PinT (Parallel in Time) algorithms for Diffusion Models

Junoh Kang

Computer Vision Laboratory
ECE, Seoul National University
junoh.kang@snu.ac.kr

Computer Vision Lab
Seoul National University

Motivation

Efficient sampling in diffusion models

Diffusion models requires heavy computation resource due to **iterative** sampling strategy.

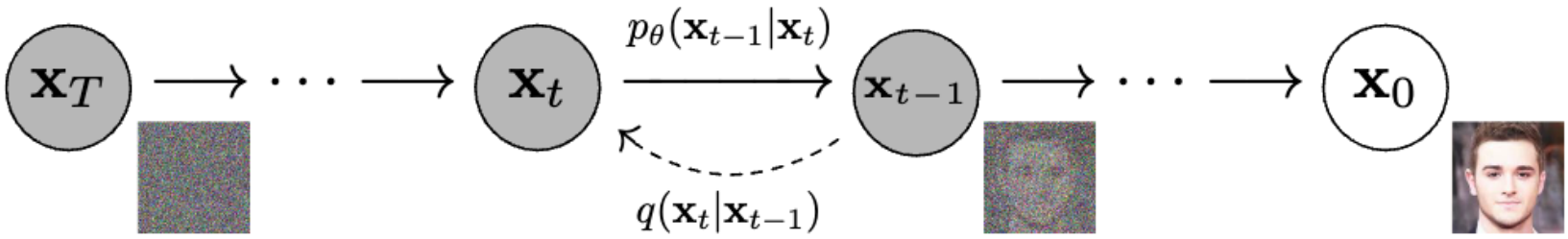


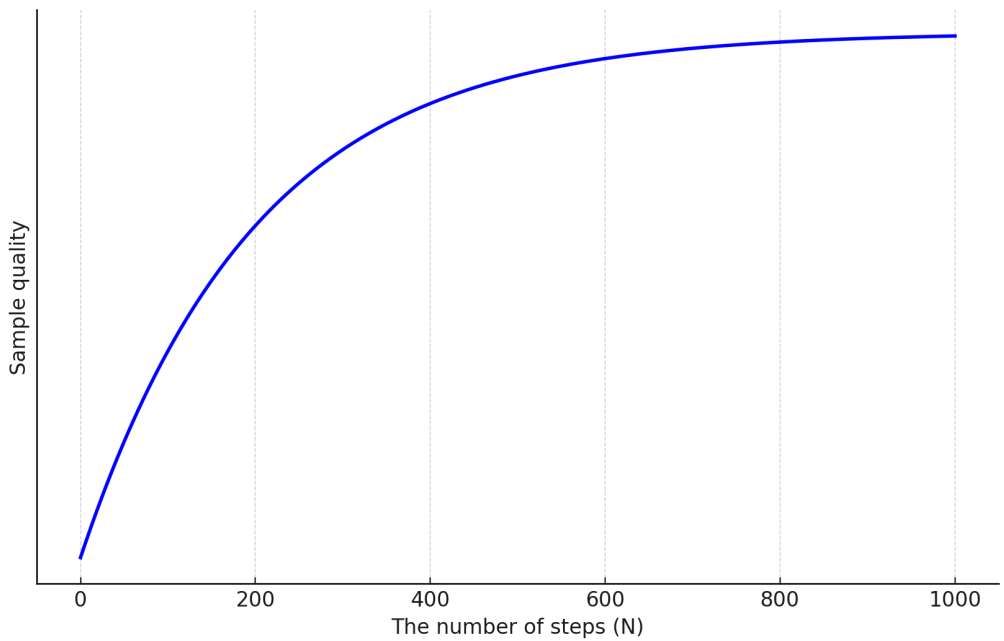
Figure 1: Graphical model of DDPM []

Motivation

Efficient sampling in diffusion models

DDIM: Accelerating Diffusion Sampling

- DDIM reduces the number of inference steps to accelerate sampling.
- The Trade-off: This often results in a degradation of sample quality.

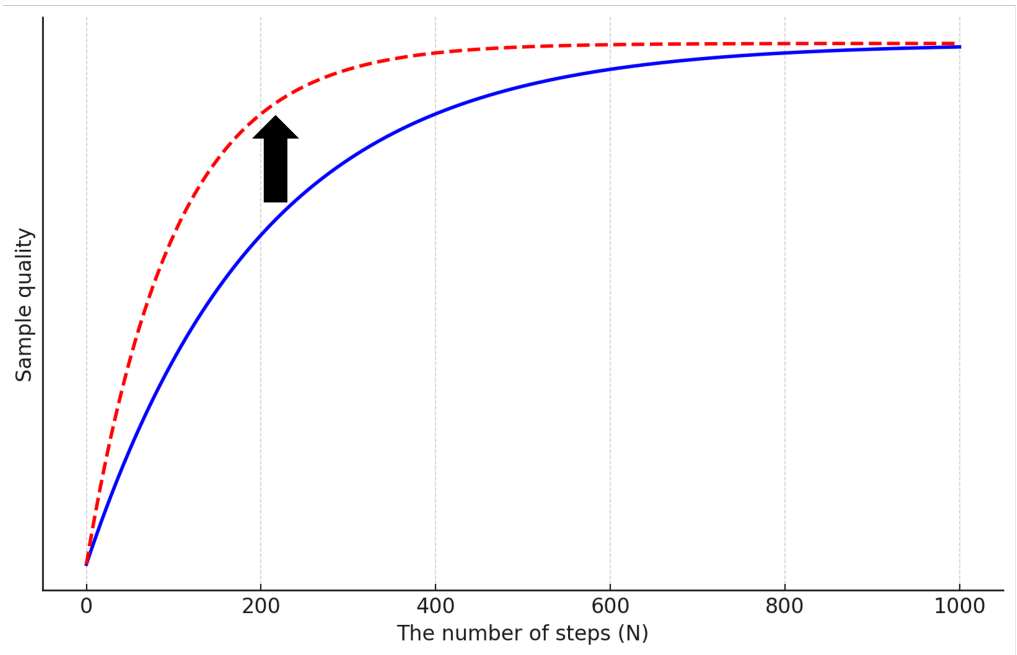


Motivation

Efficient sampling in diffusion models

Improving ODE Solving Accuracy

- Utilizing higher-order samplers and extrapolation methods deliver more accurate solutions for the same number of computational steps.
- Impact: Effectively pushes the ”steps-quality” performance curve upwards.



