This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

PCM : Picard Consistency Model for Fast Parallel Sampling of Diffusion Models

Junhyuk So[1] Jiwoong Shin[1] Chaeyeon Jang[1] Eunhyeok Park[1,2]
[1]Department of Computer Science and Engineering, POSTECH
[2]Graduate School of Artificial Intelligence, POSTECH
{junhyukso,jwshin0610, jcy2749, eh.park}@postech.ac.kr

Abstract

Recently, diffusion models have achieved significant advances in vision, text, and robotics. However, they still face slow generation speeds due to sequential denoising processes. To address this, a parallel sampling method based on Picard iteration was introduced, effectively reducing sequential steps while ensuring exact convergence to the original output. Nonetheless, Picard iteration does not guarantee faster convergence, which can still result in slow generation in practice. In this work, we propose a new parallelization scheme, the Picard Consistency Model (PCM), which significantly reduces the number of generation steps in Picard iteration. Inspired by the consistency model, PCM is directly trained to predict the fixed-point solution, or the final output, at any stage of the convergence trajectory. Additionally, we introduce a new concept called model switching, which addresses PCM's limitations and ensures exact convergence. Extensive experiments demonstrate that PCM achieves up to a 2.71x speedup over sequential sampling and a 1.77x speedup over Picard iteration across various tasks, including image generation and robotic control.

1. Introduction

Recently, diffusion probabilistic models [7] have achieved outstanding performance across diverse generative tasks in vision [30, 32, 35], audio [9, 15, 23], text [1, 18, 25], robotics [2, 12], and bioinformatics [21, 46]. Their iterative denoising process enables high-quality outputs by capturing fine details and diversity, making them versatile across applications. However, this process has a notable drawback: slow generation speed. For example, early diffusion models like DDPM [7] require 1,000 denoising steps, each involving intensive computation, resulting in high latency.

To enable faster inference, various approaches have been proposed; however, most achieve speedups at the expense of quality degradation or inconsistent convergence. For instance, ODE-based solvers like DDIM [40] often trade off image quality for fewer sampling steps. Studies like the Consistency Model [43] and Progressive Distillation [37] use distillation to reduce sampling steps, but the final output may differ from the original model due to modified weights.

In this work, we aim to explore a different question: can increased parallelism lead to faster convergence? While adding parallelism can nearly linearly enhance throughput, achieving low latency in sequential generative models demands innovation at the mathematical foundation level. Recently, ParaDiGMS [38] tackled this challenge with a parallel sampling method based on Picard iteration, reducing the sequential steps in diffusion sampling by performing more computations in parallel. Picard iteration ensures convergence to the exact output, guaranteeing the same output quality as original model; however, it does not ensure fast convergence, which may result in slower generation speeds in practice.

In this study, we introduce the Picard Consistency Model (PCM) to accelerate the convergence speed of Picard iteration. Inspired by the Consistency Model [43], PCM is trained to directly predict the fixed-point solution at any point along the Picard iteration trajectory. In addition, our novel idea, model mixing, ensures exact convergence to the original output even with PCM's acceleration. Our extensive analysis reveals that PCM with model mixing gives a 2.71x speedup over sequential denoising and 1.77x speedup over Picard iteration, with guaranteeing exact convergence in diverse image generation and robotic applications.

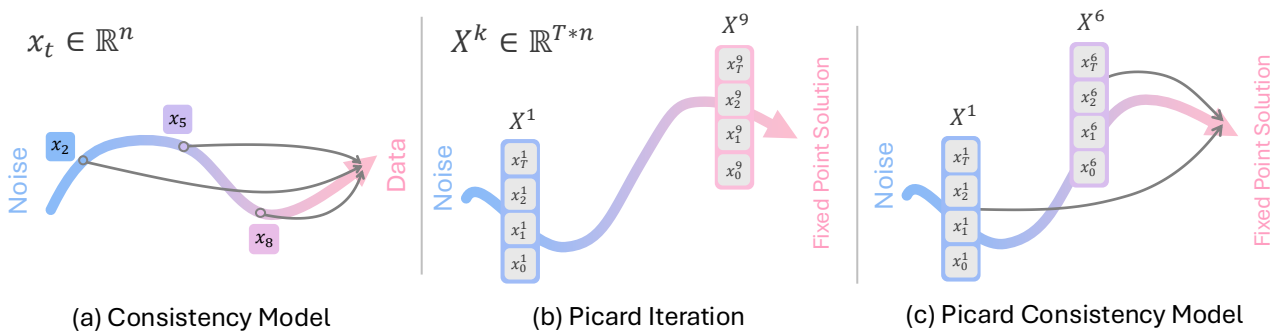
2. Preliminary

2.1. Diffusion Models

The Diffusion Probabilistic Model (DPM) was first introduced in the Denoising Diffusion Probabilistic Models [7] (DDPM) . The latent variable of diffusion model, $x_t (t \leq T)$ is defined through following diffusion forward process $q(\cdot)$.

$$q(x_t|x_T) = N(x_t; \sqrt{(\alpha(t))}x_T, (1-\alpha(t))I) \qquad (1)$$

Here, $\alpha(t)$ is scalar function that setted to $\alpha(0) = 0$, transforming $q(x_t)$ to standard gaussian $\mathcal{N}(0, I)$ with $t \rightarrow$



(a) Consistency Model (b) Picard Iteration (c) Picard Consistency Model

Figure 1. **Visual comparison** of (a) Consistency Model [43], (b) Picard iteration, and (c) our Picard Consistency Model (PCM). (a) [43] is trained to directly predict final output x_T from any point $x_t \in \mathbb{R}^n$ in denoising trajectory. (b) Picard iteration forms a denoising trajectory along tensors $X \in \mathbb{R}^{T*n}$. (c) Inspired by [43], we train PCM to predict final point X^* from any intermediate step in the Picard trajectory.

