Optimal Linear Subspace Search: Learning to Construct Fast and High-Quality Schedulers for Diffusion Models

Zhongjie Duan
East China Normal University
Shanghai, China
zjduan@stu.ecnu.edu.cn

Chengyu Wang
Alibaba Group
Hangzhou, China
chengyu.wcy@alibaba-inc.com

Cen Chen*
East China Normal University
Shanghai, China
cenchen@dase.ecnu.edu.cn

Jun Huang
Alibaba Group
Hangzhou, China
huangjun.hj@alibaba-inc.com

Weining Qian
East China Normal University
Shanghai, China
wnqian@dase.ecnu.edu.cn

ABSTRACT

In recent years, diffusion models have become the most popular and powerful methods in the field of image synthesis, even rivaling human artists in artistic creativity. However, the key issue currently limiting the application of diffusion models is its extremely slow generation process. Although several methods were proposed to speed up the generation process, there still exists a trade-off between efficiency and quality. In this paper, we first provide a detailed theoretical and empirical analysis of the generation process of the diffusion models based on schedulers. We transform the designing problem of schedulers into the determination of several parameters, and further transform the accelerated generation process into an expansion process of the linear subspace. Based on these analyses, we consequently propose a novel method called Optimal Linear Subspace Search (OLSS), which accelerates the generation process by searching for the optimal approximation process of the complete generation process in the linear subspaces spanned by latent variables. OLSS is able to generate high-quality images with a very small number of steps. To demonstrate the effectiveness of our method, we conduct extensive comparative experiments on open-source diffusion models. Experimental results show that with a given number of steps, OLSS can significantly improve the quality of generated images. Using an NVIDIA A100 GPU, we make it possible to generate a high-quality image by Stable Diffusion within only one second without other optimization techniques.

CCS CONCEPTS

• **Applied computing** → **Media arts**.

KEYWORDS

diffusion, computational efficiency, path optimization

*Corresponding author.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614999

ACM Reference Format:
Zhongjie Duan, Chengyu Wang, Cen Chen, Jun Huang, and Weining Qian. 2023. Optimal Linear Subspace Search: Learning to Construct Fast and High-Quality Schedulers for Diffusion Models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614999

1 INTRODUCTION

In recent years, diffusion models [9, 20, 35] have become the most popular framework and have achieved impressive success in image synthesis [5, 23, 28]. Unlike generative adversarial networks (GANs) [6], diffusion models generate high-quality images without relying on adversarial training processes, and do not require careful hyperparameter tuning. In addition to image synthesis, diffusion models have also been applied to image super-resolution [29], music synthesis [16] and video synthesis [4]. The research community and industry have witnessed the impressive effectiveness of diffusion models in generative tasks.

The major weakness of diffusion models, however, is the extremely slow sampling procedure, which limits their practicability [40]. The diffusion model is a family of iterative generative models and typically requires hundreds to thousands of steps to generate content. The idea of diffusion models is inspired by the diffusion process in physics. In image synthesis, various levels of Gaussian noise are incrementally added to images (or latent variables corresponding to images) in a stepwise manner, while the model is trained to denoise the samples and reconstruct the original images. The generation process is the reverse process of diffusion process and both processes include the same time discretization steps to ensure consistency. The number of steps in the training stage required is usually very large to improve the image quality, making the generation process extremely slow.

To tackle the efficiency problem of diffusion models, existing studies proposed several types of methods. For instance, Salimans et al. [31] proposed a distillation method that can reduce the number of sampling steps to half and can be applied repeatably to a model. By maximizing sample quality scores, DDSS [39] attempted to optimize fast samplers. However, these speedup methods require additional training, which makes it difficult to deploy them with limited computing resources. Another family of studies focused

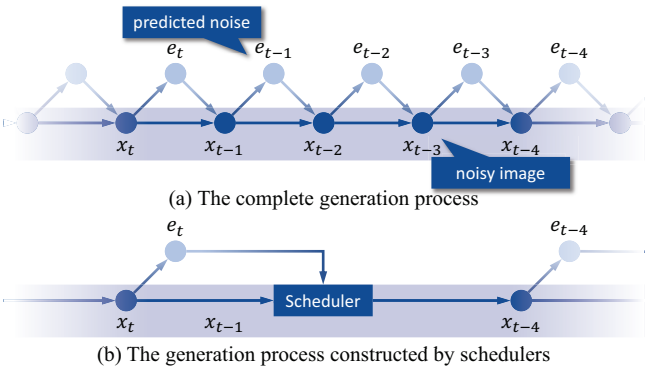


Figure 1: The complete generation process of diffusion models consists of hundreds of steps for gradual denoising. Diffusion schedulers speed up this process by skipping some steps but may make destructive changes to images.

on designing a scheduler to control the generation process, including DDIM [34], PNDM [17], DEIS [42], etc. As shown in Figure 1, these schedulers can reduce the number of steps without training. However, when we use a small number of steps, these methods may introduce new noise and make destructive changes to images because of the inconsistency between the reconstructed generation process and the complete generation process. Hence we are still faced with a trade-off between the computing resource requirement and performance.

In order to speed up the generation process, we focus on devising an adaptive scheduler that requires only a little time for learning. In this paper, we first theoretically analyze the generation process of diffusion models. With the given steps, most of the existing schedulers [13, 17, 34] generate the next sample in the linear subspace spanned by the previous samples and model outputs, thus the generation process is essentially the expansion process of linear subspaces. We also analyze the correlation of intermediate variables when generating images, and we observe that the model outputs contain redundant duplicate information at each step. Therefore, the number of steps must be reduced to prevent redundant calculation. In light of these analyses, we replace the coefficients in the iterative formula with trainable parameters to control the expansion of the linear subspaces, and then use simple least square methods [1, 14] to solve these parameters. Leveraging a path optimization algorithm, we further improve the performance by tuning the sampling steps. Path optimization and parameter solving can be performed within only a few minutes, thus it is easy to train and deploy. Our proposed scheduler, named Optimal Linear Subspace Search (OLSS), is designed to be lightweight and adaptive. We apply OLSS to several popular open-source diffusion models [26] and compare the performance of OLSS with the state-of-the-art schedulers. Experimental results prove that the approximate generation process built by OLSS is an accurate approximation of the complete generation process. The source code of our proposed method has been released on GitHub[1]. The main contribution of this paper includes:

[1]https://github.com/alibaba/EasyNLP/tree/master/diffusion/olss_scheduler

- We theoretically and empirically analyze the generation process of diffusion models and model it as an expansion process of linear subspaces. The analysis provides valuable insights for researchers to design new diffusion schedulers.
- Based on our analysis, we propose a novel scheduler, OLSS, which is capable of finding the optimal path to approximate the complete generation process and generating high-quality images with a very small number of steps.
- Using several popular diffusion models, we benchmark OLSS against existing schedulers. The experimental results demonstrate that OLSS achieves the highest image quality with the same number of steps.