Motivation

Efficient sampling in diffusion models

- These methods are sequentially estimating points in the ODE trajectory.
- They trade **quality** for **speed**.
- Instead, can we trade **computation** for **speed**?
  - Better for interactive senarios.

Parallel in Time (PinT)

PinT algorithm refers to a class of numerical methods designed to solve time-dependent problems by **exploiting parallelism across time domain**.

- Iterative correction methods : Parareal, PITA
- Multigrid-in-Time methods : MGRIT, PFASST
- Space-time methods : Picard iteration, Pseudospectral in Time

Today, the presentation introduces two papers that leverage PinT algorithm: **Parareal** and **Picard iteration**.

ParaDiGMS [Shih et al., 2023]

Parallel Sampling of Diffusion Models

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, Nima Anari
Computer Science, Stanford University
{andyshih,belkhale,ermon,dorsa,anari}@cs.stanford.edu

ParaDiGMS [Shih et al., 2023]

Concept

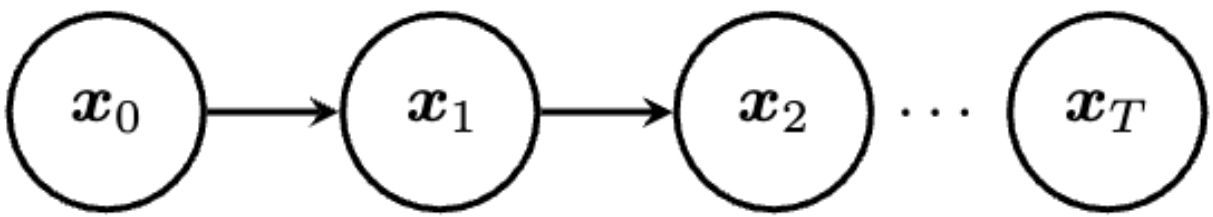
The sequential solvers numerically solve

$$dx_t = s(x_t, t)dt, \qquad x_0 = x_0, \tag{1}$$

sequentially in pseudo-time (t).

$$x_t = x_0 + \int_0^t s(x_u, u)du, \tag{2}$$

$$\Rightarrow \; x_t = x_0 + \frac{1}{T}\sum_{i=0}^{t-1} s(x_i, i/T) \tag{3}$$



ParaDiGMS [Shih et al., 2023]

Concept

ParaDiGMS is motivated from **Picard iteration**, which is an algorithm to solve **fixed-point problem**:

$$f(x_{0:T}) = x_{0:T}. \tag{4}$$

Fixed-point problem **iteratively refines the ODE trajectory** $x_{0:T}$.

ParaDiGMS [Shih et al., 2023]

Concept

Sequential solver

$$x_t = x_0 + \frac{1}{T}\sum_{i=0}^{t-1} s(x_i, i/T). \tag{5}$$

Picard iteration

$$x_t^{(k+1)} := x_0^{(k)} + \frac{1}{T}\sum_{i=0}^{t-1} s(x_i^{(k)}, i/T). \tag{6}$$

Then, for a function $h_i(x) := x + s(x, i/T)/T$,

$$x_t^{(k+1)} = h_{t-1}(x_{t-1}^{(k)}) \tag{7}$$

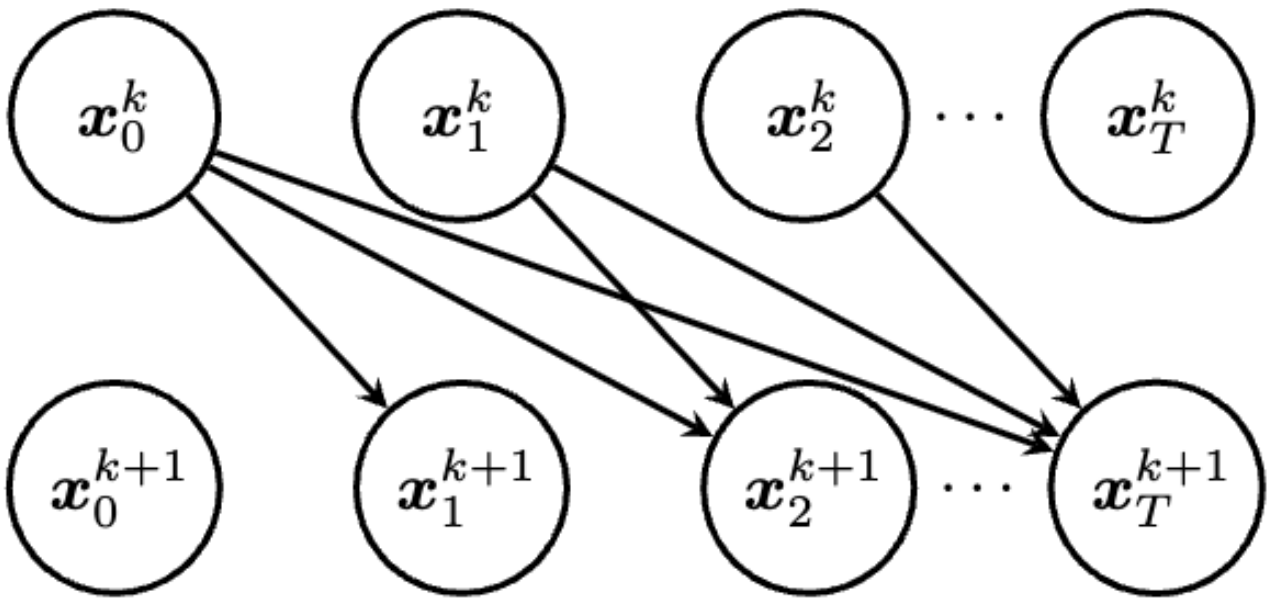
$$\begin{bmatrix} x_1^{(k+1)} \\ \vdots \\ x_T^{(k+1)} \end{bmatrix} = \begin{bmatrix} h_0(x_0^{(k)}) \\ \vdots \\ h_{T-1}(x_{T-1}^{(k)}) \end{bmatrix} \tag{8}$$

ParaDiGMS [Shih et al., 2023]

