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

Supplementary Material

1. Experimental Setup

In this section, we provide detailed hyperparameter configurations for Picard Consistency Training (PCT) on the CelebA, Stable Diffusion, and PushT datasets.

All experiments in this paper were conducted on GPU servers equipped with 8 NVIDIA GTX 3090 GPUs (each with 24 GB VRAM) and 2 AMD 7313 CPUs. The implementations were built using the PyTorch v2.4.0 and CUDA 11.8 framework. The source code for our work is included in the supplementary materials.

In Table 1, we summarize the hyperparameter configurations for PCT in the image generation tasks. Additionally, we applied Low-Rank Adaptation (LoRA) to all attention layers in our 2D-UNet architecture. For the construction of the training dataset for Stable Diffusion, we selected 20 arbitrary prompts and extracted 10 images for each prompt. For the CLIP score evaluation, we used 200 images generated from the prompts used during training and 300 images generated from unused prompts.

Table 2 also provides the hyperparameter configurations for PCT in the robotic control task, PushT. Moreover, the diffusion model for the robotic task takes the observation as input and predicts the action sequence over a defined action horizon. The model then executes the predicted action and uses the subsequent observation as input for the next step. We set the action horizon to 8, with a maximum of 200 steps in the action sequence generation. For this task, we use a 1D-state-based UNet and also apply LoRA to the attention layers within the transformer blocks.

CelebA			
# dataset	**T**	**K**	**Solver**
200	50	30	DDIM,DPMSolver
Optimizer	**learning rate**	**scheduler**	**epoch**
Adam	1e-4	Cosine	50
Stable Diffusion v1.4			
# dataset	**T**	**K**	**Solver**
200	30	15	DDIM
Optimizer	**learning rate**	**scheduler**	**epoch**
Adam	1e-4	Cosine	10

Table 1. Hyperparameters for PCT in image generation tasks.

PushT			
# dataset	**T**	**K**	**Solver**
2000	50	30	DDIM
Optimizer	**learning rate**	**scheduler**	**epoch**
Adam	5e-5	Cosine	200

Table 2. Hyperparameters for PCT in robotic control tasks.

2. Procedure of emulating Newton's method

As mentioned in the paper, our objective is to find the solution to the fixed-point problem $\Phi(X^*) = X^*$, where $x \in \mathbb{R}^d$ and $X = [x_0, x_1, \ldots, x_{T-1}] \in \mathbb{R}^{T*d}$. This fixed point problem can be reformulate as root-finding problem $\Psi(X^*) - X^* = 0$. The *inference* of Φ, Ψ are defined as

$$\Phi(X) = [x_0, F(t_1, x_0), F(t_2, x_1), \ldots, F(t_{T-1}, x_{T-2})] \tag{1}$$

$$\Psi(X) = [0, x_1 - F(t_1, x_0) \ldots, x_{T-1} - F(t_{T-1}, x_{T-2})] \tag{2}$$

Where $F(\cdot)$ is single inference of diffusion model. We start by apply newton's method to solve root-findinng problem $\Psi(\cdot)$:

$$X^{k+1} = X^k - J_{X^k}[\Psi(X^k)]^{-1}\Psi(X^k), \tag{3}$$

where $J_{X^k}[\Psi(X^k)]$ denotes the Jacobian matrix of $\Psi(X^k)$ with respect to X^k.

However, directly computing the Jacobian matrix and its inverse becomes computationally prohibitive for large-scale problems, such as diffusion models. For instance, in the case of the CelebA dataset, $d = 32 \times 32 \times 3 = 3072$ and $T = 50$, resulting in a Jacobian matrix of size $(d \cdot T) \times (d \cdot T) = 153600 \times 153600$, requiring approximately 700 GB of memory to store the Jacobian matrix.