0. Please note that we denote $p(x_T)$ as a data distribution. The reverse process, $p_\theta(x_{t+1}|x_t)$, is also paramaterized by gaussian process with predicted mean and time-dependant predefiend variance σ_t^2.

$$p_\theta(x_{t+1}|x_t) = N(x_{t+1}; \mu_\theta(x_t), \sigma_t^2 I) \tag{2}$$

As shown in [41], the notable property of the diffusion model is that the forward process can also be expressed in the form of a Stochastic Differential Equation (SDE). The SDE version of diffusion forward process is as follows :

$$dx_t = f(t)x_t dt + g(t)d\mathcal{W}, x_T \sim q(x_T). \tag{3}$$

Here, $\mathcal{W}$ is the standard Wiener process and $f(t), g(t)$ is the drift and diffusion coefficient function that chosen to match $q(x_t|x_T)$. Similarly, the reverse process can also be represented in the form of SDE.

$$dx_t = [f(t)x_t - g^2(t)\nabla_{x_t} \log q_t(x)]dt + g(t)d\mathcal{W}, \tag{4}$$

where now the term $\nabla_{x_t} \log q_t(x)$ is approximated by our neural network $S_\theta(\cdot)$, with learnable weights of θ. As known in [41], these SDE has their corresponding *probability flow ODE*, which is given as follows:

$$dx_t = [f(t)x_t - g^2(t)\nabla_{x_t} \log q_t(x)]dt. \tag{5}$$

This ODE, with the randomness term $(g(t)d\mathcal{W})$ removed, generally has slightly lower generation performance than SDE but enables inference with very few discretization steps. Now we will focus on how to effectively solve this oridinary diffrental equation with parallel computing.

2.2. Picard Iteration

Another well-explored way to solve differential equation is transforming it as a Initial Value Problem (IVP). The initial value problem of differential equation, such as Eq. 5, can be generally defined as

$$x'(t) = F(t, x(t)), x(0) = x_0, \tag{6}$$

where x' denotes the differential(dx/dt) and x and F are function that $x : \mathbb{R} \to \mathbb{R}^n$, $F : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ and x_0 is the initial value. The equivalent integral form of this IVP is

$$x(t) = x(0) + \int_0^t F(x(s), s)ds. \tag{7}$$

Intuitively, evaluating this integral for x(t) requires the values of $x(t)$ over the interval $[0, t - \delta t)$, necessitating sequential evaluation. To introduce parallelism for solving this equation, we define the integral operator Φ on x.

Definition 1 (Integral Operator) *For any integral form of IVP, we can define the integral operator,Φ as*

$$\Phi(x; F) = x(0) + \int_0^t F(x(s), s)\, ds,$$

where $\Phi : \mathcal{X} \to \mathcal{X}$ and $\mathcal{X}$ is the function space of x.

In the context of diffusion models, we can interpret $F(\cdot)$ as a single evaluation of time-conditioned diffusion neural network, $x(t)$ as the denoising trajectory at time t, and $\Phi(\cdot)$ as the inference process of a differential equation solver (e.g., DDIM [40]). As shown in Def. 1, the *inference* of Φ does not necessarily need to be sequential in t, because now we assume access to the entire trajectory of $x(t)$, even if it is not the exact solution of the differential equation. Based on this definition, the Picard iteration algorithm provides a new way to solve this IVP in parallel, leveraging the well-known Picard-Lindelöf theorem [3].

Theorem 1 (Picard–Lindelöf Theorem [3]) *Let F be continuous in t and Lipschitz continuous in x. Then Φ has a unique fixed-point solution and is a* ***contraction mapping****, satisfying $d(\Phi(X), \Phi(Y)) \leq q\, d(X, Y)$, where (X, d) is a complete metric space and $q \in [0, 1)$.*

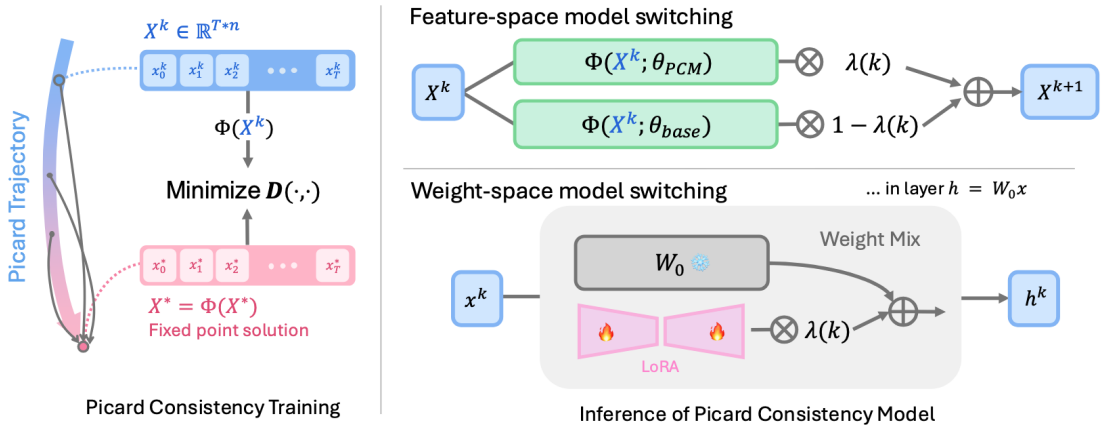


Figure 2. **Overview of our method.** Our goal is to accelerate the Picard iteration process, $X^{k+1} \leftarrow \Phi(X^k)$, by training with a loss function that minimizes the distance between a random point on the Picard trajectory and the fixed-point solution X^*. During inference, to preserve the convergence properties of the Picard iteration, we smoothly transition from our trained model θ_{PCM} to the original model θ_{base} using a scheduling function $\lambda(k)$ in feature space or weight space through LoRA.