Concept

Picard iteration iteratively refines the estimation of the sequence $x_{0:T}^{(k)}$.

$$\begin{bmatrix} x_1^{(k+1)} \\ \vdots \\ x_T^{(k+1)} \end{bmatrix} = \begin{bmatrix} h_0(x_0^{(k)}) \\ \vdots \\ h_{T-1}(x_{T-1}^{(k)}) \end{bmatrix} \tag{9}$$



ParaDiGMS [Shih et al., 2023]

Concept

Proposition 1

For numerical solution of sequential solver, $x_t^* = h_{t-1}(\dots h_0(x_0))$,

$$s(x_i^{(k)}, i/T) = s(x_i^*, i/T) \quad \forall i \leq t \quad \Rightarrow \quad x_{t+1}^{(k+1)} = x_{t+1}^*. \tag{10}$$

Proposition 1 implies $x_{0:k}^{(k)} = x_{0:k}^*$. In other words, Picard iteration converges to sequential

ParaDiGMS [Shih et al., 2023]

Compute analysis

Sequential solver

Suppose we are solving ODEs with T-steps, and we have computing power to calculate B samples per second simultaneously.

$$x_{t+1}^{(t+1)} = h_{t+1}(x_t^{(t)}). \tag{11}$$

- Number of iterations: T
- NFE per iteration: 1
- Total NFE: $T \times 1 = T$
- Effective parallel operations : 1
- Total Time: $T/1 = T$

ParaDiGMS [Shih et al., 2023]

Compute analysis

Picard iteration

$$\begin{bmatrix} x_1^{(k+1)} \\ \vdots \\ x_T^{(k+1)} \end{bmatrix} = \begin{bmatrix} h_1(x_0^{(k)}) \\ \vdots \\ h_T(x_{T-1}^{(k)}) \end{bmatrix} \tag{12}$$

- Number of iterations: K
- NFE per iteration: T
- Total NFE: $T \times K = TK$
- Effective parallel operations : $\min(B, T)$
- Total Time: $TK / \min(B, T)$

ParaDiGMS [Shih et al., 2023]

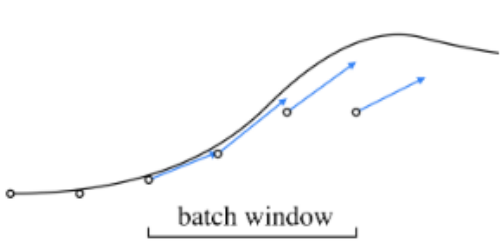
Implementation

Directly implementing Picard iteration on diffusion models has few practical challenges.

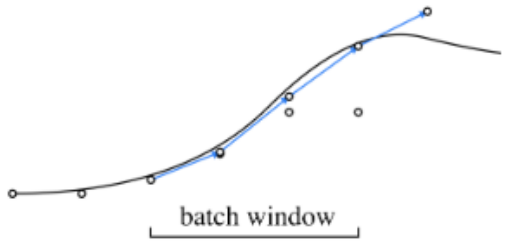
1. Performing iteration to whole $x_{0:T}$ is computationally heavy.
   - Picard iteration on $x_{t:t+p}$ with sliding window approach.
   - Sliding or stopping criterion.
2. Unable to perform stochastic sampling.
   - Sample the noise up-front and absorb stochasticity into the drift.

ParaDiGMS [Shih et al., 2023]

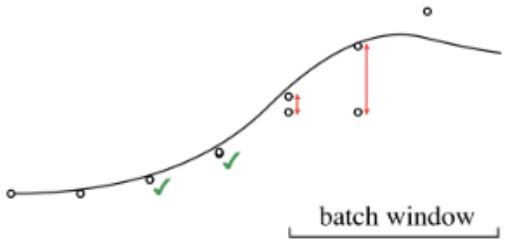
Implementation - sliding window



(a) Compute the drift of $\boldsymbol{x}_{t:t+p}^{k}$ on a batch window of size $p = 4$, in parallel



(b) Update the values to $\boldsymbol{x}_{t:t+p}^{k+1}$ using the cumulative drift of points in the window



(c) Determine how far to slide the window forward, based on the error $\|\boldsymbol{x}_i^{k+1} - \boldsymbol{x}_i^k\|^2$.

Proposition 2

Assuming that $||x_t^{(k+1)} - x_t^{(k)}||_2^2 \leq \frac{1}{2}||x_t^{(k)} - x_t^{(k-1)}||_2^2$, *using the tolerance* $||x_t^{(k)} - x_t^{(k-1)}||_2^2 \leq 4\epsilon^2\sigma_t^2/T^2$ *ensures that samples of* x_T^K *are drawn from a distribution with total variation distance at most* ϵ *from the DDPM model distribution.*

ParaDiGMS [Shih et al., 2023]

Implementation - stochastic sampling

Picard iteration is originally designed for solving ODEs. ParaDiGMS absorbs the stochastic term of SDE into drift term by sampling stochasticity up-front.

$$dx_t = s(x_t, t)dt + g(t)dW_t, \qquad x_0 = x_0. \tag{13}$$

$$\Rightarrow \; x_t = x_t + \frac{1}{T}\sum_{i=0}^{t-1} s(x_i, i/T) + g(i/T)z_i, \qquad z_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{14}$$

By fixing z_i for each i, stochasticity can be implemented to ParaDiGMS

ParaDiGMS [Shih et al., 2023]

Implementation

Algorithm 1: ParaDiGMS: parallel sampling via Picard iteration over a sliding window

Input: Diffusion model p_θ with variances σ_t^2, tolerance τ, batch window size p, dimension D
Output: A sample from p_θ

