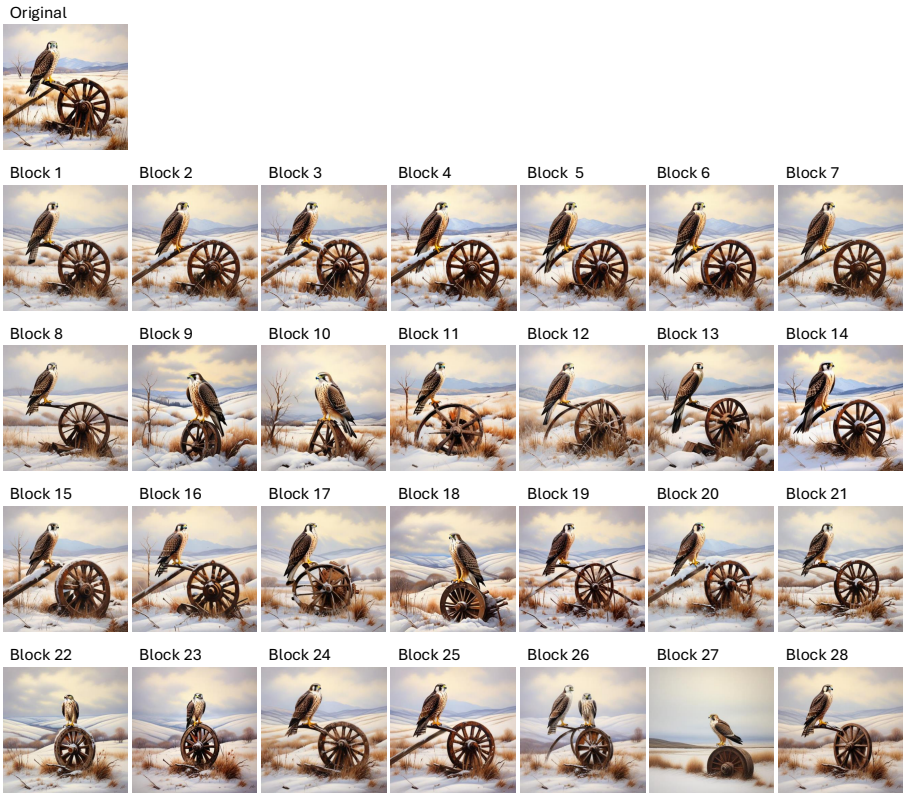


Figure 5.6: The figure shows the influence of selectively compressing individual transformer blocks (1–28) by 60% on the final generation result.

5.1.3 Block Selection Algorithms

A further key challenge after establishing the pruning criteria as the combination of CLIP-score and CMMD in model pruning 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.

Greedy

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.

Optuna

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. Here, it is 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 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 the Optuna optimization framework such that the first trails 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 pruning framework with fixed compression ratios $\pi_i = 0.6 \; \forall i \in \{1, \ldots, N_c\}$, where the number of compressed transformer blocks was chosen for each iteration individually. The procedure to determine the best transformer blocks to compress was as follows. First, a greedy algorithm was leveraged to find a prior of potentially good candidates for compression. Afterwards, the Optuna hyperparameter framework was applied to refine the block selection. In total 300 different configurations were tried out by Optuna where the decision criterion was solely the CMMD score. The pruned model was then retrained. This procedure was executed repeatedly for four iterations. Fig. 5.7 shows the prior blocks from the greedy algorithm as well as the final block selection of the Optuna algorithm. On the right side the corresponding CMMD values which were computed on a small reference dataset of 100 images are displayed for each configuration. Note this is the CMMD value before retraining to compare the prior with the final selection. The grey blocks, represent the consensus where the Optuna algorithm did not change the blocks from the prior whereas the yellow and green blocks mark the exclusive blocks selected by the greedy algorithm and the Optuna algorithm respectively. For iterations one to three, one observes that the majority of blocks from the greedy prior were maintained by the Optuna algorithm, only in iteration four do two out of three blocks differ. The large overlap is a first indication that the greedy algorithm already performs well. However, because the greedy result was used as prior the optuna framework search is focused around the greedy result which might also explain the high similarity. Overall, one can conclude that the CMMD values obtained by the Optuna algorithm are consistently smaller (better) than the ones obtained by the greedy algorithm. This indicates that at least the pre-training performance is enhanced by the refinement step.

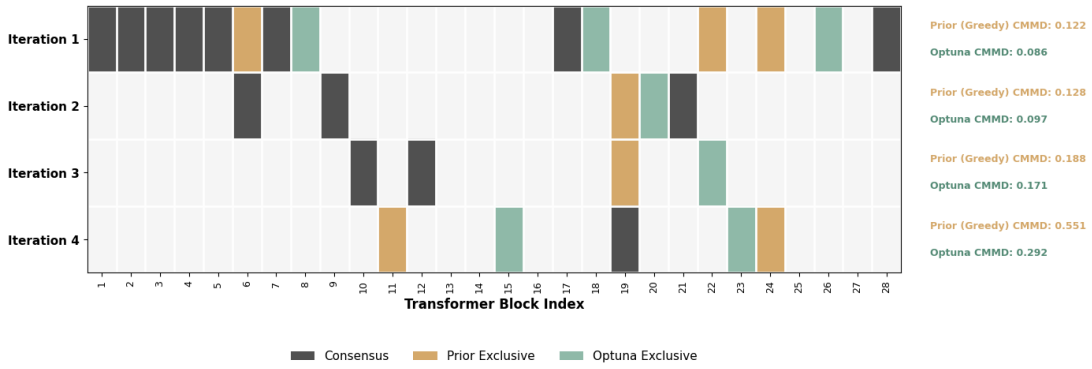
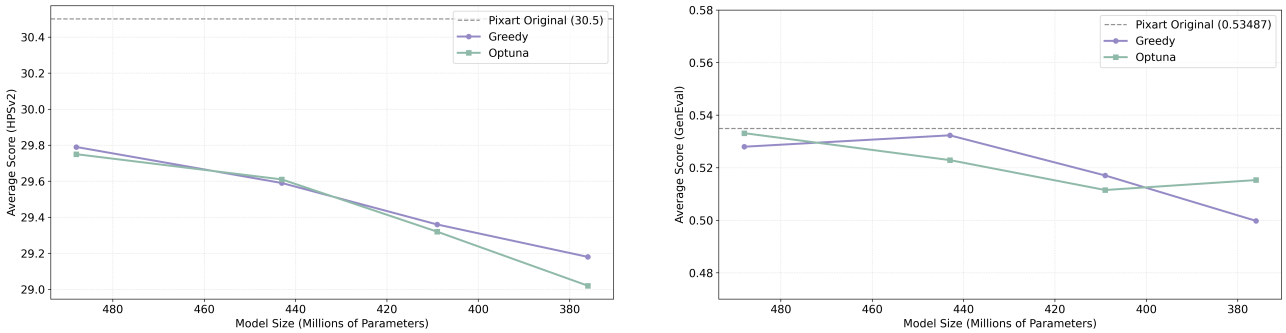


Figure 5.7: Comparison of removed blocks per iteration by the Greedy algorithm versus the Optuna framework. In each iteration, the Greedy algorithm's selection served as the initialization prior for the Optuna hyperparameter search. The heatmap visualizes the divergence in block selection:.While gray blocks represent consensus, the colored blocks highlight where Optuna (green) found a superior structural reduction compared to the Greedy baseline (yellow). The model was retrained between iterations based on the optimized configuration. The annotations demonstrate that Optuna consistently achieves lower (better) CMMD scores.

Above, it is observed that in each iteration, the Optuna framework was able to identify a block combination for compression which resulted in a smaller CMMD value compared to the prior greedy algorithm. In the next step, the model is compressed twice, once based on the greedy algorithm and once based on the Optuna algorithm as described above, each time targeting the same number of parameters (same compression rates $\pi_i = 0.6$ and same number of blocks in each iteration). Fig. 5.9 shows the corresponding distribution of blocks being compressed in each iteration. Please note that here "Greedy" specifies the compression scheme where the greedy algorithm is strictly utilized as the block selection algorithm, in contrast to the previous section,

where "Greedy" just represented the prior for the Optuna algorithm. Looking at the block distribution, one observes that the early layers are compressed heavily in both schemes first, indicating a consensus on the high redundancy of these initial blocks. In the later layers, no clear structure is visible and the selections diverge. This lack of a consistent pattern suggests that the deeper blocks are largely interchangeable regarding their impact on the model's capacity. The evaluation of the retrained compressed models (see fig. 5.8) does not provide clear evidence that the Optuna block selection algorithm led to better performing models post-retraining. One observes that for the smallest model, the HPSv2 score is even better for the greedy algorithm, and a similar trend is observed for the second and third iteration regarding the GenEval score. This demonstrates that the retraining process acts as an equalizer, effectively eliminating the small primary advantage of the Optuna algorithm.