Based on these theorems , the picard iteration algorithm is derived to find the fixed-point solution of IVP through the recursive inference on Φ.

$$\textbf{picard iteration :}\ x^{k+1} \leftarrow \Phi(x^k), x(0) = x_0 \tag{8}$$

Starting with any initial x_0, this iteration converges to a unique fixed-point solution $x^* = \Phi(x^*)$ as $k \to \infty$, as long as that F satisfies the mild assumptions in Theorem 1.

In practice, since the actual inference process of diffusion models is discretized in time, we can reinterpret this algorithm where Φ is now defined on tensors $X \in \mathbb{R}^{T\times n}$, with T denoting the total number of discretization time steps. ParaDiGMS[38] proposed the following discretized Picard iteration for discrete-time diffusion models:

$$X^{k+1} \leftarrow \Phi(X^k) \tag{9}$$

$$x_t^{k+1} \leftarrow x_0^k + \frac{1}{T}\sum_{i=0}^{t-1} s_\theta(x_i^k, i/T) \tag{10}$$

Here, $x_i^k \in \mathbb{R}^n$ represents the denoising result at the i-th time step during the k-th picard iteration. They observed that k is typically smaller than T, enabling practical and efficient parallel sampling of diffusion models.

3. Method

The mathematical foundation of Picard iteration supports convergence, but as shown in studies on numerical analysis [11, 31, 33], its convergence rate is often undefined and tends to be slow for large-scale problems, limiting practical speedup. To address this, we propose a novel method, Picard Consistency Training (PCT), to accelerate Picard iterations in diffusion models while preserving exact convergence properties.

3.1. Picard Consistency Training

Our approach is inspired by the intuitive observation that even minimal tuning can significantly enhance convergence. As shown in several diffusion distillation studies [28, 30, 37, 43], diffusion models have the capability to predict multiple steps ahead in their denoising trajectory with proper guidance from a pretrained teacher model. For instance, Consistency Models [43] are trained to reach their final output x_T directly from any point along the denoising trajectory. Similarly, while the denoising process in diffusion models forms a trajectory along a single tensor $x \in \mathbb{R}^n$, the Picard iteration can be viewed as forming a trajectory along tensors $X \in \mathbb{R}^{T*n}$. Building on this insight, we propose our Picard Consistency Model(PCM) in the style of Consistency Models: training the model to predict its final fixed-point solution from any iteration along the Picard trajectory. A visual comparison is presented in Fig. 1.

Picard Trajectory Dataset Generation. First, we collect the trajectory of the Picard iterations from the original model to make our *Picard Trajectory Dataset*. Specifically, to generate N samples for our dataset $\mathcal{D}$, we prepare our pretrained diffusion model θ_{base} and differential equation solver $\Phi(\cdot)$. Then, we sample an initial noise $x_0 \in \mathbb{R}^n$ from a standard Gaussian distribution $\mathcal{N}(0, I)$ and run the Picard iteration with θ_{base} for a total of K iterations. During each iteration of the Picard process, we store the initial noise x_0 and the current result of each Picard iteration $X^k \in \mathbb{R}^{T\times n}$. We provide the detailed algorithm for generating the trajec-

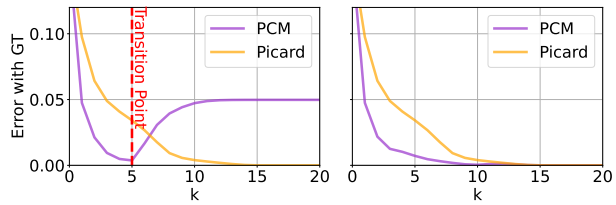


Figure 3. Comparison of convergence errors between Picard iteration at k and the ground truth. (a) PCM w/o model switching converges faster than the naive Picard, but the error begins to increase at the *transition point* and converges to a different point due to the modified weights. (b) PCM with model switching safely converges to the exact same output while ensuring accelerated convergence speed. Experiments are conducted on CelebA using DDIM.

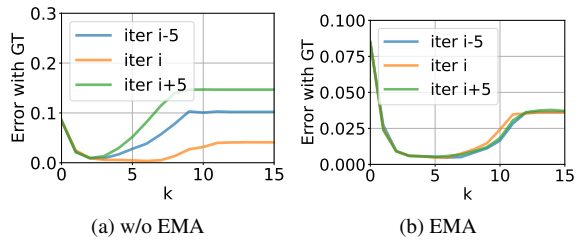


Figure 4. Training dynamics of PCM. (a) Naive PCT shows lots of training instability where transition point became different and final convergence error also different at every training iteration. (b) By using EMA, the error curve smoothly moved across the training iteration. Experiments are conducted on CelebA using DDIM.

tory dataset in Algorithm 1.