$t, k \leftarrow 0, 0$
$\boldsymbol{z}_i \sim \mathcal{N}(\mathbf{0}, \sigma_i^2 \boldsymbol{I}) \quad \forall i \in [0, T)$ // Up-front sampling of noise (for SDE)
$\boldsymbol{x}_0^k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \qquad \boldsymbol{x}_i^k \leftarrow \boldsymbol{x}_0^k \quad \forall i \in [1, p]$ // Sample initial condition from prior
while $t < T$ **do**
  $\boldsymbol{y}_{t+j} \leftarrow p_\theta(\boldsymbol{x}_{t+j}^k, t+j) - \boldsymbol{x}_{t+j}^k \quad \forall j \in [0, p)$ // Compute drifts in parallel
  $\boldsymbol{x}_{t+j+1}^{k+1} \leftarrow \boldsymbol{x}_t^k + \sum_{i=t}^{t+j} \boldsymbol{y}_i + \sum_{i=t}^{t+j} \boldsymbol{z}_i \quad \forall j \in [0, p)$ // Discretized Picard iteration
  error $\leftarrow \{\frac{1}{D} \|\boldsymbol{x}_{t+j}^{k+1} - \boldsymbol{x}_{t+j}^k\|^2 : \forall j \in [1, p)\}$ // Store error value for each timestep
  stride $\leftarrow \min\left(\{j : \text{error}_j > \tau^2 \sigma_j^2\} \cup \{p\}\right)$ // Slide forward until we reach tolerance
  $\boldsymbol{x}_{t+p+j}^{k+1} \leftarrow \boldsymbol{x}_{t+p}^{k+1} \quad \forall j \in [1, \text{stride}]$ // Initialize new points that the window now covers
  $t \leftarrow t + \text{stride}, \qquad k \leftarrow k + 1$
  $p \leftarrow \min(p, T - t)$
return $\boldsymbol{x}_T^k$

ParaDiGMS [Shih et al., 2023]

Compute analysis

	# Iter.	NFE/iter.	# NFE	Parallelization	Time
Sequential solver	T	1	T	1	T
Picard iteration	K	T	KT	$\min(B, T)$	$KT/\min(B, T)$
ParaDiGMS	K	p	Kp	$min(B, p)$	$KP/min(B, p)$

ParaDiGMS [Shih et al., 2023]

Experiments

Diffusion policy

Table 1: Robosuite Square with ParaDiGMS using a tolerance of $\tau = 0.1$ and a batch window size of 20 on a single A40 GPU. Reward is computed using an average of 200 evaluation episodes, with sampling speed measured as time to generate $400/8 = 50$ samples.

Robosuite Square	Sequential			ParaDiGMS				Speedup
	Model Evals	Reward	Time per Episode	Model Evals	Parallel Iters	Reward	Time per Episode	
DDPM	100	0.85 ± 0.03	37.0s	392	25	0.85 ± 0.03	10.0s	3.7x
DDIM	15	0.83 ± 0.03	5.72s	47	7	0.85 ± 0.03	3.58s	1.6x
DPMSolver	15	0.85 ± 0.03	5.80s	41	6	0.83 ± 0.03	3.28s	1.8x

Image diffusion

Table 4: Evaluating CLIP score of ParaDiGMS on StableDiffusion-v2 over 1000 random samples from the COCO2017 captions dataset, with classifier guidance $w = 7.5$. CLIP score is evaluated on ViT-g-14, and sample speed is computed on A100 GPUs.

Stable Diffusion-v2	Sequential			ParaDiGMS					Speedup
	Model Evals	CLIP Score	Time per Sample	Tol. τ	Model Evals	Parallel Iters	CLIP Score	Time per Sample	
DDPM	1000	32.1	50.0s	1e-1	2504	50	32.1	14.6s	3.4x
DPMSolver	200	31.7	10.3s	1e-1	422	15	31.7	2.6s	4.0x
DDIM	200	31.9	10.3s	1e-1	432	15	31.9	2.6s	4.0x
DDIM	100	31.9	5.3s	5e-2	229	19	31.9	2.0s	2.7x
DDIM	50	31.9	2.6s	5e-2	91	17	31.9	1.1s	2.4x
DDIM	25	31.7	1.3s	1e-2	93	17	31.7	1.0s	1.3x

ParaDiGMS [Shih et al., 2023]

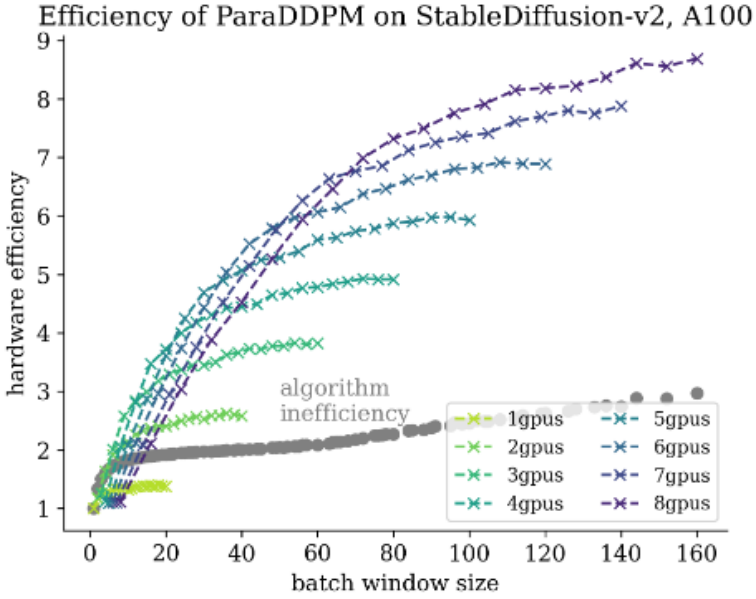
Experiments

(a) "a beautiful castle, matte painting"

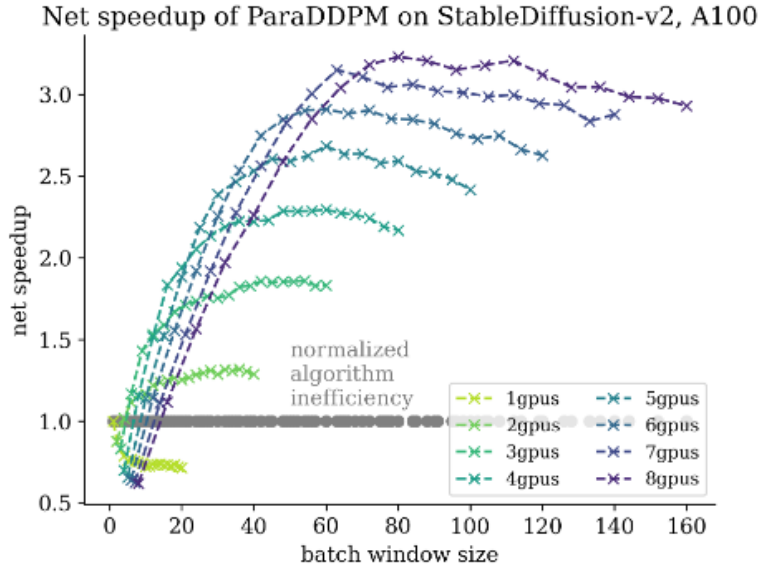
(b) "a batter swings at a pitch during a baseball game"

(c) "several sail boats in the water at night"

(d) "a grey suitcase sits in front of a couch"



(e) Hardware efficiency overtakes algorithm inefficiency as number of GPUs increase.



