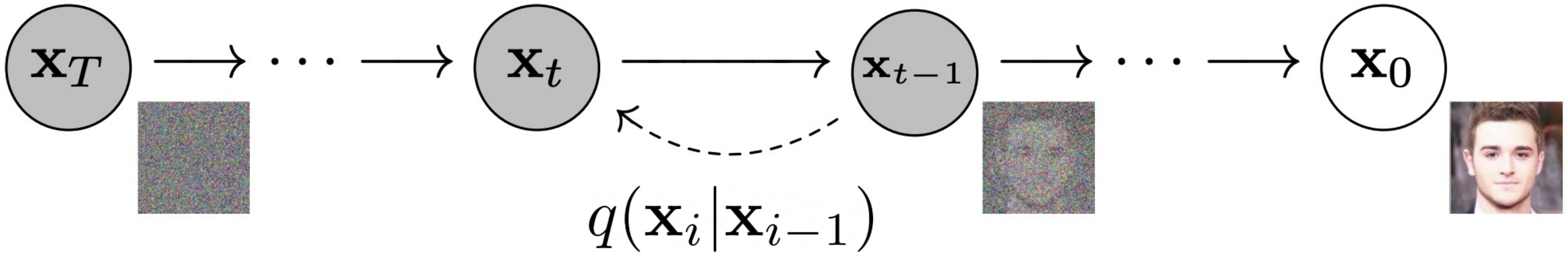
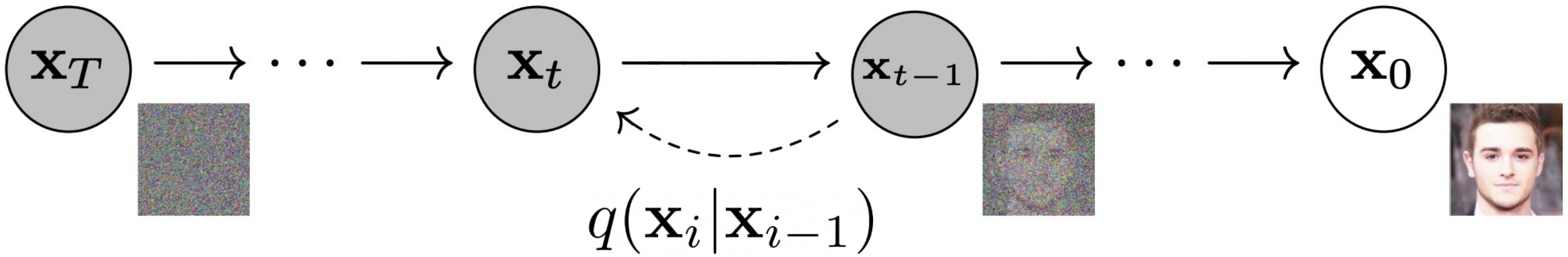
Parallel Sampling of Diffusion Models

with Suneel, Stefano, Dorsa, Nima



$$q(\mathbf{x}_i|\mathbf{x}_{i-1}) = \mathcal{N}(\mathbf{x}_i; \sqrt{1-\beta_i}\mathbf{x}_{i-1}, \beta_i\mathbf{I})$$

adding noise



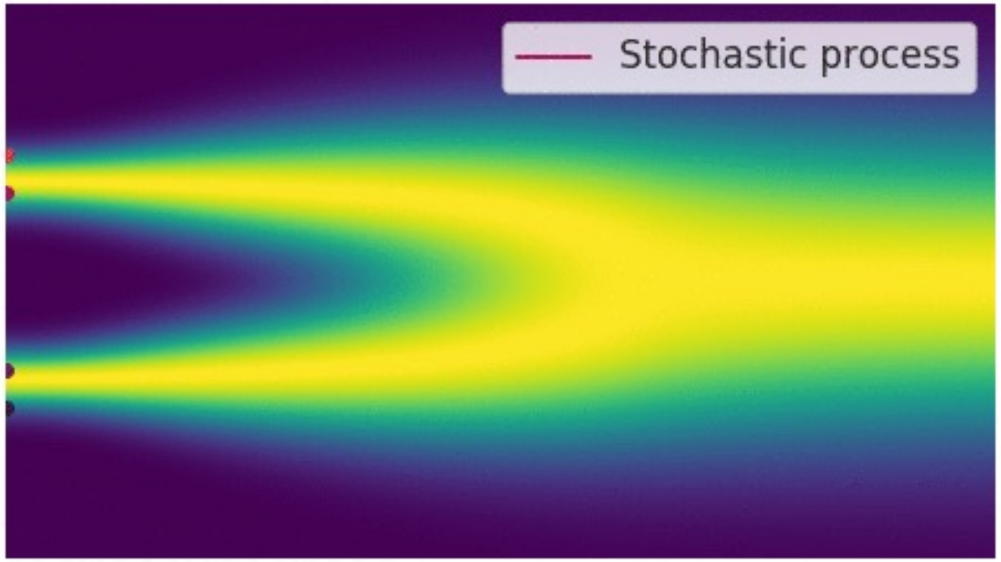
$$q(\mathbf{x}_i|\mathbf{x}_{i-1}) = \mathcal{N}(\mathbf{x}_i; \sqrt{1-\beta_i}\mathbf{x}_{i-1}, \beta_i \mathbf{I})$$

$$\mathbf{x}_i = \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\mathbf{z}_{i-1}, \quad i = 1, \cdots, N$$

adding noise

$$\mathrm{d}\boldsymbol{x}_t = f(t)\boldsymbol{x}_t \mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}_t$$

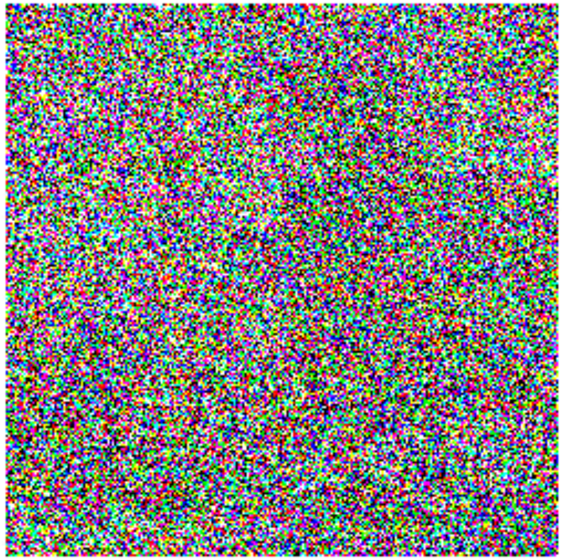
like an SDE!

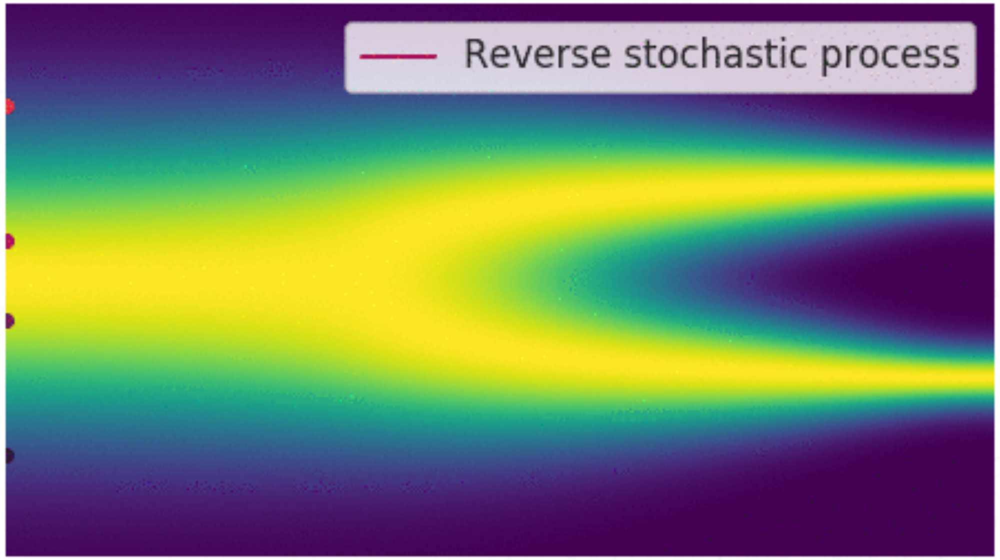


[Song 2021]

$$\mathrm{d}\boldsymbol{x}_t = f(t)\boldsymbol{x}_t\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{w}_t$$

