University of Stuttgart

Institute for Artificial Intelligence
Machine Learning for Simulation Science

Universitätsstraße 32
70569 Stuttgart

Master Thesis

SwinDiffuser: Accelerating Diffusion Models through Parallel Processing

Yun Ye

Study program:	Automonous System
1. Examiner:	Prof. Dr. Mathias Niepert
2. Examiner:	Prof. Dr. Steffen Staab
Advisors:	Anji Liu, M.Sc.
start date:	15.05.2024
end date:	15.07.2024

Abstract

Diffusion models have emerged as a powerful generative approach in artificial intelligence, particularly for image, video, and audio synthesis. Despite their success, these models suffer from significant computational demands due to the iterative nature of the denoising process. This thesis introduces the SwinDiffuser, a novel method designed to accelerate diffusion models by leveraging parallel processing. The proposed method divides high-resolution images into smaller patches, allowing for simultaneous processing by multiple diffusers. Key innovations include the integration of global feature extractors and shifting windows to maintain coherence across patches, and the utilization of a U-Net architecture for noise prediction. Experimental results demonstrate that the SwinDiffuser achieves comparable image quality to standard diffusion models while significantly reducing generation time. This advancement paves the way for practical applications of diffusion models in real-time scenarios and resource-constrained environments.

Kurzfassung

Diffusionsmodelle haben sich als mächtiger generativer Ansatz in der künstlichen Intelligenz etabliert, insbesondere für die Bild-, Video- und Audiogenerierung. Trotz ihres Erfolgs sind diese Modelle aufgrund der iterativen Natur des Rauschunterdrückungsprozesses mit erheblichen Rechenanforderungen verbunden. Diese Arbeit stellt den SwinDiffuser vor, eine neuartige Methode zur Beschleunigung von Diffusionsmodellen durch parallele Verarbeitung. Die vorgeschlagene Methode unterteilt hochauflösende Bilder in kleinere Abschnitte, die gleichzeitig von mehreren Diffusoren verarbeitet werden können. Zu den wichtigsten Innovationen gehören die Integration globaler Merkmalsextraktoren und verschiebbarer Fenster, um die Kohärenz zwischen den Abschnitten zu gewährleisten, sowie die Nutzung einer U-Net-Architektur zur Rauschvorhersage. Experimentelle Ergebnisse zeigen, dass der SwinDiffuser eine mit Standard-Diffusionsmodellen vergleichbare Bildqualität erreicht und gleichzeitig die Generierungszeit erheblich reduziert. Diese Weiterentwicklung ebnet den Weg für praktische Anwendungen von Diffusionsmodellen in Echtzeitszenarien und ressourcenbeschränkten Umgebungen.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI Artificial Intelligence. 15

CNN Convolutional Neural Network. 35

CPU Central Proceccing Unit. 43

D3PM Discrete Denoising Diffusion Probabilistic Models. 15

DDP Distributed Data Parallel. 35

DDPM Diffusion Denoising Probablistic Model. 15

DSM Denoising Score Machting. 18

DWT Discrete wavelet transform. 44

ELBO Evidence Lower Bound. 17

EMA Exponential Moving Average. 29

GAN Generative Adversarial Network. 15

GPU Graphics Proceccing Unit. 24

HR High Resolution. 41

i.i.d. Independent and identically distributed. 40

IQA Image Quality Assessment. 33

KL Kullback-Leibler. 17

LPIPS Learned Perceptual Image Patch Similarity. 34

LR Low Resolution. 24

LR PSNR Low Resolution Peak signal-to-noise ratio. 35

MSE Mean Squared Error. 33

NN Neural Network. 17

ODE Ordinary Differential Equation. 19

PSNR Peak signal-to-noise ratio. 33

RRDB Residual-in-Residual Dense Block. 25

SDE Stochastic Differential Equation. 7

SR Super Resolution. 24

SSIM Structural Similarity Index. 34

VAE Variational Autoencoder. 20

1 Introduction

1.1 Introduction

Diffusion models have recently gained significant attention in the field of generative Artificial Intelligence (AI). The core concept involves perturbing the input with Gaussian noise to progressively transform it into random noise. A neural network is then trained to predict the added noise, effectively reversing the noising process, as illustrated in Fig. 1.1. This technique allows the generation of highly diverse samples from complex distributions. Building on the basic version, advanced diffusion models have achieved great success in image synthesis, surpassing Generative Adversarial Network (GAN) [DN21][DN21], and demonstrating their value in various domains. These models have been applied in fields such as image, video, and audio generation [KPH+21; LZL+24; RBL+22]. Additionally, Discrete Denoising Diffusion Probabilistic Models (D3PM)[AJH+23], a discretized version of Diffusion Denoising Probablistic Model (DDPM) have expanded their usage to data in discrete spaces. Diffusion models have also been utilized in molecular generation [VKS+23]and in solving combinatorial optimization problems[SY23].

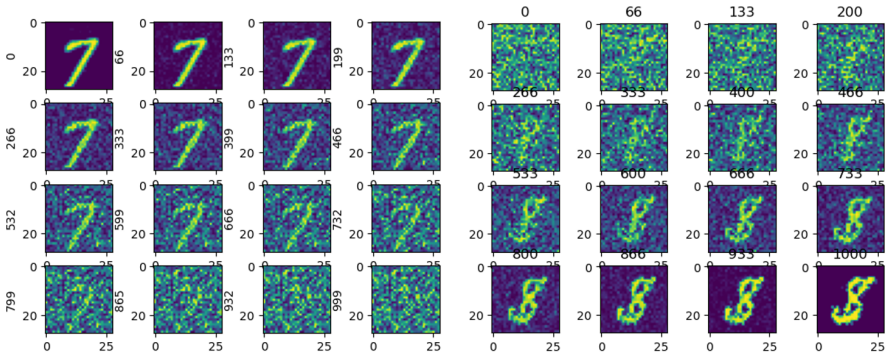

Figure 1.1: D3PM on MNIST Dataset. right: the forward diffusion process with discrete gaussian transition kernel, the image is corrupted during the time. left: the backward process, the image is generated from random noise sampled from a gaussian distribution.

However, a significant drawback of diffusion models is that the progressive denoising operation typically requires substantial time and computational resources due to the iterative processing by neural networks. This makes inference slower compared to other generative approaches like GANs and relatively high-performance devices. Consequently, this limitation hinders the application of diffusion models in scenarios such as real-time image generation and autonomous driving. Numerous research efforts have been directed at addressing this issue, including projecting and extracting information into smaller latent spaces[RBL+22] before applying diffusion, and using distillation techniques to reduce the necessary sampling steps[SH22].

In this thesis, we introduce the concept of parallel diffusion and propose a new method called 'SwinDiffuser' to overcome the generation time bottleneck in single sample generation use cases. This method aims to distribute the computational workload across multiple devices while preserving the generation quality and diversity of the diffusion model. In the first chapter, we provide a brief introduction to diffusion models and related works. Chapter 2 focuses on the design of our model, SwinDiffuser. The implementation details and performance evaluation are covered in Chapter 3. Finally, we present the conclusion and final evaluation in the last chapter.

1.2 Related Works

1.2.1 Diffusion Denoising Probablistic Model

The diffusion model was first introduced in [SWMG15] and further developed in Ho et al. [HJA20]. The concept of diffusion originates from physics, describing the redistribution process of mass to achieve an equilibrium concentration. Similarly, the diffusion proposed in the DDPM progressively inserts noise into a sample. In the forward diffusion process, the sample is gradually transformed from a complex unknown distribution p_{data} in to a prior distribution p_{prior}, from which samples can be easily drawn, such as a Gaussian distribution. In the backward path, the sample starts as random noise and is gradually transformed into a valid sample by a pre-trained denoised network $p_\theta(x_{t-1}|x_t)$ (figure 1.2).

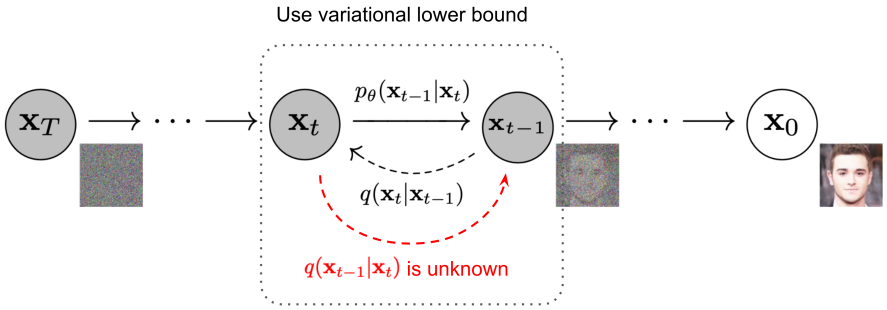

Figure 1.2: Overview of DDPM model: $q_(x_t|x_{t-1})$ is the forward diffusion process and $p_\theta(x_{t-1}|x_t)$ is the learnable backward process.(from paper [HJA20])

The diffusion kernel is typically Gaussian. The forward process can be described by the following formula1.1:

$$\mathbf{q}(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}), \quad \forall t, 0 < t < T. \tag{1.1}$$

Here x_t represents the sample at a particular time step t and β_t denotes the noise schedule, where $0 < \beta_1, \beta_2, ..., \beta_T$. The recursive formulation can be rewritten using some useful properties of Gaussian distributions, allowing direct sampling from x_0:

$$\mathbf{q}(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\overline{\alpha_t}}\mathbf{x}_0, (1-\overline{\alpha_t})\mathbf{I}), \quad \alpha_t = 1-\beta_t \quad and \quad \overline{\alpha_t} = \prod_{i=0}^{t} \alpha_i. \tag{1.2}$$

Intuitively, the value $\overline{\alpha_t}$ gradually decreases to 0. Therefore, for the sample $\mathbf{x}_t = \sqrt{\overline{\alpha_t}}\mathbf{x}_0 + \sqrt{1-\overline{\alpha_t}}\mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$, the percentage of the original sample x_0 shrinks, eventually leaving pure random Gaussian noise.