2 RELATED WORK

2.1 Image Synthesis

Image synthesis is an important task and has been widely investigated. In the early years, GANs [11, 21, 24] are the most popular methods. By adversarial training a generator and a discriminator, we can obtain a network that can directly generate images. However, it is difficult to stabilize the training process [30]. Diffusion models overcome this issue by modeling the synthesis task as a Markovian diffusion process. Theoretically, diffusion models include Denoising Diffusion Probabilistic Models [33], Score-Based Generative Models [35], etc. In recent years, Latent Diffusion [26], a diffusion model architecture that denoises images in a latent space, became the most popular model architecture. Utilizing cross-attention [37] and classifier-free guidance [10], Latent Diffusion is able to generate semantically meaningful images according to given prompts. Leveraging large-scale text-image datasets [7, 32], diffusion models with billions of parameters have achieved impressive success in text-to-image synthesis. Diffusion models are proven to outperform GANs in image quality [3], and are even competitive with human artists [5, 23, 28]. However, the slow sampling procedure becomes a critical issue, limiting the practicability of diffusion models.

2.2 Efficient Sampling for Diffusion Models

The time consumed of generating an image using diffusion models is in direct proportion to the number of inference steps. To speed up the generation process, existing studies focus on reducing the number of inference steps. Specifically, some schedulers are proposed for controlling this denoising process. DDIM (Denoising Diffusion Implicit Models) [34] is a straightforward scheduler that converts the stochastic process to a deterministic process and skips some steps. Some numerical ODE algorithms [2, 13] are also introduced to improve efficiency. Liu et al. [17] pointed out that numerical ODE methods may introduce additional noise and are therefore less efficient than DDIM with only a small number of steps. To overcome this pitfall, they modified the iterative formula and improved their effectiveness. DEIS (Diffusion Exponential Integrator Sampler) [42], another study based on ODE, stabilizes the approximated generation process leveraging an exponential integrator and a semilinear structure. DPM-Solver [18] made further refinements by calculating a part of the solution analytically. Recently, an enhanced version [19] of DPM-Solver adopted thresholding methods and achieved state-of-the-art performance.

3 METHODOLOGY

3.1 Review of Diffusion Models

Different from GAN-based generative models, diffusion-based models require multi-step inference. The iterative generation process significantly increases the computation time. In the training stage, the number of steps may be very large. For example, the number of steps in Stable Diffusion [26] while training is 1000.

In the complete generation process, starting from random Gaussian noise $\boldsymbol{x}_T$, we need to calculate $\boldsymbol{x}_{T-1}, \ldots, \boldsymbol{x}_0$ step by step, where T is the total number of steps. At each step t, the diffusion model ϵ_θ takes $\boldsymbol{x}_t$ as input and output $\boldsymbol{e}_t = \epsilon_\theta(\boldsymbol{x}_t, t)$. We obtain $\boldsymbol{x}_{t-1}$ via:

$$\begin{aligned}\boldsymbol{x}_{t-1} =& \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\boldsymbol{e}_t}{\sqrt{\alpha_t}}\right)}_{\text{predicted } \boldsymbol{x}_0} \\ &+ \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2}\boldsymbol{e}_t}_{\text{direction pointing to } \boldsymbol{x}_t} + \underbrace{\sigma_t \boldsymbol{\epsilon}_t}_{\text{random noise}},\end{aligned} \tag{1}$$

where α_t, σ_t are hyper-parameters used for training. Note that $\sigma_t \boldsymbol{\epsilon}_t$ is the additional random noise to increase the diversity of generated results. In DDIM [34], σ_t is set to 0, making this process deterministic given $\boldsymbol{x}_T$.

To reduce the steps, in most existing schedulers, a few steps $t(1), \ldots, t(n)$ are selected as a sub-sequence of $\{T, T-1, \ldots, 0\}$, and the scheduler only calls the model to calculate $\boldsymbol{e}_{t(i)}$ at these n steps. For example, DDIM directly transfers Formula (1) to an n-step generation process:

$$\begin{aligned}\boldsymbol{x}_{t(i+1)} =& \sqrt{\alpha_{t(i+1)}}\left(\frac{\boldsymbol{x}_{t(i)} - \sqrt{1-\alpha_{t(i)}}\boldsymbol{e}_{t(i)}}{\sqrt{\alpha_{t(i)}}}\right) \\ &+ \sqrt{1-\alpha_{t(i+1)}}\boldsymbol{e}_{t(i)}.\end{aligned} \tag{2}$$

The final tensor $\boldsymbol{x}_0$ obtained by DDIM is an approximate result of that in the complete generation process. Another study [13] focuses on modeling the generation process as an ordinary differential equation (ODE) [2]. Consequently, forward Euler, a general numerical ODE algorithm, can be employed to calculate the numerical solution of $\boldsymbol{x}_0$:

$$\boldsymbol{x}_{t(i+1)} = \boldsymbol{x}_{t(i)} + \Big(t(i+1) - t(i)\Big)\frac{\mathrm{d}\boldsymbol{x}_{t(i)}}{\mathrm{d}t(i)}, \tag{3}$$

where

$$\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t} = -\frac{\mathrm{d}\alpha_t}{\mathrm{d}t}\left(\frac{\boldsymbol{x}_t}{2\alpha_t} - \frac{\boldsymbol{e}_t}{2\alpha_t\sqrt{1-\alpha_t}}\right). \tag{4}$$

PNDM [17] is another ODE-based scheduler. It leverages Linear Multi-Step Method and constructs a pseudo-numerical method. We simplify the iterative formula of PNDM as:

$$\boldsymbol{x}_{t(i+1)} = \frac{\sqrt{\alpha_{t(i+1)}}}{\sqrt{\alpha_{t(i)}}}\boldsymbol{x}_{t(i)} - \frac{1}{\sqrt{\alpha_{t(i)}}}\alpha'_{t(i)}\boldsymbol{e}'_{t(i)}, \tag{5}$$

where

$$\boldsymbol{e}'_{t(i)} = \frac{1}{24}(55\boldsymbol{e}_{t(i)} - 59\boldsymbol{e}_{t(i-1)} + 37\boldsymbol{e}_{t(i-2)} - 9\boldsymbol{e}_{t(i-3)}), \tag{6}$$

$$\alpha'_{t(i)} = \frac{\alpha_{t(i+1)} - \alpha_{t(i)}}{\sqrt{(1-\alpha_{t(i+1)})\alpha_{t(i)}} + \sqrt{(1-\alpha_{t(i)})\alpha_{t(i+1)}}}. \tag{7}$$