f and g are
design choices





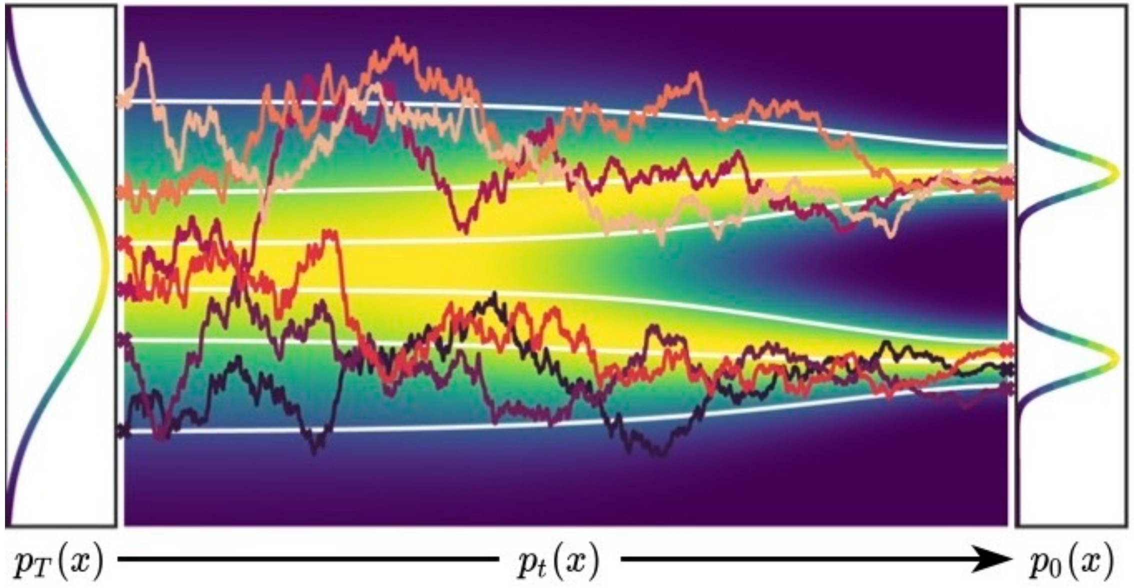
[Song 2021]

$$\mathrm{d}\boldsymbol{x}_t = [f(t)\boldsymbol{x}_t - g^2(t)\nabla_{\boldsymbol{x}} \log q_t(\boldsymbol{x}_t)]\mathrm{d}t + g(t)\mathrm{d}\bar{\boldsymbol{w}}_t$$

[Anderson 1982]

can reverse
in general,
still an SDE

	DDPM [Ho 2020]			
Sample Method	SDE (euler maruyama)			
Speed	Slow 1000 steps			
Quality	Best			



if we only care about marginals (the white lines)

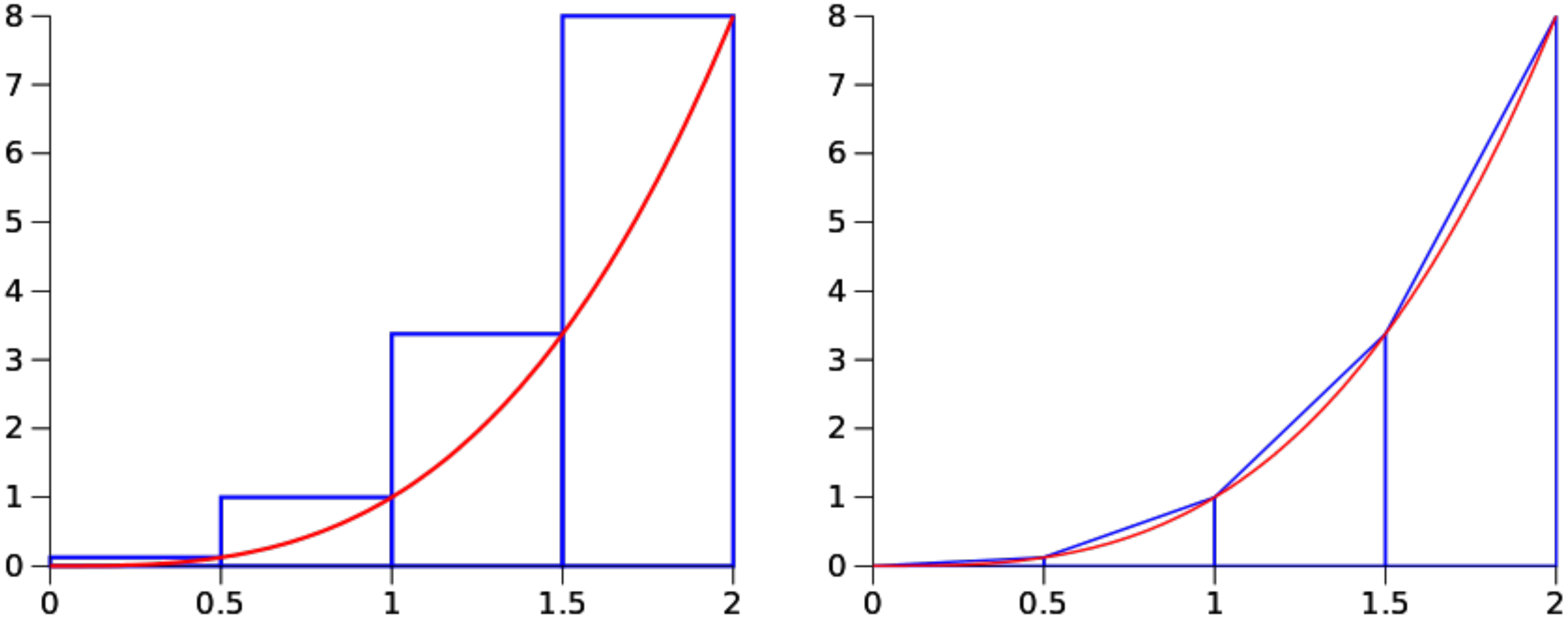
[Song 2021]

$$\frac{\mathrm{d}\boldsymbol{x}_t}{\mathrm{d}t} = f(t)\boldsymbol{x}_t - \frac{1}{2}g^2(t)\nabla_{\boldsymbol{x}} \log q_t(\boldsymbol{x}_t)$$

can also write as ODE!

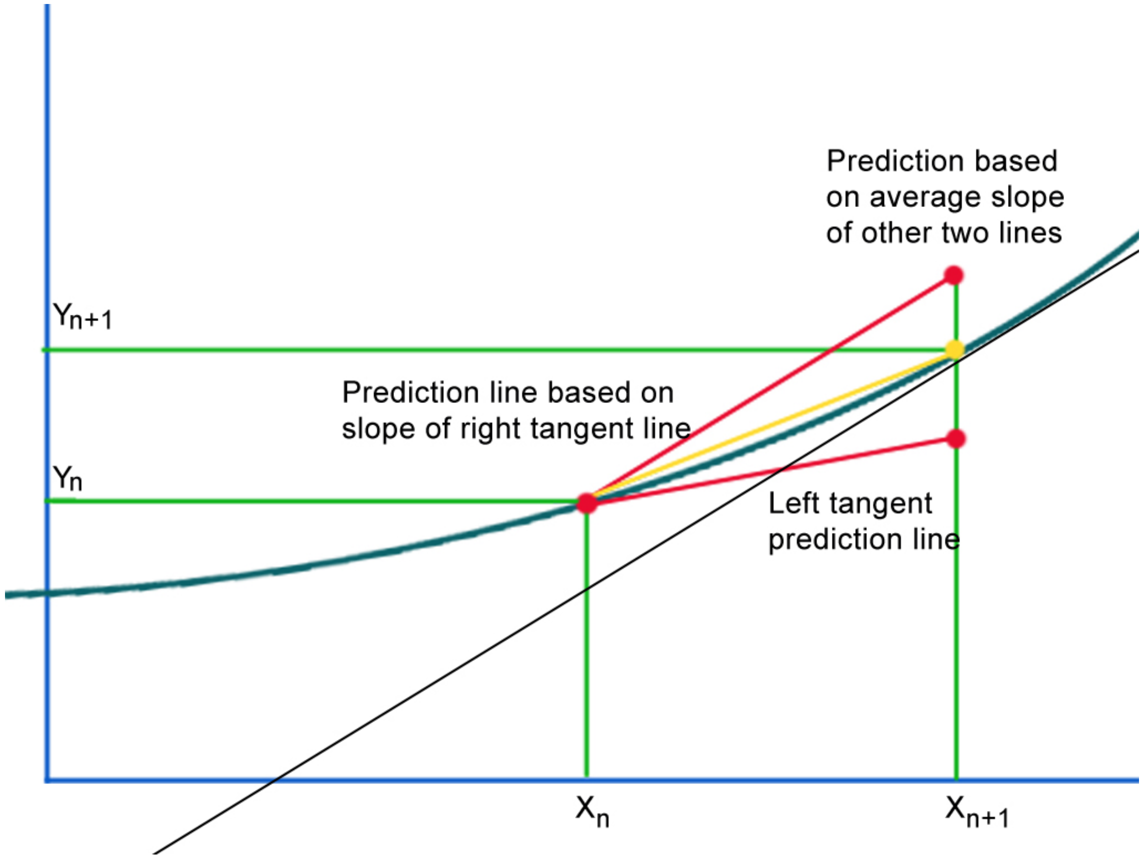
[Maoutsa 2020]