The objective is to train a Neural Network (NN) $p_\theta(\mathbf{x}_t - 1|\mathbf{x}_t)$ to reverse the diffusion process. The network is designed to model a Gaussian $p_\theta(\mathbf{x}_{t-1}|x_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t))$. $\mu_\theta(\mathbf{x}_t, t)$ and $\Sigma_\theta(\mathbf{x}_t, t)$ are learnable parameters representing the mean and variance of the Gaussian distribution, respectively. In practice, Σ_θ is often substituted with a predefined function. Although the exact loss function for training the network is intractable, we can derive the Evidence Lower Bound (ELBO) using the Markov property:

$$L \leq \mathbb{E}_{\mathbf{x}_0 \sim p_{data}, \mathbf{x}_{1:T} \sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}] \tag{1.3}$$

$$\leq \mathbb{E}_{\mathbf{x}_0, \mathbf{x}_{1:T}}[\sum_{t=1}^{T} D_{KL}[q(\mathbf{x}_{t-1}|\mathbf{x}_t, x_0)|p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)]]. \tag{1.4}$$

This equation shoes that the loss reduces to the Kullback-Leibler (KL) Divergence between $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and the learned $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ The tern $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ can be analytically derived using bayesian rule:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \tag{1.5}$$

$$= \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha_t}}}\mathbf{z}_1) + \sqrt{\frac{1-\overline{\alpha_{t-1}}}{1-\overline{\alpha_t}}\beta_t}\mathbf{z}_2, \quad \mathbf{z}_1, \mathbf{z}_2 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{1.6}$$

Since both elements in the KL divergence are Gaussian, there is an analytical solution for the KL divergence. The ELBO is further simplified to:

$$D_{KL} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\overline{\alpha_t})}\left|\mathbf{z} - D_\theta(\sqrt{\overline{\alpha_t}} + \sqrt{1-\overline{\alpha_t}}\mathbf{z}, t\right|)_2^2] \tag{1.7}$$

$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}[\left|\mathbf{z} - D_\theta(\sqrt{\overline{\alpha_t}} + \sqrt{1-\overline{\alpha_t}}\mathbf{z}, t\right|)_2^2]. \tag{1.8}$$

In equation 1.7, the final form of the ELBO is shown. The paper suggests disregarding the scaling factor in front to improve sample quality, as we did in equation 1.7. Intuitively, this means we need to train a network to predict the noise added to a clean image x_0.

For inference, we obtain μ_{t-1} using formula 1.5, giving the inference form:

$$\mathbf{x}_t = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\overline{\alpha_t}}}\mathbf{D}_\theta(\mathbf{x}_t, t)) + \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{1.9}$$

Here, the original noise $\mathbf{z}_2$ is substituted by the noise prediction $textbf{D}_\theta$.

1.2.2 Score function

Unlike the traditional log-likelihood maximization approach, score-based methods create a map indicating the ascent direction for the probability of x using a score function $s(x)$. This approach avoids the intractability of the normalization factor by combining logarithms and gradients.

(1.10)
$$s(\mathbf{x}) \equiv \nabla \log p_{data}(\mathbf{x}) \in \mathbb{R}^d.$$

This means that for every data point $\mathbf{x}$ fed into the score function, the output will be an increment direction $\triangle\mathbf{x}$, so that the new data point $\mathbf{x}_{new} = \mathbf{x} + \alpha \triangle \mathbf{x}$ is more likely to be in the data distribution. If the score model is known, we can sample from the data distribution by taking the gradient ascent path:

(1.11)
$$\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha s(\mathbf{x}_t) + \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

where α is the step size and z is the injected noise. This method, also known as the Langevin Dynamics. The question now becomes how do we get a proper estimation of the score function. The challenge for that is the original score function is intractable and randomly drawn from the distribution p_{data} is insufficient to achieve a good estimation across the entire sample space, especially in low-density region. As shown in figure 1.3. The way to address both problems

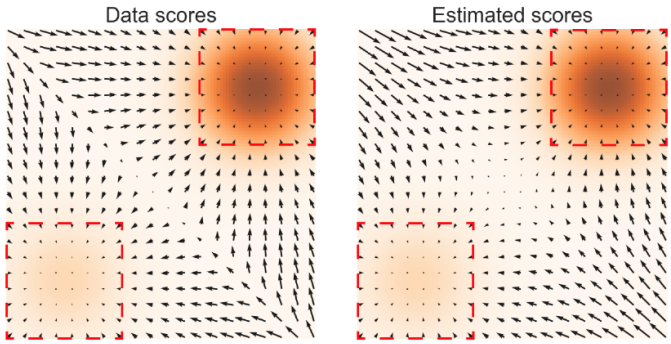

Figure 1.3: The score matching problem: the score function is only correct in the high density region, the low density region suffer from not enough samples.(from paper [SE20])

simultaneously is through Denoising Score Machting (DSM), as suggested in paper [SE20]. This method can be intuitively understood as the linear diffusion process (smoothing) of the probability space with a Gaussian kernel. After convolution, the probability distribution becomes more üniform", addressing the second issue mentioned. In DSM, the original intractable part of the objective(equation 1.12) is replaced by the gaussian diffusion kernel(1.13). The reverse sampling process is called annealed Langevin dynamics, where the noisy distribution at the beginning guides the sample through the low-density area, and a clearer guide is used when the density estimate is trustworthy. This procedure is similar to the process in diffusion models.

(1.12)
$$J(\theta) = \mathbb{E}_{p_{data}(\mathbf{x})}[|\mathbf{s}_\theta(\mathbf{x}) - \nabla \log p_{data}(\mathbf{x})|_2^2]$$

(1.13)
$$J_D(\theta) = \mathbb{E}_{p^{\theta}_{data}(\tilde{x},x)}[\left|\mathbf{s}_\theta(\mathbf{x}) - \nabla \log p^{\sigma}_{\mathcal{N}}(\tilde{\mathbf{x}}|\mathbf{x})\right|_2^2].$$

If we examine this formula closely, we can derive the relationship between the score function and the denoise function $\mu_\theta(x_t, t)$:

(1.14)
$$\mathbf{s}_\theta(\mathbf{x}_t) = \nabla \mathbf{x}_t \log p(\mathbf{x}_t) = \frac{\sqrt{\alpha_t}\boldsymbol{\mu}\theta(\mathbf{x}_t, t) - \mathbf{x}_t}{1 - \alpha_t}.$$

This indicates an alternative way to approximate the score function by utilizing the denoise function from the diffusion model. If the diffusion model is already trained, the score function can be obtained for free.

Another connection is proposed in Song et al. [SSK+21]. In this work, they extend the diffusion model using SDE by transitioning the model into continuous time space. The original DDPM proposed by Ho et al. operates in discrete time with time step $0 < t \in \{1, 2, ..., T\}$.

Song et al. demonstrate that if we extend the time step to infinity, we obtain the forward diffusion process in continuous time, described by the following SDE:

(1.15)
$$d_{\mathbf{x}_t} = -\frac{1}{2}\beta_t \mathbf{x}_t d_t + \sqrt{\beta_t} d\boldsymbol{\omega}_t,$$

where $d\omega_t$ is the Wiener process(or the brownian motion). Furthermore, this process can be reversed using the following equation:

(1.16)
$$d_{\mathbf{x}_t} = [-\frac{1}{2}\beta_t \mathbf{x}_t - \beta_t \nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)]d_t + \sqrt{\beta_t} d\overline{\boldsymbol{\omega}_t}.$$

This provides the reverse SDE to transition from the prior distribution back to the original distribution from which our data samples are drawn. The key component here is the score function $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. The visualization of the forward and backward diffusion SDE in 1D is depicted in Figure 1.4.

Starting from the left is the data distribution $p_0(\mathbf{x})$ with each slice representing the marginal distribution $p_t(\mathbf{x})$. At time step T(in the middle), it is transformed into the prior distribution $p_T(\mathbf{x})$. And the right half completes the reverse process.

This approach also opens up new possibilities for optimizing the diffusion process, potentially leading to more efficient and accurate generative models. If we discard the Wiener process $d\overline{\omega_t}$ in the backward process, we obtain the Ordinary Differential Equation (ODE) formulation of the diffusion process, as indicated by the white lines in Figure 1.4. To perform inference, we need to sample from the prior distribution and use a numerical ODE solver to reverse the entire process.

1.2.3 Potential Problem

While diffusion models have achieved high-quality sample generation, they also have evident drawbacks. Compared to GANs, inference is extremely slow due to the iterative denoising architecture. The noise is gradually removed at each time step, meaning the sample must pass

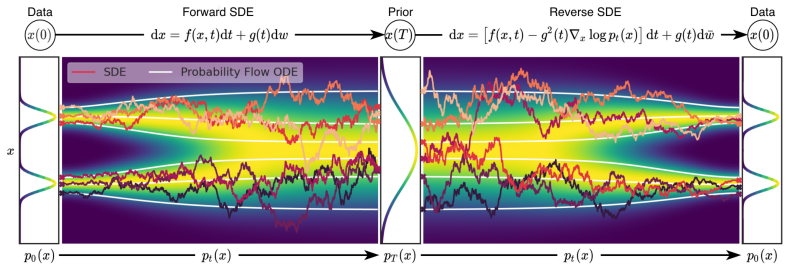

Figure 1.4: Overview of Diffusion model with SDE

through the network T times (usually set to 1000). As the number of time steps increases, the computational burden escalates, making it impractical for real-time applications. Addressing this limitation is crucial for broadening the applicability of diffusion models in various domains, such as autonomous systems and interactive AI applications. Currently, there are methods to accelerate this

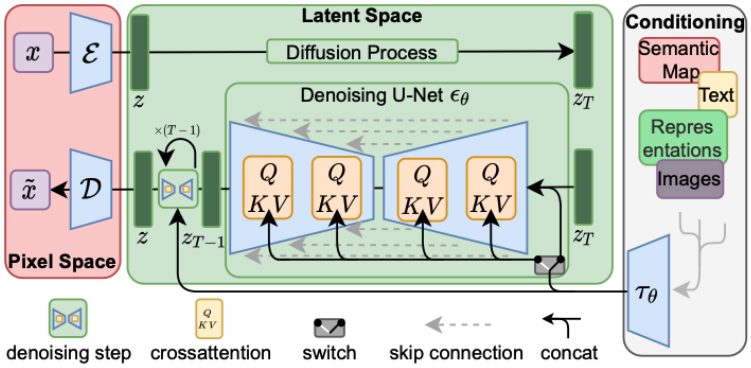

Figure 1.5: Overview of latent diffusion model[RBL+22]

process[RBL+22; SH22], which combines diffusion models with Variational Autoencoder (VAE). In this method, the image is first downsampled to a latent space via VAE to reduce dimensionality, thus accelerating the diffusion process in the latent space. Another approach focuses on reducing the inference steps using distillation[SH22]. Here, a pre-trained diffusion model is used to 'teach' the student network to mimic its behavior but with fewer time steps. This method significantly reduces inference time without a substantial performance drop. Both of these methods aim to optimize the trade-off between computational efficiency and sample quality.

However, these methods also have drawbacks. Both introduce additional complexity into the training phase. Latent diffusion requires training an extra VAE and incorporates additional adversarial architecture to ensure quality. The training of the encoder cannot be parallelized with the training of the diffusion model. Regarding progressive distillation, it adds complexity to the implementation and involves multi-stage learning, which prolongs the training time and relies heavily on the quality of the teacher models. The need for additional components and extended training times can limit the scalability and flexibility of these methods. Meanwhile, the neural networks employed in these models are typically large and challenging to execute on low-end devices. Consequently, inference is generally conducted on high-performance machines, further limiting their applicability.