Loss function. For training, we randomly sample data X from our trajectory dataset $\mathcal{D}$ and choose a random index k from a uniform distribution $\mathcal{U}[0, K-1]$. We then select two tensors: X^k, a random point on the trajectory, and $X^K = X^*$, the solution fixed point. Initialized with the pretrained diffusion model θ_{base} used to generate the trajectory dataset, we train our PCM θ_{PCM} to directly predict the Picard solution X^* by using X^k as the input of a single Picard iteration. The model is trained by minimizing the following loss function:

$$\mathcal{L} = \mathbb{E}_{X\sim\mathcal{D},\, k\sim\mathcal{U}[0,K-1]}\, \alpha(k)\, D(X^*, \Phi(X^k; \theta_{\text{PCM}})), \quad (11)$$

where $D(\cdot,\cdot)$ is the metric function used to measure the distance between two data distributions (e.g., L2). One important aspect to note is the weighting function $\alpha(k)$, which is used to mitigate the noise variance gap between X^k and X^*. Intuitively, since the early stages (smaller k) of the Picard iteration are farther from the solution point, the images tend to be noisier compared to later stages. We use $\alpha(k) = \frac{1}{\sqrt{\text{Var}(k)}}$, where $\text{Var}(k)$ is the predefined variance schedule of the original diffusion model.

3.2. Feature-space Model Switching

However, because the PCT modifies the original weights, the convergent result of the θ_{PCM} differs from the original model θ_{base}, sacrificing the primary advantage of parallel sampling. In Fig. 3 (a), we depict the error between the result of picard iteration generated by θ_{PCM} (purple) and the ground-truth solution generated by θ_{base} (yellow). As shown in the figure, PCM initially converges to a lower error region faster than the baseline picard; however, it reaches a ***transition point*** where the error increases, and the final convergence error remains greater than zero.

To address this issue, we propose the **Model Switching** method, which uses θ_{PCM} in the early stages of the Picard iteration and switches to θ_{base} in the later stages. Specifically, we employ a linear interpolation between the output estimated by θ_{PCM} and θ_{base}, controlled by a mixing schedule $\lambda(k)$, in our new Picard iteration, X^{k+1} is computed as:

$$X^{k+1} \leftarrow \lambda(k) * \Phi(X^k, \theta_{PCM}) + (1-\lambda(k)) * \Phi(X^k, \theta_{base})$$

$$\lambda(k) = \max(0, \min(1, 1 - s * k/K)). \quad (12)$$

While we can use any decreasing function from 1 to 0 for $\lambda(k)$, we employ the linear function with stiffness s, controlling the switching speed of the schedule. K denotes the predefined maximum iteration of picard iteration. As shown in Fig. 3 (b), with the Model Switching method, we can observe that the final convergence error approaches zero while maintaining the accelerated convergence speed.

3.3. Training Stabilization with EMA

We found that PCT often destabilizes training, particularly affecting the transition point and final convergence of the Picard iteration. As shown in Fig. 4 (a), the transition point and final convergence error of Picard iterations vary significantly with each training iteration. This variability makes it challenging to select the best model using the validation set, and it is especially difficult to predefine the model-switching schedule $\lambda(k)$, as the transition point changes with each training iteration. To address this, we follow prior research on stabilizing diffusion model training [13] and apply Exponential Moving Average (EMA) updates at each training iteration i, as shown below.

$$\hat{\theta}_{\text{PCM}} \leftarrow \mu\hat{\theta}_{\text{PCM}} + (1-\mu)\theta^i_{\text{PCM}}, \quad (13)$$

where μ is a decay factor close to 1 (e.g., 0.999). As shown in Fig 4 (b), with EMA, we can notice that the transition point of θ_{PCM} and final convergence error moves smoothly across the training iteration.

3.4. Parameter-space Model Switching with LoRA

The remaining issues of PCT are that (1) it requires twice the model storage because both trained and untrained versions of the weights are needed, (2) the inference cost is doubled, even if parallelizable, and (3) full backpropagation of the diffusion model is required, which can be computationally intensive. To address these issues, we present an efficient version of our algorithm inspired by weight mixing [47] and LoRA [8].

First, before training, we apply LoRA to every layer in the diffusion model. For example, for a linear layer $h = W_0 x$ where $W_0 \in \mathbb{R}^{d\times s}$, the LoRA ΔW is injected as

$$h = W_0 x + \Delta W x = W_0 x + BAx,$$

where $A \in \mathbb{R}^{r\times s}$ and $B \in \mathbb{R}^{d\times r}$ are the trainable up- and down-projection layers of LoRA, while W_0 freezed during training. Since typically $r \ll s, d$, the size of ΔW is usually much smaller than θ_{PCM}.

During inference, now we apply Model Switching in the weight space rather than in the output space. Inspired by LoRA's weight-mixing strategy, we implement model switching by adjusting the scale λ of LoRA weights according to picard iteration k.

$$h^k = W_0 x^k + \lambda(k)\Delta W x^k = (W_0 + \lambda(k)\Delta W)x^k, \quad (14)$$

where h^k denotes the featue map h at the k-th picard iteration As with score mixing in the output space, $\lambda(k)$ is implemented using a linear schedule with stfiness s. This weight mixing operates quickly offline before each Picard iteration, allowing it to be implemented with the same latency as the original Picard iteration.

In the experiment section, we assume that PCM uses feature-space model switching by default. We explicitly denote PCM-LoRA when LoRA-based parameter-space model switching is applied in addition to PCM.