	DDPM [Ho 2020]	DDIM [Song 2021]	DPMSolver [Lu 2022]	
Sample Method	SDE (euler maruyama)	ODE (euler)	ODE (heun)	
Speed	Slow 1000 steps	Slow 1000 steps	Slow 1000 steps	
Quality	Best			



riemann : trapezoidal :: euler : heun

higher-order integration rule (heun's method)



wikipedia

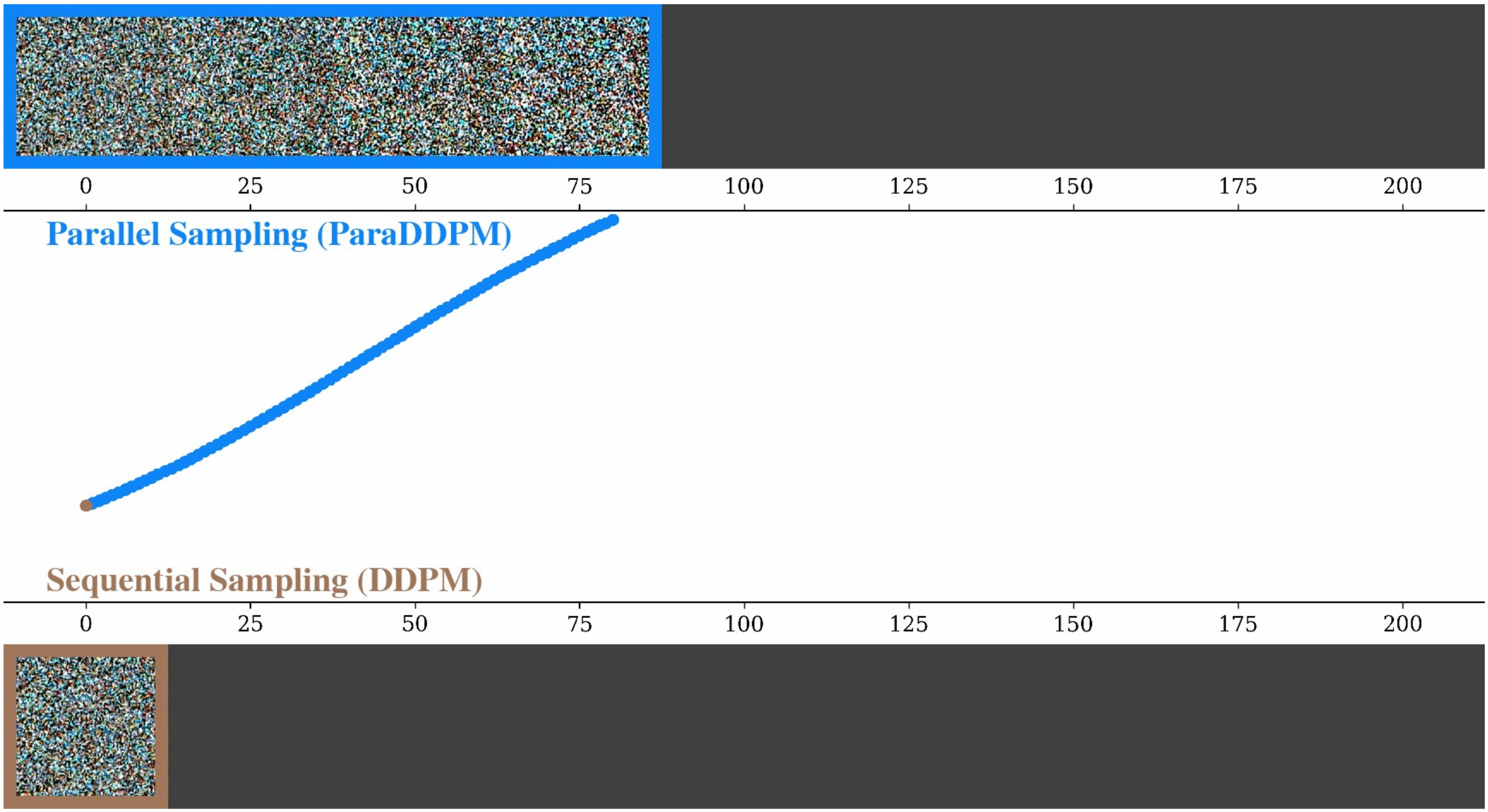
	DDPM [Ho 2020]	DDIM [Song 2021]	DPMSolver [Lu 2022]	
Sample Method	SDE (euler maruyama)	ODE (euler)	ODE (heun)	
Speed	Slow 1000 steps	Slow 1000 steps	Slow 1000 steps	
Quality	Best			

	DDPM [Ho 2020]	DDIM [Song 2021]	DPMSolver [Lu 2022]	
Sample Method	SDE (euler maruyama)	ODE (euler)	ODE (heun)	
Speed	Slow 1000 steps	Fast 50 steps	Fast 50 steps	
Quality	Best	Good	Good	

trade quality for speed

trade quality for speed

	DDPM [Ho 2020]	DDIM [Song 2021]	DPMSolver [Lu 2022]	ParaDiGMS [our method!]
Sample Method	SDE (euler maruyama)	ODE (euler)	ODE (heun)	ODE (picard+ euler/heun)
Speed	Slow 1000 steps	Fast 50 steps	Fast 50 steps	Fast 1000 steps
Quality	Best	Good	Good	Best
		trade quality for speed	trade quality for speed	trade compute for speed

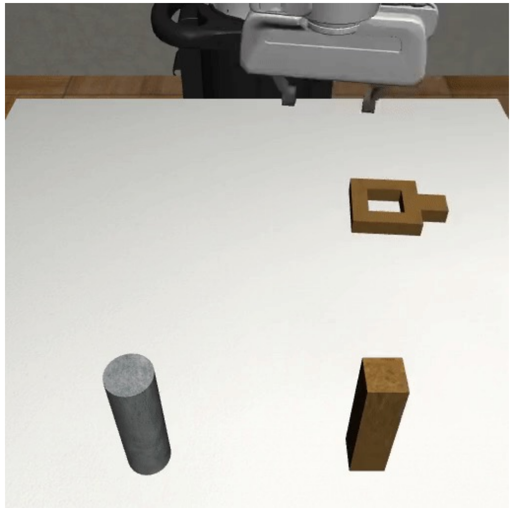


0
25
50
75
100
125
150
175
200
Parallel Sampling (ParaDDPM)
Sequential Sampling (DDPM)
0
25
50
75
100
125
150
175
200

Preview of results

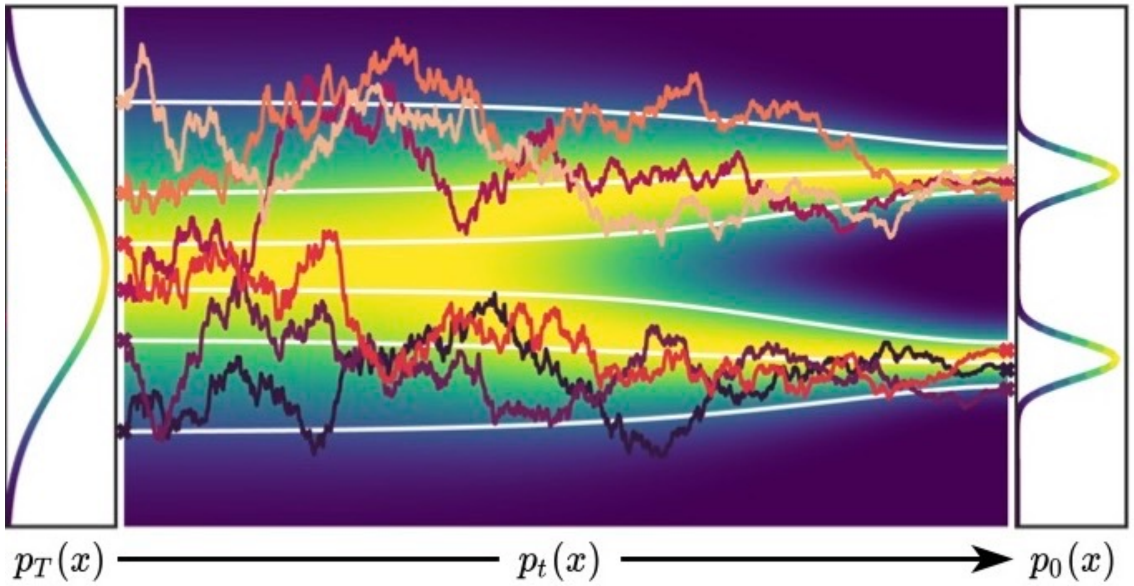
No drop in sample quality!!

