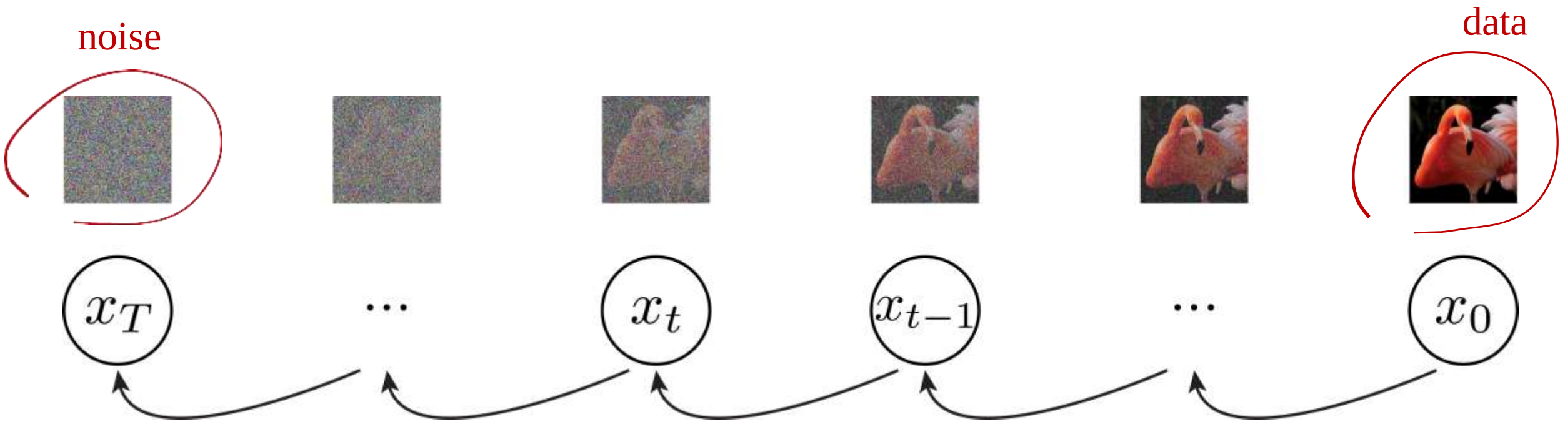
Diffusion Models in a nutshell

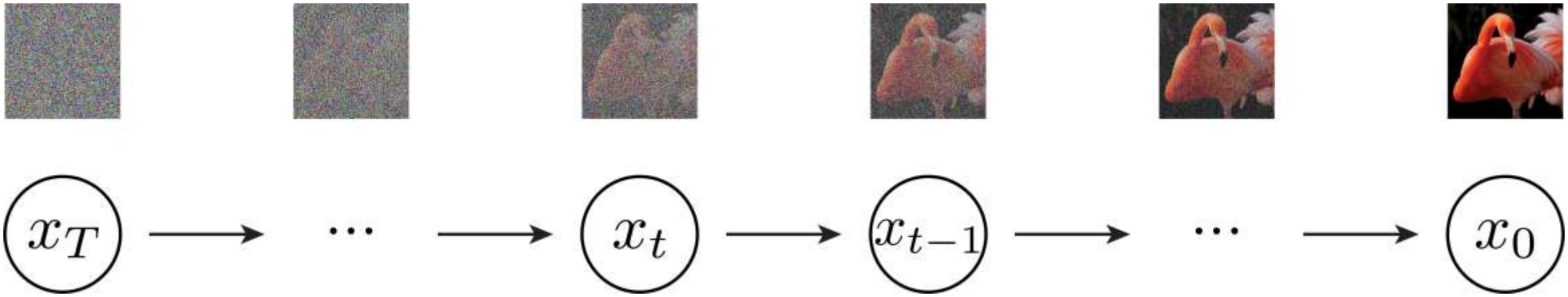


Forward process: add noise