To implicitly emulate the Newton iteration, we start by multiplying the Jacobian matrix on both sides:

$$J_{X^k}[\Psi(X^k)]X^{k+1} = J_{X^k}[\Psi(X^k)]X^k - \Psi(X^k). \tag{4}$$

Rearranging terms, we get:

$$J_{X^k}[\Psi(X^k)](X^{k+1} - X^k) = -\Psi(X^k). \tag{5}$$

In matrix form, this becomes:

$$\begin{bmatrix} \frac{\partial F(t_1,x_0^k)}{\partial x_0^k} & I & 0 & \dots & 0 \\ 0 & \frac{\partial F(t_2,x_1^k)}{\partial x_1^k} & I & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & \dots & I \end{bmatrix} \cdot \begin{bmatrix} x_1^{k+1} - x_1^k \\ x_2^{k+1} - x_2^k \\ \vdots \\ x_T^{k+1} - x_T^k \end{bmatrix} = \begin{bmatrix} F(t_1, x_0^k) - x_1^k \\ F(t_2, x_1^k) - x_2^k \\ \vdots \\ F(t_T, x_{T-1}^k) - x_T^k \end{bmatrix} \tag{6}$$

solving this equation is equivalent as :

$$x_{t_{i+1}}^{k+1} = F(t_{i+1}, x_t^k) + \frac{\partial F(t_{i+1}, x_{t_i}^k)}{\partial x_{t_i}^k}(x_{t_i}^{k+1} - x_{t_i}^k) \tag{7}$$

As shown in the equation, the computation of $\frac{\partial F(t_{i+1}, x_{t_i}^k)}{\partial x_{t_i}^k}(x_{t_i}^{k+1} - x_{t_i}^k)$ corresponds to a Jacobian-vector product (JVP), which can be efficiently computed using autograd frameworks such as PyTorch. By applying this equation and computing the JVP sequentially for each t_i, we can emulate single iteration of Newton's method in $O(T)$ sequential time with feasible computational resources.

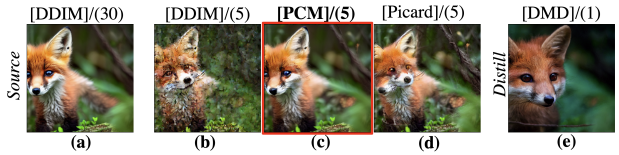


Figure 1. [x]/(y) denotes [Method]/(Sequential Step)

3. Comparison with Distillation-based methods

We avoid direct comparison between PCM with distillation-based methods(*Distill*), such as CM, DMD, and CTM, since the two approaches have different goals. *Distill* sacrifices diversity (and quality in some cases) for lower latency, which generates different output from the source model due to modified weights. See Fig.1(a)(source) and (e)(*Distill*). This problem can be crucial in scenario where even minor quality degradation is unacceptable, such as robotic control. In contrast, PCM accelerates inference **without any quality degradation**, as shown in Fig.1 (c). This exact convergence property is guaranteed by the Picard theorem, making PCM apart from typical optimization methods that trade latency for quality. In fact, many parallel sampling works [1, 2] did not compare against *Distill* for this reason. Furthermore, PCM offers advantages in training cost: *Distill* typically requires days, whereas PCM takes only an hour.

4. Effect of weight-mixing

In Table 3, we present the effects of weight mixing to better highlight the novelty of PCM. As shown, without weight mixing, the FID improvement of PCM over Picard is marginal, even resulting in a higher FID score than the naive Picard. However, when weight mixing is applied, PCM achieves a significantly better FID score, outperforming Picard.

LDM-CelebA (FID↓)	k=1	k=2	k=3	k=4	k=5
Picard	382.48	257.83	109.67	60.82	42.66
PCM w/o Mix	124.93	76.75	63.54	54.46	50.07
PCM w. Mix	**124.93**	**67.77**	**50.14**	**41.94**	**38.36**

Table 3. Effect of Weight Mixing

5. Evaluation on various metric

In Table 4, we report the results of multiple evaluation metrics for generative models—including FID, sFID, IS, Precision, Recall, and CMMD—measured on LDM-CelebA with $k = 5$. The results demonstrate that PCM consistently outperforms both Picard and Sequential across all metrics, except for IS, which is often considered *noisy* as it reflects the variance of the generated images.