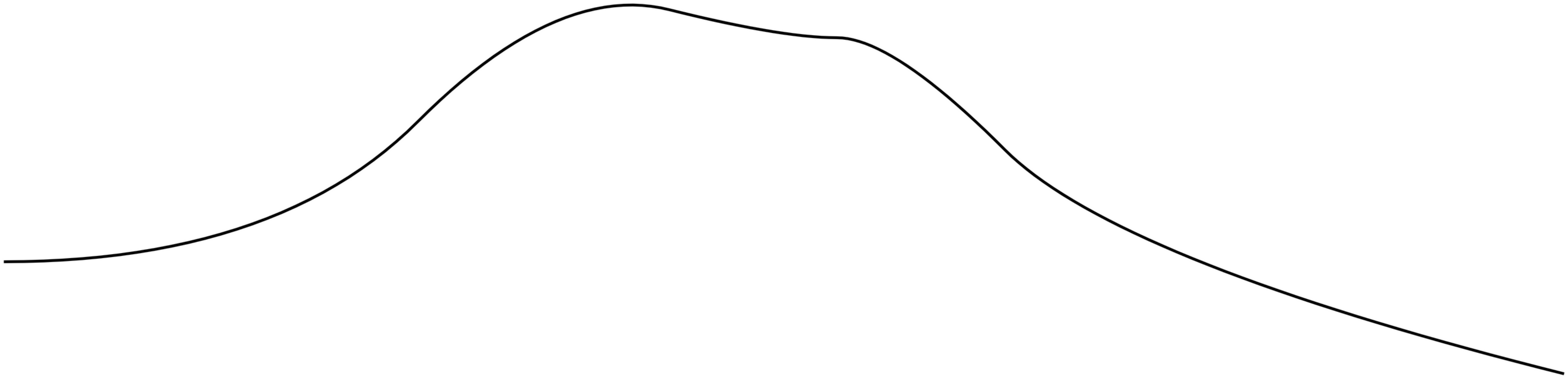
robosuite square 200 evaluation episodes	Method	DDPM	DDIM	DPE-solver
Base Sampler	Samples per Second	10.8 ± 0.6	70 ± 4	69 ± 4
ParaDiGM	Samples per Second	40 ± 2 **(3.7x faster)**	112 ± 7 **(1.6x faster)**	122 ± 7 **(1.8x faster)**



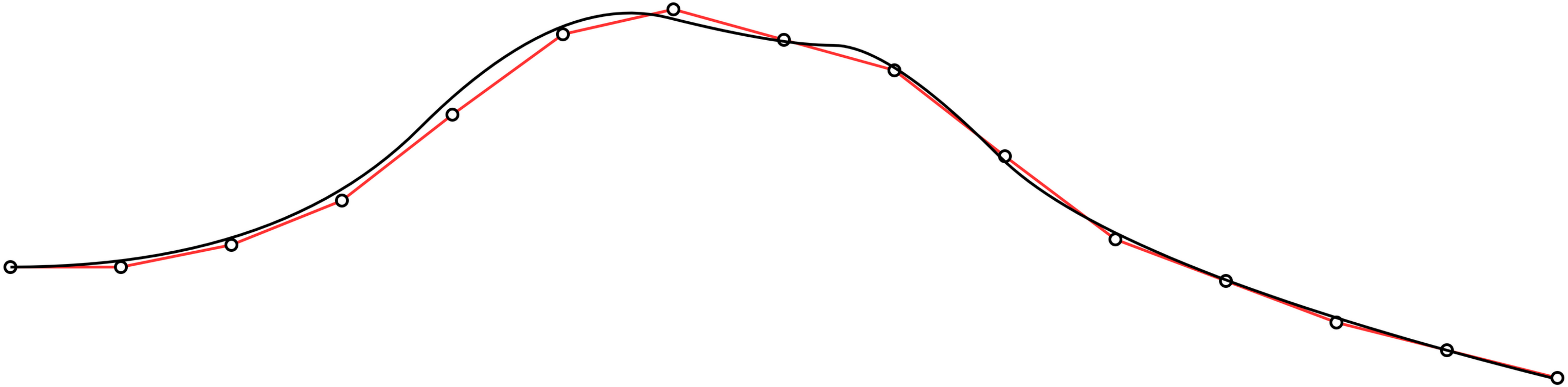
stable diffusion	Method	DDPM	DDIM	DDIM
Base Sampler	Samples per Minute	1.2	5.8	5.8
ParaDiGM	Samples per Minute	3.7 **(3.1x faster)**	10.4 **(1.8x faster)**	10.4 **(1.8x faster)**

we want to solve this (white lines) fast!





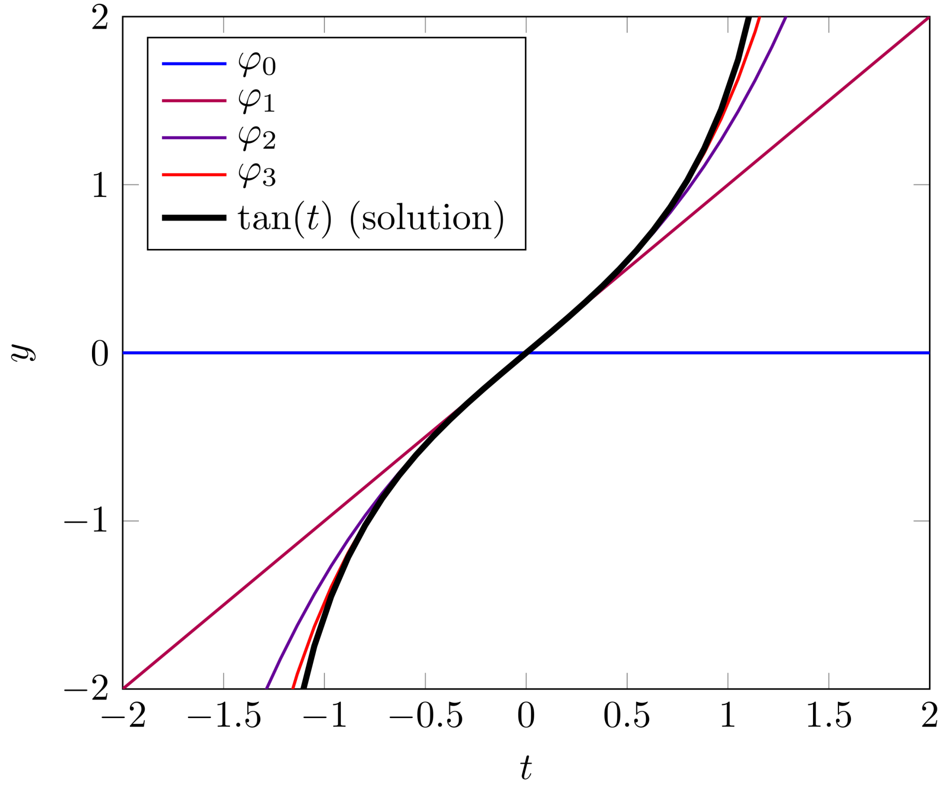
solve this ODE (only pointwise gradient information)



solve this ODE (only pointwise gradient information)
discretize, take one small step at a time

Picard–Lindelöf

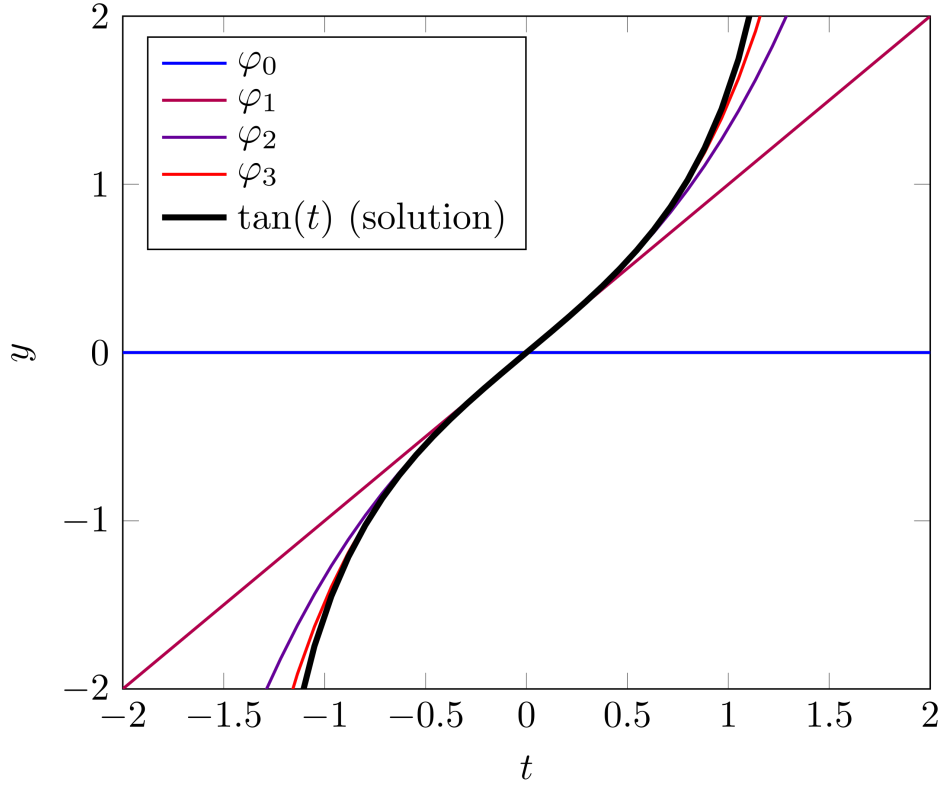
Solve (analytically) an ODE by iterating until convergence