Algorithm 1 Picard Consistency Training

Require: Original diffusion model θ_{base}, ODE-Solver Φ, picard step K, dataset size N

▷ Dataset Generation
while $n < N$ **do**
  $\mathcal{J}, k \leftarrow \{\}, 0$ ▷ Initialize
  $X^0 \sim \mathcal{N}(0, I)$ ▷ Sample initial noise
  while $k < K$ **do**
    $X^{k+1} = \Phi(X^k)$ ▷ Picard iteration
    Append X^k to $\mathcal{J}$, $k \leftarrow k + 1$
  end while
  Append $\mathcal{J}$ to $\mathcal{D}$, $n \leftarrow n + 1$
end while

▷ Training
$\theta_{PCM}, \hat{\theta}_{\text{PCM}} \leftarrow \theta_{base}$ ▷ Start from pretrained.
while not converged **do**
  Sample $\mathcal{J} \sim \mathcal{D}, k \sim U[0, K-1]$
  Get X^k, X^* from $\mathcal{J}$and compute loss $\mathcal{L}$ using Eq. 11
  Backprop. θ_{PCM} using $\mathcal{L}$
  Update EMA $\hat{\theta}_{\text{PCM}}$ using Eq. 13
end while
return $\hat{\theta}_{\text{PCM}}$

Algorithm 2 Inference of Picard Consistency Model

Require: Original diffusion model θ_{base}, PCM θ_{PCM}, ODE-Solver Φ, picard step K

$k \leftarrow 0$
$X^0 \sim N(0, I)$ ▷ Sample initial noise
while $k < K$ **do**
  $\lambda \leftarrow \lambda(k)$ ▷ Eq. 12
  if θ_{PCM}`.use-lora` **then**
    for `lora-layers` **in** θ_{PCM} **do**
      `lora-layers.scale` $\leftarrow$ λ
    end for
    $X^{k+1} = \Phi(X^k; \theta_{PCM})$
  else
    $X_{PCM}, \leftarrow \Phi(X^k; \theta_{PCM})$
    $X_{base} \leftarrow \Phi(X^k; \theta_{base})$
    $X^{k+1} \leftarrow \lambda * X_{PCM} + (1 - \lambda) * X_{base}$
  end if
  $k \leftarrow k + 1$
end while
return X_T^K ▷ Return last image

4. Experiments

To validate the effectiveness of our method, we conducted experiments across various tasks, including image generation and robot control. For image generation, we tested on the Latent Diffusion Model [35] (LDM) , which uses a U-Net [36] architecture that operates denoising procedure in latent space of VAE, and performed experiments on the CelebA [24] dataset and Stable Diffusion [35] (SD), to conduct experiment on both unconditional and conditional sampling cases. For robotic control, we follow setups in Diffusion Policy [2]. Detailed experimental setups are provided in the relevant section.

For evaluation, we performed qualitative and quantitative analysis in both tasks. For the qualitative experiments, in image generation, we demonstrate that our method achieves generated exactly the same results compared to sequential denoising and our PCM generates higher-quality images within the same number of iterations compared to the naive Picard [38]. In robotic control tasks, we visualize an episode of pushT tasks using naive Picard and our

(a) Sequential (b) Picard (c) PCM

Figure 5. Qualitative comparison of Sequential (a), Picard (b), and PCM (c) on Stable Diffusion v1.4 using DDIM. All methods converge to the exact same solution, with PCM requiring 2.8 times less latency than Sequential.

Figure 6. Qualtitive comparison of Picard (up-row) and PCM(down-row) in same iteration k on LDM-CelebA with DDIM. While Picard generate inaccurate images in early iteration, PCM generate data that are more closely resembles actual images.

PCM, demonstrating PCM safely generate correct motion sequences while naive Picard failed.

In the quantitative experiments, for CelebA, we measured the latency speedup and Fréchet Inception Distance (FID) [6] using 5,000 generated images. For stable diffusion, we report the theoretical speedup and CLIP score [5] to measure the correctness of image and given prompt. For robotic control tasks, we report the latency speedup and average reward of episodes. The Picard Consistency Training and dataset generation are conducted on high-performance servers equipped with 8×RTX3090 and evaluated on 1×A6000 GPUs. We use CUDA 11.8 and Pytorch 2.4.0 for experiments.

4.1. Image Generation

First, we conducted quantitative experiments on image generation task by measuring FID, CLIP score, and latency speedup. For CelebA, we generated 500 samples for the dataset for Picard Consistency Training and trained the model over 50 epochs using the Adam optimizer with a learning rate of 1e-4. The final model was selected based on the highest performance across 10 noise configurations used for evaluation. We use total time step T for 50 for picard iteration. For SD, we setup similar experimental setup and report the theoretical speed-up using ratio of sequential steps. We depicts details and hyperparameters in our Appendix. For PCM-LoRA, following prior work [8], we applied LoRA to the linear layers in attention in U-Net.

As shown in Table 1, for LDM-CelebA, while sequential DDIM requires 18 steps to converge, PCM converges in just 6 steps, achieving a speedup of 2.11x. Although the Picard [38] also converges in a similar number of steps, PCM consistently achieves a lower FID than Picard. Furthermore, as the sequential step k in Picard iterations decreases, Picard [38] generates lower-quality images, whereas PCM rapidly generates feasible image quality. For example, while Picard produces entirely incorrect images at 2 steps with an FID of 257.82, PCM generates images with FID of 67.74.

Table 2 presents similar results for SD. As shown, PCM consistently produces better CLIP score than Picard under the same sequential iteration k, even surpassing the performance of Picard at $k = 9$ when using PCM-LoRA at $k = 8$.