Diffusion Models in a nutshell

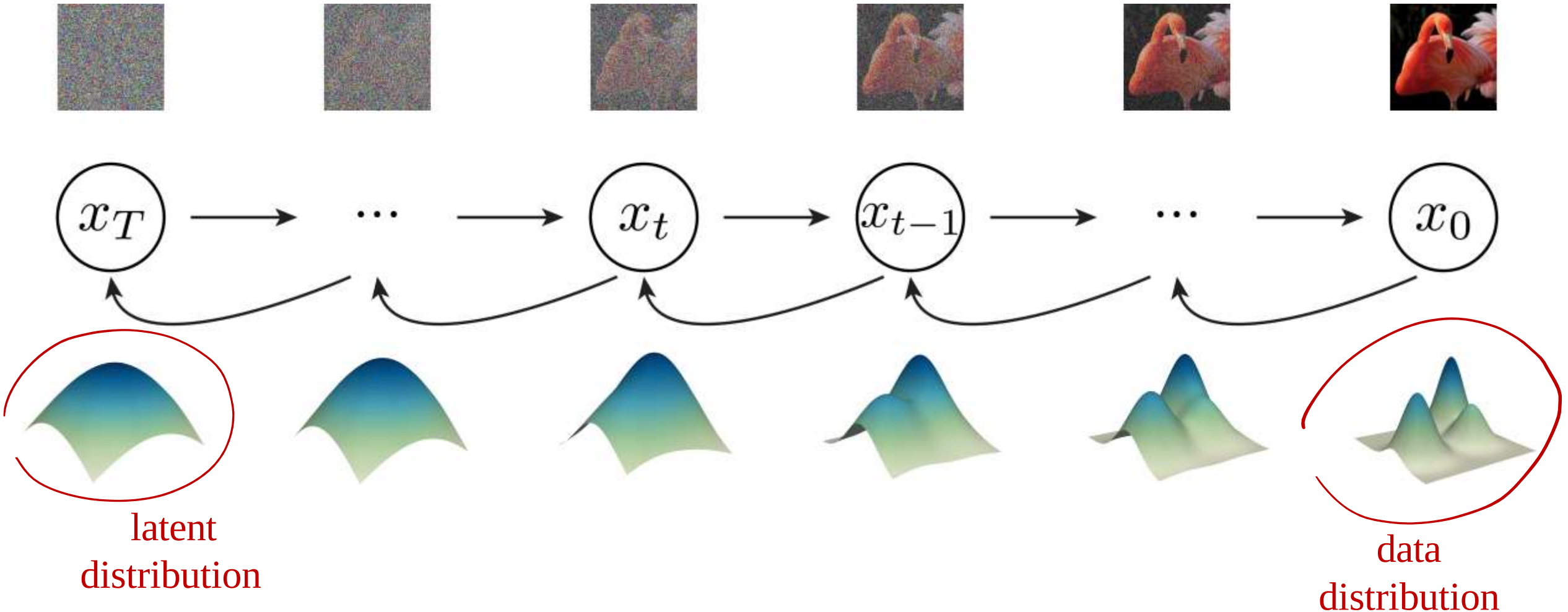
noise

data

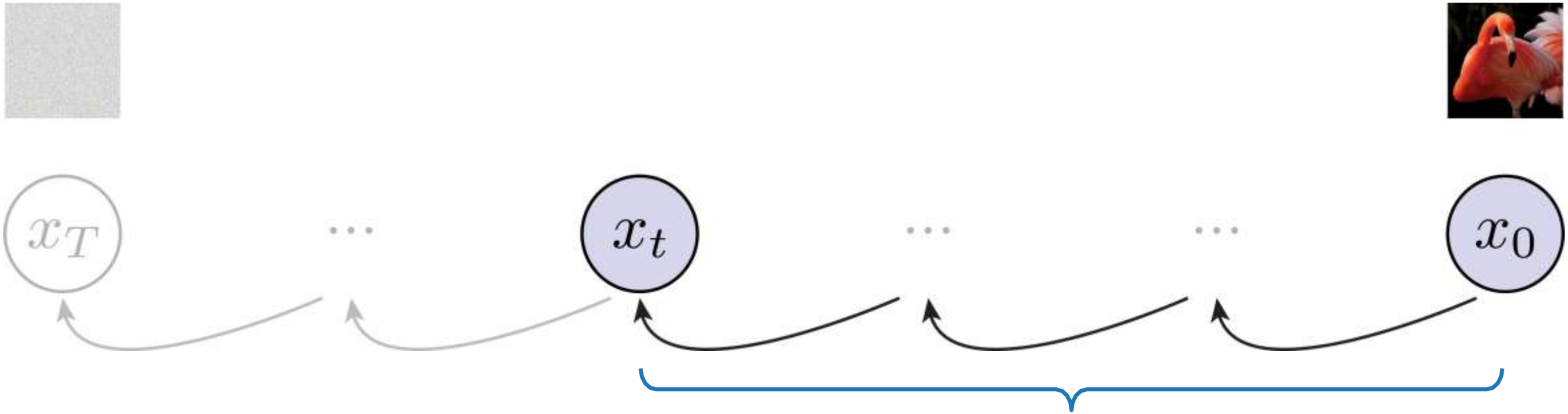


Reverse process: denoise

What is noise?