$$\varphi_{k+1}(t) = y_0 + \int_{t_0}^{t} f(s, \varphi_k(s))ds$$

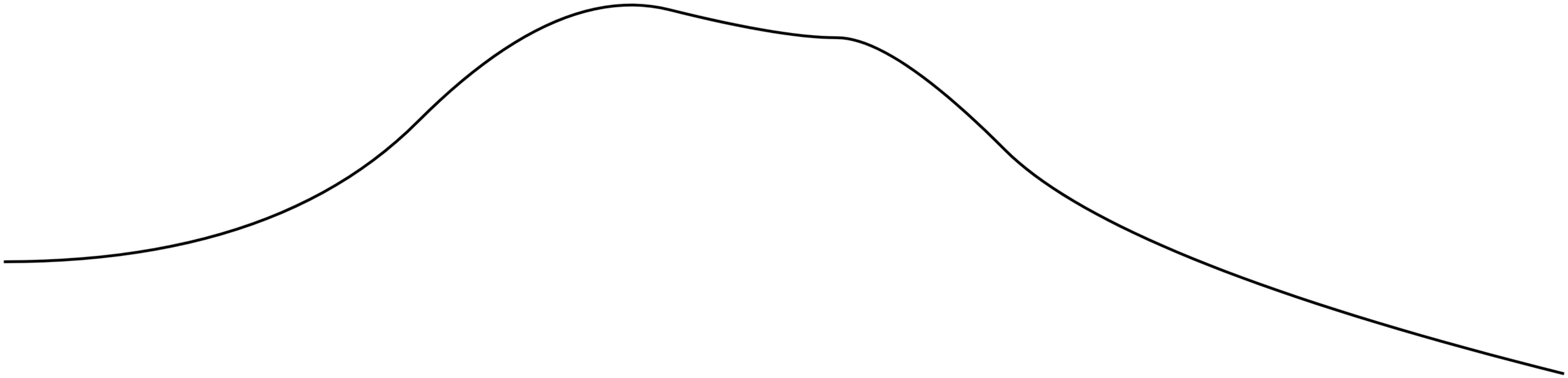
Picard + Euler

Solve discretized ODE by iterating until convergence

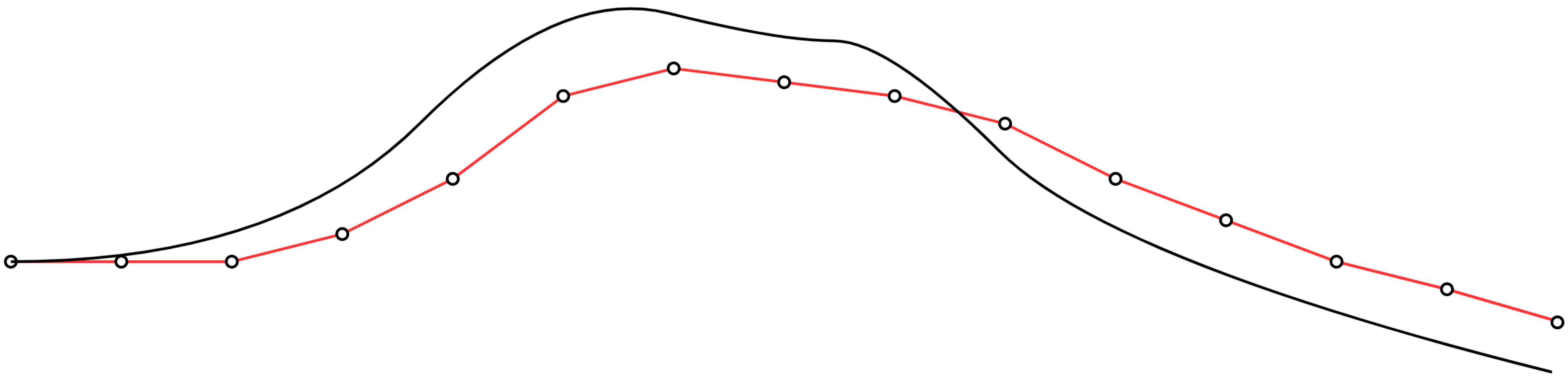


$$\varphi_{k+1}(t) = y_0 + \int_{t_0}^{t} f(s, \varphi_k(s))ds$$

$$\varphi_{k+1}(\frac{j}{N}) = y_0 + \frac{1}{N}\sum_{i=0}^{j-1} f(\frac{i}{N}, \varphi_k(\frac{i}{N}))$$



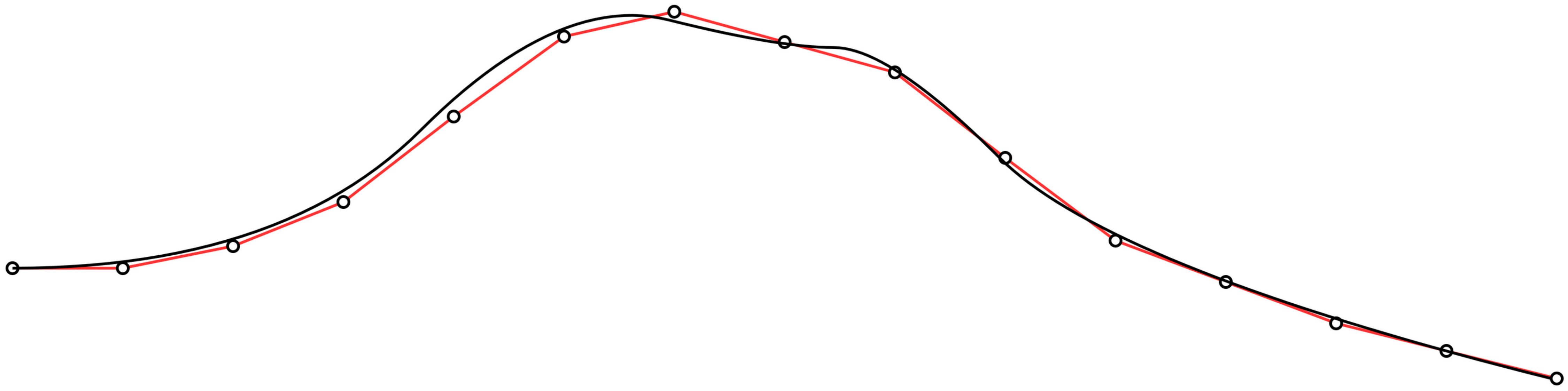
solve this ODE (only pointwise gradient information)



solve this ODE (only pointwise gradient information)

discretize, make a guess everywhere, iterate

$$\varphi_{k+1}\left(\frac{j}{N}\right) = y_0 + \frac{1}{N}\sum_{i=0}^{j-1} f\left(\frac{i}{N}, \varphi_k\left(\frac{i}{N}\right)\right)$$



solve this ODE (only pointwise gradient information)

discretize, make a guess everywhere, iterate…until convergence

$$\varphi_{k+1}(\frac{j}{N}) = y_0 + \frac{1}{N}\sum_{i=0}^{j-1} f(\frac{i}{N}, \varphi_k(\frac{i}{N}))$$

Practical Issues

- Isn’t that (number of iterations) * (number of steps)? Seems even slower!
  - Parallel computation! (GPUs, multi-GPUs)

Practical Issues

- Isn't that (number of iterations) * (number of steps)? Seems even slower!
  - Parallel computation! (GPUs, multi-GPUs)

- Out of memory? (1000x memory)
  - batching!

Practical Issues

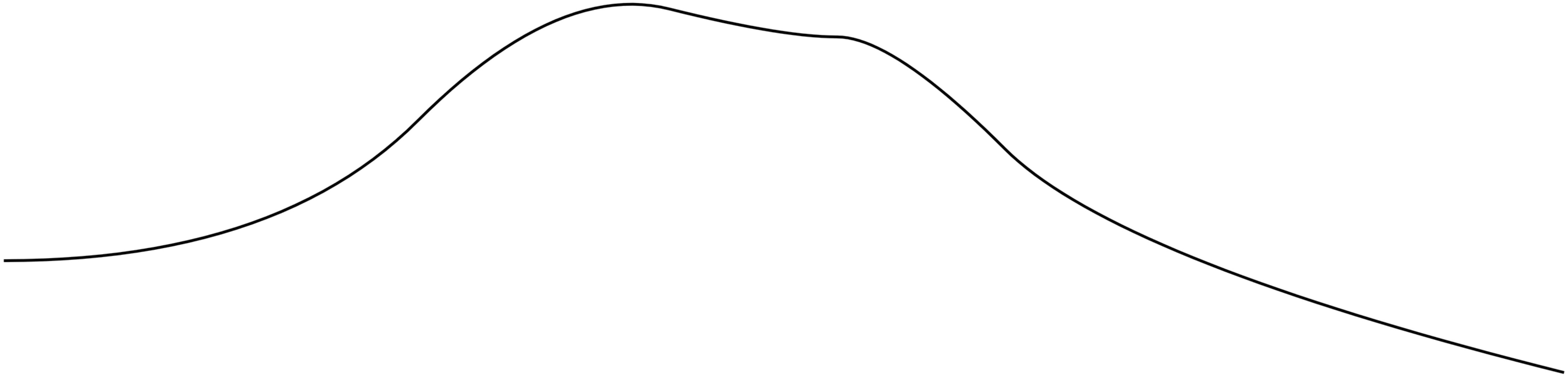
- Isn't that (number of iterations) * (number of steps)? Seems even slower!
  - Parallel computation! (GPUs, multi-GPUs)

- Out of memory? (1000x memory)
  - batching!

- Approximate method?
  - Yes, but in practice no quality degradation (w.r.t. standard metrics)!