These observations inspired us to develop a new method to accelerate the diffusion process while preserving high-quality performance without introducing additional difficulties into the training procedure and maintaining the original diffusion architecture.

2 Parallel Diffusion

As the size of the diffusion model grows, the computational complexity also increases dramatically. The size of the diffusion model, training duration, and inference time all grow accordingly. Currently, the time required to synthesize high-resolution images remains relatively long due to the recursive denoising process. Another significant issue, as previously discussed, is that executing inference on devices with insufficient computational resources is typically unfeasible. This limitation further exacerbates the challenge of deploying these models in resource-constrained environments. Although the inference time problem can be partially solved by distillation[SH22], which reduces the intermediate steps necessary to denoise, or by bringing the original image to a smaller latent space[RBL+22]. However, this issue still remains, making diffusion models difficult to apply in real-time generation problems, such as optical flow prediction. Therefore, we focus on another aspect of the problem: using the divide and conquer principle to separate a large-scale task into smaller pieces and solve it on different machines to achieve a speed improvement for a single sample.

For example, in image generation, we can divide a high-resolution picture into smaller patches and apply smaller-scale diffusion kernels to each of them. As the image size is smaller, the throughput will also be reduced. The computational complexity can be easily distributed across different machines, so the overall generation process will be faster. This patch-based approach ensures that each smaller image segment is processed independently, allowing for more efficient parallelization. By leveraging multiple devices to handle different patches simultaneously, the overall inference time can be significantly reduced, enabling faster generation of high-resolution images.

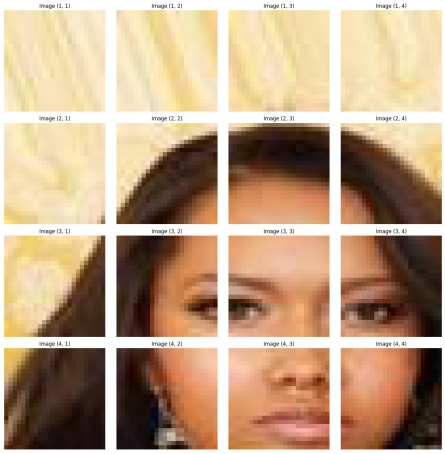

Figure 2.1: The original image divided into 4 by 4 patches. Each patch will be operated independently.

Although the multiple Graphics Proceccing Unit (GPU) acceleration can also be applied to normal diffusion process by using data distribution to enable larger batch sizes, allowing more images to be generated simultaneously, this method does not accelerate the generation of a single sample. The focus of the parallel diffusion is to boost the sampling time for a single sample to achieve semi-real-time performance. This approach is particularly advantageous for tasks that require immediate feedback or rapid processing, such as real-time image enhancement or dynamic video generation. In this chapter, we will focus on image generation and image Super Resolution (SR) tasks. Images can be naturally separated into small patches, as is done in Vision Transformer.[DBK+20] Though it's a straightforward idea, it requires extra operations like global feature extraction and pattern shifting to solve potential problems. In this chapter, we will go through the basic idea of the major components of the parallel diffusion architecture we proposed:

I. Global Feature Extractor
II. Shifting Window
III. Neural Network

These components are designed to work in harmony to address the challenges associated with dividing and processing image patches independently. By incorporating global feature extraction, the model ensures that each patch is aware of the overall image context, reducing artifacts and improving coherence in the generated outputs. And at the end, we will provide an overview of the model architecture in training and inference.

2.1 Global Feature Extractor

The basic idea of parallel diffusion is to use the diffusion kernel to generate a large-scale image one step at a time by separating a large sample space into patches, with each single patch handled by an independent diffuser, as shown in Figure 2.1.

However, the problem with this design is that the diffuser, which generates the patches independently, only has access to local information. This constraint introduces artificial boundaries among the edges of patches, similar to JPEG artifacts. Therefore, the diffuser should also be conditioned to a global state that can coordinate among all the independent diffusers. The global state is chosen to be the Low Resolution (LR) images for SR tasks. This can be easily transferred to an image synthesis problem by replacing the LR image with the current sample X_t. Another important aspect is that the diffuser should know the location of the patch to be optimized. Traditional methods like directly inputting the Cartesian coordinates of the bounding box are not preferable here since they require additional encoding and take longer for the network to capture the semantic meaning. Thus, we propose another method: Gaussian attention mask encoding. This method helps in reducing boundary artifacts and ensures that the generated patches integrate seamlessly with each other. We encode the location information of the patch as a Gaussian kernel on the image, as shown in Figure 2.2. Suppose the LR image we use as a condition i $x_{LR} \in \mathbb{R}^{C\times h\times w}$, where C is the number of color channels and H and W the height and width, respectively. The single patches we get after the division are $(x_{0,0}, x_{0,1}, ..., x_{d,d}) \in \mathbb{R}^{C\times \frac{h}{d}\times \frac{w}{d}}$. For each single patch, we provide a condition input with the LR image x_{LR} it self and the additional gaussian attention mask $M \in \mathbb{R}^{1\times h\times w}$ We concatenate them along the color channel to obtain the condition tensor $C \in \mathbb{R}^{(1+C)\times h\times w}$

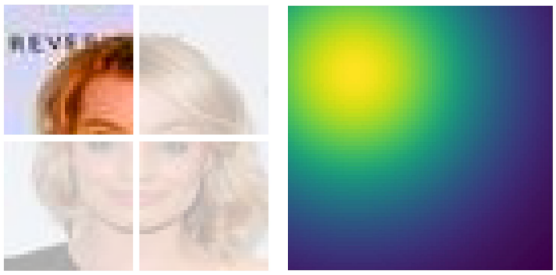

Figure 2.2: Gaussian attention mask example. Right: The patchified image. Left: The positional encoding for the patch on top left.

With the attention mask, the diffuser will not only get the positional encoding directly associated with the original LR input but also pay extra attention to the neighborhood around it. Unlike a Heaviside function, the Gaussian kernel has a much smoother isotropic drop. In the experiment, we also tried to include the current state x_t as part of the condition. However, despite the overhead computation caused by this operation, the improvement was negligible. Therefore, we chose the simpler version.

After that, the condition tensor will first go through a learnable module to extract more fine-grained features for the network. There are two types of global feature extractors used in the experiment:

I. A sequential series of convolutional layers to extract the structure details.
II. Residual-in-Residual Dense Block (RRDB) from the ESRGAN paper [WYW+18], which is frequently applied as the condition extractor in diffusion SR tasks.

The convolutional layers generally perform well in feature extraction tasks. However, recently, the RRDB structure has also become quite popular.

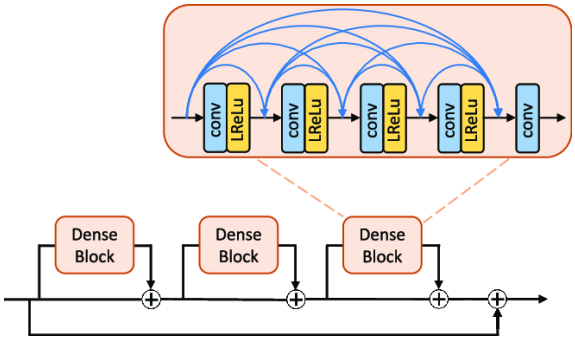

Figure 2.3: Illustration of the Residual-in-Residual-Dense-Block architecture

The idea is to use a series of residual blocks containing dense convolution layers with residual connections. This configuration not only boosts performance in generating high-quality images but also stabilizes the training process. The RRDB module alone can perform a SR inference. However, it can also be used as a feature extractor by discarding the final output and concatenating all outputs from the dense blocks as the global feature. The RRDB network can be trained beforehand as a separate SR task and frozen during the diffusion training or trained together with the diffuser.

However, the global feature extractor alone remains insufficient in fully addressing the issue of artifacts. This is due to the static nature of the condition in relation to the generation process. Consequently, we lack control over the dynamic changes during inference. Therefore, an additional mechanism is required to facilitate dynamic guidance during sampling.

2.2 Shifting Window

Figure 2.4: Experiment on CelebA dataset 64 x 64, scaling factor 2, without shifting. The patches don't fit with each other on the boundaries.

Although the diffuser has access to the global feature and the location of the patches, artifacts can still occur, as shown in Figure 2.4 Sometimes, even when the boundaries are smooth enough, the structure across different windows may be misaligned because a single diffuser only focuses on the same region at every timestep without knowing what happens outside its boundaries.

To further tackle this problem, we also introduce shifting. The inspiration comes from Swin Transformer [LLC+21], where they use a shifting window to build connections across different windows, exploiting the hierarchical architecture of vision transformers. It turns out that this idea can also be used to reduce artifacts. The basic idea is that the patches the diffuser works on are not fixed. They will shift according to a specific pattern, so every time the image is denoised, it is denoised with a different division pattern. This means the boundaries in the inference are constantly changing, allowing information to flow through different parts of the image progressively. So that no single region is processed in isolation for too long.

The shifting window is implemented as shown in Figure 2.5. For each input x_{HR}, before its division into patches, the image will be cropped according to a specific shifting pattern to ensure the location of the diffuser is constantly changing, and there are no fixed boundaries for a single patch diffuser. This implies that certain regions will be processed by multiple diffusers. These regions will function as information buffers, allowing information to flow across boundaries. As a result, different diffusers will gain knowledge of the activities of their neighbors. The shifting pattern is offset from the origin of the image described by a tuple (x, y). The cropped image has width w' and heighth'. To ensure that the cropped image has the same size as before, the image should be either resized or padded depending on the shifting step size s.

Two shifting patterns are provided:

I. Circular shift: the shift pattern is randomly chosen in the tuple $((0,0),(0,s),(s,0),(s,s))$ during training and is selected in turn during sampling to ensure consistence denoising.

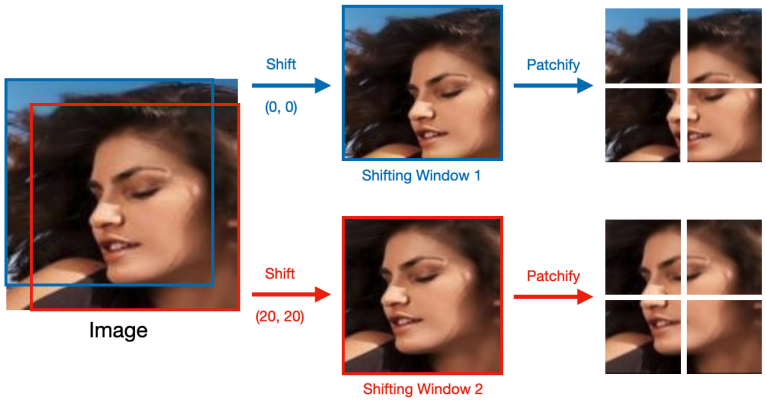

Figure 2.5: The implementation illustration of shifting window

II. Random shift: the offsets are sampled from a discrete uniform distribution $x, y \sim U\{0, s\}$ both during training and sampling.