(f) Over 3x net wall-clock speedup for 1000-step ParaDDPM

SRDS [Selvam et al., 2024]

Self-Refining Diffusion Samplers: Enabling Parallelization via Parareal Iterations

Nikil Roashan Selvam **Amil Merchant** **Stefano Ermon**
Department of Computer Science
Stanford University
{nrs,amil,ermon}@cs.stanford.edu

SRDS [Selvam et al., 2024]

Concept

SRDS is motivated from **Parareal** (Parallel in real time). Parareal algorithm utilizes two solvers:

A slow but accurate fine solver $\mathcal{F}$

$$x_{i+1} = \mathcal{F}(x_i, t_i, t_{i+1}) \tag{15}$$

A fast but inaccurate coarse solver $\mathcal{G}$

$$x_{i+1} = \mathcal{G}(x_i, t_i, t_{i+1}) \tag{16}$$

Rough idea of Parareal:

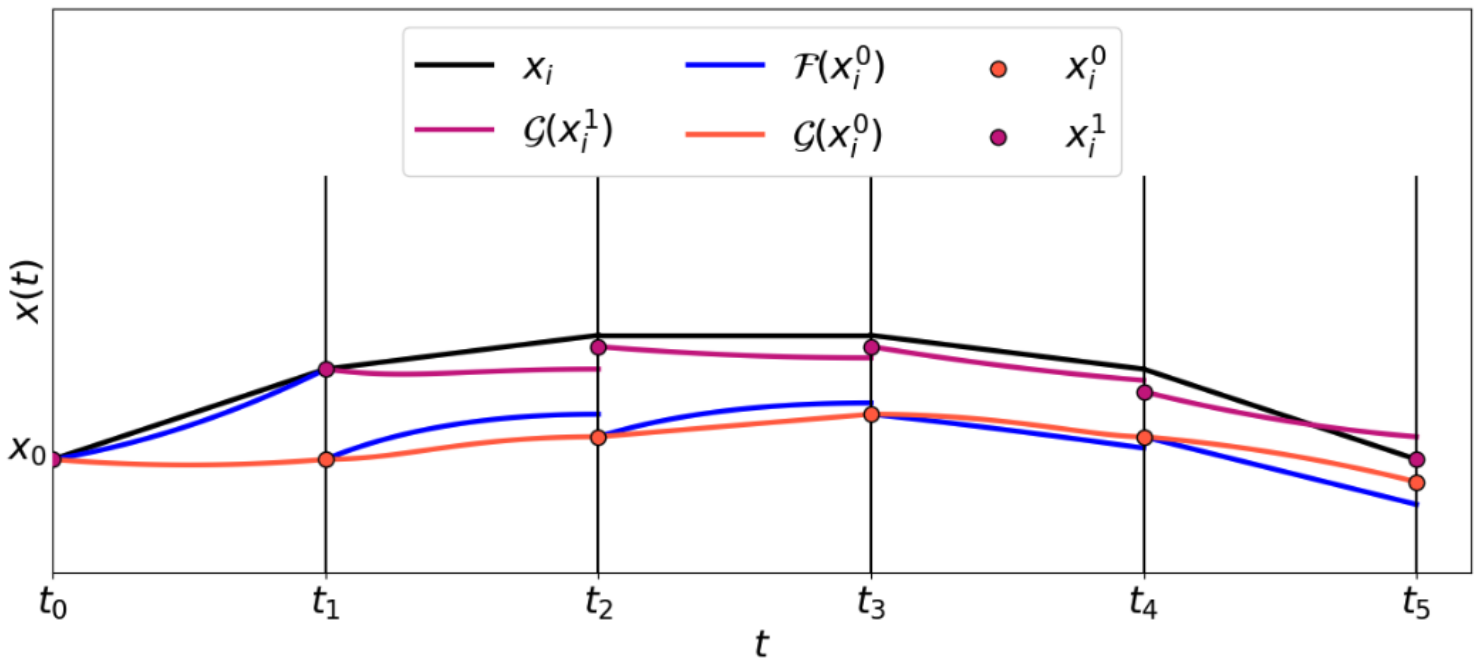
1. Corase solver solves ODE for $[t_o, t_N]$ sequentially.
2. Fine solver solves ODE for each $[t_i, t_{i+1}]$ in parallel.

SRDS [Selvam et al., 2024]

Concept

Parareal algorithm

1. $x_{i+1}^{(o)} = \mathcal{G}(x_i^{(o)})$
2. Compute $\mathcal{F}(x_i^{(k)})$ in parallel.
3. $x_{i+1}^{(k+1)} = \mathcal{G}(x_i^{(k+1)}) + (\mathcal{F}(x_i^{(k)}) - \mathcal{G}(x_i^{(k)}))$
4. Repeat 2 and 3 until convergence.