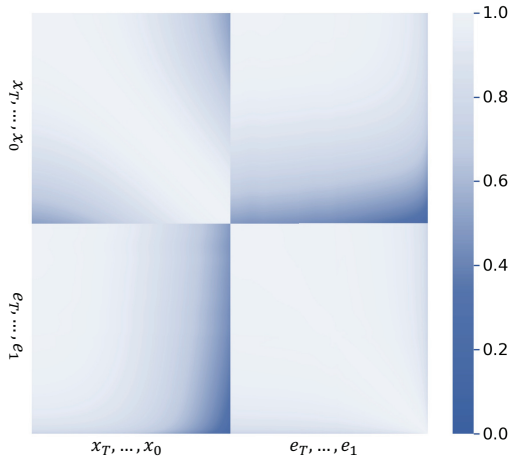


Figure 2: The heat map of the correlation coefficients of latent variables, which records the whole generation process.

In DDIM (2) and forward Euler (3), $\boldsymbol{x}_{t(i+1)}$ is a linear combination of $\{\boldsymbol{x}_{t(i)}, \boldsymbol{e}_{t(i)}\}$. In PNDM (5), $\boldsymbol{x}_{t(i+1)}$ is a linear combination of $\{\boldsymbol{x}_{t(i)}, \boldsymbol{e}_{t(i)}, \boldsymbol{e}_{t(i-1)}, \boldsymbol{e}_{t(i-2)}, \boldsymbol{e}_{t(i-3)}\}$. Generally, all these schedulers satisfy

$$\boldsymbol{x}_{t(i+1)} \in \mathrm{span}\{\boldsymbol{x}_{t(i)}, \boldsymbol{e}_{t(1)}, \ldots, \boldsymbol{e}_{t(i)}\}. \tag{8}$$

Recursively, we can easily prove that

$$\boldsymbol{x}_{t(i+1)} \in \mathrm{span}\{\boldsymbol{x}_{t(1)}, \boldsymbol{e}_{t(1)}, \ldots, \boldsymbol{e}_{t(i)}\}. \tag{9}$$

Therefore, the generation process is the *expansion process of a linear subspace*. This linear subspace is spanned by the initial Gaussian noise and the previous model outputs. At each step, we obtain the model output and add it to the vector set. The core issue of designing a scheduler is to determine the coefficients in the iterative formula. The number of non-zero coefficients does not exceed $\frac{1}{2}n^2 + \frac{3}{2}n$.

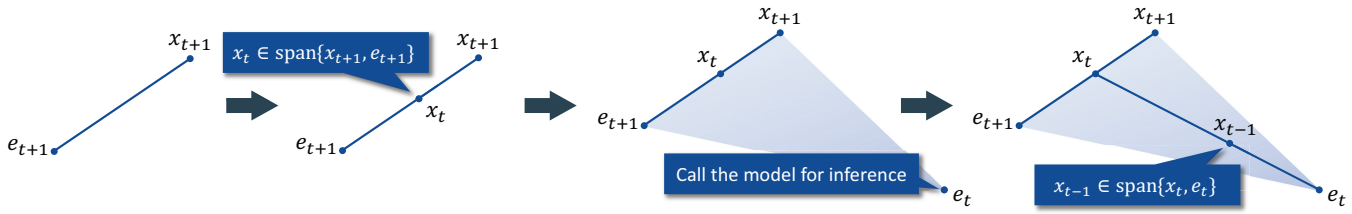
3.2 Empirical Analysis of Generation Process

To empirically analyze what happens in the whole generation process, we use Stable Diffusion to generate several images and store the latent variables, including $\boldsymbol{x}_T, \ldots, \boldsymbol{x}_0, \boldsymbol{e}_T, \ldots, \boldsymbol{e}_1$. As shown in Figure 2, we plot a heat map showing the correlation coefficients between these variables. We have the following findings:

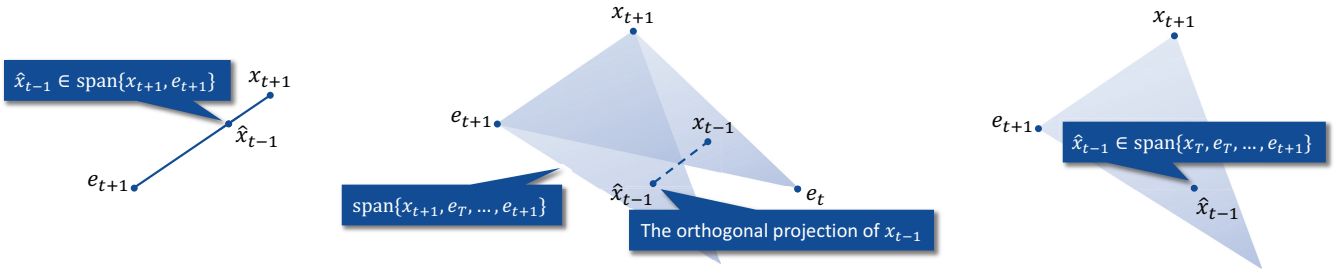
(1) $\boldsymbol{x}_t$ is similar to that in neighboring steps and differs from that in non-neighboring steps. As the denoising process proceeds, $\boldsymbol{x}_t$ is updated continuously.
(2) The correlation between $\boldsymbol{x}_t$ and the predicted noise $\boldsymbol{e}_t$ is strong in the beginning and becomes weak in the end. The reason is that $\boldsymbol{x}_t$ consists of little noise in the last few steps.
(3) The correlation between $\boldsymbol{e}_T, \ldots, \boldsymbol{e}_1$ is significantly strong, indicating that the outputs of the model contain redundant duplicate information.