We also depict the qualitative comparison of Picard and PCM. As shown in Fig. 5, both Picard (Fig. 5 (b)) and PCM (Fig. 5 (c)) converged to the exact same image as the ground truth of sequential denoising in Fig. 5 (a). This result demonstrates that our PCM successfully restores the output distribution of the original model through model switching, even when starting with different weights. In Fig. 6, we present the results of the naive Picard (up-row) and our PCM (down-row) for the same sequential Picard iteration k. As seen in the figure, especially at low k, Picard generates almost incorrect images, while PCM produces data that more closely resembles actual images.

4.2. Robotic Control

Recently, research on creating robust robot action sequences using diffusion models has been showing increasingly promising results [2]. Compared to traditional methods, diffusion models demonstrate high robustness and quality in motion generation across various situation and modalities. However, since increasing the speed of motion control is crucial for improving industrial productivity and user satisfaction, the high latency from the iterative nature of diffusion models poses a significant obstacle to their commercialization. Therefore, we experimented our PCM with a latency-critical robotic control task. Following setups from Diffusion Policy [2], a recently proposed diffusion model framework for behavioral cloning, we conduct an experiment using the PushT dataset, which aims to generate motion sequences for locating the T-block in the desired location. We used 1D state based U-Net that takes observation of current state as input and use action horizon with 8 and maximum trajectory length with 200 for the experiment. For PCT, we generated a total of 2000 samples for datasets and use Adam optimizer with learning rate of 1e-4 for 10 epochs.

In Table 3, we present the average reward and latency speedup of sequential, Picard, and PCM methods using

LDM-CelebA				
(DDIM)	**Steps↓**	**FID↓**	**latency↓**	**Speedup↑**
Sequential[40]	9	51.97	1.42s	1x
Picard[38]	4	60.82	0.88s	1.61x
PCM	3	56.80	0.91s	1.56x
PCM-LoRA	3	**50.10**	**0.66s**	**2.15x**
Sequential[40]	15	38.87	2.36s	1x
Picard[38]	5	42.67	1.10s	2.14x
PCM	5	40.09	1.52s	1.55x
PCM-LoRA	5	**38.35**	**1.10s**	**2.14x**
Sequential[40]	18	36.09	2.83s	1x
Picard[38]	6	36.19	1.34s	2.11x
PCM	6	36.67	1.87s	1.51x
PCM-LoRA	6	**35.97**	1.34s	2.11x
Comparison under same sequential step				
Sequential[40]	2	366.92	0.32s	-
Picard[38]	2	257.83	0.44s	-
PCM	2	77.62	0.60s	-
PCM-LoRA	2	**67.74**	0.44s	-
(DPM-Solver)	**Steps↓**	**FID↓**	**latency↓**	**Speedup↑**
Sequential	7	64.06	1.21s	1x
Picard	5	64.68	1.10s	1.09x
PCM	4	64.14	1.20s	1.01x
PCM-LoRA	4	**62.61**	**0.88s**	**1.37x**

Table 1. Image generation performance comparison on LDM-CelebA dataset.

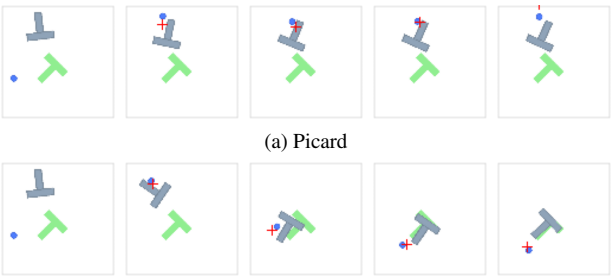
Stable Diffusion(SD) V1.4			
(DDIM)	**Steps↓**	**CLIP-Score↑**	**Theoretical Speedup↑**
Sequential[40]	14	26.20	1x
Picard[38]	5	26.24	2.8x
PCM	5	26.25	2.8x
PCM-LoRA	5	**26.26**	**2.8x**
Sequential[40]	25	26.34	1x
Picard[38]	9	26.34	2.78x
PCM	9	26.34	2.78x
PCM-LoRA	8	**26.35**	**3.13x**
Comparison under same sequential step			
Sequential[40]	4	25.00	-
Picard[38]	4	25.91	-
PCM	4	25.92	-
PCM-LoRA	4	**26.02**	-

Table 2. Image generation performance comparison on Stable Diffusion v1.4.

DDIM on the PushT task. As shown in the table, while the Sequential method requires 25 steps to reach a reward of 0.8, PCM converges in only 8 steps, achieving a even higher reward of 0.83 with 2.7x wall-clock speed-up and approximately twice the acceleration compared to Picard iteration. Even at lower reward levels, PCM achieves a higher reward with faster speed-up than Sequential and surpasses Picard iteration with the same number of steps. We also depict the qualitative comparison of Picard and PCM in Fig. 7. As shown in the figure, under the same latency budget (the same k), (a) While naive Picard fails to locate T-block in correct location, (b) PCM successfully completes the task.

PushT				
(DDIM)	**Steps↓**	**Reward↑**	**Time ↓**	**Speedup↑**
Sequential[2]	25	0.80	5.14s	1x
Picard[38]	15	0.80	3.57s	1.44x
PCM	**8**	**0.83**	2.01s	2.55x
PCM-LoRA	**8**	**0.83**	1.90s	**2.71x**
Sequential[2]	7	0.45	1.44s	1x
Picard[38]	5	0.43	1.19s	1.21x
PCM	**5**	**0.48**	1.25s	1.15x
PCM-LoRA	**5**	**0.47**	1.19s	**1.21x**
Sequential[2]	4	0.12	0.82s	1x
Picard[38]	3	0.12	0.71s	1.15x
PCM	**3**	**0.15**	0.75s	**1.10x**
PCM-LoRA	**3**	**0.15**	0.71s	**1.15x**

Table 3. Robotic control performance comparison on PushT dataset.