Methods	FID↓	sFID↓	IS↑	Precision↑	Recall↑	CMMD↓
Sequential	375.08	54.75	2.57	0.00	0.00	4.95
Picard	42.66	20.36	**3.07**	0.43	0.37	1.95
PCM	**38.36**	**15.13**	2.76	**0.44**	**0.42**	**1.71**

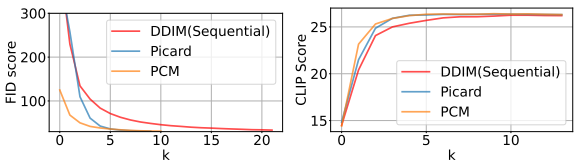
Table 4. More metric evaluations

6. Pareto Front Comparison

In Fig. 2, we present a Pareto front comparison of Sequential(DDIM), Picard, and our proposed PCM, evaluated based on their difference from the ground truth (GT) over the same sequential iteration k. As illustrated in the figures, our PCM consistently outperforms all other methods given the same number of sequential iterations, achieving a complete Pareto front in both CelebA and Stable diffuison dataset.

7. Limitation

In this study, we propose the Picard Consistency Model, which significantly accelerates the convergence speed of Picard Sampling. The primary limitation of our approach is that it requires training, which incurs a higher computational cost compared to existing training-free acceleration methods. However, our method does not require additional datasets, completes training within relatively few epochs. Also, since training is a one-time cost, PCM introduces no



(a) CelebA (b) Stable Diffusion

Figure 2. Pareto front comparison of diffrent sampling methods, Sequeuntial, Picard and our PCM. Our PCM show consistent improvement in every iteration compared to oteher methods.

overhead during the inference. The second limitation of our approach is a challenge shared by all parallel inference methods: reducing latency necessitates increased computational resources and energy. However, considering the current trend in computing, which is moving toward maximizing parallelism, we believe that parallel computation costs will become less of a concern. Instead, reducing latency is likely to deliver greater value in practice.

8. Additional Visualization Examples

In Fig. 3,4, we provide additional visual examples for CelebA and Stable Diffusion, comparing the quality of image generation using DDIM (Sequential), Picard, and our proposed PCM. As shown in the figures, while DDIM and Picard converge slowly during the initial iterations and produce inaccurate images, our PCM generates plausible images earlier and converges more quickly.

In Figures 5 and 6, we illustrate the action trajectories of Picard and our PCM using three different random seeds starting from $k = 4$. As shown in the figures, while Picard fails to generate accurate motions in most cases, our PCM successfully produces correct motions in nearly all scenarios.

References

[1] Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[2] 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. 2

Figure 3. Stable Diffusion v1.4 Qualtitive comparison of DDIM(up-row), Picard (mid-row) and PCM(down-row) in same iteration k. The prompt is "a warrior in gleaming armor, standing on a battlefield, dramatic lighting, ultra realistic, intricate details, vivid, hdr, cinematic".