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.8: Compare the performance of pruned models based on the Optuna and Greedy block selection algorithm.

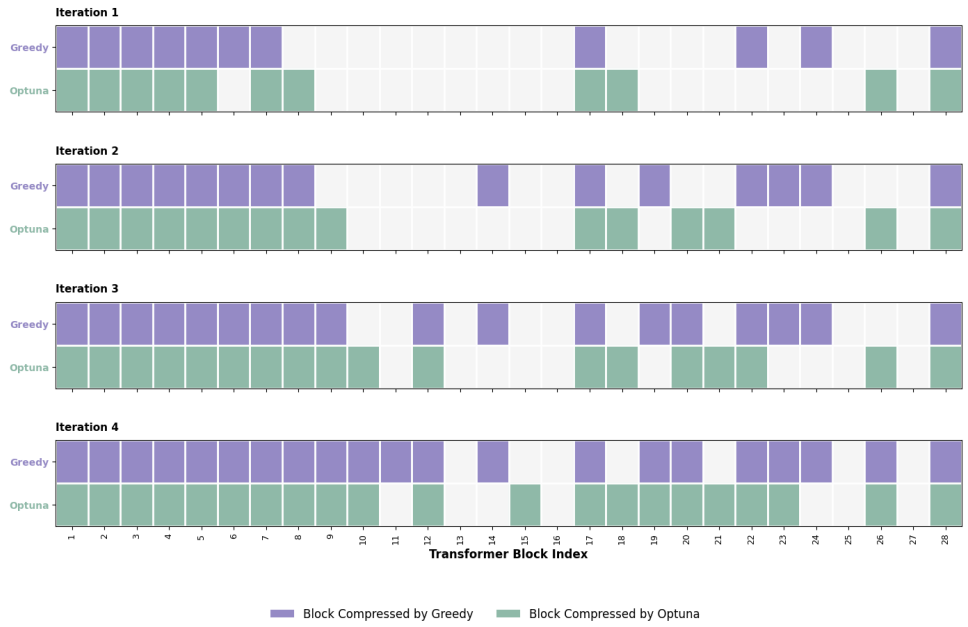
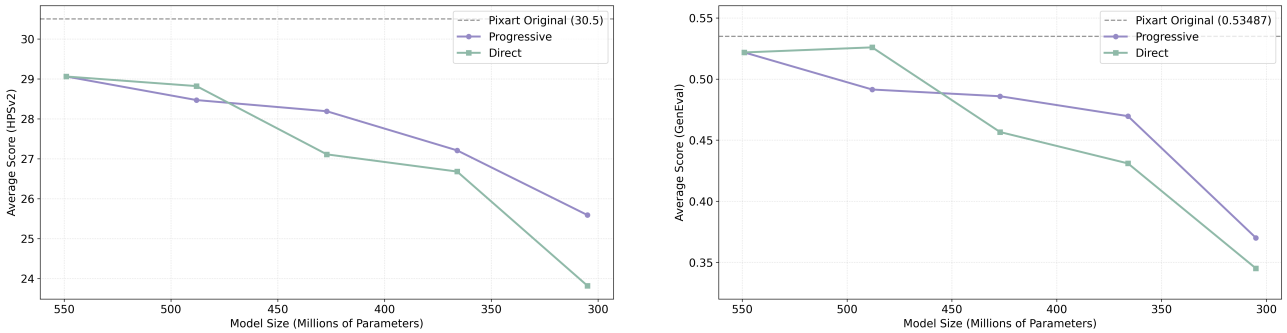


Figure 5.9: Here, all compressed blocks based on a greedy and optuna pruning scheme are displayed per iteration.

5.1.4 One-Shot vs Progressive Model Pruning

One-shot model pruning compresses the model to target size in a single step, while progressive pruning employs an iterative scheme of incrementally removing parameters followed by recovery training phases. Both frameworks are tested using the complete block removal approach. The number of blocks removed for each model and the

number of training steps are summarized in tab. 5.2. The exact blocks which are removed for each model can be found in the appendix **??** (see fig. **??**). Fig. 5.10 displays the performance of the one-shot versus progressive pruning on the HPSv2 and GenEval benchmarks across various compression ratios. To ensure a fair comparison, the models for both methods are trained for the same amount of training steps and on exactly the same data. Both benchmarks demonstrate that one-shot compression is a viable strategy in the low compression regime (10% to 20% parameter reduction). However, as the compression ratio increases the progressive approache becomes advantageous because the incremental removal of parameters allows the model to recover its generative capabilities more effectively compared to the abrupt removal of large amount of parameters at once.



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.10: Performance comparison between progressive and one-shot pruning across varying degrees of model compression.

Table 5.2: Training and compression settings for compressed PixArt-Σ models.

Model Variant (Remaining %)	**Number of Removed Blocks**	**Training Steps (in Thousands)**
Model 90%	3	88.5
Model 80%	6	265.5
Model 70%	8	531.0
Model 60%	11	708.0
Model 50%	14	973.5

5.1.5 Knowledge Distillation Loss

In this section the standard setting (see sec. 5.1.1) except the learning objective is varied to investigate the effectiveness of the individual components of the knowledge distillation loss (see eq. 2.90). In total, five different loss functions are testet:

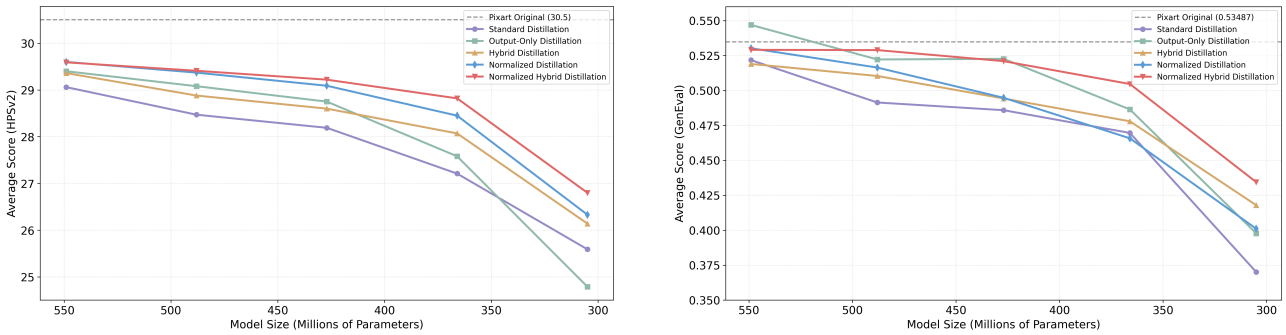
- **Output-Only Distillation** ($\mathcal{L}_{\mathbf{OutKD}}$):
  The MSE between the final output of the student and the teacher serves as the sole learning signal.
- **Standard Distillation** ($\mathcal{L}_{\mathbf{OutKD}} + \mathcal{L}_{\mathbf{FeatKD}}$):
  The unweighted sum of the final output loss and the intermediate feature loss is leveraged as learning objective.
- **Hybrid Distillation** ($\mathcal{L}_{\mathbf{OutKD}} + \mathcal{L}_{\mathbf{FeatKD}} + \mathcal{L}_{\mathbf{Task}}$):
  The original diffusion loss is added to the standard distillation loss.

- **Normalized Distillation** ($\mathcal{L}_{\mathbf{OutKD,norm}} + \mathcal{L}_{\mathbf{FeatKD,norm}}$):
  The final output and the intermediate feature loss are normalized (see sec. 4.5.1) before summed together to ensure that individual layers do not dominate the total loss.

- **Normalized Hybrid Distillation** ($\mathcal{L}_{\mathbf{OutKD,norm}} + \mathcal{L}_{\mathbf{FeatKD,norm}} + \mathcal{L}_{\mathbf{Task}}$):
  This configuration extends the normalized distillation objective by incorporating the original diffusion task loss.