(a) Picard

(b) PCM

Figure 7. Qualitative comparison of Picard and PCM at PushT dataset. We use DDIM sampler and record 1 episode using k=6. (a) While naive Picard fails to locate the T-block in correct location (b) PCM successfully completes the task.

4.3. Ablation Study

In this section, we conducted an ablation study to analyze the impact of different components in our PCM. Specifically, we examined the effects of loss weighting and model switching on the feature and weight spaces with different values of stiffness s.

4.3.1. Effect of Loss Weighting

In Table 4, we compare the results with and without the loss-weighting term $\alpha(k)$ from Eq. 11. As shown in the table, without loss weighting, the model converges to a sub-optimal solution due to the variance gap in the Picard iterations. This result highlights the importance of applying loss weighting based on variance levels, similar to the approach used in standard diffusion model training.

4.3.2. Effect of Model Switching

In Fig. 8, we compare the convergence error between the ground truth and PCM iterations at various k values, us-

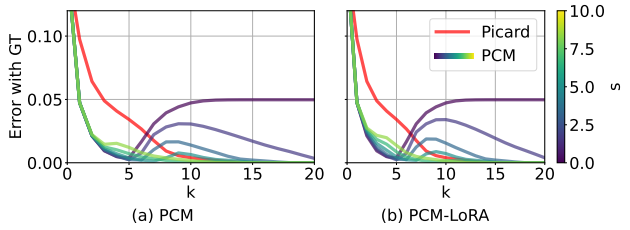


Figure 8. Effect of model switching on different stiffness $s \in \{0, 2, 4, 6, 8, 10\}$. Experiments are conducted on LDM-CelebA with DDIM sampler.

$\alpha(k)$	Reward ↑
1	0.71
$1/\sqrt{var(k)}$	**0.83**
$1/k$	0.78

Table 4. Effect of $\alpha(k)$. Experiments are conducted with PushT task on $k = 8$.

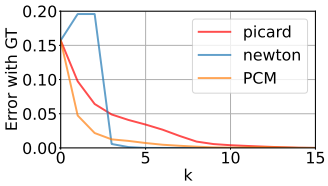


Figure 9. Comparison with Newton's method. We use LDM-CelebA and DDIM 50-step.

ing different stiffness s from Eq. 12. As shown, both (a) the feature space and (b) the weight space converge to the zero-error region when an appropriate s is used. However, when s is too small, the error converges to a non-zero point, and when s is too large, the error curve fluctuates slightly between $k = 3$ and $k = 4$, resulting in slower convergence.

4.4. Comparision with Newton's Method

In this section, we conducted a comparison experiment with Newton's method [34], a well-known technique for accelerating the root-finding problem. Specifically, finding the solution to the fixed-point problem can be converted into a root-finding problem: $\Psi(X^*) = \Phi(X^*) - X^* = 0$. To solve this root-finding problem, we apply Newton's method: $X^{k+1} \leftarrow X^k - J_{X^k}[\Psi(X^k)]^{-1}\Psi(X^k)$, where $J[\cdot]$ is the Jacobian. While computing the Jacobian inverse is generally infeasible for large-scale problems like neural networks, we can emulate Newton's method with $O(T)$ sequential time by leveraging the MCMC properties of the denoising process. Although this approximated method does not improve latency, it allows us to compare the speed of convergence and assess the potential margin for PCM. Details of the Newton algorithm are provided in the Appendix.

In Fig. 9, we compare the convergence results of Newton's method and PCM on the LDM-CelebA dataset. As shown, the points where Newton's method converges at a low error level are nearly the same as the convergence points of PCM, indicating that the convergence speed of PCM is actually close to that of the oracle. Additionally, unlike Newton's method, which often struggles with poor initialization, PCM shows stable convergence.

5. Related Works

5.1. Efficient Diffusion Models

To accelerate the slow generation speed of diffusion models, various studies have introduced methods to create more efficient diffusion models. One of the most active research areas focuses on developing differential equation solvers [26, 27, 40, 48], which aim to preserve generation quality while reducing the number of sampling steps. Additionally, to achieve extremely low sampling steps, distillation-based approaches [30, 37, 43] have also been widely studied. Alternatively, to reduce the computational cost, optimization methods such as quantization [10, 19] and caching techniques [29, 39] are also being actively explored.

5.2. Parallel inference of Sequential Models

Several pioneering studies have aimed to reduce the latency of sequential models by leveraging parallel computation. An early study [42] introduced a parallel inference algorithm for sequential models using Jacobi and Gauss-Seidel iterations. Building on this foundation, [22] proposed an accelerated parallel inference method for Jacobi iteration using quasi-newton methods. The ParaDiGMS framework [38] was the first to apply parallelization to denoising steps in diffusion models through Picard iteration. Furthermore, ParaTAA [45] reframed fixed-point Picard iterations as a nonlinear root-finding problem, proposing an acceleration technique based on Anderson Acceleration. However, Picard iteration often suffers from slow convergence in practice, and ParaTAA requires additional computation and memory for the quasi-newton matrix during inference. Parallel inference for language models has also become an active research area, with methods like speculative decoding [14, 17, 20, 44] and Jacobi decoding [4, 42]. Recently, similar to our approach, CLLM [16] accelerated jacobi decoding by minimizing the distance between a random trajectory and the fixed-point solution.