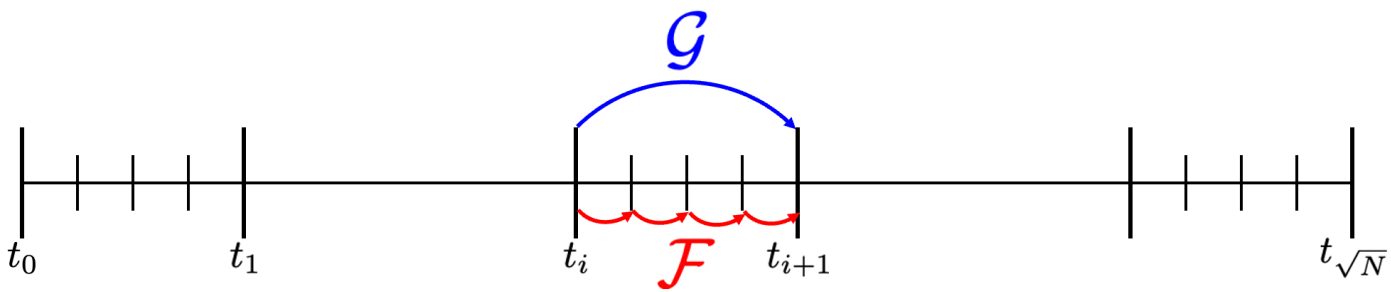
SRDS [Selvam et al., 2024]

Implementation

1. Consider a $\sqrt{N}$-discretization: $t_{i+1} = t_i + T/\sqrt{N}$.
2. Define coarse and fine solver with n-step solver $h(x_t, t, s, n)$.

$$\mathcal{G}(x_i, t_i, t_{i+i}) = h(x_t, t_i, t_{i+1}, 1) \tag{17}$$

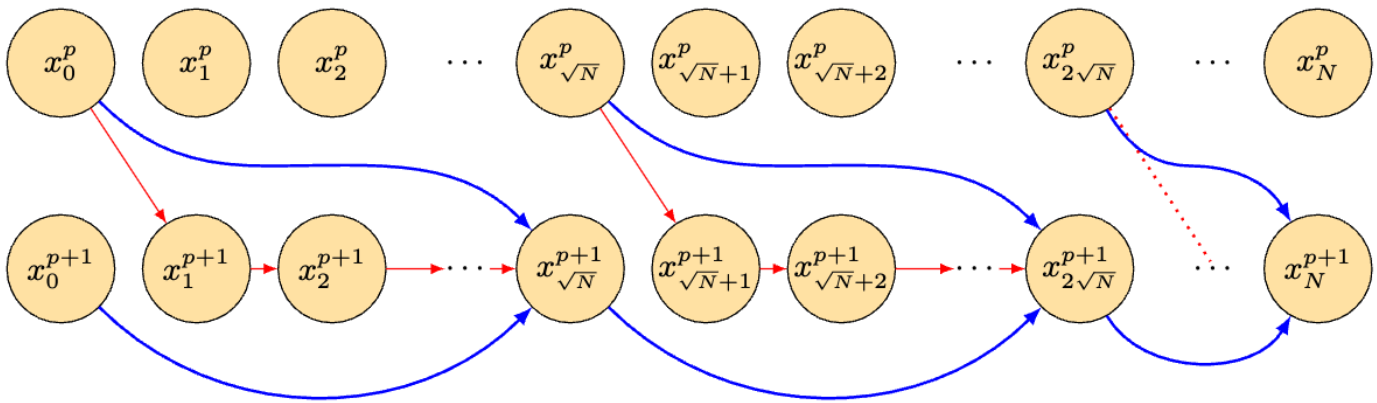
$$\mathcal{F}(x_i, t_i, t_{i+1}) = h(x_t, t_i, t_{i+1}, \sqrt{N}). \tag{18}$$



SRDS [Selvam et al., 2024]

Implementation

3. Apply Parareal until $||x^{(k+1)}_{\sqrt{N}} - x^{(k)}_{\sqrt{N}}|| < \tau$



(b) SRDS

SRDS [Selvam et al., 2024]

Implementation

Proposition 3

The sample output by SRDS converges to the outuput of the N-step sequential solver in at most $\sqrt{N}$ refinement iterations.

SRDS [Selvam et al., 2024]

Implementation

Algorithm 1 SRDS: Self-Refining Diffusion Sampler

Require: Diffusion model p_θ with denoising steps N, tolerance τ, and corresponding DDIM solver $h(\boldsymbol{x}, t_{start}, t_{end}, steps)$

Ensure: A sample from p_θ

$\boldsymbol{x}_0^0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ // Sample initial condition for Initial Value Problem from prior

for $i \leftarrow 1$ to $\sqrt{N}$ do

  $\boldsymbol{prev}_i \leftarrow h\left(\boldsymbol{x}_{i-1}^0, t_{i-1}, t_i, 1\right)$

  $\boldsymbol{x}_i^0 \leftarrow \boldsymbol{prev}_i$ // Initialize $\boldsymbol{x}$ with a coarse solve

$p \leftarrow 1$ // SRDS refinement iteration number

while $p \leq \sqrt{N}$ do

  for $i \leftarrow 1$ to $\sqrt{N}$ in parallel do

    $\boldsymbol{y}_i \leftarrow h\left(\boldsymbol{x}_{i-1}^{p-1}, t_{i-1}, t_i, \sqrt{N}\right)$ // Perform fine solves in parallel

  for $i \leftarrow 1$ to $\sqrt{N}$ do // Perform a coarse sweep

    $\boldsymbol{cur}_i \leftarrow h\left(\boldsymbol{x}_{i-1}^p, t_{i-1}, t_i, 1\right)$

    $\boldsymbol{x}_i^p \leftarrow \boldsymbol{y}_i + \boldsymbol{cur}_i - \boldsymbol{prev}_i$ // Take predictor-corrector step

    $\boldsymbol{prev}_i \leftarrow \boldsymbol{cur}_i$

  if $|\boldsymbol{x}_{\sqrt{N}}^p - \boldsymbol{x}_{\sqrt{N}}^{p-1}| < \tau$ then // Check for convergence

    break

  $p \leftarrow p + 1$

return $\boldsymbol{x}_{\sqrt{N}}^{p-1}$

SRDS [Selvam et al., 2024]