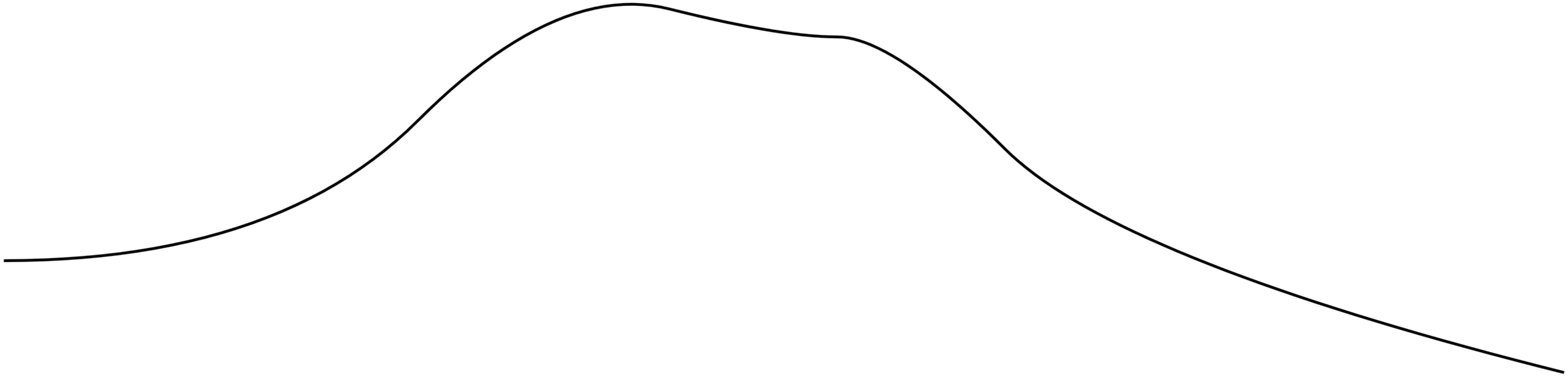
Batching

- Best explained with a GIF



Batching

- Window size
  - Can make small to fit on GPU
  - Even if no memory issues, still a good idea to batch!
    - Initial guesses at tail end of ODE is poor anyways, don’t bother with them



Batching

- Tolerance (when to slide forward)
    - too low = not much speedup, too high = risk of degradation
    - 0.1 * noise gives 2-4x speedup with no measurable degradation

Batching

- Tolerance (when to slide forward)
  - too low = not much speedup, too high = risk of degradation
  - 0.1 * noise gives 2-4x speedup with no measurable degradation

If $$||\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i||_2^2 \leq 4\epsilon^2\sigma_i^2/b^2$$

Then
$$D_{TV}(\mathcal{N}(\hat{\boldsymbol{x}}_i, \sigma_i^2\boldsymbol{I}) \,||\, \mathcal{N}(\boldsymbol{x}_i, \sigma_i^2\boldsymbol{I})) \leq \sqrt{\frac{1}{2}D_{KL}(\mathcal{N}(\hat{\boldsymbol{x}}_i, \sigma_i^2\boldsymbol{I}) \,||\, \mathcal{N}(\boldsymbol{x}_i, \sigma_i^2\boldsymbol{I}))}$$
$$= \sqrt{\frac{||\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i||_2^2}{4\sigma_i^2}} \leq \frac{\epsilon}{b}$$

So chance of faithful sample is $c = (1 - \frac{\epsilon}{b})^b \geq 1 - \epsilon$

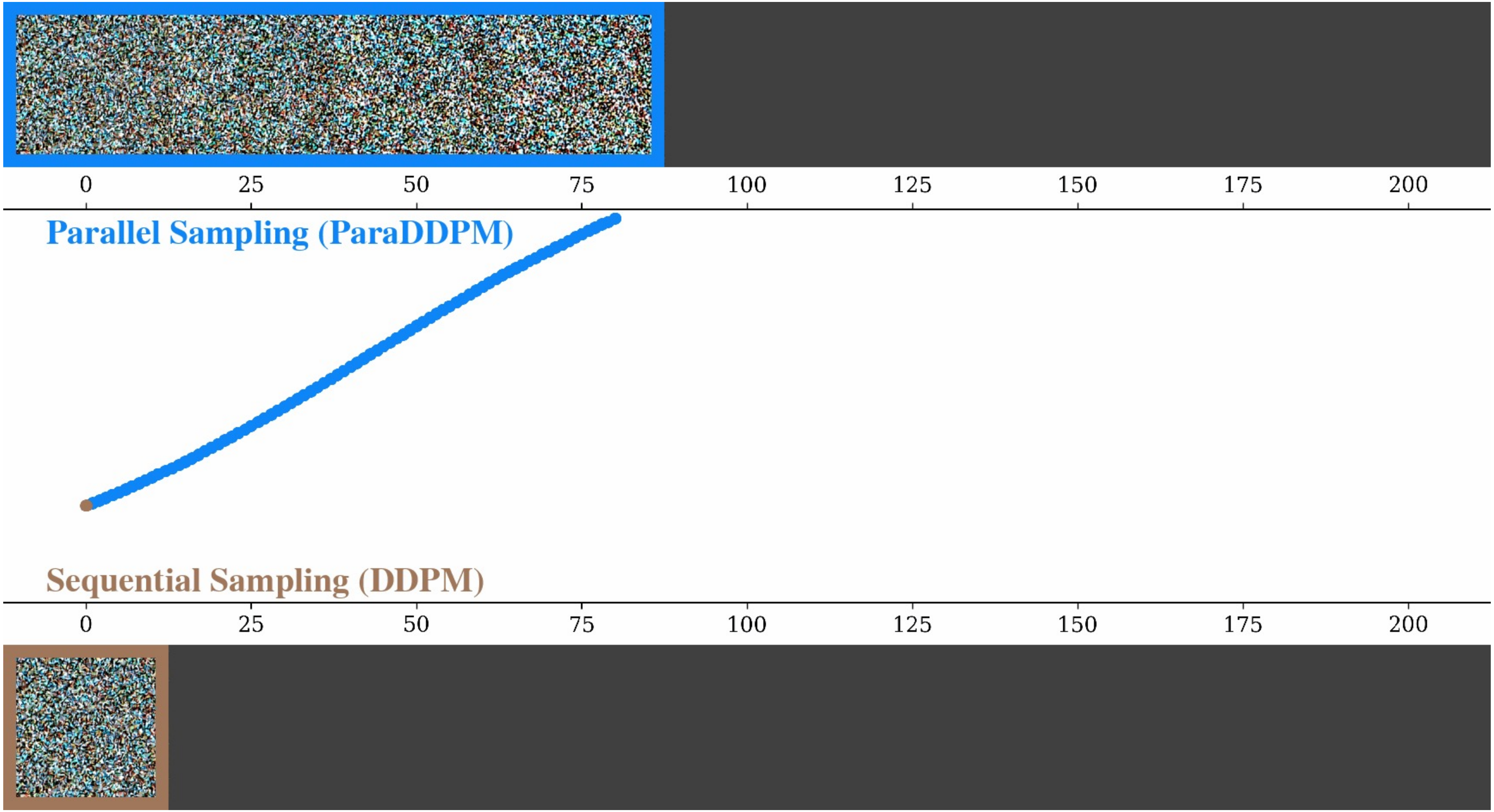
this is worst-case,
too conservative in practice

		DDPM	=	ParaDDPM	Picard + SDE
ParaDiGMS	+	DDIM	=	ParaDDIM	Picard + Euler
		DPMSolver	=	ParaDPMSolver	Picard + Heun

		DDPM	=	ParaDDPM	Picard + SDE 😟
ParaDiGMS	+	DDIM	=	ParaDDIM	Picard + Euler
		DPMSolver	=	ParaDPMSolver	Picard + Heun

		DDPM	=	ParaDDPM	Picard + ODE
ParaDiGMS	+	DDIM	=	ParaDDIM	Picard + Euler
		DPMSolver	=	ParaDPMSolver	Picard + Heun

just pre-sample variance to remove stochasticity


0
25
50
75
100
125
150
175
200
Parallel Sampling (ParaDDPM)
Sequential Sampling (DDPM)
0
25
50
75
100
125
150
175
200

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

$t, k \leftarrow 0, 0$

$\boldsymbol{z}_i \sim \mathcal{N}(\boldsymbol{0}, \sigma_i^2 \boldsymbol{I}) \quad \forall i \in [0, T)$ // Up-front sampling of noise (for SDE)

$\boldsymbol{x}_0^k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \qquad \boldsymbol{x}_i^k \leftarrow \boldsymbol{x}_0^k \quad \forall i \in [1, p]$ // 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
- $\text{error} \leftarrow \{\frac{1}{D} \|\boldsymbol{x}_{t+j}^{k+1} - \boldsymbol{x}_{t+j}^k\|^2 : \forall j \in [1, p)\}$ // Store error value for each timestep
- $\text{stride} \leftarrow \min\left(\{j : \text{error}_j > \tau \sigma_j^2\} \cup \{p\}\right)$ // 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 x_T^k

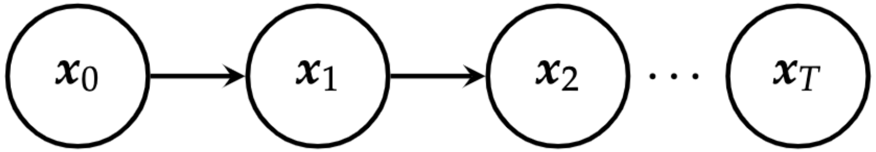


Figure 1: Computation graph of sequential sampling by evaluating $p_\theta(\boldsymbol{x}_{t+1} \mid \boldsymbol{x}_t)$, from the perspective of reverse time.