3.3 Constructing a Faster Scheduler

On the basis of existing schedulers and our analysis, we propose OLSS, a new diffusion scheduler. In this method, we first run the complete generation process to catch the intermediate variables



(a) The complete generation process of DDIM form x_{t+1} to x_{t-1}



(b) The generation process of DDIM form x_{t+1} to x_{t-1}, skipping x_t

(c) The parameter-solving process of OLSS

(d) The generation process of OLSS form x_{t+1} to x_{t-1}, skipping x_t

Figure 3: An interpretation of OLSS. (a) In the complete generation process of DDIM from x_{t+1} to x_{t-1}, the scheduler first computes $x_t \in \text{span}\{x_{t+1}, e_{t+1}\}$ and then computes $x_{t-1} \in \text{span}\{x_t, e_t\}$. We have $x_{t-1} \in \text{span}\{x_{t+1}, e_{t+1}, e_t\}$. (b) If we use DDIM to skip x_t, the scheduler will compute $\hat{x}_{t-1} \in \text{span}\{x_{t+1}, e_{t+1}\}$ using the iterative Formula (2). (c) If we use OLSS to skip x_t, we use the orthogonal projection of x_{t-1} in $\text{span}\{x_{t+1}, e_T, \dots, e_{t+1}\}$ as the estimation of x_{t-1}. In the parameter-solving process, we compute the orthogonal projection matrix and construct the new iterative formula. (d) The linear subspace spanned by $\{x_{t+1}, e_T, \dots, e_{t+1}\}$ is equivalent to the linear subspace spanned by $\{x_T, e_T, \dots, e_{t+1}\}$. In the generation process of OLSS, we directly compute $\hat{x}_{t-1} \in \text{span}\{x_T, e_T, \dots, e_{t+1}\}$. Compared with DDIM, the estimation of x_{t-1} is in a higher dimensional linear subspace, thus it is more accurate.

and then construct an approximate process using these variables, instead of leveraging mathematical theories to design a new iterative formula.

Assume that we have selected n steps $\{t(1), \dots, t(n)\}$, which is a sub-sequence of $\{T, T-1, \dots, 0\}$ with $t(1) = T$. Calling the diffusion model for inference is only allowed at these n steps. At the i-th step, we have obtained intermediate variables $\boldsymbol{x}_{t(1)}, \dots, \boldsymbol{x}_{t(i)}$ and $\boldsymbol{e}_{t(1)}, \dots, \boldsymbol{e}_{t(i)}$. In the complete generation process, it needs to call the model for $t(i+1)-t(i)-1$ times. To reduce time consumption, we naturally come up with a naive method. As we mentioned in Section 3.2, the correlation between $\boldsymbol{e}_T, \dots, \boldsymbol{e}_1$ is significantly strong, thus we can estimate the model's output $\boldsymbol{e}_{t(i)-1}$ using a simple linear model trained with intermediate variables. Formally, let

$$\hat{\boldsymbol{e}}_{t(i)-1} = \arg\min_{\boldsymbol{e} \in \mathcal{E}} ||\boldsymbol{e} - \boldsymbol{e}_{t(i)-1}||_2^2, \tag{10}$$

where the feasible region is

$$\mathcal{E} = \text{span}\{\boldsymbol{x}_{t(1)}, \boldsymbol{e}_{t(1)}, \dots, \boldsymbol{e}_{t(i)}\}. \tag{11}$$

In other words, we use the orthogonal projection of $\boldsymbol{e}_{t(i)-1}$ in $\mathcal{E}$ as the estimation of $\boldsymbol{e}_{t(i)-1}$. Similarly, we can obtain all estimated intermediate variables in the missing steps before $t(i+1)$ and finally calculate $\hat{\boldsymbol{x}}_{t(i+1)}$. According to Equations (9, 10, 11), it is obvious to see the estimated $\boldsymbol{x}_{t(i+1)}$ still satisfies Equation (9). However, note that the estimation may have non-negligible errors and the errors are accumulated into subsequent steps. To address this issue, we design a simplified end-to-end method that directly estimates $\boldsymbol{x}_{t(i+1)}$. The simplified method only contains one linear model, containing $i+1$ coefficients $w_{i,0}, w_{i,1}, \dots, w_{i,i}$. The estimated $\boldsymbol{x}_{t(i+1)}$ is formulated as:

$$\hat{\boldsymbol{x}}_{t(i+1)} = w_{i,0}\boldsymbol{x}_{t(1)} + \sum_{j=1}^{i} w_{i,j}\boldsymbol{e}_{t(j)}. \tag{12}$$

The decision space in this simplified method is consistent with the naive method mentioned above. Leveraging least square methods [1], we can easily minimize the mean square error $||\hat{\boldsymbol{x}}_{t(i+1)} - \boldsymbol{x}_{t(i+1)}||_2^2$. Note that we use only $\boldsymbol{x}_{t(1)}$ (i.e., $\boldsymbol{x}_T$) instead of all intermediate variables $\{\boldsymbol{x}_{t(1)}, \dots, \boldsymbol{x}_{t(i)}\}$ in Equation (12), because $\{\boldsymbol{x}_{t(1)}, \boldsymbol{e}_{t(1)}, \dots, \boldsymbol{e}_{t(i)}\}$ is a linearly independent set and the other vectors $\{\boldsymbol{x}_{t(2)}, \dots, \boldsymbol{x}_{t(i)}\}$ can be linearly represented by these vectors. Additionally, the linear independence makes it easier to solve the least squares problem using QR-decomposition algorithms [14], which is faster and more numerically stable than directly computing the pseudo-inverse matrix [25].

We provide an interpretation of OLSS in Figure 3. When we skip $\boldsymbol{x}_t$, OLSS computes an estimation of $\boldsymbol{x}_{t-1}$ in the linear subspace spanned by the initial Gaussian noise and the previous model outputs. Essentially, this estimation is the orthogonal projection of $\boldsymbol{x}_{t-1}$ in the linear subspace. Each time we call the model for inference, the linear subspace is expanded by the predicted noise. Therefore, the generation process is the expansion process of the linear subspace.

Algorithm 1 Step Searching with error upper bound D

Input: Error upper bound D
Input: Previous steps $\{t(1), \dots, t(i)\}$
$t_l = t(i), t_r = 0$
while $t_l > t_r$ **do**
  $t_m = \lfloor \frac{t_l + t_r}{2} \rfloor$
  if $d(t(1), \dots, t(i), t_m) > D$ **then**
    $t_r = t_m + 1$
  else
    $t_l = t_m$
  end if
end while
$t(i+1) = t_l$
return $t(i+1)$

Algorithm 2 Path searching with error upper bound D

Input: Error upper bound D
$t(1) = T$
for $i = 1, 2, \dots, n$ **do**
  Find $t(i+1)$ using Algorithm 1
  if $t(i+1)$ does not exist **then**
    return None
  end if
end for
if $t(n+1) > 0$ **then**
  return None
else
  return $\{t(1), t(2), \dots, t(n)\}$
end if

Algorithm 3 Path optimization

Input: The required absolute error ϵ of optimal error limit
$D_l = 0$
$D_r = 10$ (a sufficiently large value)
while $D_r - D_l > \epsilon$ **do**
  $D_m = \frac{t_l + t_r}{2}$
  Find a path $\mathcal{T}$ with error limit D_m using Algorithm 2
  if $\mathcal{T}$ is not None **then**
    $D_r = D_m$
  else
    $D_l = D_m$
  end if
end while
Find a path $\mathcal{T}$ with error limit D_r
return $\mathcal{T}$

3.4 Searching for the Optimal Path

Another problem to be addressed is how to select the n steps $\{t(1), \dots, t(n)\}$. In Section 3.2, we observe that the correlation between $\boldsymbol{x}_t$ and $\boldsymbol{e}_t$ is different at each step, indicating that the difficulty level of generating low-error $\boldsymbol{x}_t$ is also varied. In most existing schedulers, these steps are selected in $\{T, T-1, \dots, 0\}$ with equal intervals. For example, in PNDM, Equation (6) comes from the Linear Multi-Step Method, which enforces that the steps must be uniformly selected. However, there is no restriction on the selection of steps in our method. We can search for the optimal path to generate high-quality images.