Compute analysis

SRDS

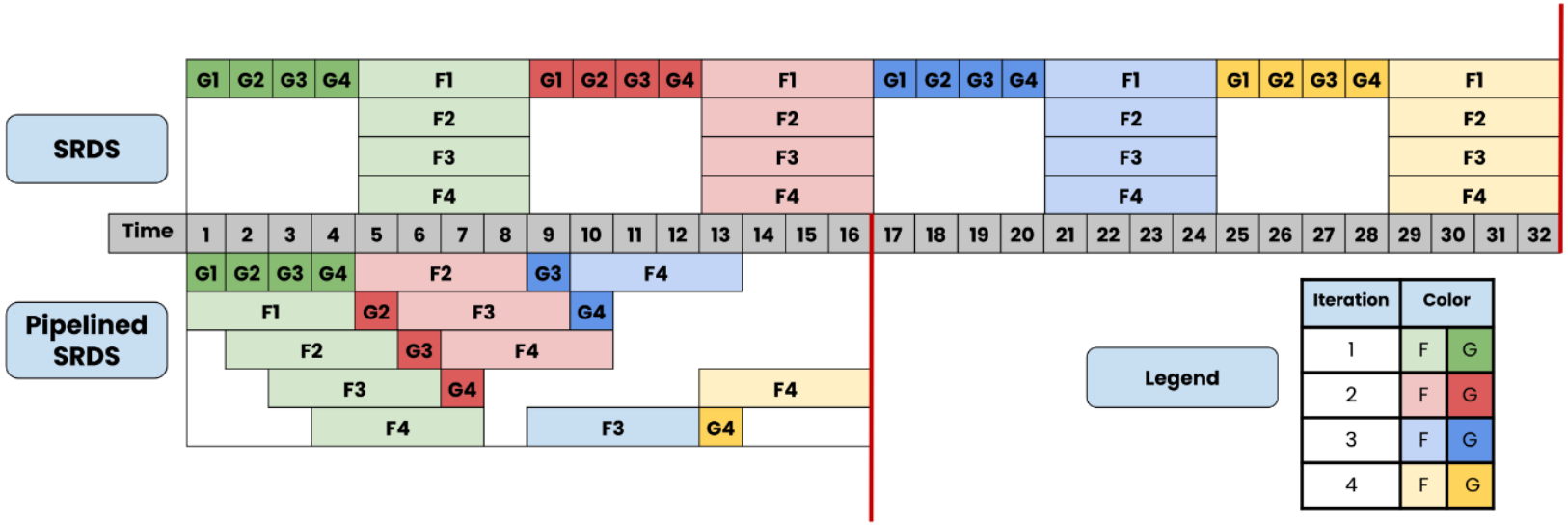
- Number of iterations: $K \leq \sqrt{N}$
- NFE per iteration: $\underbrace{\sqrt{N}}_{\mathcal{G}} + \underbrace{N}_{\mathcal{F}}$
- Total NFE: $K(\sqrt{N} + N)$
- Effective parallel operations : 1 for $\mathcal{G}$, $\min(B, \sqrt{N})$ for $\mathcal{F}$
- Total Time: $K(1 + N/\min(B, \sqrt{N}))$

SRDS [Selvam et al., 2024]

Implementation

Pipelining can accelerate SRDS more.

- Skip *i*-th segment after *i*-th iteration.
- Better GPU utilization.



SRDS [Selvam et al., 2024]

Compute analysis

Pipelined SRDS

- Number of iterations: $K \leq \sqrt{N}$
- NFE of k-th iteration: $\underbrace{\sqrt{N} - (k-1)}_{\mathcal{G}} + \underbrace{\sqrt{N}(\sqrt{N} - (k-1))}_{\mathcal{F}}$
- Total NFE: $K(\sqrt{N} - \frac{K-1}{2}) + K\sqrt{N}(\sqrt{N} - \frac{K-1}{2})$

SRDS [Selvam et al., 2024]

Experiments

Table 1: Evaluating FID score (lower is better) of SRDS on various datasets using 5000 samples generated using a DDIM solver. Effective serial evals refers to the number of serial model evaluations taken by the pipelined SRDS algorithm (counting all model evaluations simultaneously performed in parallel as one evaluation) . Total evals refers to the total number of model evaluations.

Dataset	Sequential		SRDS			
	Serial Evals	FID Score	SRDS Iters	Eff. Serial Evals	Total Evals	FID Score
LSUN Church	1024	12.8	5.7	209	5603	12.8
LSUN Bedroom	1024	10.0	5.8	212	5692	10.0
Imagenet	1024	9.0	4.6	175	4612	9.0
CIFAR	1024	7.6	3.7	147	3771	7.6

Table 2: CLIP scores of SRDS on StableDiffusion-v2 over 1000 samples from the COCO2017 captions dataset, with classifier guidance $w = 7.5$, evaluated on ViT-g-14. Time is measured on 4 A100 GPUs **without pipeline parallelism**, showcasing speedups with early convergence of the SRDS sample.

Stable Diffusion-v2	Sequential			Vanilla SRDS					
	Serial Evals	CLIP Score	Time per Sample	Max Iter	Eff. Serial Evals	Total Evals	CLIP Score	Time per Sample	Speedup
DDIM	100	31.9	4.6	1	19	119	31.9	2.0	2.3x
DDIM	25	31.7	1.2	1	9	34	31.4	0.8	1.5x
DDIM	25	31.7	1.2	3	17	74	31.9	1.7	0.7x

SRDS [Selvam et al., 2024]

Experiments

Table 3: Evaluation of additional speedup offered by pipelined version of SRDS.

Method	Serial Model Evals	SRDS Eff. Serial Evals	SRDS Time Per Sample	Pipelined SRDS Eff. Serial Evals	Pipelined SRDS Time Per Sample
DDIM	961	93	12.30	63	10.31
DDIM	196	42	3.30	27	2.85
DDIM	25	15	0.82	9	0.69

Table 4: Comparison of wallclock speedups offered by Pipelined SRDS and ParaDiGMS with various thresholds, with respect to Serial image generation. These StableDiffusion experiments are performed on identical machines (4 40GB A100 GPUs) for a fair comparison.