For all loss configurations models of different size (90% to 50%) are trained whereas the same settings as described in tab. 5.2 are utilized. Fig. 5.11 presents the performance for the performance of models with varying degrees of compression based on the different learning objective.
According to the HPSv2 benchmark which measures overall image quality, the normalization of the intermediate features and the final output loss proves advantageous (see blue and red lines in Fig. 5.11). Both loss configurations which utilize normalization achieve a consistently higher HPSv2 score compared to the unnormalized counterparts. The normalization ensures that the contribution of each intermediate feature to the overall loss is balanced. This prevents the optimization process from prioritizing the later layers where the errors are numerically larger to the detriment of the earlier layer. The numerically larger discrepancies between the teacher and the student stem from the larger feature magnitudes in deeper layers as displayed in fig. 5.12 which presents the RMS of the intermediate features after every transformer block in PixArt-Σ. The increase in feature magnitude results from to the signal accumulation inherent to the PixArt-Σ architecture which follows the residual formulation $\mathbf{x}_{l+1} = \mathbf{x}_l + B_l(\mathbf{x}_l)$ where B_l representing the l^{th} transformer block.
Of the two loss configurations applying normalizaiton, the normalized hybrid distillation performs best. It is advantageous for the pruned model if besides the teacher's learning signal the original diffusion loss is leveraged too. The original loss becomes critical when the teacher is uncertain itself about a prediction as it anchors the student to the ground truth.
The benefit of the original diffusion loss is also confirmed by the unnormalized settings where the hybrid distillation consistently achieves a higher HPSv2 score compared to the standard distillation approach.
Interestingly, the output-only approach outperforms both standard and hybrid distillation settings for shallow compressed models. This suggests that the unbalanced feature loss prevents effective learning. However, for strongly compressed models the output-only loss configuration leads to an abrupt drop in quality. This shows that the intermediate feature loss despite being suboptimally distributed serves as a critical stabilizer of the training process.

The GenEval benchmark mainly confirms the observations from the HPSv2 benchmark, that the normalized hybrid distillation approach performs best overall and that adding the original diffusion loss boosts performance. Notably, the output-only configuration results are even more pronounced by performing comparably good or better to the normalized hybrid distillation approach in the low compression regime (90% to 70%). This suggests that at low compression, removing the constraints of feature matching allows the student to optimize semantic alignment very effectively. However perfrmance drops severly for larger compression rates. Furthermore, it is worth noting, that for the high compression of 60% or 50% the unnormalized hybrid distillation approach slightly outperforms the normalized distillation approach. This indicates that in the high compression regime the original diffusion loss is critical for maintaining the text-to-image alignment.



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.11: Impact of different loss configurations on performance across varying compression ratios.

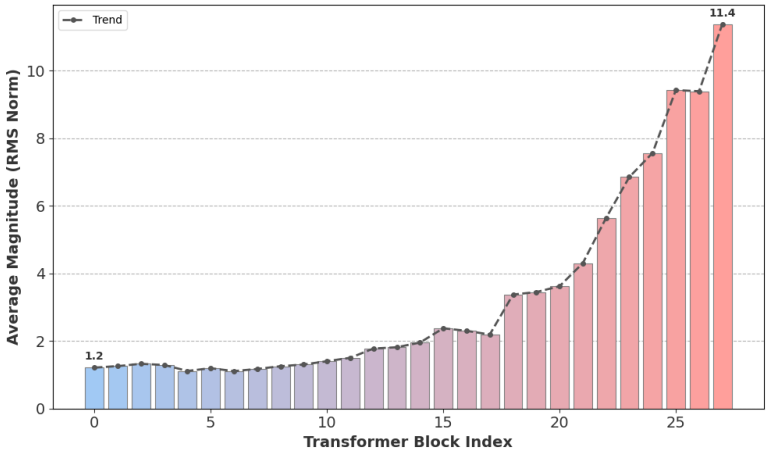


Figure 5.12: The RMS norm of the output features of the individual transformer blocks from PixArt-Σ varies significantly. Features from later layers have a larger magnitude than earlier layers. The RMS norm is computed on 100 prompts from LAION dataset and averaged.

5.1.6 Finetuning Protocol

After removing blocks from the PixArt-Σ architecture, the reduced model is finetuned to recover its image generation capabilities. In total, three different finetuning protocols are tested to identify the most effecive one. In all protocols, the model was trained for 88.5k optimization steps to enable fair comparison.

- **Three Stage Finetuning Protocol**
  This is the default finetuning protocol described in sec. 5.1.1. It is repeated for the reader's convinience.

  1. Local Stabilization: The blocks immediately preceding the removed blocks are finetuned for 38k steps on the LAION dataset while all other parts of the model are frozen.
  2. Global Optimization: The complete model is optimized for one epoch (38k steps) on the LAION dataset.
  3. Domain Specific Refinement: The LAION-Pixart dataset is leveraged to finetune the pruned model for 12.5k steps on its teachers domain.

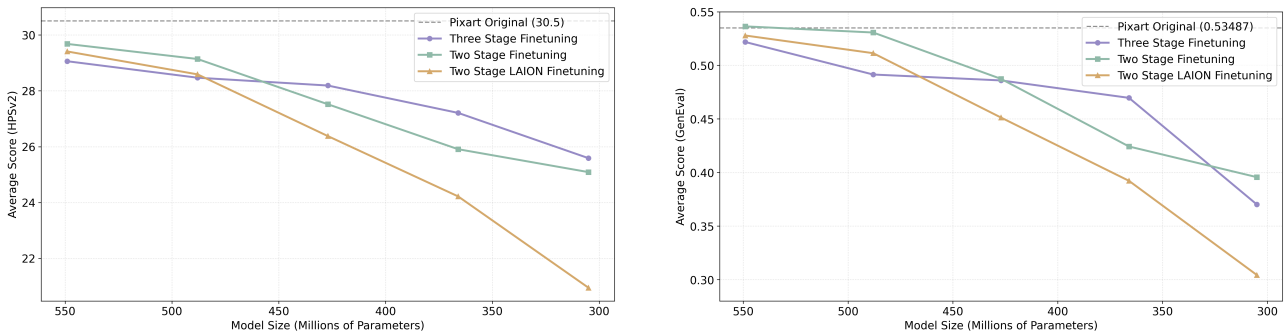
- **Two Stage Finetunig Protocol**
  In the following protocol, instead of performing local finetuning, the entire model is directly finetuned for 2 epochs.

1. Global Optimization: The complete model is optimized for two epoch (76k steps) on the LAION dataset.
2. Domain Specific Refinement: The LAION-Pixart dataset is leveraged to finetune the pruned model for 12.5k steps on its teachers domain.

- **Two Stage LAION only Finetuning Protocol**
  The absence of the domain specific refinement is investigated in this protocol.

1. Local Stabilization: The blocks immediately preceding the removed blocks are finetuned for 38k steps on the LAION dataset while all other parts of the model are frozen.
2. Global Optimization: The complete model is optimized for two epoch (50.5k steps) on the LAION dataset.

The evaluation of the performance obtained for models of different pruninng ratios (90% to 50%) can be found in fig. 5.13. For both the HPSv2 and GenEval benchmarks, fine-tuning solely on the LAION dataset (yellow line) leads to a sharp decline for strongly compressed models. This is a strong indication that the image quality being higher in the LAION-PixArt dataset compared to the LAION dataset is highly relevant for the recoverage of the generation capability of a pruned transformer-based diffusion model. Moreover, it could be observed that for shallow compressed models (10% to 20%) the two stage finetuning protocol (green line) performs better than the three stage protocol (purple line). This indicates that in the low compression regime the model benefits from the extended global finetuning and no local repair is needed. For the middle compressed models (70%-60%), the three stage finetuning is consistently better or on par with the two stage protocol. This performance gap suggests that the local stabilization step provides necessary structural regularization when the model capacity is reduced to a critical threshold. For strong compression there are contradictory results. According to the HPSv2 benchmark the three stage approach is still marginally better than the two stage approach. However, the GenEval benchmark suggests that it is reverse. Given the high degree of degradation in both models at this stage, these marginal differences may also reflect the inherent instability of training extremely pruned networks rather than a definitive superiority of one method over the other.



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.13: Impact of different finetuning protocols on performance across varying compression ratios.