These shifting patterns provide flexibility in how the patches are processed, allowing the model to adapt to different configurations and improving the overall robustness of the generated outputs. The circular shift ensures a systematic coverage of the image, while the random shift introduces variability that can help in achieving smoother boundaries. The comparison of both methods are in chapter 3.

But this design brings another problem in the inference. Since the diffusion window is moving (raw images are being cropped), not all the pixels are denoised in a single step. This causes the pixels on the edges, especially in the corners, to be insufficiently denoised, leading to imbalanced diffusion overall."

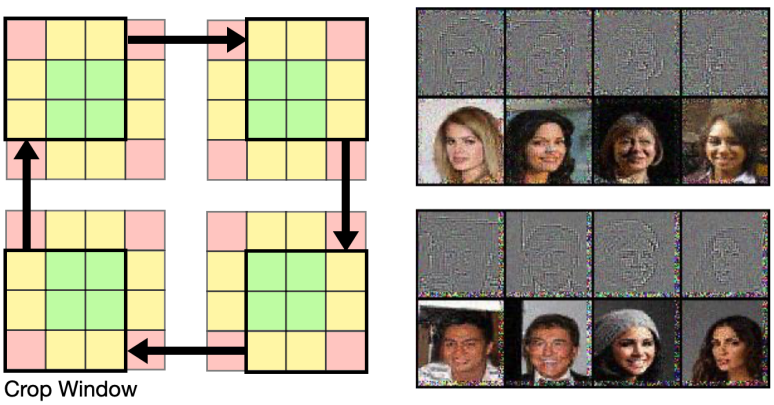

Figure 2.6: Navie shifting strategy is not enough. Left: Illustration of shifting window using circular shift(assume a 4 by 4 patch and shift step equals to a patch length). Right Top: Apply shifting window with circular shift in inference. Right Bottom: Apply no window shifting in inference.

From the above Figure 2.6, one can observe that shifting windows during inference will introduce noisy edges, which is more obvious if shifting is disabled. The insufficiently denoised edges will also affect inference by introducing unexpected information, which will confuse the diffuser, as the pixels in the patch are in different states, causing unsatisfactory results in the experiments. Therefore, another shifting maneuver is needed. Following the strategy of rolling shift in Swin Transformer [LLC+21], we also implemented the rolling shift alongside the padding shift. s shown in Figure 2.7, instead of vanishing, blocks A, B, and C appear on the other side of the image after shifting towards the upper left corner.

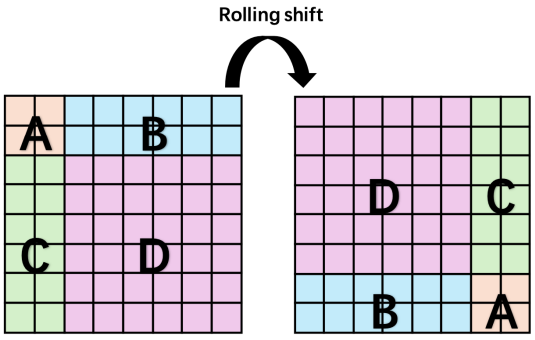

Figure 2.7: Illustration of rolling shift

Rolling shift preserves the original shape of the image, so padding and resizing are not necessary in this case. During inference, all the pixels will be processed equally. Hence, we will apply rolling shift in all upcoming experiments.

And this rolling shift pattern also affect the global feature generation. Since the positional encoding (attention mask) is no longer the same in each iteration for inference, additional operations need to be performed to locate the Gaussian kernel in the right position.

2.3 Neural Network Architecture

As in many other works regarding diffusion models, we choose to use the U-Net architecture as the noise predictor. The U-Net architecture is well-suited for this task due to its ability to capture multi-scale features and its strong performance in various image processing tasks. The detailed structure can be found in Figure 2.8

The model $D_\theta(x_t, C, t)$ takes current state $x_t \in \mathbb{R}^{b\times c_1\times \frac{H}{d}\times \frac{W}{d}}$ condition feature $c \in \mathbb{R}^{b\times c_2\times \frac{H}{d}\times \frac{W}{d}}$ and time step $t \in \mathbb{R}^b$ as input, where c_1 is number of color channels of the original image(3 in our case) and c_2 is the number of feature dimensions. The conditional feature will first go through the global feature extractor explained in previous section. The input convolution will initially convert the dimension of input to align with the condition embedding output. The time embedding is a sinusoidal encoding, which converts $t \in \mathbb{R}^b$ into time feature vector $v_t \in \mathbb{R}^{b\times f_t}$, where f_t is a hyper-parameter indicating the length of the time embedding features. The time feature will go through a learnable fully connected layer before being fed into the downsample convolution layer.

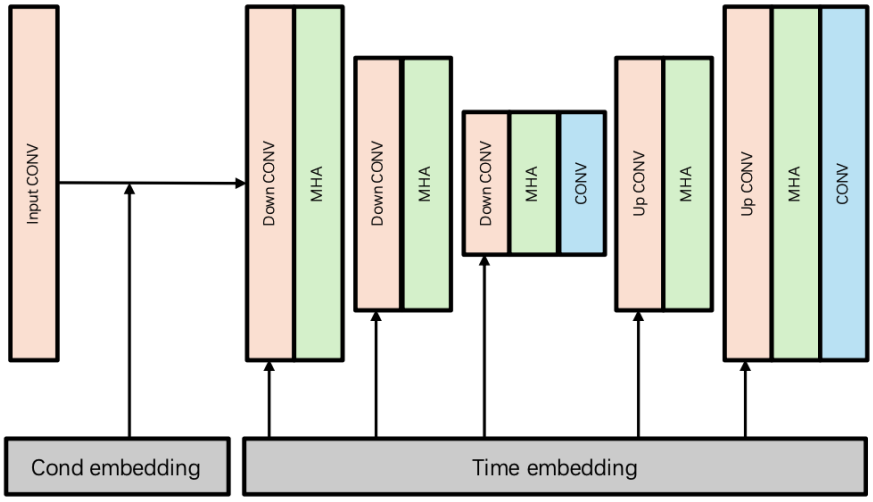

Figure 2.8: U-Net

The downsample convolution layer consists of a series of max pooling, convolution, layer norm, and residual connections to shrink the dimensionality. The upsample convolution layer works the other way around. Between each convolution block, we have also added a couple of multi-head attention layers to increase the expressiveness of the network and better capture long-range dependencies. Additionally, as in typical U-Net architecture, residual connections are built between upsample and downsample blocks. At the end, a convolutional layer is applied to shrink the number of channels back to the normal image.

The optimizer we used in update is the normal Adam optimizer(Learning rate set to 3×10^{-4}). For the sake of stability in training, we also utilize the idea of Exponential Moving Average (EMA). The use of EMA helps to smooth out the parameter updates, reducing the impact of noise and improving the overall stability and performance of the training process. The parameters will be gradually updated with hyper-parameter μ with equation:

$$\boldsymbol{\theta}^{EMA} = (1-\mu)\boldsymbol{\theta}^{model} + \mu\boldsymbol{\theta}^{EMA}. \tag{2.1}$$

In the implementation, the μ value is set to 0.95 to balance between the speed of convergence and reduction of oscillation.

We also include two schedulers to control the learning rate. The first one is for warm up the EMA, which gradually raises the learning rate from zero to the preset value using a simple linear function: $\frac{current\_step}{total\_warmup\_step}$. The second one is a Cosine Annealing scheduler to control the overall changes in the learning rate, which will be updated after every complete epoch.

2.4 Model Pipeline

We are now ready to discuss the actual model of the Swin parallel diffuser with the SR task. The training procedure is illustrated in Figure 2.9 and Algorithm 2.1. The input is an image pair consisting of the original image $X_{hr} \in \mathbb{R}^{b\times C\times H\times W}$ and downsampled LR image $X_{lr} \in$

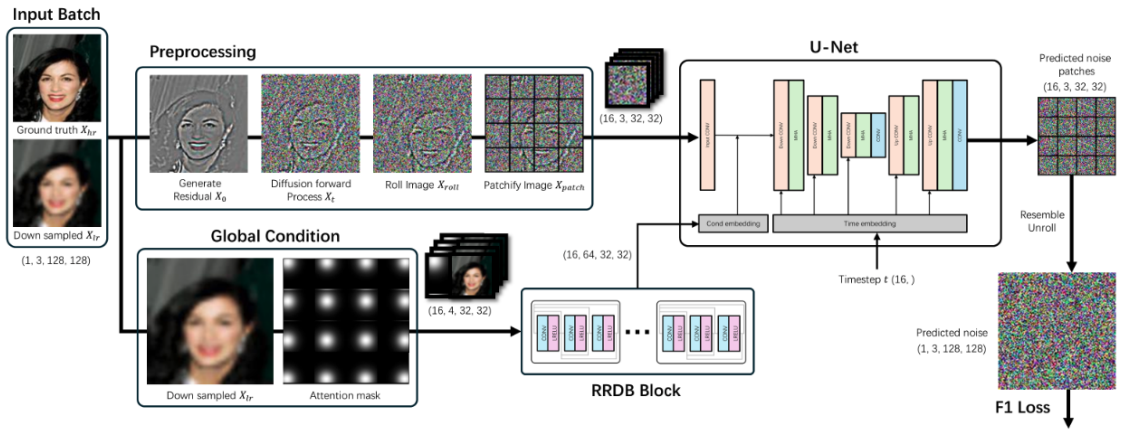

Figure 2.9: Overview of the model training pipeline

$\mathbb{R}^{b\times C\times h\times w}$(downsampled with MATLAB imresize implementation). Then X_hr will go through the pre-processing procedure, which includes adding noise, rolling shifting, and division (patchify) to convert the original image batch into a patch batch $X_{patch} \in \mathbb{R}^{B\times C\times \frac{H}{d}\times \frac{W}{d}}$, where $B = b \times d^2$ is the new batch size and d is the scaling factor.

On the other hand, the LR image x_{LR} will be passed to the global condition generator. First, it will be duplicated along the batch dimension to match the patch batch size B. Then, according to the size, an attention mask $M \in \mathbb{R}^{B\times 1\times h\times w}$ will be generated to encode the location information. Finally X_{LR} and M will be concatenated before being fed into the RRDB feature extractor and retrieve the feature tensor C.

After that, X_{patch}, condition C and time step t will be processed by the U-Net to produce the output $X_{predict}$, which predicts the noise. The L_1 loss will be calculated and backpropogated until the network converges.