6. Conclusion

In this paper, we introduce the Picard Consistency Model (PCM) to enhance parallel sampling in diffusion models. Inspired by Consistency Models, our Picard Consistency Training enables the diffusion model to predict the final fixed-point solution at each Picard iteration, accelerating convergence notably. Additionally, we propose a model-switching method to preserve the exact convergence properties of Picard iteration. Extensive experiments demonstrate that PCM achieves approximately a 2.71x speedup compared to sequential denoising and around a 1.77x speedup over standard Picard iteration without compromising generation quality.

Acknowledgments This work was supported by IITP and NRF grants funded by the Korea government(MSIT) (Nos. RS-2019-II191906, RS-2023-00213611, RS-2024-00415602, RS-2024-00457882).

References

[1] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021. 1

[2] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023. 1, 5, 6, 7

[3] Earl A. Coddington, Norman Levinson, and T. Teichmann. Theory of ordinary differential equations. *Physics Today*, 9 (2):18, 1956. 2

[4] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding. *arXiv preprint arXiv:2402.02057*, 2024. 8

[5] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021. 6

[6] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 6

[7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1

[8] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 5, 6

[9] Rongjie Huang, Jiawei Huang, Dongchao Yang, Yi Ren, Luping Liu, Mingze Li, Zhenhui Ye, Jinglin Liu, Xiang Yin, and Zhou Zhao. Make-an-audio: Text-to-audio generation with prompt-enhanced diffusion models. In *International Conference on Machine Learning*, pages 13916–13932. PMLR, 2023. 1

[10] Yushi Huang, Ruihao Gong, Jing Liu, Tianlong Chen, and Xianglong Liu. Tfmq-dm: Temporal feature maintenance quantization for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7362–7371, 2024. 8

[11] Martin Hutzenthaler, Thomas Kruse, and Tuan Anh Nguyen. On the speed of convergence of picard iterations of backward stochastic differential equations. *arXiv preprint arXiv:2107.01840*, 2021. 3

[12] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. Dall-e-bot: Introducing web-scale diffusion models to robotics. *IEEE Robotics and Automation Letters*, 8(7):3956–3963, 2023. 1

[13] Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24174–24184, 2024. 4

[14] Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36, 2024. 8

[15] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020. 1

[16] Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. Cllms: Consistency large language models. *arXiv preprint arXiv:2403.00835*, 2024. 8

[17] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023. 8

[18] Xiang Li, John Thickstun, Ishaan Gulrajani, Percy S Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *Advances in Neural Information Processing Systems*, 35:4328–4343, 2022. 1

[19] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023. 8

[20] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024. 8

[21] Zehui Li, Yuhao Ni, Tim August B Huygelen, Akashaditya Das, Guoxuan Xia, Guy-Bart Stan, and Yiren Zhao. Latent diffusion model for dna sequence generation. *arXiv preprint arXiv:2310.06150*, 2023. 1

[22] Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length. *arXiv preprint arXiv:2309.12252*, 2023. 8

[23] Haohe Liu, Zehua Chen, Yi Yuan, Xinhao Mei, Xubo Liu, Danilo Mandic, Wenwu Wang, and Mark D Plumbley. Audioldm: Text-to-audio generation with latent diffusion models. *arXiv preprint arXiv:2301.12503*, 2023. 1

[24] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018. 5

[25] Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Forty-first International Conference on Machine Learning*. 1

[26] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022. 8

[27] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion

probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022. 8

[28] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023. 3

[29] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15762–15772, 2024. 8

[30] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14297–14306, 2023. 1, 3, 8

[31] Olavi Nevanlinna. Linear acceleration of picard-lindelöf iteration. *Numerische Mathematik*, 57:147–156, 1990. 3

[32] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 1

[33] Sara Pollock, Leo G Rebholz, and Mengying Xiao. Anderson-accelerated convergence of picard iterations for incompressible navier–stokes equations. *SIAM Journal on Numerical Analysis*, 57(2):615–637, 2019. 3

[34] Boris T Polyak. Newton's method and its use in optimization. *European Journal of Operational Research*, 181(3): 1086–1096, 2007. 8

[35] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022. 1, 5

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

[37] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022. 1, 3, 8

[38] Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. 1, 3, 5, 6, 7, 8

[39] Junhyuk So, Jungwon Lee, and Eunhyeok Park. Frdiff: Feature reuse for universal training-free acceleration of diffusion models. *arXiv preprint arXiv:2312.03517*, 2023. 8

[40] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 1, 2, 7, 8

[41] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020. 2

[42] Yang Song, Chenlin Meng, Renjie Liao, and Stefano Ermon. Accelerating feedforward computation via parallel nonlinear equation solving. In *International Conference on Machine Learning*, pages 9791–9800. PMLR, 2021. 8

[43] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023. 1, 2, 3, 8

[44] Ziteng Sun, Ananda Theertha Suresh, Jae Hun Ro, Ahmad Beirami, Himanshu Jain, and Felix Yu. Spectr: Fast speculative decoding via optimal transport. *Advances in Neural Information Processing Systems*, 36, 2024. 8

[45] Zhiwei Tang, Jiasheng Tang, Hao Luo, Fan Wang, and Tsung-Hui Chang. Accelerating parallel sampling of diffusion models. In *Forty-first International Conference on Machine Learning*, 2024. 8

[46] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023. 1

[47] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022. 5

[48] Shuchen Xue, Mingyang Yi, Weijian Luo, Shifeng Zhang, Jiacheng Sun, Zhenguo Li, and Zhi-Ming Ma. Sa-solver: Stochastic adams solver for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. 8