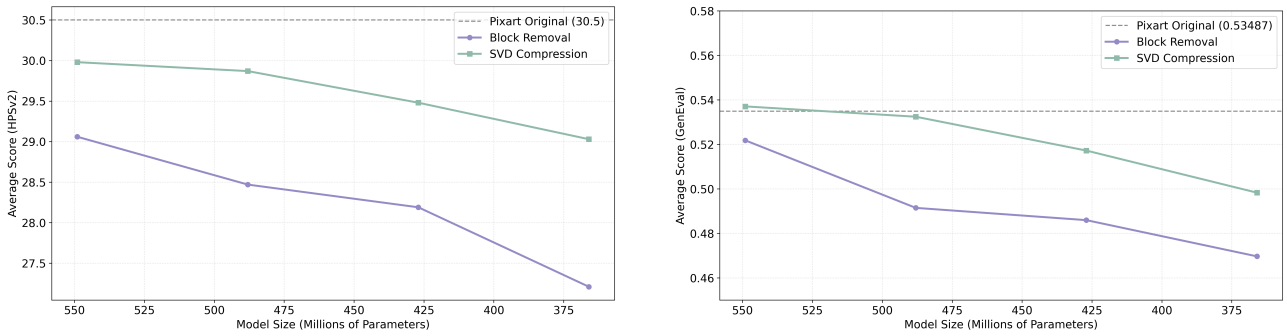
5.1.7 Compression Strategy

As an alternative to the pruning strategy of removing entire transformer blocks from PixArt-Σ, a compression strategy utilizing SVD compression is investigated. This decomposition is applied to the MLP, self-attention and cross-attention matrices within a transformer block. To evaluate the efficiency of the SVD compression method against the structural block removal method, a representative operating point is chosen. The rank for the MLP is set to $r_{\text{MLP}} = 512$ corresponding to a compression of approximately 44% of its parameters and the

ranks for the self- and cross-attention matrices are fixed to $r_{\text{self-attn}} = r_{\text{cross-attn}} = 128$ representing a parameter reduction of approximately 78%. This serves as a starting point. Later, the choice of compressing attention matrices more aggressively than the MLP matrices is further investigated (see sec. 5.1.8).
Tab. 5.3 compares the configurations of both methods, specifically the number of blocks removed or compressed and corresponding training duration for each model. Note that the number of training steps differs significantly, as the default criterion - a CMMD value exceeding 0.1 - was employed to the pruning rate per iteration, consistent with the default settings. The observation that the SVD allows for more parameters to be removed with a less severe impact on the CMMD value on the small reference dataset is an initial indication that compression is a more nuanced method than block removal. Fig. 5.14 presents the performance for both methods on the HPSv2 and GenEval benchmarks. For both benchmarks the SVD compression method outperforms the complete block removal method in low and high compression regimes. This suggests that distributing the parameter reduction across multiple blocks via compression is less adverse to model performance compared to removing individual blocks completely.

Table 5.3: Comparing training steps and modified blocks for block removal versus SVD compression method.

Model Variant (Remaining %)	**Number of Removed Blocks**	**Number of SVD Compressed Blocks**	**Training Steps (in Thousands)**
Model 90%	3	-	88.5
SVD Model 90%	-	6	88.5
Model 80%	6	-	265.5
SVD Model 80%	-	11	88.5
Model 70%	8	-	531.0
SVD Model 70%	-	17	265,5
Model 60%	11	-	708.0
SVD Model 60%	-	22	442,5



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.14: Impact of different compression strategies on performance across varying compression ratios.

5.1.8 Sensitivity Analysis of MLP and Attention Layers under SVD Compression

In this section the SVD compression strategy is applied but otherwise the default settings remains. The goal is to investigate how to distribute the parameter removal of the individual components of one transformer blocks best. Therefore, individual compression ratios are applied to the MLP matrix and the attention matrices whereby both self- and cross-attention matrices are prunded by the same amount. In total three different scenarios are explored:

- **Equal Compression**:
  Both MLP and attention matrices are compressed by the same percentage $p_{\text{MLP}} = p_{\text{Attn}} = 60\%$.

- **Attention-Dominant Compression**:
  The attention matrices are more strongly compressed $p_{\text{Attn}} = 80\%$ compared to the MLP matrix $p_{\text{MLP}} = 40\%$.

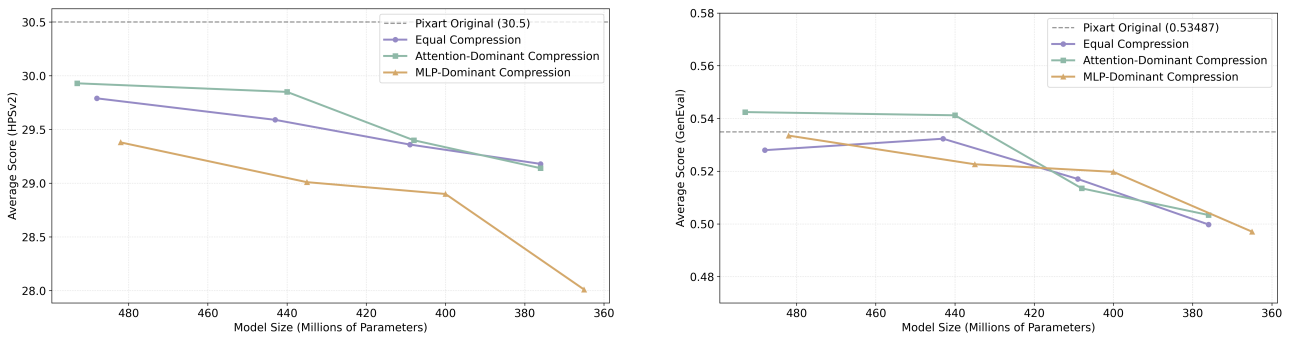
- **MLP-Dominant Compression**:
  The MLP matrix are more strongly compressed $p_{\text{Attn}} = 80\%$ compared to the attention matrices $p_{\text{MLP}} = 40\%$.

Note that the combined parameter count of self- and cross-attention is approximately equal to the parameter count of the MLP. Therefore, all configurations remove a similar amount of total parameters of one transformer block.

The scores in fig. 5.15 reveal two distinct trends. The HPSv2 score demonstrates that prioritizing the compression of the attention matrices consistently yields higher image quality than the method focused on compressing the MLP matrices. The GenEval benchmark, however, shows that although the attention-dominant approach is advantageous in the low-compression regime, it declines sharply under heavier compression, eventually falling below the MLP-dominant approach for models with approximately 400 million parameters.

One possible explanation for the divergence between the two benchmarks is the distribution of tasks among the different components of a transformer block. Recent studies suggest [39] that MLPs are crucial for knowledge storage and the processing of visual features. Consequently, strong compression of these layers leads to a significant decrease in image quality (see HPSv2 benchmark yellow line). In contrast, the GenEval benchmark measures text-image alignment. Since the cross-attention mechanism is responsible for integrating the text prompts into the visual latents its compression in many transformer blocks severially compromise severily compromising this linking. While the image quality suffers less than with the strong MLP compression, the model's text comprehension deteriorates disproportionately. This explains why the attention-dominant approach loses its advantage for heavy compression, and is in some cases even worse than the MLP-dominant approach in the GenEval benchmark.

The equal compression strategy exhibits results similar to the attention-dominant compression approach in the high-compression regimes but is slightly worse in the low-compression regime.



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.15: Impact of varying degrees of compression of individual transformer block components on performance across varying compression ratios.

5.1.9 SVD Compression Scheduling

In previous experiments, SVD compression was applied to each block at most once. However, restricting compression to untouched blocks may be suboptimal and incrementally reducing the size of an already compressed block could potentially preserve model performance better than pruning a dense block. To investigate this, we apply the linear progressive compression scheme defined in sec. 4.6.3 is investigated in the following. Hereby, three different scenarios are investigated where different number of blocks (7, 14 or 21, 28) are compressed per

iteration. This examines whether it is advantageous to compress more blocks but to a lesser extent, or whether the parameters to be removed should be better distributed across fewer blocks. Here, in each iteration 10% of the original number of parameters are removed.

Untersuche ob immer neue Blöcke gepruned werden oder eher dieselbenr mehrfach

5.1.10 Summary of Best Settings for Structural Model Pruning

Considering the results of all experiments, they lead to the following best configuration for structural model pruning:

- **Block Importance Analysis**
  The experiments (sec. 4.2) showed that using the CMMD value as primary block selection criteria is working well. Particularly in linear SVD compression scheduling, where the number of compressed parameters varies per block, the CMMD can be weighted by the number of removed parameters to ensure a fair comparison.
  As block selection method, we found that the greedy algorithm already performs well enough such that more advanced methods like Optuna do not provide substantial benefits that would justify the additional compute.

- **Pruning Method**
  The progressive model pruning has profen advantageous over the one-shot model pruning in the high-compression regime where models are compressed more than 70% of the original size (sec. 5.1.4)). Therefore, this is the first choice.

- **Knowledge Distillation Loss**
  The results indicates that the combination of the original diffusion loss together with the normalized feature and output loss leads to the strongest performance in all pruning regimes (sec. 5.1.5).