The inference, as illustrated in figure 2.10 and algorithm 2.2, is also intuitive. The noise generated at time step X_t from the prior diffusion $\mathcal{N}(0, I)$ will go through the same preprocessing pipeline to be converted into patches. The condition is generated with the already given LR references and attention masks. Then, a single patch batch will be divided into smaller sub-batches depending on the number of available GPUs by the Data Parallel module from PyTorch. This allows a single batch to be parallelly calculated on multiple devices to achieve speed acceleration. After the calculation, the results will be collected and reassembled to construct the noise prediction in the form of a patch batch. Post-processing is still needed, e.g., unpatchify and unrolling, before obtaining the actual output. Then, we calculate the next state and continue this procedure until time step 0 is achieved.

In this chapter, we focused on the main architecture and design of the SwinDiffuser used for parallel diffusion. We proposed several methods and techniques to address the challenges arising from the division of correlated information. Our approach includes the use of global feature extractors, Gaussian attention masks, and shifting windows to ensure seamless integration of patches and reduce artifacts.

Algorithm 2.1 Train Swin Parallel Diffuser

procedure TRAIN(SRDataset D, Noise scheduler α_t, Network NN_θ)
  $\boldsymbol{\theta}^{EMA} \leftarrow \boldsymbol{\theta}$
  while not converge **do**
    $\mathbf{x}_{LR}, \mathbf{x}_{HR} \sim D$
    $t \sim Uniform(0, T)$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_H R + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$
    $roll\_pattern$ = generate_roll_pattern()
    $\mathbf{x}_{rolled}$ = rolling_shift($\mathbf{x}_t, roll\_pattern$)
    $\mathbf{x}_{patch}$ = patchify($\mathbf{x}_{rolled}$)
    $\mathbf{c}$ = generate_condition($\mathbf{x}_{LR}, inverse(roll\_pattern)$)
    $\boldsymbol{\epsilon}_{pred} = NN_\theta(\mathbf{x}_{patch}, \mathbf{c}, t)$
    $Loss = \|\boldsymbol{\epsilon}_{pred} - \text{patchify}(\boldsymbol{\epsilon})\|_1$
    $Loss.backprop()$
    $\boldsymbol{\theta}^{EMA} = (1-\mu)\boldsymbol{\theta} + \mu\boldsymbol{\theta}^{EMA}$
  end while
end procedure

Algorithm 2.2 Sample Swin Parallel Diffuser

procedure TRAIN(LR image $\mathbf{x}_{lr}$, Noise scheduler α_t, Network NN_θ)
  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  for t = T, ..., 1 **do**
    if t > 1 **then**
      $\mathbf{z} \sim \mathcal{N}(0, I)$
    else
      $\mathbf{z} = 0$
    end if
    $roll\_pattern$ = generate_roll_pattern()
    $\mathbf{c}$ = generate_condition($x_{LR}, inverse(roll\_pattern)$)
    $\mathbf{x}_{rolled}$ = rolling_shift($\mathbf{x}_t, roll\_pattern$)
    $\mathbf{x}_{patch}$ = patchify($\mathbf{x}_{rolled}$)
    $\boldsymbol{\epsilon}_{pred} = NN_\theta(\mathbf{x}_{patch}, \mathbf{c}, t)$
    $\boldsymbol{\epsilon}_{unpatch}$ = unpatchify($\mathbf{x}_{\epsilon_{pred}}$)
    $\boldsymbol{\epsilon}$ = unrolling_shift($\boldsymbol{\epsilon}_{unpatch}$)
    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)}}\epsilon) + \sigma\mathbf{z}$
  end for
end procedure

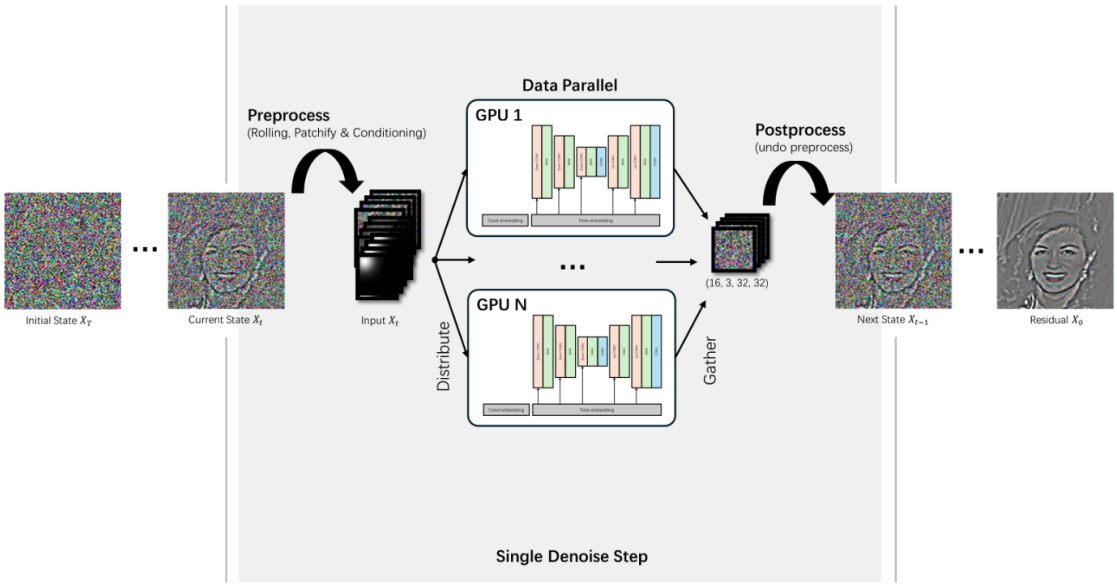

Figure 2.10: Overview of the model inferencing pipeline

By carefully selecting hyper-parameters, we demonstrated that it is possible to significantly minimize artifacts, achieving quality comparable to standard diffusion models. [1] The SwinDiffuser not only maintains the high-quality output expected of diffusion models but also accelerates the generation process, providing a practical solution for computationally intensive tasks. In the next chapter, we will present the experimental results, showcasing the performance improvements and quality comparisons across different settings.

[1] The implementation can be found online https://github.com/Lukasye/Swin_Parallel_Diffusion

3 Experiment

In this chapter, we will discuss the detailed settings of the model, analyze the hyper-parameters, and evaluate whether the SwinDiffuser design can achieve similar or even better performance while reducing image generation duration. The chapter is organized as follows: first, we will discuss the SR metrics we use, given that the Swin framework is focused on SR tasks. Then we will go through the individual experiments and discuss the results:

I. Step Size
II. Shift Pattern
III. Kernel Size
IV. Time Usage

Afterward, we will conduct larger-scale experiments to demonstrate that the Swin architecture indeed brings speed improvements in inference. By systematically analyzing each component and parameter, we aim to establish a robust framework for accelerating diffusion models.

3.1 Metrics

3.1.1 Peak signal-to-noise ratio

Peak signal-to-noise ratio (PSNR) is the most commonly used measurement in Image Quality Assessment (IQA). It is simply the ratio between the signal, represented by the maximum amplitude of the image, and the Mean Squared Error (MSE) between the predicted image and the ground truth, as described in the following formula 3.1:

$$PSNR(\mathbf{y}_{SR}, \mathbf{y}_{HR}) = 10 \times \log_{10} \frac{L^2}{\frac{1}{N} \sum_{i=1}^{N} \left| \mathbf{y}_{SR} - \mathbf{y}_{HR} \right|^2}. \tag{3.1}$$

The idea is intuitive and easy to implement, making it straightforward to compare different models or algorithms. As it is widely used in various SR benchmarks, we include this as part of the quality measurement approach. However, this method is too simple as it only considers pixel-wise differences and fully ignores dependencies in the neighborhood. Consequently, the results might not align with human perception and may neglect the measurement of structural similarity. Despite its limitations, PSNR provides a baseline for evaluating image quality and helps in initial comparisons between different models.

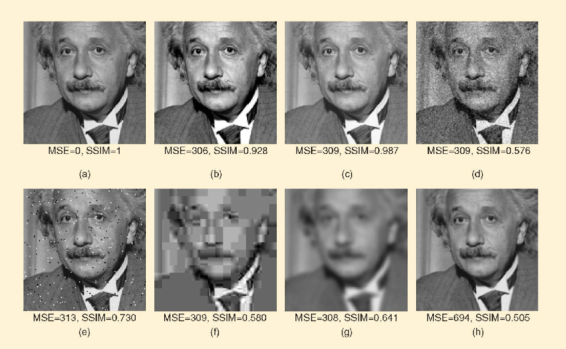

Figure 3.1: Comparison of MSE(PSNR) and SSIM: The non-structure distortion which causing significant changes in MSE is actual acceptable for human perception. But structure distortion, which is worse won't be detected by PSNR Loss.

3.1.2 Structural Similarity Index

Structural Similarity Index (SSIM) is another popular method for IQA. As proposed in the paper [WBSS04], this method focuses on extracting structure similarity, instead of naively compute pixel-wise errors. It states that the subjective nature of our visual system cannot be represented by simple L1 or L2 loss. Non-structural distortions (luminance, contrast changes) do not affect human assessment as severely as structural distortions, such as blurring, noise, and JPEG artifacts. PSNR is not designed to measure these structural distortions, as shown in Figure 3.1. Therefore, they designed a new measurement to detect structural similarities by generating quality maps via sliding windows for the images.

$$SSIM(\mathbf{y}_{SR}, \mathbf{y}_{HR}) = [C_l(\mathbf{y}_{SR}, \mathbf{y}_{HR})]^{\alpha} \times [C_c(\mathbf{y}_{SR}, \mathbf{y}_{HR})]^{\beta} \times [C_s(\mathbf{y}_{SR}, \mathbf{y}_{HR})]^{\gamma}. \tag{3.2}$$

The loss, as shown in formula 3.2, can be divided into comparisons of luminance, contrast, and structural similarity based on statistics like mean and variance. The parameter α, β and γ is the weighting factor for different criteria. For details of derivation, please refer to the original paper.

Compare to the PSNR, SSIM evaluates the image in a way more similar to human perception. It uses local information to analyze structural differences and is more sensitive to structural distortions. SSIM has now become a standard measurement in SR tasks.

3.1.3 Learned Perceptual Image Patch Similarity

Learned Perceptual Image Patch Similarity (LPIPS), proposed in Zhange et al [ZIE+18], utilizes pretrained neural networks like VGG or AlexNet to achieve more realistic evaluation results. The approach involves taking two images, the SR prediction y_{pred} and ground-truth y_{target}, and feeding them to a pre-trained network. he network acts as a feature extractor, and a series of breakpoints in the network are set. The MSE loss is compared along the forward path from all these breakpoints l, as shown in formula 3.3, where ω_l is the weighting factor, and H_l and W_l correspond to the dimension of the breakpoint layers.

$$LPIPS(\mathbf{y}_{SR}, \mathbf{y}_{HR}) = \sum_{l=1}^{L} \frac{1}{H_l W_l} \sum_{h,w} \left| \omega_l (\mathbf{y}^l_{SR,h,w} - \mathbf{y}^l_{HR,h,w}) \right|_2^2. \tag{3.3}$$

The idea is that the Convolutional Neural Network (CNN) will gradually extract details from coarse to fine during the forward pass. Therefore, the MSE loss will naturally compare the two images from details to the global overview perspective. All these comparisons are done in the learned feature space rather than the original image space, making it more flexible and easier to capture high-level information, providing a more comprehensive assessment of image quality.