For convenience, we use $d(t(1), \dots, t(i+1))$ to denote the distance from $\boldsymbol{x}_{t(i+1)}$ to its orthogonal projection in the linear subspace spanned by $\{\boldsymbol{x}_{t(1)}, \boldsymbol{e}_{t(1)}, \dots, \boldsymbol{e}_{t(i)}\}$ (i.e., the error of $\hat{\boldsymbol{x}}_{t(i+1)}$). We add an additional step $t(n+1) = 0$. To find the optimal path $\mathcal{T} = \{t(1), \dots, t(n+1)\}$, we formulate the path optimization problem as:

$$\mathcal{T} = \underset{t(1),\dots,t(n+1)}{\arg\min} \max_{i=1}^{n} d(t(1), \dots, t(i+1)), \tag{13}$$

$$\text{s.t. } T = t(1) \geq t(2) \geq \cdots \geq t(n) \geq t(n+1) = 0. \tag{14}$$

We intend to minimize the largest error in the n steps. Setting an error upper bound D, we hope the error of every step does not exceed D. Such a path always exists if D is sufficiently large, thus we can use a binary search algorithm to compute the minimal error upper bound D when a path exists. The pseudo-code is presented in Algorithm 3. In this binary search algorithm, we have to design another algorithm to check whether the path with the error upper bound D exists.

According to the conclusions in Section 3.2, the more steps we skip, the larger errors we have. However, if we only skip a small number of steps to reduce the error, the path will not end at 0 within n steps. Therefore, we use another binary search algorithm to search for the next step based on a greedy strategy. By skipping more steps as possible, we can find a path with the error upper bound D if it exists. The pseudo-code of finding the next step with error limitation is presented in Algorithm 1, and the pseudo-code of finding the path is presented in Algorithm 2.

The whole path optimization algorithm includes three loops. From inner to outer, the pseudo-codes of the three loops are presented in Algorithm 1-3. The first one is to find the next step with an error limitation. The second one is to check if such a path exists. Algorithm 2 will return the path to Algorithm 3 if it exists. The third one is to find the minimal error upper bound. Leveraging the whole path optimization algorithm, we obtain the optimal path for constructing the generation process.

3.5 Efficiency Analysis

We analyze the time complexity of constructing an OLSS scheduler. The most time-consuming component is solving the least square problem. We need to solve the least square problem $O(n)$ times if the path is fixed, and $O(n \log \frac{1}{\epsilon} \log T)$ times to perform path optimization, where ϵ is the absolute error of optimal D. Empirically, when we use OLSS to improve the efficiency of Stable Diffusion, usually a few minutes of computation on the CPU is sufficient to compute the optimal path and solve all the parameters $\{w_{i,j}\}$ after all the required intermediate variables are collected from the complete generation process. In the inference phase, the computation on schedulers is negligible compared to the computation on the models. The total time consumed of generating an image is in direct proportion to the number of steps.

4 EXPERIMENTS

To demonstrate the effectiveness of OLSS, we conduct comparative experiments. We further investigate the factors that affect the quality of images generated by OLSS.

4.1 Comparison of Diffusion Schedulers

We compare OLSS with 7 baseline schedulers, including its variant OLSS-P and 6 existing schedulers. 1) **OLSS-P**: A variant of our proposed OLSS that selects the steps $\{t(1), \ldots, t(n)\}$ uniformly instead of using the path optimization algorithm. 2) **DDIM** [34]: A straightforward method that directly skips some steps in the generation process. It has been widely used in many diffusion models. 3) **Euler** [13]: An algorithm of calculating ODEs' numerical solution. We follow the implementation of k-diffusion[2] and use the ancestral sampling version. 4) **PNDM** [17]: An pseudo numerical method that improves the performance of Linear Multi-Step method. 5) **DEIS** [42]: A fast high-order solver designed for diffusion models. It consists of an exponential integrator and a semi-linear structure. 6) **DPM-Solver** [18]: A high-order method that analytically computes the linear part of the latent variables. 7) **DPM-Solver++** [19]: An enhanced scheduler based on DPM-Solver. It solves ODE with a data prediction model and uses thresholding methods. DPM-Solver++ is the most recent state-of-the-art scheduler.

4.1.1 Experimental Settings. The comparative experiments consist of two parts. The first part is to benchmark the speed-up effect of these schedulers on open-domain image synthesis and analyze the relationship between different schedulers. We compare the schedulers on two popular large-scale diffusion models in the research community, including Stable Diffusion[3] and Stable Diffusion 2[4]. The architecture of both models consists of a CLIP-based text encoder [22], a U-Net [27], and a VAE [15], where the U-Net is trained to capture the pattern of noise. The implementation of baseline schedulers is mainly based on Diffusers [38]. We randomly sample 1000 prompts in LAION-Aesthetics V2[5] as the conditional information input to models. The guidance scale is set to 7.0. The second part is to further investigate the effect of these schedulers on close-domain image synthesis. We fine-tune Stable Diffusion on CelebA-HQ [12] (256×256) and LSUN-Church [41] (256×256) for 5000 steps respectively. CelebA-HQ is a high-quality version of CelebA, which is a human face dataset. LSUN-Church is a part of LSUN and includes photos of churches. The training and generating process is performed without textual conditional information. We generate images using the fine-tuned model and compare them with real-world images in each dataset. In both two parts of the experiments, in order to avoid the influence of random noise on the experimental results, we use the same random seed and the same pre-generated $\boldsymbol{x}_T$ for every scheduler. For DPM-Solver and DPM-Solver++, we use their multi-step version to bring out their best generative effects. For OLSS, we run the complete generation process to generate 32 images and then let our algorithm construct the approximate process.