- **Finetuning Protocol**
  The finding suggests that incorporating the LAION-PixArt dataset into the training procedure is critical as it contains images of higher quality demonstrated in the higher HPSv2 and GenEval scores for the three and two stage finetuning protocols. However, the results regarding the superiority of three stage protocal over the two stage protocal are inconclusive. It is not clear if local finetuning is beneficial or not for high compression regimes (sec. 5.1.6). Therefore, the two stage approach is chosen for the final setting.

- **Structural Compressin Strategy**
  The experiments demonstrated that leveraging SVD compression over complete block removal leads to superior performance even for less training steps (sec. 4.4).

- **Sensitivity Analysis of MLP and Attention Layers under SVD Compression**
  The sensitivity analysis for MLP and attention layers revealed that strong compression of MLPs led to more severe deterioration than heavy compression of attention layers. However, compressing both MLP and attention layers equally yielded performance comparable to the strong attention compression experiment. Consequently, the equal compression scheme was adopted for the final configuration.

- **SVD Compression Scheduling**

5.2 Flux.1-dev Pruning

Flux.1-dev is a 11.9 billion parameter rectified flow model capable of generating detailed, high-quality, and high-aesthetic images. Due to its high parameter count, it is one primary target for model reduction techniques in the literature and should be used to demonstrate the effectiveness of the best settings found for PixArt-Σ by comparing it to various competitors. In addition, as training of Flux.1-dev is extremely computatinally intensive, the training-free GRASP method (see sec. 4.4.4) originally developped for LLM compression is transfered to Flux1.dev.

5.2.1 Training Details

In this section, all experiments are built upon the training and inference code for Flux.1-dev published in the following git repo [82]. The hyperparameters used for finetuning Flux.1-dev are specified in Tab. 5.4. For pruning Flux.1-dev the best settings found for PixArt-Σ are leveraged and specified here again for completeness:

- **Block Importance Analysis**
  To determine the importance of individual blocks in Flux.1-dev, every block is individually compressed by 60% and a small reference dataset (100 images) is generated which is leveraged to compute the CMMD score. Afterwards, the CMMD score is weighted by the number of removed parameters, e.g. a single-stream block contains less parameters than a double-stream block. In contrast to the experiments with PixArt-Σ, the scoring is computed in a single initial pass rather than leveraging a dynamic greedy algorithm that would iteratively reevaluate the remaining blocks after each sequential compression due to the high computational demands of Flux.1-dev.

- **Pruning Method**
  Progressive structural model pruning is performed.

- **Knowledge Distillation Loss**
  The original flow matching loss as well as the normalized intermediate feature and output losses are utilized whereby the normalization is computed for features from double-stream and single-stream block seperately

$$\mathcal{L} = \mathcal{L}_{\text{Task}} + \lambda_{\text{OutKD}}\mathcal{L}_{\text{OutKD}} + \lambda_{\text{SingleFeatKD}}\mathcal{L}_{\text{SingleFeatKD}} + \lambda_{\text{DoubleFeatKD}}\mathcal{L}_{\text{DoubleFeatKD}} \quad . \tag{5.1}$$

  with $\lambda_{\text{OutKD}} = 10$, $\lambda_{\text{SingleFeatKD}} = 0.01$ and $\lambda_{\text{DoubleFeatKD}} = 0.1$.

- **Finetuning Protocol**
  Only the LAION-Flux dataset is utilized, and a complete finetuning is performed directly.

- **Structural Compression Strategy**
  The SVD compression is leveraged for compressing Flux.1-dev blocks. The individual components of double-stream and single-stream blocks and their corresponding parameters can be found in Tab. 4.2.

- **SVD Compression Scheduling**

A specific

Table 5.4: Configuration details for the compressed model finetuning run, including optimizer settings and flow fatching specific parameters. This parameters represent the config setup for the training using the git repository [82].

Hyperparameter	**Value**
Optimization & Scheduler	
Optimizer Type	Adafactor
Optimizer Arguments	relative_step=False, scale_parameter=False, warmup_init=False
Learning Rate	1.8×10^{-6}
LR Scheduler	constant_with_warmup
LR Warmup Steps	4240
Gradient Accumulation	4
Batch Size	1
Model Precision & Hardware	
Mixed Precision	bf16
Weight Precision (Base)	fp8
Full bfloat16 Training	True
Gradient Checkpointing	True
SDPA (Scaled Dot Product Attn)	True
GPU	$1 \times$ H200
Flow Matching & Loss Settings	
Discrete Flow Shift	3
Timestep Sampling	sigmoid
Max Timestep	1000
Loss Type	L2
Huber Parameter (c)	0.1
Huber Schedule	snr
Model Prediction Type	raw
Guidance Scale	1.0
Sample Sampler	euler_a
Architecture & Data Settings	
T5 Max Token Length	512
Clip Skip	1
Apply T5 Attn Mask	True
Data Loader Workers	6
Seed	42

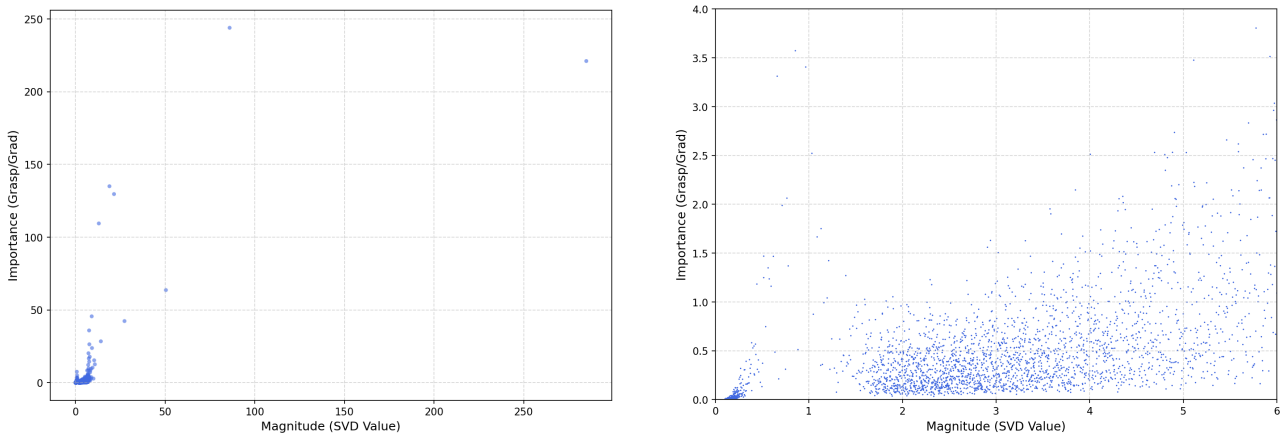
5.2.2 Training-Free Pruning

In this section, it is investigated whether GRASP (see Sec. 4.4.4) is transferable from LLMs to flow-based image generation models like Flux.1-dev. The goal is to identify singular values based on their importance for the image generation task rather than solely based on their magnitude as is standard practise in ordinary SVD compression. The performance of GRASP is compared to Eco-Diff [178] another recently proposed training-free method to prune Flux.1-dev and to the simple application of SVD without retraining.

First, we examine the extent to which the gradient-based method GRASP selects different singular values than a normal SVD, which selects them by magnitude. This experiment focuses exclusively on the compression of double-stream blocks from Flux.1-dev as they contain the most parameters. Exclusively for this setup a pre-existing importance ranking for the Flux.1-dev transformer blocks from [16] (see Sec. 6.3.1) is utilized. In total four different compression levels are tested. The compression rate for the double-stream blocks is fixed to 76% such that the levels differs in the number of compressed blocks. For the exact ranks used during SVD for the individual components please refer to Tab. 6.1.

Fig. 5.16 illustrates exemplarily the magnitude and the importance estimated by GRASP for each singular

value of the image modulation matrix from double-stream block number four. The total view (Fig. 5.16a) reveals a few significant outliers with very high magnitude and importance. Therefore, to investigate the correlation between magnitude and importance for the majority of singulare values Fig. 5.16b provides a zoomed-in version. Here, one can observe a general positive trend where higher magnitude correlates with higher importance. However, there are numerous singular values with low magnitude but high importance and high magnitude values with negligible important values. This discrepancy indicates that GRASP might effectively identify important features that magnitude-based methods would miss, potentially leading to an improvement over ordinary SVD.