Method	Serial Model Evals	Serial Time Per Sample	Pipelined SRDS Time Per Sample	ParaDiGMS Threshold 1e-3	ParaDiGMS Threshold 1e-2	ParaDiGMS Threshold 1e-1
DDIM	961	44.88	10.31 (4.3x)	275.29	20.48	14.30
DDIM	196	9.17	2.85 (3.2x)	29.45	5.08	3.42
DDIM	25	1.18	0.69 (1.7x)	1.98	1.51	0.77

SRDS [Selvam et al., 2024]

Experiments

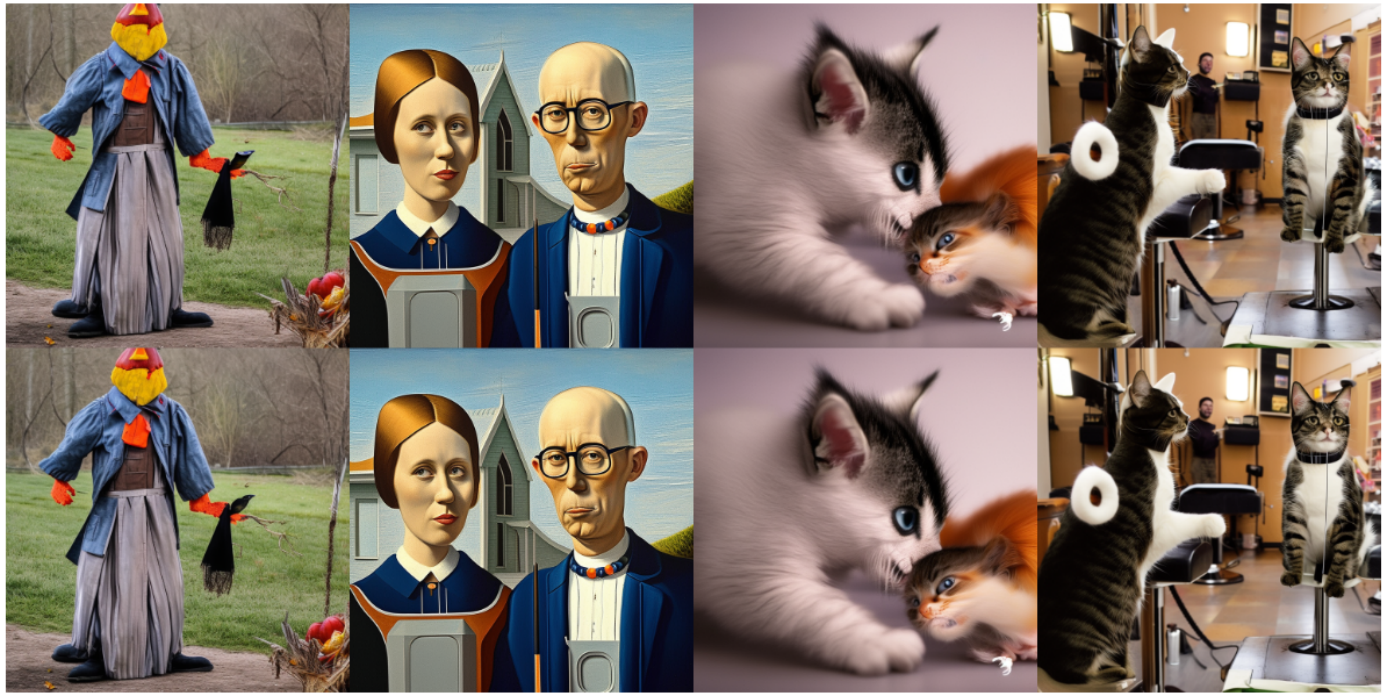


Figure 6: Sample generation from StableDiffusion-v2 with the SRDS algorithm with text prompts based on examples from DrawBench. We plot the early converged SRDS figure (top) and the result of the serial trajectory (bottom); the two rows are essentially indistinguishable, highlighting the approximation-free nature of SRDS.

SRDS [Selvam et al., 2024]

Experiments

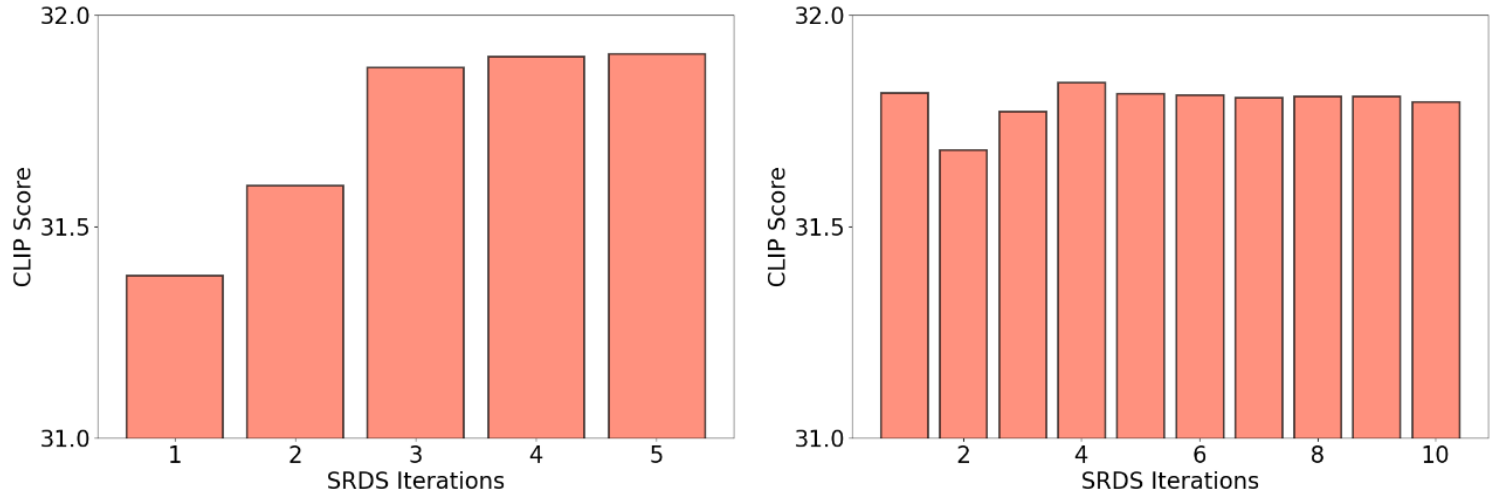


Figure 5: Convergence of the SRDS algorithm for a trajectory of length 25 (left) and 100 (right) showcase how early termination of the algorithm can yield equivalent sample quality. In particular, longer trajectories with increased parallelism appear to converge faster.

Reference I

Selvam, N. R., Merchant, A., and Ermon, S. (2024).

Self-refining diffusion samplers: Enabling parallelization via parareal iterations.

NeurIPS.

Shih, A., Belkhale, S., Ermon, S., Sadigh, D., and Anari, N. (2023).

Parallel sampling of diffusion models.

NeurIPS.