4.1.2 Evaluation Metrics. We compare the quality of generated images by these methods with the same number of steps. Note that the time consumed by each scheduler is different even if the number of steps is the same, but we cannot measure it accurately because it is negligible compared to the time consumed on the model. We use FID (Frechet Inception Distance) [8], i.e., the Frechet Distance of features extracted by Inception V3 [36], to measure the similarity between two sets of images. A smaller FID indicates that the distributions of the two sets of images are more similar. In the first part, For each scheduler, we run the generation program with 5, 10, and 20 steps respectively. Considering that the generation process with fewer steps is an approximate process of the complete generation process, we compare each one with the complete generation process (i.e., DDIM, 1000 steps). Additionally, we also compare each one with the 100-step schedulers to investigate the consistency. In the second part, we compute the FID scores between 10,000 generated images and real-world images. The experimental results of the two parts are shown in Table 1 and Table 2.

4.1.3 Experimental Results. In Table 1, we can clearly see that OLSS reaches the best performance with the same steps. The FID between images generated by OLSS and those by 1000-step DDIM is lower than other schedulers, and the gap is significant when we use only 5 steps. Considering the consistency of different schedulers, we observe that most schedulers generate similar images with the same settings except Euler. The FID of Euler is even larger than that of DDIM. This is because Euler's method computes the numerical solution of $\boldsymbol{x}_0$ iteratively along a straight line [17], making the solution far from the origin $\boldsymbol{x}_0$. PNDM, another ODE-based scheduler, overcomes this pitfall by adjusting the linear transfer part in the iterative formula. Comprehensively, DPM-Solver++ performs the best among all baseline methods but still cannot outperform our method. Comparing OLSS and OLSS-P, we find that OLSS performs better than OLSS-P. It indicates that the path optimization algorithm further improves the performance.

In Table 2, the FID scores of OLSS are also the lowest. Even without the path optimization algorithm, OLSS-P can still outperform other schedulers. The gap between OLSS and other schedulers is more significant than that in Table 1, indicating that OLSS is more effective in generating images with a similar style. The main reason is that the generation process of images in a close domain follows a domain-specific pattern, and the learnable parameters in OLSS make it more suitable for the generation task. Additionally, the FID of PNDM is lower than DDIM, which is different from the first part of the experiments. We suspect that PNDM constructs a new generation pattern to generate realistic images rather than constructing an approximate process of the original process.

4.2 Efficiency Study

When a scheduler is applied to a diffusion model, there is a trade-off between efficiency and quality. Fewer steps can reduce the inference time but usually results in lower image quality. With the same settings as above, we apply OLSS to Stable Diffusion and calculate the FID scores with varying numbers of steps. We run the program on an NVIDIA A100 GPU and record the average time of generating an image. The results are plotted in Figure 4. As the number of steps increases, the FID score decreases rapidly at the

[2] https://github.com/crowsonkb/k-diffusion
[3] https://huggingface.co/CompVis/stable-diffusion-v1-4
[4] https://huggingface.co/stabilityai/stable-diffusion-2-1
[5] https://laion.ai/blog/laion-aesthetics

Table 1: The FID ↓ scores between diffusion schedulers with different steps. The best results are in bold, and the second best are underscored. †: the generation result of 1000-step DDIM is the most high-quality irrespective of the computational efficiency.

Model	Steps / Scheduler		100 Steps						1000 Steps†
			DDIM	Euler	PNDM	DEIS	DPM -Solver	DPM -Solver++	DDIM
Stable Diffusion (512 × 512)	5 Steps	DDIM	82.62	78.14	84.40	84.91	85.00	85.24	84.55
		Euler	132.19	111.18	134.62	135.06	135.28	135.72	134.28
		PNDM	75.53	89.53	74.29	75.50	75.31	75.13	75.41
		DEIS	56.31	**55.89**	58.44	58.03	58.21	58.54	57.68
		DPM-Solver	55.39	55.94	57.49	57.09	57.27	57.59	56.76
		DPM-Solver++	54.78	56.11	56.82	56.41	56.59	56.91	56.08
		OLSS-P	48.67	61.27	48.62	48.80	48.77	48.72	48.94
		OLSS	**48.34**	61.11	**48.40**	**48.61**	**48.60**	**48.57**	**48.79**
	10 Steps	DDIM	52.52	59.87	53.71	54.37	54.46	54.66	54.13
		Euler	79.63	60.04	81.97	81.70	81.91	82.33	81.32
		PNDM	55.77	72.31	54.84	56.01	55.88	55.77	56.03
		DEIS	39.11	**55.07**	41.75	40.96	41.13	41.40	40.97
		DPM-Solver	38.93	55.33	41.53	40.75	40.92	41.17	40.73
		DPM-Solver++	38.42	55.72	41.06	40.17	40.33	40.59	40.02
		OLSS-P	37.33	59.28	38.64	38.36	38.36	38.38	38.60
		OLSS	**36.46**	58.09	**37.68**	**37.66**	**37.66**	**37.67**	**37.79**
	20 Steps	DDIM	38.04	57.14	39.07	40.90	40.99	41.17	41.13
		Euler	65.38	**49.92**	67.18	66.85	67.05	67.39	66.32
		PNDM	37.26	60.57	35.55	38.38	38.39	38.42	38.55
		DEIS	23.37	58.88	28.64	25.87	25.91	25.99	26.28
		DPM-Solver	24.38	60.03	28.73	26.16	26.12	26.08	26.72
		DPM-Solver++	26.21	61.43	29.61	27.31	27.20	27.08	27.94
		OLSS-P	22.91	58.96	28.20	25.02	25.04	25.11	25.42
		OLSS	**20.78**	58.71	**27.26**	**22.98**	**23.04**	**23.09**	**23.23**
Stable Diffusion 2 (768 × 768)	5 Steps	DDIM	93.50	80.83	95.12	98.10	98.16	98.38	97.62
		Euler	170.79	119.12	173.09	175.24	175.28	175.51	174.43
		PNDM	105.08	99.88	105.82	108.61	108.54	108.67	108.28
		DEIS	55.88	**58.62**	57.30	59.04	59.20	59.50	58.65
		DPM-Solver	54.90	58.77	56.31	58.02	58.18	58.48	57.63
		DPM-Solver++	54.25	59.13	55.67	57.30	57.45	57.75	56.91
		OLSS-P	50.03	65.20	51.38	51.95	51.87	51.83	51.95
		OLSS	**49.18**	64.89	**50.64**	**51.17**	**51.11**	**51.07**	**51.11**
	10 Steps	DDIM	55.32	62.05	56.01	58.74	58.83	59.03	58.47
		Euler	96.86	61.75	98.57	100.88	100.99	101.26	100.15
		PNDM	56.44	64.58	56.94	59.52	59.62	59.81	59.28
		DEIS	39.83	**60.85**	42.11	42.59	42.74	43.02	42.33
		DPM-Solver	39.70	61.22	41.93	42.41	42.56	42.81	42.17
		DPM-Solver++	39.40	61.81	41.62	41.89	42.04	42.30	41.66
		OLSS-P	38.39	64.21	40.46	40.70	40.70	40.74	40.63
		OLSS	**38.10**	63.81	**40.15**	**40.46**	**40.48**	**40.54**	**40.45**
	20 Steps	DDIM	40.19	61.44	40.83	44.12	44.25	44.42	44.03
		Euler	74.53	**48.51**	76.08	77.80	77.94	78.18	77.24
		PNDM	40.93	62.20	40.26	44.19	44.35	44.61	43.97
		DEIS	27.27	64.34	32.74	30.25	30.29	30.39	30.53
		DPM-Solver	28.18	65.52	33.47	30.44	30.40	30.38	30.82
		DPM-Solver++	29.81	67.08	34.66	31.34	31.24	31.15	31.88
		OLSS-P	26.74	63.82	**32.17**	30.27	30.31	30.40	30.59
		OLSS	**26.44**	63.64	**32.17**	**29.98**	**30.01**	**30.12**	**30.27**