This approach leverages the hierarchical feature representations learned by deep networks to compare images in a more perceptually meaningful way. However, LPIPS has greater computational complexity than traditional metrics, and its performance depends on the quality of the pre-trained networks.

3.1.4 Low Resolution Peak signal-to-noise ratio

Low Resolution Peak signal-to-noise ratio (LR PSNR) is another version of the standard PSNR method proposed in paper [LDGT20]. It additionally compares the downsampled SR image with the original LR image to ensure consistency. The calculation is almost the same as normal PSNR.

$$PSNR(\mathbf{y}_{SR}, \mathbf{y}_{LR}) = 10 \times \log_{10} \frac{L^2}{\frac{1}{N}\sum_{i=1}^{N} \left| downsample(\mathbf{y}_{SR}) - \mathbf{y}_{LR} \right|^2}. \tag{3.4}$$

This measurement complements the standard PSNR by ensuring that the generated image is based on the given condition, meaning the SR image should not deviate too much from the low-resolution one.

In the experiments, we use all four of these IQA metrics to measure the quality of the generated samples. Using a combination of these metrics allows for a comprehensive evaluation of the image quality, considering both pixel-level accuracy and perceptual similarity.

3.2 Experiments

The Swin Diffuser architecture depends on multiple hyper-parameters, such as shift step size, shift pattern, and kernel size. In the following sections, we conduct a series of experiments to find the most suitable parameters for image generation tasks with different scaling factors. We perform our experiments on dataset CelebA [LLWT15], a typical face attributes dataset containing over 202K face images with annotations. We used the pre-aligned version, which crops the image based on the given landmark. Under the same network architecture and optimizer settings, but with different parameters, we trained the models for 7 epochs and compared their results using the given metrics.

To keep the training procedure simple and efficient, we chose the 2x scaling SR task, where we downsample the original image to 64x64 as the ground truth and 32x32 as the low-resolution condition. The learning rate is set to $3e^{-4}$. We use a total of 100 denoising steps. The model is parallelly trained with PyTorch's Distributed Data Parallel (DDP) module on 2 NVIDIA RTX A4000 GPUs, each with 16 GB RAM. For a regular SwinDiffuser, it takes about 7 hours to converge.

3.2.1 Step Size

The step size is a crucial parameter in the SwinDiffuser as it ensures that information can flow among different diffusers, resulting in smoother transitions at the boundaries. The size of the step size controls the ratio of information exchange. When the step size is too small, information hardly flows around the patches, leading to artifacts. However, if the step size is too large, convergence becomes difficult, as most regions are constantly operated on by different diffusers, amplifying errors. Therefore, the step size plays a significant role in balancing the trade-off between reducing artifacts and maintaining inference stability.

In the experiments, we compare the step size based on the relative size of the patches. For 64x64 images, we choose step sizes of 0 (no shift), 4, 8, 12, and 16, corresponding to ratios of 0%, 12.5%, 25%, 37.5% and 50% . The results are shown in the Figure 3.2 and Table 3.1.

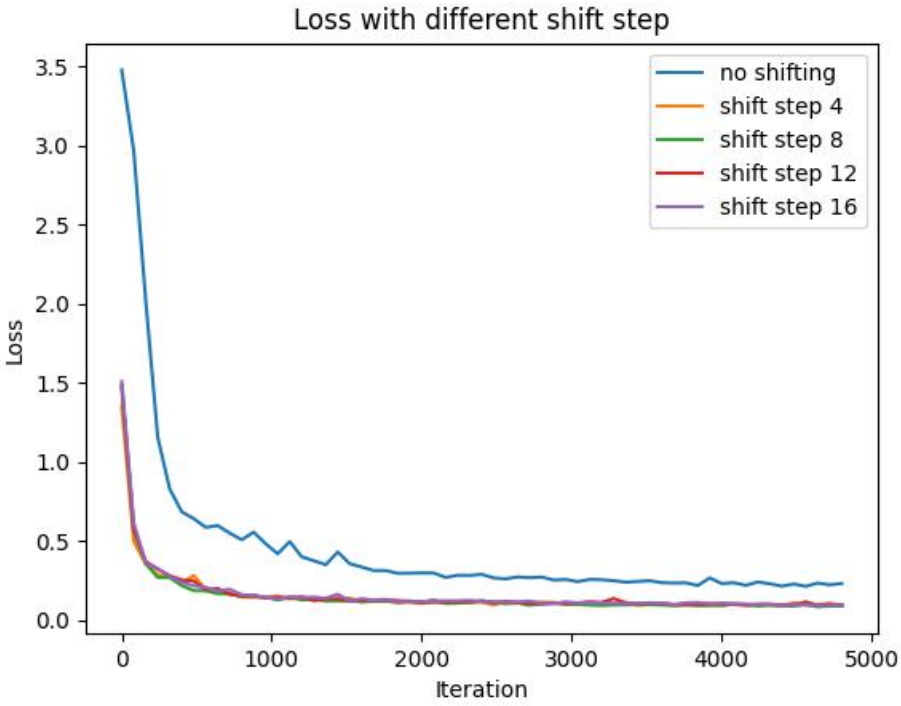

Figure 3.2: Training loss for different step sizes

As discussed in the previous chapter, the vanilla version (without any shifting) leads to sub-optimal results. This is evident from Figure 3.2, where the loss with 0 shift converges slower and with higher residual error compared to the others. Adding a small step size significantly improves convergence.

As listed in Table 3.1, a step size 8(ratio 25%) achieved the best result in both SSIM and LPIPS, balancing artifact reduction and inference stability. Although a step size of 16 achieved a relatively better PSNR score, it did not reconstruct structural details as well as the best one. Overall, the ratio 25% is the optimal choice, which we applied in the following experiments.

	PSNR ↑	SSIM ↑	LPIPS ↓	LR-PSNR ↑
no shift (0%)	25.773	0.854	0.029	33.664
step 4 (12.5%)	27.556	0.896	0.017	**34.764**
step 8 (25%)	28.179	**0.908**	**0.014**	34.441
step 12 (37.5%)	27.893	0.902	0.015	34.503
step 16 (50%)	**28.182**	0.906	0.015	34.578

Table 3.1: Metrics comparison of different shifting step size: the best result is indicated with bold font.

(a) LR condition **(b)** Diffusion Model Output **(c)** Final Result

Figure 3.3: Sample of Swin Diffuser step size 8

3.2.2 Shift Pattern

Another important factor is the shift pattern, which determines how the diffuser moves after each episode. In the previous chapter, we proposed two shift patterns: the circular shift with predefined step length and the random shift that draws step sizes from a uniform distribution. The circular shift ensures systematic coverage, while the random shift introduces variability, which can be beneficial in different scenarios. In this experiment, we fix all other parameters (step size set to 8, with the maximum for the random pattern clipped at 8) and vary only the shift pattern. We also include the non-shift version as a baseline. The results are shown in the Figure 3.4 and Table 3.2.

	PSNR ↑	SSIM ↑	LPIPS ↓	LR-PSNR ↑
no shift	25.773	0.854	0.029	33.664
random pattern	27.283	0.888	0.018	34.112
fix pattern	**28.179**	**0.908**	**0.014**	**34.441**

Table 3.2: Metrics comparison of different shift pattern: the best result is indicated with bold font.

As shown in Figure 3.4, both shift patterns perform better than the non-shift version, as expected.

In Table 3.2, the fixed pattern outperforms the random pattern in all criteria. This is reasonable since we use only 100 denoising steps. The fixed pattern, divided into four stages, ensures that all pixels are equally denoised during inference. In contrast, the random pattern may not cover all pixels equally, leading to worse performance with a short denoising step configuration. The situation might change with a longer denoising schedule, such as 1000 steps, where the random pattern could leverage stochasticity to produce smoother edges. For a smaller number of steps, like in our setting, the fixed pattern is a better choice.

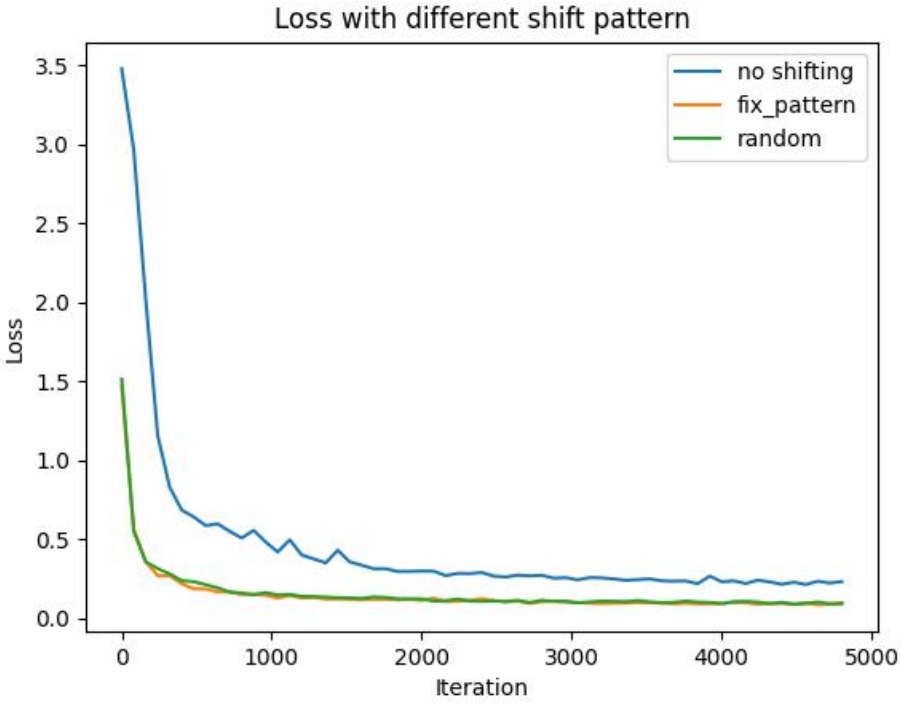

Figure 3.4: Training loss for different shifting pattern

3.2.3 Kernel Size

Next, we evaluate the size of the diffusion kernel, which directly relates to computational complexity. Choosing the appropriate kernel size is critical for balancing computational efficiency and image quality. The efficiency of the diffusion model scales quadratically with the size of the images. A smaller kernel size means the NN would be smaller and more efficient to calculate, allowing more patches to be generated and distributed for multi-GPU training. However, smaller patches mean restricted access to information and more difficult coordination, often leading to poorer results.