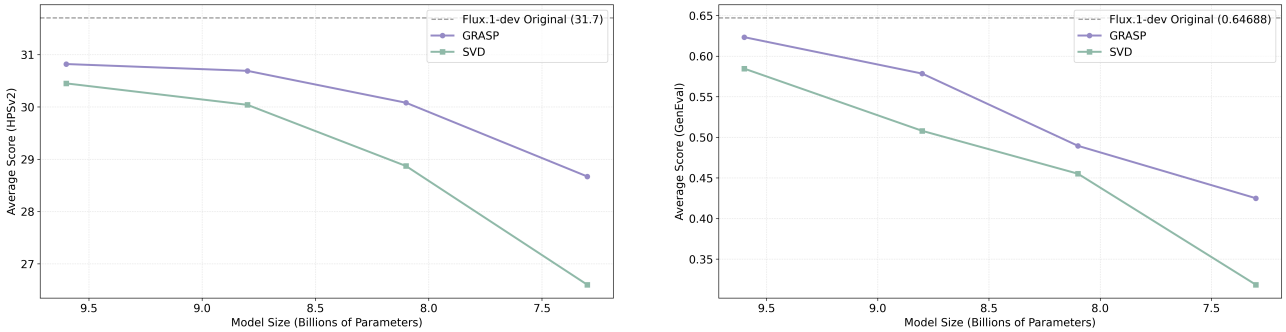


(a) Total view

(b) Zoomed in visualization

Figure 5.16: Comparing the magnitude (x-axis) with the gradient-based importance (y-axis) of singular values from SVD. The visualization is from the 4^{th} block of the image modulation matrix.

The potential for improvements over SVD implied by the analysis of the correlation between magnitude and importance is confirmed by the evaluation of both methods on the HPSv2 and GenEval benchmarks (see Fig. 5.17). The results show that models obtained using the most important singular values (GRASP) instead of those with the highest magnitude outperform standard SVD across all compression ratios on both benchmarks. Fig. 5.18 displays example images, demonstrating that especially for high compression (68% and 60%) the divergence between both methods is pronounced. SVD based models exhibit significant degradation in image quality and detail at these compression ratios. Furthermore, their text-generation capabilities degrades more severely than those of GRASP based models as the kangaroo image still correctly displays "Welcome Friends" at the strongest compression for GRASP but not for SVD (see second column in Fig. 5.18). However, models compressed with GRASP also struggles to generate text consistently correctly at high compression (see fourth columns) and for the highest compression rate some severe artifacts (see third column) can be observed. Nevertheless, both image quality and text generation capabilities are superior with the GRASP method. As image details are typically high-frequency information the results suggest that the high-magnitude singular values primarily encode low-frequency information and the low-magnitude singular values often discarded by ordinary SVD store the high-frequency information. By prioritizing gradients, GRASP better preserves these subtle but important singular values and, thereby mainiting superior text-image alignment and preserving finer details.



(a) HPSv2 Benchmark

(b) GenEval Benchmark

Figure 5.17: Impact of singular value selection based on magnitude (SVD) or importance (GRASP) on performance across varying compression ratios.

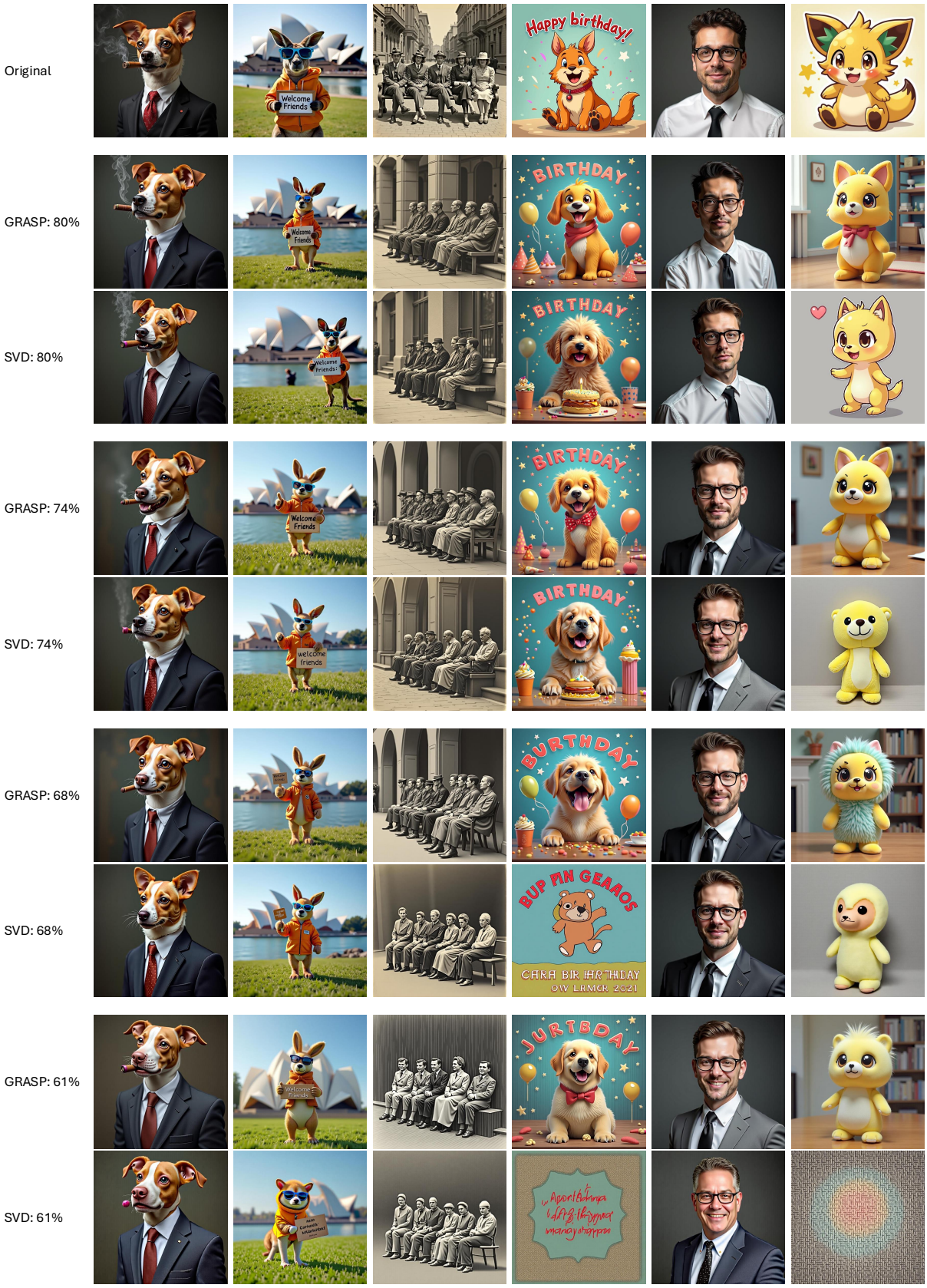


Figure 5.18: Images from compression Flux.1-dev with SVD or GRASP respectively.

Finally, GRASP and SVD are compared to EcoDiff (see Tab. 5.5) demonstrating that these approaches are

significantly superior in maintaining image quality for the same degree of parameter reduction. For a parameter reduction of 20%, the SVD-based methods obtain a GenEval score of 0.58 and 0.62 wheras EcoDiff drops to 0.18 indicating a loss of text-alignment. Fig. 5.19 presents example images from GenEval Benchmark providing visual proof of the strong degeneration and poor text-alignment for compressed Flux.1-dev being smaller than 80% of the original size using EcoDiff. Similarly, the HPSv2 score of EcoDiff measuring human preferences is only 23.6 while SVD and GRASP reach 30.45 and 30.82. Notably, when compressing Flux.1-dev by 30% utilizing the EcoDiff method, the model completely collapses. Visual inspection show that the model ceases to generate meaningful images, producing grey images with undefined texture, in contrast to the performance observed with GRASP. This performance gap indicates that strucutrally removing entire neurons breaks the Markovian generation trajectory of Flux.1-dev and validates the effectiveness of the general concept of SVD that approximates the original information flow instead of relying on the hard removal of neurons as in EcoDiff. Nevertheless, while all methods effectively reduce the VRAM consumption during inference, standard implementations of SVD and GRASP do not inherently reduce the inference time as EcoDiff. This is because low-rank decomposition replaces one large matrix by two smaller matrices which doubles the number of matrix multiplications.