Table 2: The FID ↓ scores of diffusion schedulers with different steps on two real-world datasets. The underlying diffusion model is fine-tuned for 5,000 steps.

Dataset	Scheduler	Steps	
		5 Steps	10 Steps
CelebA-HQ	DDIM	80.55	31.72
	Euler	83.80	35.85
	PNDM	57.63	25.33
	DEIS	24.12	11.68
	DPM-Solver	23.03	11.68
	DPM-Solver++	21.82	11.44
	OLSS-P	14.24	11.40
	OLSS	**11.65**	**11.37**
LSUN-Church	DDIM	109.57	38.55
	Euler	216.07	84.21
	PNDM	29.55	14.58
	DEIS	55.27	13.67
	DPM-Solver	51.66	12.99
	DPM-Solver++	48.93	11.77
	OLSS-P	19.46	11.10
	OLSS	**15.18**	**10.21**

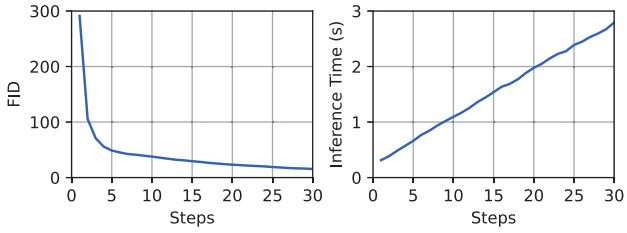


Figure 4: The FID scores (between images generated by our method and the complete generation process) and inference time of OLSS.

beginning and gradually converges. The inference time increases almost linearly with the number of steps. Setting the number of steps to 5, OLSS is able to generate an image within only *one second*, while still achieving satisfactory quality.

4.3 Visualization

To intuitively see how diffusion models generate images with different schedulers, we select 32 examples randomly generated by Stable Diffusion in the above experiments. We catch the generation path $\{\boldsymbol{x}_{t(0)}, \boldsymbol{x}_{t(1)}, \ldots, \boldsymbol{x}_{t(n)}\}$ and then embed these latent variables into a 2D plane using Principal Component Analysis (PCA). The embedded generation paths of three schedulers are shown in Figure 5. Starting from the same Gaussian noise $\boldsymbol{x}_{t(0)}$, these generation processes finally reach different $\boldsymbol{x}_{t(n)}$. In the complete generation process, $\boldsymbol{x}_i$ is updated gradually along a curve. The three 10-step schedulers construct an approximate generation process. We can see that the errors in the beginning steps are accumulated in the subsequent steps, thus the errors at the final steps become larger than those at the beginning. The generation path of OLSS is the

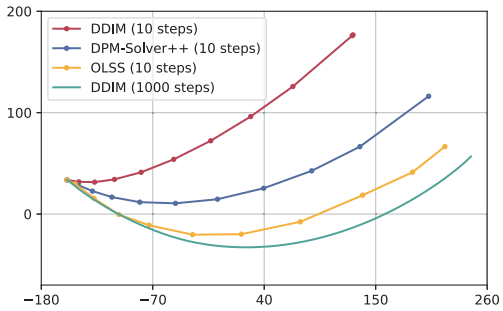


Figure 5: The generation path $\{\boldsymbol{x}_{t(0)}, \boldsymbol{x}_{t(1)}, \ldots, \boldsymbol{x}_{t(n)}\}$ of three schedulers. We embed the latent variables to 2D using PCA to see the generation process intuitively.

closest one to the complete generation process, and the generation path of DPM-Solver++ is the second closest.

4.4 Case Study

In Figure 6, we present examples of images generated using Stable Diffusion 2 with DDIM, DPM-Solver++, and OLSS. Benefiting from the excellent generation ability of Stable Diffusion 2, we can generate exquisite artistic images. With only 5 steps, DDIM may make destructive changes to images. For instance, the castle in the first example looks smoggy. DPM-Solver++ tends to sharpen the entity at the expense of fuzzing details up. Only OLSS can clearly preserve the texture in the generated images (see the fur of the wolf in the second example and the flowers in the third example). In the fourth example, we observe that sometimes both DPM-Solver++ and OLSS generate images in a different style, where OLSS tends to generate more detail. Despite being generated with significantly fewer steps than 1000-step DDIM, the images generated by OLSS still look satisfactory. Hence, OLSS greatly improves the quality of generated images within only a few steps.

5 CONCLUSION AND FUTURE WORK

In this paper, we investigate the schedulers in diffusion models. Specifically, we propose a new scheduler (OLSS) that is able to generate high-quality images within a very small number of steps. OLSS is a simple yet effective method that utilizes linear models to determine the coefficients in the iterative formula, instead of using mathematical theories. Leveraging the path optimization algorithm, OLSS can construct a faster process as an approximation of the complete generation process. The experimental results demonstrate that the quality of images generated by OLSS is higher than the existing schedulers with the same number of steps. In future work, we will continue investigating the generation process and explore improving the generative quality based on the modification in the latent space.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under grant number 62202170, Fundamental Research Funds for the Central Universities under grant number YBNLTS2023-014, and Alibaba Group through the Alibaba Innovation Research Program.