In the experiments, we tried two patch sizes: 16x16 and 32x32. Further experiments are restricted by the size of the images (64x64), as the patches still need to be downsampled in the U-Net. Training the model on a larger scale of the ground truth would be too expensive. The results are shown in Figure 3.5 and Table 3.3.

	PSNR ↑	**SSIM ↑**	**LPIPS ↓**	**LR-PSNR ↑**
16x16	24.811	0.827	0.0433	32.362
32x32	**28.179**	**0.908**	**0.014**	**34.441**

Table 3.3: Metrics comparison of different patch size: the best result is indicated with bold font.

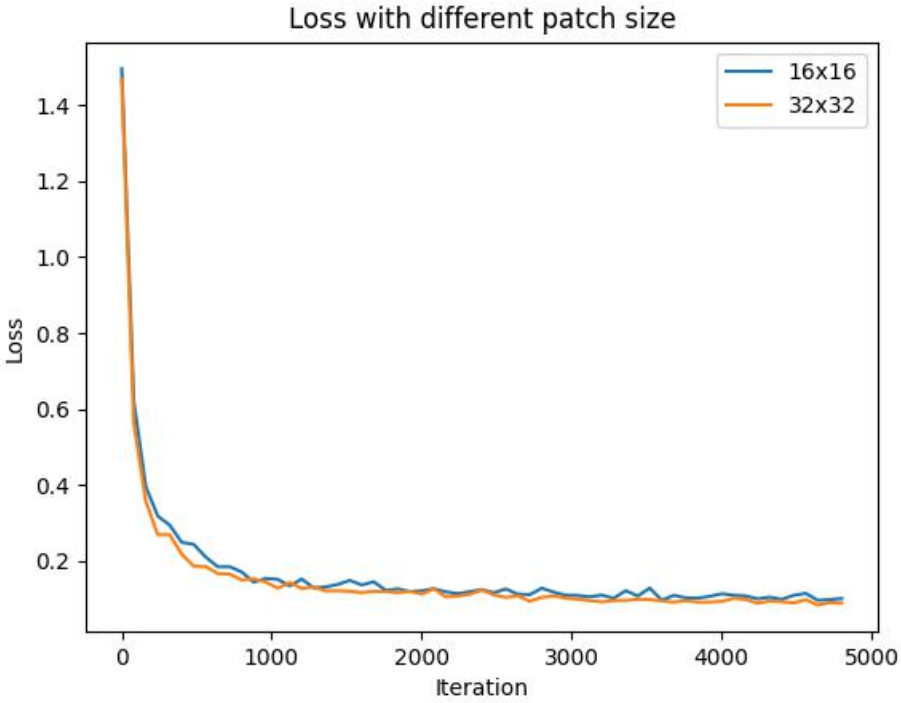

Figure 3.5: Training loss for different patch size

The results show that a larger kernel size indeed brings better results. In this case, the 16x16 kernel is too small to get a reasonable outcome, as the information it can access is insufficient. Additionally, due to the local behavior, it takes more epochs for the patches to converge, resulting in noisier behavior and more difficult convergence during training.

To conclude, the kernel size should be carefully chosen to balance efficiency and convergence speed in training. Larger kernels provide better access to contextual information, improving image quality and training stability, while smaller kernels enhance computational efficiency but may compromise on quality and convergence.

3.2.4 Time Usage

After completing the experiments on 64x64 images, we also measured the time consumed for inference. This helps determine the feasibility of the model for practical tasks. The results are shown in Table 3.4.

	1 GPU	2 GPUs
64x64	0.924	4.141

Table 3.4: Time usage of generating a single sample

It turns out that the SwinDiffuser makes the inference even four times slower than directly applying vanilla diffusion. The reason is that the patch size (32x32) is too small for a diffuser and the overhead of preprocessing and post-processing becomes significant, as both are not fully optimized for GPU calculation. This results in extra time to move them around and calculate on the CPU, preventing the device from utilizing its full capacity for the task.

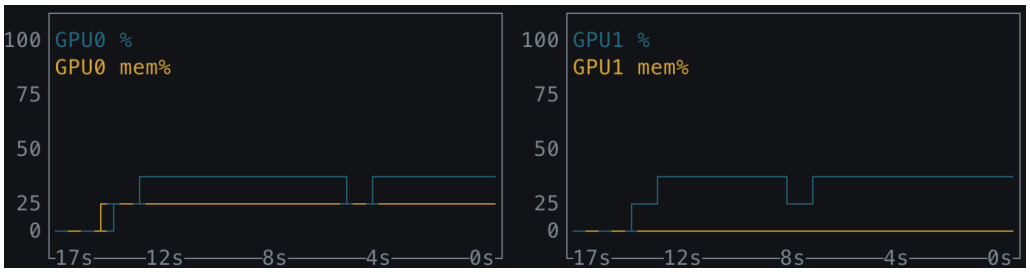

Figure 3.6: GPU usage by 64x64 inference experiment

As shown in Figure 3.6, when running the inference task, the GPU uses under 50% of its full capacity. In comparison, the vanilla version utilizes over 75%. The additional computational overhead makes the SwinDiffuser extremely slow in small-scale experiments. To truly observe the performance boost, we need to conduct larger-scale experiments.

3.2.5 Independent and identically distributed batch

Another point we noticed is that in the training pipeline, the samples in the patchified batch are no longer Independent and identically distributed (i.i.d.). This might negatively affect the training. So, we constructed another kind of i.i.d. batch and compared the results. We aim to understand the impact of batch composition on the training performance of the SwinDiffuser.

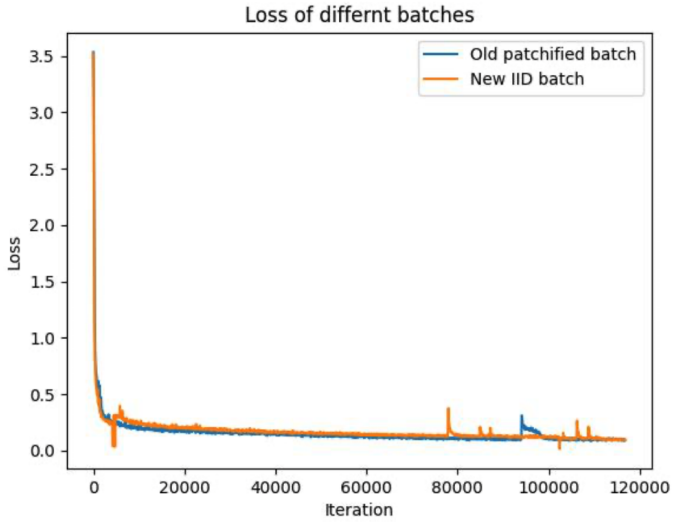

Figure 3.7: Experiment of i.i.d. batches

The experiment results shown in Figure 3.7 indicate that there is no difference between these two implementations This finding suggests that the non-i.i.d. nature of the patchified batches does not significantly impact the training performance.

3.3 Larger Scale

In this section, we scale up the experiments to provide a more comprehensive evaluation of the SwinDiffuser's performance and its ability to handle higher-resolution images. We use High Resolution (HR) images of size 128x128 and perform a 4x scale SR task. The configuration of the model, optimizer, and Swin trainer remains the same as in the 64x64 experiments. Initially, the results were not satisfactory, so we fine-tuned the hyper-parameters, such as the ratio of the Gaussian in the attention masks, experimented with different noise schedules (which did not significantly affect the results), and increased the number of layers in the RRDB blocks to make the conditional unit more expressive.

We used SRDiff[LYC+21] as the baseline for comparison, as it has a similar architecture utilizing the diffusion model to predict the residuals in SR tasks. The results are shown in Table 3.5 and Table 3.6,and illustrated in Figure 3.8.

Figure 3.8: Outcome of SwinDiffuser 128x128

	PSNR ↑	SSIM ↑	LPIPS ↓	LR-PSNR ↑
SwinDiffuser(Ours)	26.37	0.813	0.077	38.039
SRDiff(baseline)	**28.78**	**0.879**	**0.042**	**56.293**

Table 3.5: Metrics comparison of SwinDiffuser and baseline.

	1 GPU	2 GPUs
SwinDiffuser(Ours)	7.229	4.893
SRDiff(baseline)	5.59	–

Table 3.6: Time usage of generating a single sample

The results indicate that SRDiff outperforms SwinDiffuser. The additional structure of the SwinDiffuser complicates the neural network's learning process during training, resulting in longer convergence times. Despite the presence of the global feature extractor, local restrictions still affect generation quality, as shown in Figure 3.9, where the condition block RRDB does not capture long-distance correlations effectively. Consequently, the SwinDiffuser's performance is inferior to SRDiff.

Figure 3.9: Problem of SwinDiffuser: The woman has different eye colors due to the 32x32 diffuser processing the eyes independently without sufficient coordinating.

Regarding inference time, SRDiff is faster when using a single GPU due to less overhead from extra transformations (e.g., shifting, patchifying). However, this changes with multiple GPUs. Like other diffusion methods, SRDiff is not designed for parallel inference, whereas SwinDiffuser can easily achieve this. By doubling the number of devices, SwinDiffuser gains a 30% of performance boost. The performance gap is expected to widen with more devices until it reaches a saturation point where the overhead outweighs the benefits, as seen in the 64x64 example. This implies that for large, computationally complex tasks, SwinDiffuser can significantly accelerate the inference process.

The experiment demonstrates that SwinDiffuser strikes a balance between performance and faster inference. It allows the execution of complex models, previously requiring high-performance devices, on parallelly distributed edge devices with less computational capacity, achieving sub-optimal yet similar results.

4 Conclusion and Outlook

4.1 Conclusion

In this thesis, we proposed a novel architecture, the SwinDiffuser, designed to distribute the computational workload of diffusion models. This approach aims to reduce inference time and the computational demand for a single machine, thereby enabling the application of diffusion models in fields where high sampling frequency is crucial or where devices lack sufficient computational resources. The core idea involves dividing individual samples into smaller patches and distributing them across different machines to perform inference concurrently. To facilitate this, the SwinDiffuser architecture leverages shifting windows and a global feature extractor to mitigate generation artifacts and ensure seamless integration of patches processed independently. It overcomes the aforementioned problem and utilizes the unique structure to be able to distributed to parallel devices.

Our experiments, conducted on the CelebA dataset, demonstrated the effectiveness of the SwinDiffuser in handling SR tasks, By optimizing hyper-parameters, we achieved significant acceleration in the inference phase while maintaining results comparable to the baseline. Despite its promise, the SwinDiffuser still faces challenges, including suboptimal image quality and incomplete GPU optimization. The generated image quality is still not optimal and can be improved with a new global feature extractor. Additionally, the implementation is not fully optimized for GPUs: overhead can be reduced by migrating more computation from Central Proceccing Unit (CPU) to the GPU, where it can be done more efficiently. Incorporating more patch-wise simplification, as proposed in recent work by Liu et al.[LDP+24], which uses a confidence map to adaptively choose the denoising step based on reconstruction difficulties, might also enhance performance.