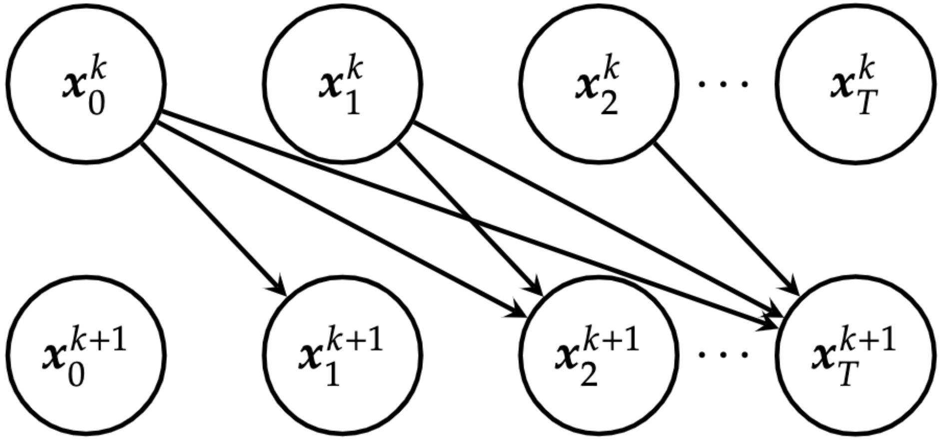
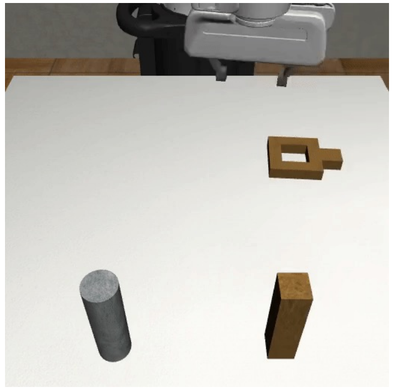


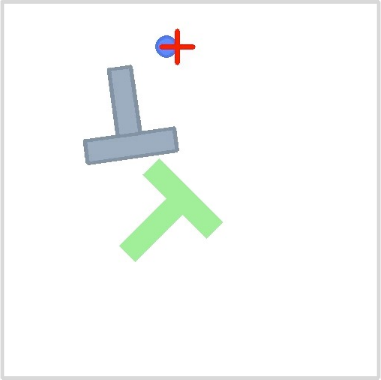
Figure 2: Computation graph of Picard iterations, which introduces skip dependencies.

	DDPM [Ho 2020]	DDIM [Song 2021]	DPMSolver [Lu 2022]	ParaDiGMS [our method!]
Sample Method	SDE (euler maruyama)	ODE (euler)	ODE (heun)	ODE (picard+ euler/heun)
Speed	Slow 1000 steps	Fast 50 steps	Fast 50 steps	Fast 1000 steps
Quality	Best	Good	Good	Best
		trade quality for speed	trade quality for speed	trade compute for speed

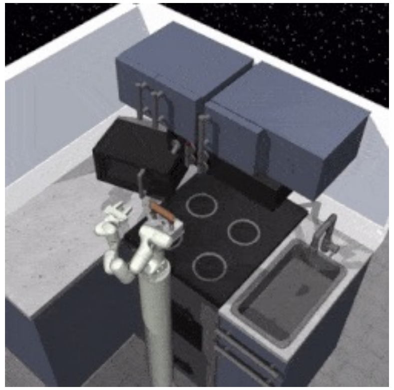
Experiments



square



push t



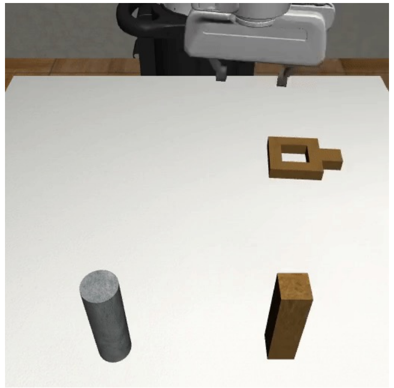
kitchen

stable diffusion v2

LSUN church

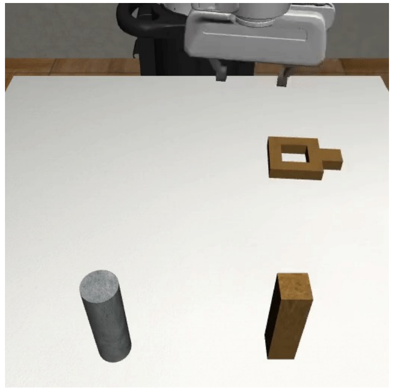
Square

NVIDIA a40 GPU robosuite square task 400k training steps 200 evaluation episodes	Method	DDPM	DDIM	DPE-solver
	Description	100 step reverse SDE	15 step reverse ODE	15 step reverse ODE with 3-point integration rule
Base Sampler	Reward	0.85 ± 0.03	0.83 ± 0.03	0.85 ± 0.03
	Function Evaluations	100	15	15
	Samples per Second	10.8 ± 0.6	70 ± 4	69 ± 4



Square

NVIDIA a40 GPU robosuite square task 400k training steps 200 evaluation episodes	Method	DDPM	DDIM	DPE-solver
	Description	100 step reverse SDE	15 step reverse ODE	15 step reverse ODE with 3-point integration rule
Base Sampler	Reward	0.85 ± 0.03	0.83 ± 0.03	0.85 ± 0.03
	Function Evaluations	100	15	15
	Samples per Second	10.8 ± 0.6	70 ± 4	69 ± 4
ParaDiGM (Tolerance: 0.1 x noise)	Reward	0.85 ± 0.03	0.85 ± 0.03	0.83 ± 0.03
	Function Evaluations	~430	~60	~50
	Parallel Iterations	~27	~8	~7
	Samples per Second	40 ± 2 **(3.7x faster)**	112 ± 7 **(1.6x faster)**	122 ± 7 **(1.8x faster)**

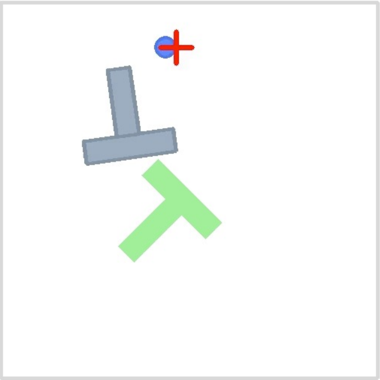


No drop in sample quality!

Faster! 🏎️ 🏎️

Push T

NVIDIA a40 GPU push-t task 400k training steps 200 evaluation episodes	Method	DDPM	DDIM	DPE-solver
	Description	100 step reverse SDE	15 step reverse ODE	15 step reverse ODE with 3-point integration rule
Base Sampler	Reward	0.81 ± 0.03	0.78 ± 0.03	0.79 ± 0.03
	Function Evaluations	100	15	15
	Samples per Second	9.3 ± 0.6	71 ± 4	71 ± 4
ParaDiGM (Tolerance: 0.1 x noise)	Reward	0.85 ± 0.03	0.77 ± 0.03	0.82 ± 0.03
	Function Evaluations	~430	~60	~50
	Parallel Iterations	~27	~8	~7
	Samples per Second	36 ± 2 **(3.9x faster)**	118 ± 7 **(1.7x faster)**	140 ± 7 **(2.0x faster)**

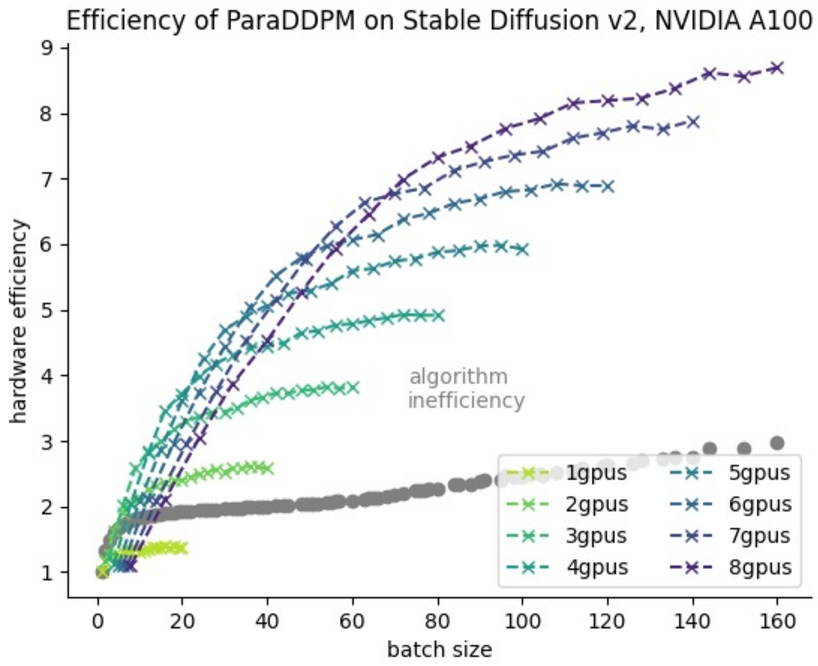


No drop in sample quality!

Faster!