DDIM, 5 steps DPM-Solver++, 5 steps OLSS, 5 steps DDIM, 1000 steps

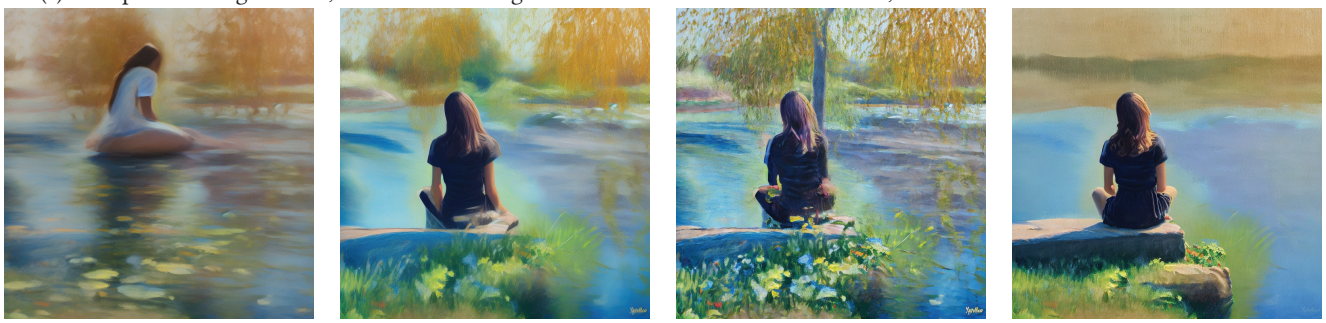
(a) Prompt: "Fantasy magic castle on floating island. A very beautiful art painting."

DDIM, 5 steps DPM-Solver++, 5 steps OLSS, 5 steps DDIM, 1000 steps

(b) Prompt: "The leader of the wolves bared its ferocious fangs. High-resolution digital painting."

DDIM, 5 steps DPM-Solver++, 5 steps OLSS, 5 steps DDIM, 1000 steps

(c) Prompt: "On the grassland, there is a towering tree with white flowers in full bloom, and under the tree are colorful flowers."



DDIM, 5 steps DPM-Solver++, 5 steps OLSS, 5 steps DDIM, 1000 steps

(d) Prompt: "The girl sitting by the river looks at the other side of the river and thinks about life. Oil painting."

Figure 6: Some examples generated by Stable Diffusion 2 with different schedulers and steps.

REFERENCES

[1] Åke Björck. 1990. Least squares methods. *Handbook of numerical analysis* 1 (1990), 465–652.

[2] John C Butcher. 2000. Numerical methods for ordinary differential equations in the 20th century. *J. Comput. Appl. Math.* 125, 1-2 (2000), 1–29.

[3] Prafulla Dhariwal and Alexander Nichol. 2021. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems* 34 (2021), 8780–8794.

[4] Patrick Esser, Johnathan Chiu, Parmida Atighehchian, Jonathan Granskog, and Anastasis Germanidis. 2023. Structure and content-guided video synthesis with diffusion models. *arXiv preprint arXiv:2302.03011* (2023).

[5] Zhida Feng, Zhenyu Zhang, Xintong Yu, Yewei Fang, Lanxin Li, Xuyi Chen, Yuxiang Lu, Jiaxiang Liu, Weichong Yin, Shikun Feng, et al. 2023. ERNIE-ViLG 2.0: Improving text-to-image diffusion model with knowledge-enhanced mixture-of-denoising-experts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10135–10145.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[7] Jiaxi Gu, Xiaojun Meng, Guansong Lu, Lu Hou, Niu Minzhe, Xiaodan Liang, Lewei Yao, Runhui Huang, Wei Zhang, Xin Jiang, et al. 2022. Wukong: A 100 million large-scale chinese cross-modal pre-training benchmark. *Advances in Neural Information Processing Systems* 35 (2022), 26418–26431.

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

[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems* 33 (2020), 6840–6851.

[10] Jonathan Ho and Tim Salimans. 2021. Classifier-Free Diffusion Guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*.

[11] Xun Huang, Yixuan Li, Omid Poursaeed, John Hopcroft, and Serge Belongie. 2017. Stacked generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5077–5086.

[12] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2017. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196* (2017).

[13] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. 2022. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems* 35 (2022), 26565–26577.

[14] Andrew Kerr, Dan Campbell, and Mark Richards. 2009. QR decomposition on GPUs. In *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*. 71–78.

[15] Diederik P Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

[16] Jinglin Liu, Chengxi Li, Yi Ren, Feiyang Chen, and Zhou Zhao. 2022. Diffsinger: Singing voice synthesis via shallow diffusion mechanism. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 36. 11020–11028.

[17] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. 2021. Pseudo Numerical Methods for Diffusion Models on Manifolds. In *International Conference on Learning Representations*.

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

[19] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095* (2022).

[20] Alexander Quinn Nichol and Prafulla Dhariwal. 2021. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*. PMLR, 8162–8171.

[21] Augustus Odena, Christopher Olah, and Jonathon Shlens. 2017. Conditional image synthesis with auxiliary classifier gans. In *International conference on machine learning*. PMLR, 2642–2651.

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

[23] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* (2022).

[24] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. 2016. Generative adversarial text to image synthesis. In *International conference on machine learning*. PMLR, 1060–1069.

[25] Phillip A Regalia. 1993. Numerical stability properties of a QR-based fast least squares algorithm. *IEEE Transactions on Signal Processing* 41, 6 (1993), 2096–2109.

[26] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10684–10695.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. 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*. Springer, 234–241.

[28] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. 2022. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems* 35 (2022), 36479–36494.

[29] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J Fleet, and Mohammad Norouzi. 2022. Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022).

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

[31] Tim Salimans and Jonathan Ho. 2021. Progressive Distillation for Fast Sampling of Diffusion Models. In *International Conference on Learning Representations*.

[32] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. 2022. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in Neural Information Processing Systems* 35 (2022), 25278–25294.

[33] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*. PMLR, 2256–2265.

[34] Jiaming Song, Chenlin Meng, and Stefano Ermon. 2020. Denoising Diffusion Implicit Models. In *International Conference on Learning Representations*.

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

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

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

[38] Patrick von Platen, Suraj Patil, Anton Lozhkov, Pedro Cuenca, Nathan Lambert, Kashif Rasul, Mishig Davaadorj, and Thomas Wolf. 2022. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers.

[39] Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. 2022. Learning fast samplers for diffusion models by differentiating through sample quality. In *International Conference on Learning Representations*.

[40] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Yingxia Shao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. 2022. Diffusion models: A comprehensive survey of methods and applications. *arXiv preprint arXiv:2209.00796* (2022).

[41] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. 2015. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365* (2015).

[42] Qinsheng Zhang and Yongxin Chen. 2022. Fast Sampling of Diffusion Models with Exponential Integrator. In *The Eleventh International Conference on Learning Representations*.