Table 5.5: Performance Comparison of Compression Methods across HPSv2, GenEval, and DPG Benchmarks

Ratio	**Method**	**HPSv2 ↑**	**GenEval ↑**	**DPG ↑**
90%	EcoDiff	29.67	-	-
	SVD			
	GRASP			
80%	EcoDiff	23.633	0.1873	
	SVD	30.45	0.5845	81.30
	GRASP	**30.82**	**0.6232**	**83.12**
70%	EcoDiff	7.26		
	SVD	28.87	0.45516	
	GRASP	**30.08**	**0.48951**	

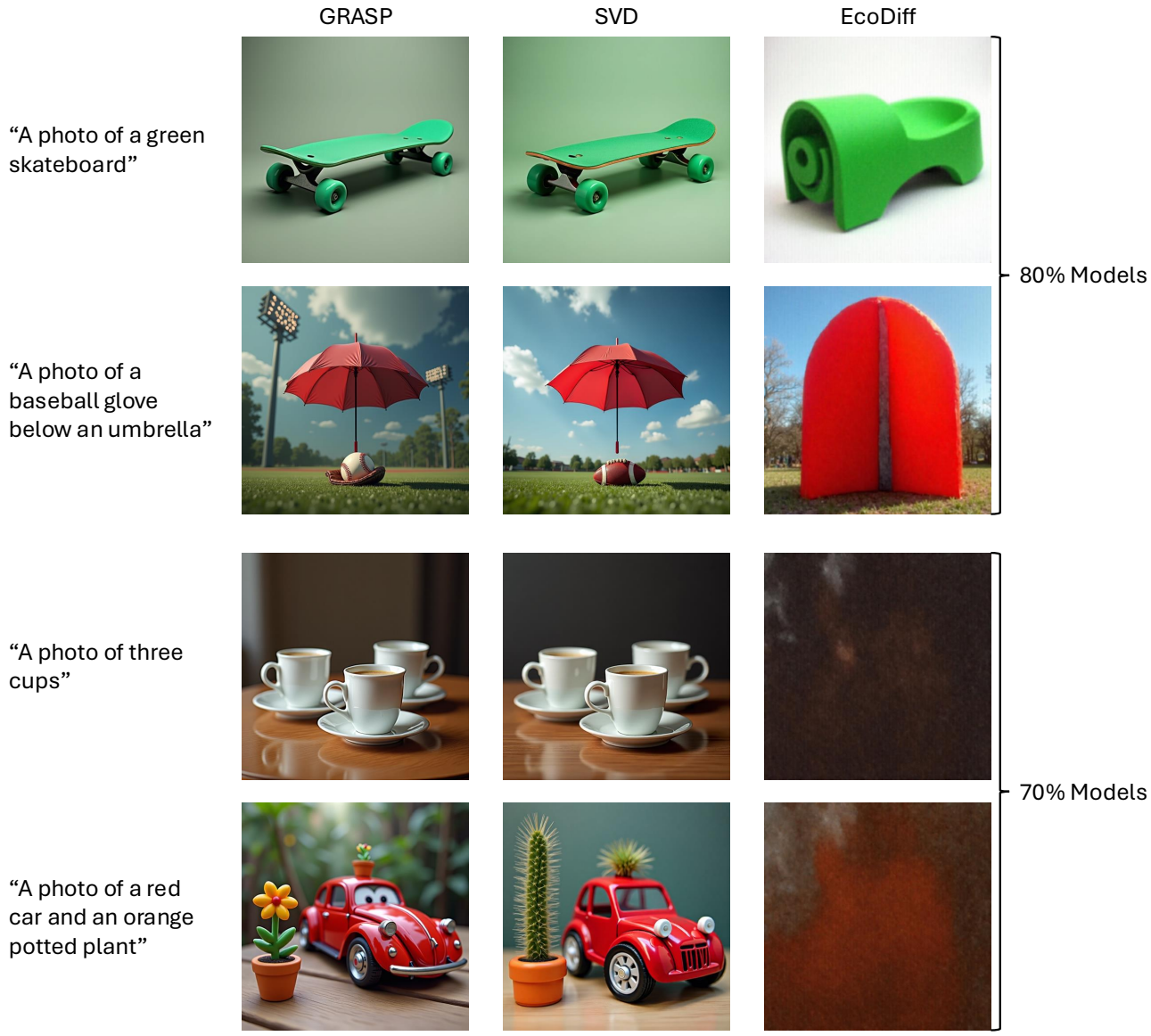


Figure 5.19: Examples from GenEval benchmark for Flux.1-dev compressed by 10%, 20% and 30% using EcoDiff, SVD and GRASP.

5.2.3 Linear Progressive SVD Compression

Chapter 6

Appendix

6.1 Appendix A

6.1.1 Block Analysis

Heatmap for cka: Block importance for different time steps

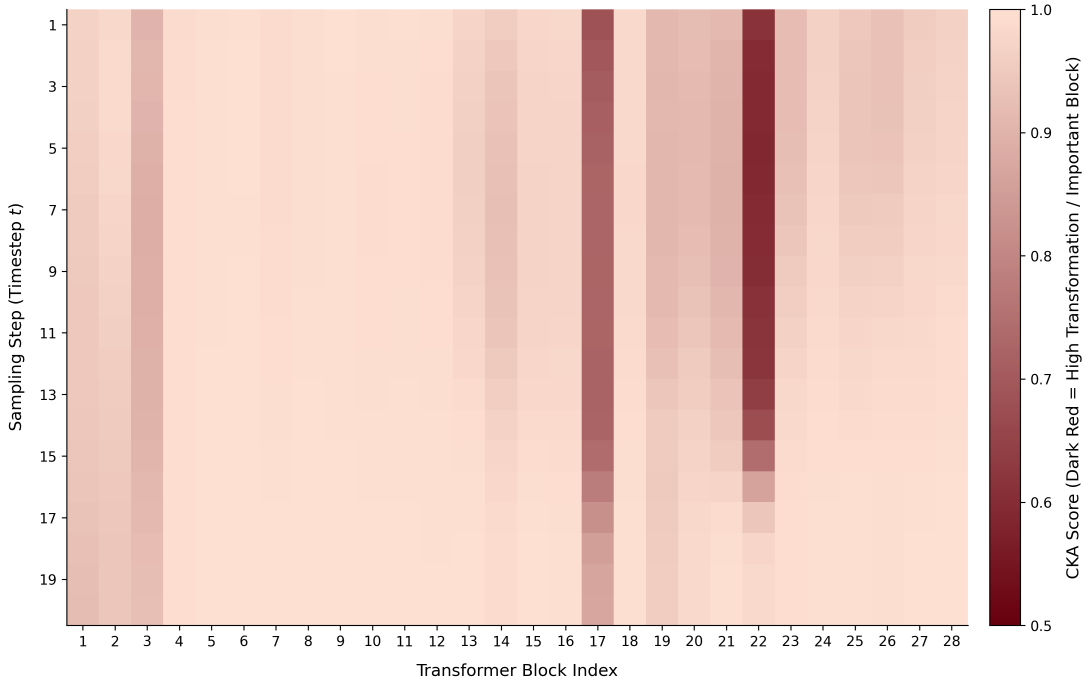


Figure 6.1: CKA-Kernel Clip

6.2 Appendix B - Detailed Overview of Selected Blocks Experiments PixArt-Σ

In this section, the removed or compressed blocks for each experiments are visualized. Different colors indicates during which iteration of the progressive pruning framework the specific block was selected.

6.3 Experimen Flux.1-dev

6.3.1 GRASP

Block Importance Ranking

For this experiment the importance of blocks determined in [16] was leveraged. Below the figure from the corresponding work is presented.

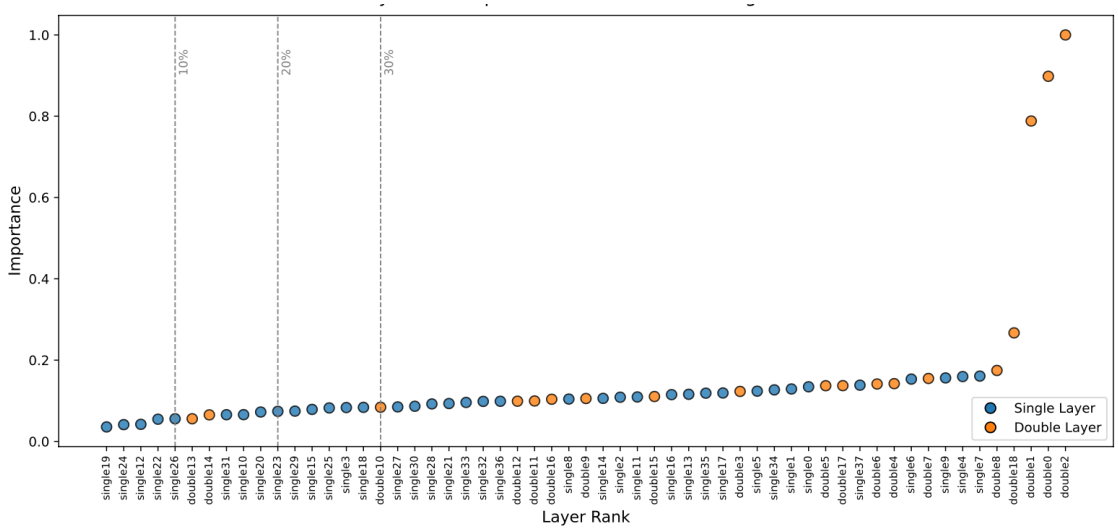


Figure 6.2: Block importance for Flux.1-dev taken from [16].