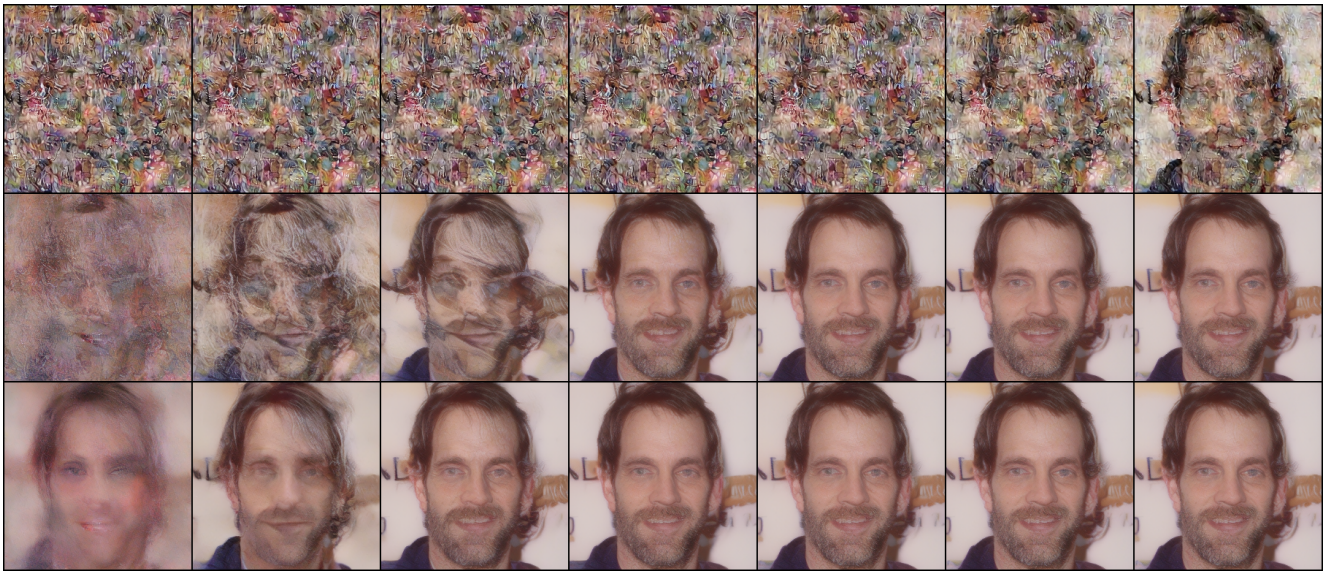


Figure 4. CelebA Qualtitive comparison of DDIM(up-row), Picard (mid-row) and PCM(down-row) in same iteration k.

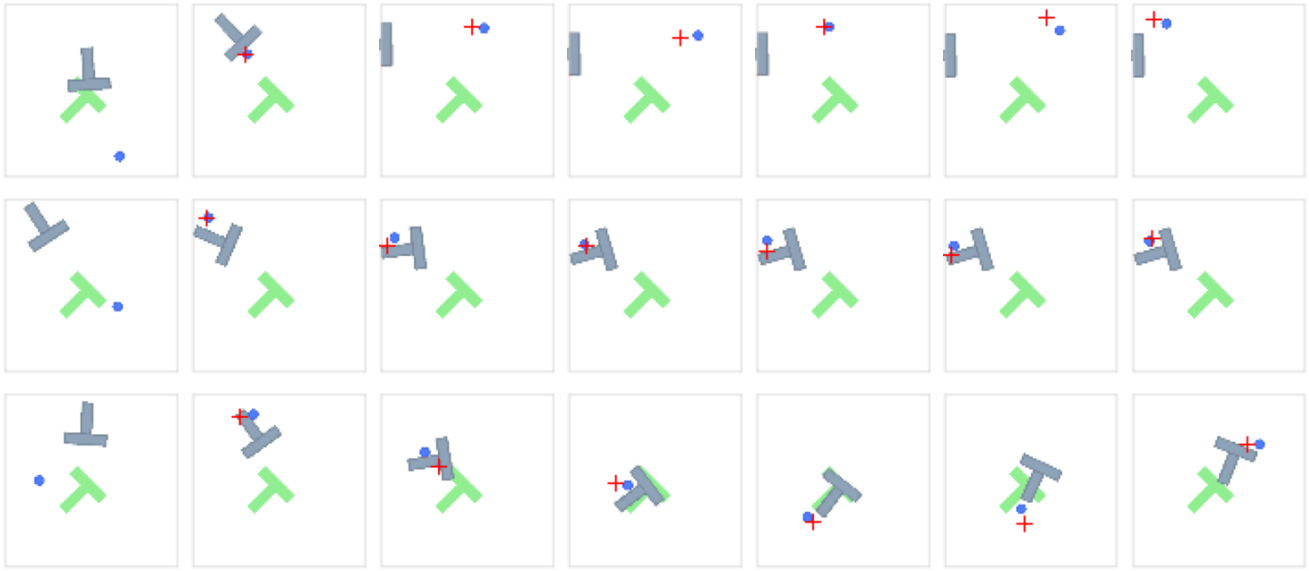


Figure 5. Generated action episode using Picard from $k = 4$, we randomly sample 3 epsiodes using different random seeds.

Figure 6. Generated action episode using PCM from $k = 4$, we randomly sample 3 epsiodes using different random seeds.