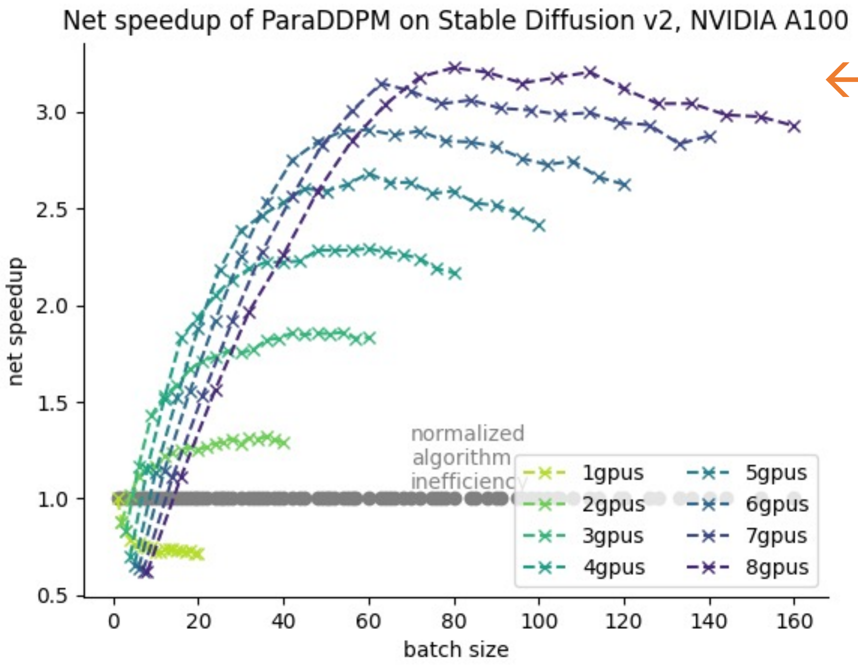
Stable Diffusion v2

- 768 x 768 images
- Diffusion in latent space 4 x 96 x 96



← efficiency of GPUs

← cost of iterating

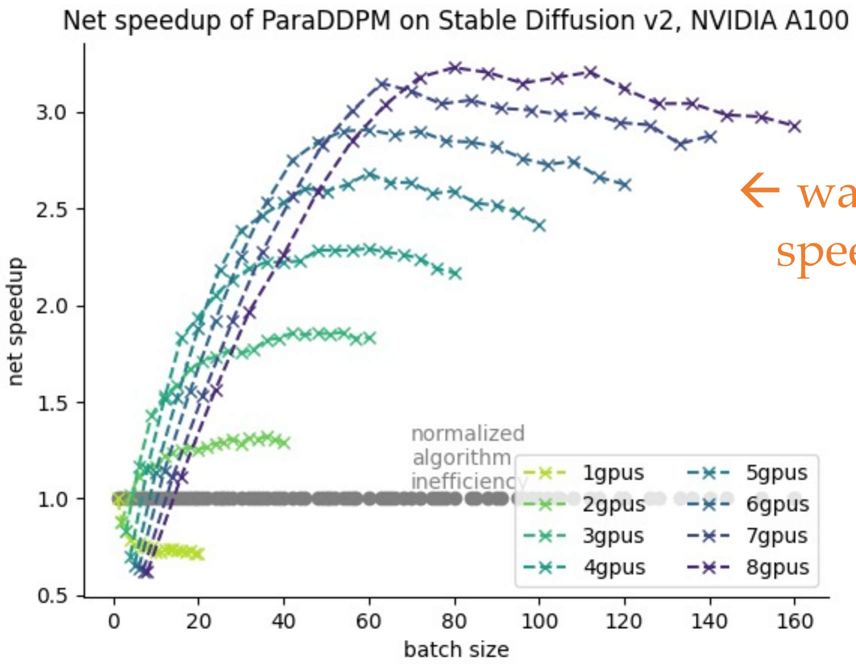


← wallclock speedup

Stable Diffusion v2

- 768 x 768 images
- Diffusion in latent space 4 x 96 x 96

(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"

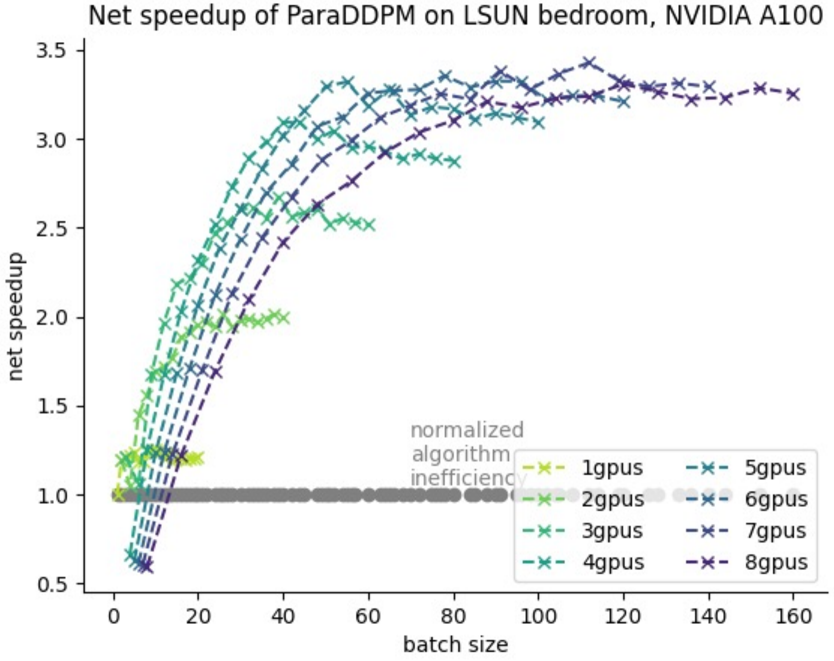


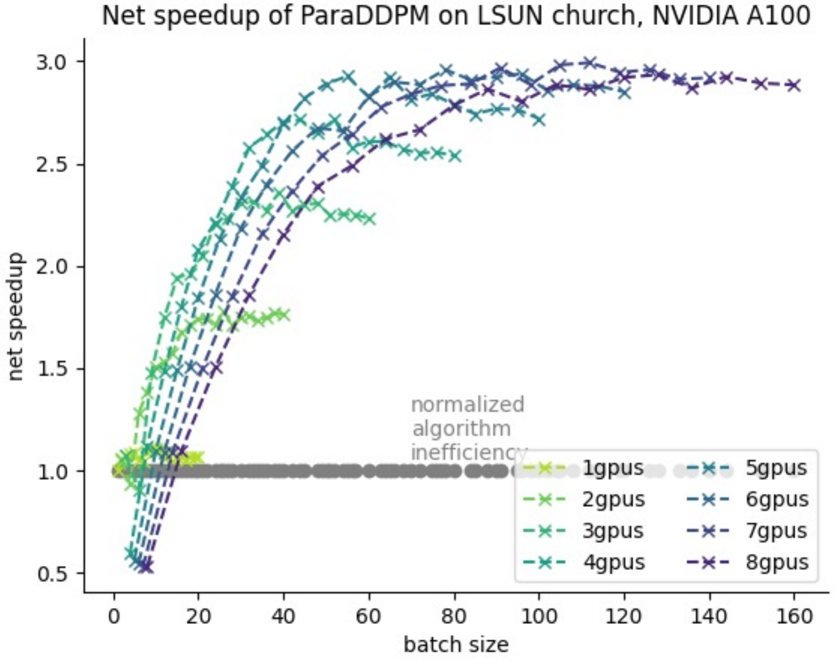
← wallclock speedup

StableDiffusion-v2	Sequential Model Evals	Sequential CLIP Score	Sequential Time per Sample	ParaDiGMS Model Evals	ParaDiGMS Parallel Iters	ParaDiGMS CLIP Score	ParaDiGMS Time per Sample	Speedup
DDPM	1000	32.1	50.0s	2040	44	32.1	16.2s	3.1x
DDIM	200	31.9	10.3s	425	16	31.9	5.8s	1.8x
DPMSolver	200	31.7	10.3s	411	16	31.7	5.8s	1.8x

LSUN Church / Bedroom

- 256 x 256 images
- Diffusion in pixel space 3 x 256 x 256





LSUN Church / Bedroom

- 256 x 256 images
- Diffusion in pixel space 3 x 256 x 256

LSUN Church	Sequential			ParaDiGMS				Speedup
	Model Evals	FID Score	Time per Sample	Model Evals	Parallel Iters	FID Score	Time per Sample	
DDPM	1000	12.8	24.0s	2556	42	12.9	8.2s	2.9x
DDIM	500	15.7	12.2s	1502	42	15.7	6.3s	1.9x

	DDPM [Ho 2020]	DDIM [Song 2021]	DPMSolver [Lu 2022]	ParaDiGMS [our method!]
Sample Method	SDE (euler maruyama)	ODE (euler)	ODE (heun)	ODE (picard+ euler/heun)
Speed	Slow 1000 steps	Fast 50 steps	Fast 50 steps	Fast 1000 steps
Quality	Best	Good	Good	Best
		trade quality for speed	trade quality for speed	trade compute for speed

ParaDiGMS

Parallel Diffusion Generative Model Sampler
instead of ***trading quality for speed***
we enable ***trading compute for speed***

Can be combined with other sampling methods for 2-4x speedup
(ParaDDPM, ParaDDIM, ParaDPMSolver)

ParaDiGMS

Parallel Diffusion Generative Model Sampler

instead of *trading quality for speed*

we enable *trading compute for speed*

Can be combined with other sampling methods for 2-4x speedup

(ParaDDPM, ParaDDIM, ParaDPMSolver)

single GPU for diffusion-policy!

multi GPU for image diffusion models