Compression Details

Table 6.1: Overview of compressed double-stream block components and the respecive rank used in SVD.

Components Double Block	Rank for SVD
Double-Stream Image Linear Layer (Attention)	307
Double-Stream Image MLP	491
Double-Stream Image Modulation	526
Double-Stream Text Linear Layer (Attention)	307
Double-Stream Text MLP	491
Double-Stream Text Modulation	526

Table 6.2: Overview of Pruned Models and Block Configurations

Model Size	# Pruned	Pruned Block IDs
80%	9	13, 14, 10, 12, 11, 16, 9, 15, 3
74%	12	13, 14, 10, 12, 11, 16, 9, 15, 3, 5, 17, 6
68%	15	13, 14, 10, 12, 11, 16, 9, 15, 3, 5, 17, 6, 4, 7, 8
60%	18	13, 14, 10, 12, 11, 16, 9, 15, 3, 5, 17, 6, 4, 7, 8, 18, 1, 0

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/claude-3-haiku`.

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

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

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

[9] Jason Baldridge, Jakob Bauer, Mukul Bhutani, Nicole Brichtova, Andrew Bunner, Lluis Castrejon, Kelvin Chan, Yichang Chen, Sander Dieleman, Yuqing Du, et al. Imagen 3. *arXiv preprint arXiv:2408.07009*, 2024.

[10] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[12] 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.

[13] 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.

[14] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Leo Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators. *OpenAI Blog*, 1(8):1, 2024.

[15] 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.

[16] 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.

[17] Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13169–13178, 2020.

[18] Junsong Chen, Chongjian Ge, Enze Xie, et al. Pixart-sigma: Official pytorch implementation. `https://github.com/PixArt-alpha/PixArt-sigma`, 2024.

[19] 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.

[20] 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.

[21] 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.

[22] Lei Chen, Yuan Meng, Chen Tang, Xinzhu Ma, Jingyan Jiang, Xin Wang, Zhi Wang, and Wenwu Zhu. Q-dit: Accurate post-training quantization for diffusion transformers. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 28306–28315, 2025.

[23] 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.

[24] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[25] 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.

[26] Yu-Hui Chen, Raman Sarokin, Juhyun Lee, Jiuqiang Tang, Chuo-Ling Chang, Andrei Kulik, and Matthias Grundmann. Speed is all you need: On-device acceleration of large diffusion models via gpu-aware optimizations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4651–4655, 2023.

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

[28] 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.

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

[30] 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.

[31] 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.

[32] Javier Martín Daniel Verdú. Flux.1 lite: Distilling flux1.dev for efficient text-to-image generation. 2024.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

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

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

[39] Yihe Dong, Lorenzo Noci, Mikhail Khodak, and Mufan Li. Attention retrieves, mlp memorizes: Disentangling trainable components in the transformer. *arXiv preprint arXiv:2506.01115*, 2025.

[40] 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.

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

[42] 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.

[43] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

[44] 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.

[45] 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.

[46] Determine Filters'Importance. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[47] fpgaminer. JoyCaption: An open, uncensored image captioning VLM. `https://github.com/fpgaminer/joycaption`, 2024. Accessed: 2026-01-22.

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[55] 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.

[56] 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.

[57] Yefei He, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Efficientdm: Efficient quantization-aware fine-tuning of low-bit diffusion models. *arXiv preprint arXiv:2310.03270*, 2023.

[58] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36:13237–13249, 2023.

[59] Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. Ptqd: Accurate post-training quantization for diffusion models. *Advances in Neural Information Processing Systems*, 36:13237–13249, 2023.

[60] 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.

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

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

[63] 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`.

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

[65] 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.

[66] 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.

[67] 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.

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

[69] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

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

[71] 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.

[72] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

[73] 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.

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

[75] 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.

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

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

[78] 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.

[79] 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.

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

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

[82] kohya ss. sd-scripts. `https://github.com/kohya-ss/sd-scripts`, October 2024.

[83] 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.

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

[85] Young D Kwon, Rui Li, Sijia Li, Da Li, Sourav Bhattacharya, and Stylianos I Venieris. Hierarchicalprune: Position-aware compression for large-scale diffusion models. *arXiv preprint arXiv:2508.04663*, 2025.

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

[87] 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.

[88] 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.

[89] 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.

[90] B. Lefaudeux, F. Massa, D. Liskovich, W. Xiong, V. Caggiano, S. Naren, M. Xu, J. Hu, M. Tintore, S. Zhang, P. Labatut, D. Haziza, L. Wehrstedt, J. Reizenstein, and G. Sizov. xformers: A modular and hackable transformer modelling library. `https://github.com/facebookresearch/xformers`, 2022.

[91] 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.

[92] 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`.

[93] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. *arXiv preprint arXiv:2411.05007*, 2024.

[94] 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.

[95] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023.

[96] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. *Advances in Neural Information Processing Systems*, 36:20662–20678, 2023.

[97] 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.

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

[99] 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.

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

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

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

[103] Kainan Liu, Yong Zhang, Ning Cheng, Zhitao Li, Shaojun Wang, and Jing Xiao. Grasp: Replace redundant layers with adaptive singular parameters for efficient model compression. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26344–26359, 2025.

[104] 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.

[105] 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.

[106] 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.

[107] 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.

[108] 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.

[109] 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.

[110] 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.

[111] 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.

[112] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International conference on machine learning*, pages 7197–7206. PMLR, 2020.

[113] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 4990–4999, 2017.

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

[115] 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.

[116] 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.

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

[118] 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.

[119] 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.

[120] 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.

[121] 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.

[122] 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.

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

[124] 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.

[125] 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.

[126] 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.

[127] 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`.

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

[129] 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.

[130] 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`.

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

[132] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation, 2023. URL: `https://arxiv.org/abs/2208.12242`, `arXiv:2208.12242`.

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

[134] 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.

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

[136] 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.

[137] 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.

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

[139] 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.

[140] 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.

[141] 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.

[142] 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.

[143] 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.

[144] 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`.

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

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

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

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

[149] 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.

[150] 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.

[151] 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.

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

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

[154] 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.

[155] 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.

[156] Siao Tang, Xin Wang, Hong Chen, Chaoyu Guan, Yansong Tang, et al. Lightweight diffusion models with distillation-based block neural architecture search. *arXiv preprint arXiv:2311.04950*, 2023.

[157] 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.

[158] TencentARC. flux-mini. `https://huggingface.co/TencentARC/flux-mini`, 2024. Hugging Face Model Hub.

[159] 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.

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

[161] 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.

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

[163] 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.

[164] 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.

[165] 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.

[166] 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.

[167] 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.

[168] Chenglin Yang, Celong Liu, Xueqing Deng, Dongwon Kim, Xing Mei, Xiaohui Shen, and Liang-Chieh Chen. 1.58-bit flux. *arXiv preprint arXiv:2412.18653*, 2024.

[169] 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.

[170] 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.

[171] 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.

[172] 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.

[173] 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.

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

[175] 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.

[176] 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.

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

[178] Yang Zhang, Er Jin, Wenzhong Liang, Yanfei Dong, Ashkan Khakzar, Philip Torr, Johannes Stegmaier, and Kenji Kawaguchi. Learnable sparsity for vision generative models. *arXiv preprint arXiv:2412.02852*, 2024.

[179] Zhihua Zhang. The singular value decomposition, applications and beyond. *arXiv preprint arXiv:1510.08532*, 2015.

[180] Yang Zhao, Yanwu Xu, Zhisheng Xiao, Haolin Jia, and Tingbo Hou. Mobilediffusion: Instant text-to-image generation on mobile devices. In *European Conference on Computer Vision*, pages 225–242. Springer, 2024.

[181] 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.