With these optimizations, the SwinDiffuser has the potential to achieve real-time performance, provided that computation is efficiently distributed and optimized. There are numerous possibilities for extending the SwinDiffuser model. For example, combining it with stable diffusion models that learn features in latent space could further enhance performance. In the following sections, we will discuss some interesting research directions that could pave the way for future projects.

4.2 Optimization Probabilities

4.2.1 Distillation

One promising approach for optimizing the SwinDiffuser is the application of model distillation. The technique of progressive distillation, as described by Salimans and Ho [SH22], involves using a pre-trained model (teacher) to train a smaller model (student) to achieve similar performance with reduced computational steps. In the context of diffusion models, the teacher and student share the

same architecture, with the student model learning to achieve comparable results in fewer steps. By halving the number of denoising steps iteratively, the inference process can be significantly accelerated, as shown in Figure 4.1

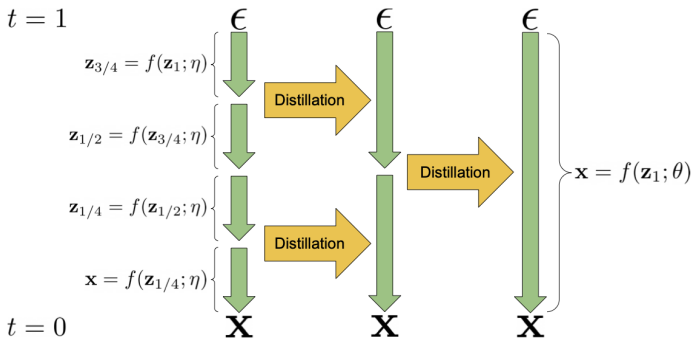

Figure 4.1: Overview of the progressive distillation

Since the computational overhead is caused by pre- and post-processing between denoise steps, he SwinDiffuser architecture is naturally suitable for this extension, making it a feasible and realistic approach to achieve real-time performance

4.2.2 Discrete wavelet transform

Another idea is inspired by the paper[MFR+23], which performs the diffusion operation in the frequency domain. It occurs to us that Swin diffusion is suitable to transfer in the wavelet domain. The image that is transferred through the Discrete wavelet transform (DWT) will automatically be divided into 4 separate pieces: LA (low frequency), LH (high frequency horizontal), LV (high frequency vertical), and LD (high frequency diagonal). In this case, no shifting is needed, as the different patches are independent. The LA part will serve as condition and LR reference. Three different diffusers will be responsible for reproducing the high-frequency details, which can be run in parallel on different machines, as shown in Figure 4.2.

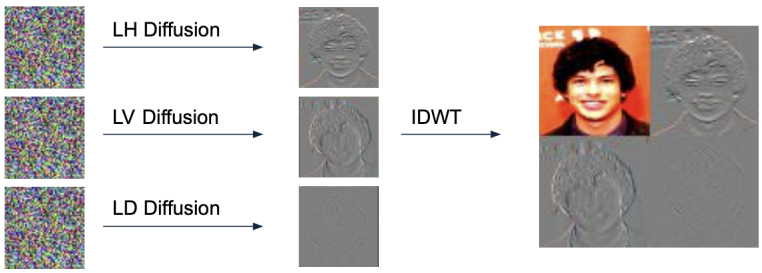

Figure 4.2: Overview of the DWT diffusion

This idea has also been tested during the project (figure 4.3). However, the overall structure still needs to be improved to overcome some issues. For example, the LD patch is difficult to converge and tends to be noisy in the inference, introducing noise to the final results. Introducing a hierarchical architecture to not only perform 2x scaling SR tasks but also incorporate a vanilla diffusion model to transform the task to image generation.

Figure 4.3: Outcome of the DWT diffusion: noise is introduced by the LD patches.

4.2.3 Vision Transformer as condition extractor

As already mentioned in chapter 3, the Swin Diffuser can still be improved in larger-scale tasks by equipping it with a new conditional block. The problem is that the Swin Diffuser needs to capture long-distance correlations when the patch size is small for computational efficiency. This is where the transformer excels.

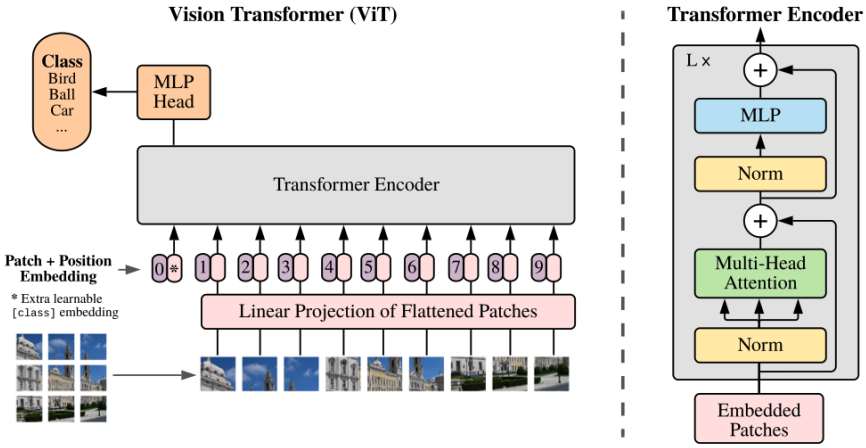

Figure 4.4: Overview of vision transformer: the original image is separated into smaller patches and fed into the transformer with embedding vectors.

The vision transformer includes the idea of dividing images into small patches and using key, value, and query matrices to explore the relationship between different blocks, which is exactly what we want. To extract the global representation, an additional input (classification token) is added to gather the information from all the patches. Only this output will be finally used as the feature for classification or any other downstream tasks. This is perfect for being the global condition extractor for the Swin Diffuser. Furthermore, some modifications can also be applied: not only the classification token but also the other outputs from the transformer can be selectively preserved. For example, the patch the particular diffuser is working on and its neighbors should be concatenated

to the final output as additional augmented local conditions. The vision transformer can be either pre-trained or trained simultaneously with the Swin Diffuser, which might probably achieve higher quality.

4.2.4 Putting together with Latent Diffusion

Combining the ideas of Swin and latent diffusion to further accelerate the inference. The basic idea is to bring the sample from the original space to the latent space with a smaller size to accelerate the generation. This transformation process is accomplished by a VAE. There might be some tricks we can apply to guide the VAE to learn the features in a way that can be better processed by the Swin Diffuser in the latent space.

Bibliography

[AJH+23] J. Austin, D. D. Johnson, J. Ho, D. Tarlow, R. van den Berg. *Structured Denoising Diffusion Models in Discrete State-Spaces*. 2023. arXiv: 2107.03006 [cs.LG] (cit. on p. 15).

[DBK+20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929 (cit. on p. 24).

[DN21] P. Dhariwal, A. Nichol. *Diffusion Models Beat GANs on Image Synthesis*. 2021. arXiv: 2105.05233 [cs.LG] (cit. on p. 15).

[HJA20] J. Ho, A. Jain, P. Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG] (cit. on p. 16).

[KPH+21] Z. Kong, W. Ping, J. Huang, K. Zhao, B. Catanzaro. *DiffWave: A Versatile Diffusion Model for Audio Synthesis*. 2021. arXiv: 2009.09761 [eess.AS] (cit. on p. 15).

[LDGT20] A. Lugmayr, M. Danelljan, L. V. Gool, R. Timofte. *SRFlow: Learning the Super-Resolution Space with Normalizing Flow*. 2020. arXiv: 2006.14200 [cs.CV] (cit. on p. 35).

[LDP+24] Y. Liu, H. Dong, J. Pan, Q. Dong, K. Chen, R. Zhang, X. Mei, L. Fu, F. Wang. *PatchScaler: An Efficient Patch-Independent Diffusion Model for Super-Resolution*. 2024. arXiv: 2405.17158 [cs.CV] (cit. on p. 43).

[LLC+21] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, B. Guo. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV] (cit. on pp. 26, 28).

[LLWT15] Z. Liu, P. Luo, X. Wang, X. Tang. *Deep Learning Face Attributes in the Wild*. 2015. arXiv: 1411.7766 [cs.CV] (cit. on p. 35).

[LYC+21] H. Li, Y. Yang, M. Chang, H. Feng, Z. Xu, Q. Li, Y. Chen. *SRDiff: Single Image Super-Resolution with Diffusion Probabilistic Models*. 2021. arXiv: 2104.14951 [cs.CV] (cit. on p. 41).

[LZL+24] Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, L. He, L. Sun. *Sora: A Review on Background, Technology, Limitations, and Opportunities of Large Vision Models*. 2024. arXiv: 2402.17177 [cs.LG] (cit. on p. 15).

[MFR+23] B. Moser, S. Frolov, F. Raue, S. Palacio, A. Dengel. *Waving Goodbye to Low-Res: A Diffusion-Wavelet Approach for Image Super-Resolution*. 2023. arXiv: 2304.01994 [cs.CV] (cit. on p. 44).

[RBL+22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2022. arXiv: 2112.10752 [cs.CV] (cit. on pp. 15, 20, 23).

[SE20] Y. Song, S. Ermon. *Generative Modeling by Estimating Gradients of the Data Distribution*. 2020. arXiv: 1907.05600 [cs.LG] (cit. on p. 18).

[SH22] T. Salimans, J. Ho. *Progressive Distillation for Fast Sampling of Diffusion Models*. 2022. arXiv: 2202.00512 [cs.LG] (cit. on pp. 15, 20, 23, 43).

[SSK+21] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, B. Poole. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG] (cit. on p. 19).

[SWMG15] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, S. Ganguli. *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. 2015. arXiv: 1503.03585 [cs.LG] (cit. on p. 16).

[SY23] Z. Sun, Y. Yang. *DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization*. 2023. arXiv: 2302.08224 [cs.LG] (cit. on p. 15).

[VKS+23] C. Vignac, I. Krawczuk, A. Siraudin, B. Wang, V. Cevher, P. Frossard. *DiGress: Discrete Denoising diffusion for graph generation*. 2023. arXiv: 2209.14734 [cs.LG] (cit. on p. 15).

[WBSS04] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861 (cit. on p. 34).

[WYW+18] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, Y. Qiao, X. Tang. *ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks*. 2018. arXiv: 1809.00219 [cs.CV] (cit. on p. 25).

[ZIE+18] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, O. Wang. *The Unreasonable Effectiveness of Deep Features as a Perceptual Metric*. 2018. arXiv: 1801.03924 [cs.CV] (cit. on p. 34).

All links were last followed on May 25, 2024.