Forward Process



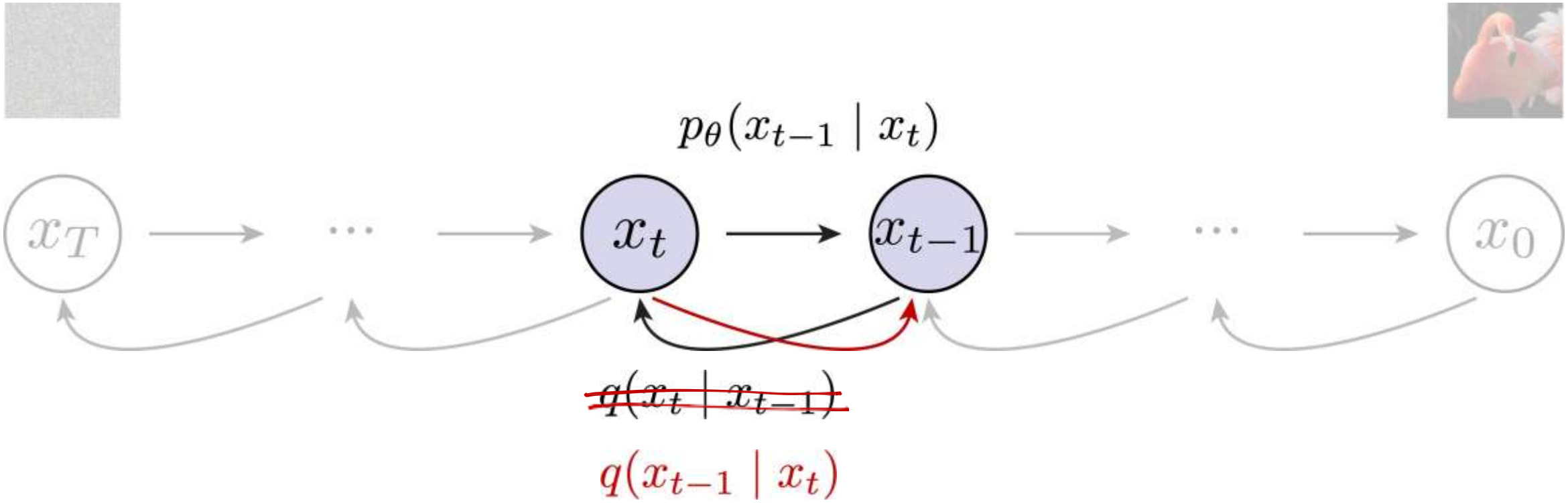
- sampling without simulation
- x_t from x_0 in closed form

$$q(x_t \mid x_0) = \mathcal{N}(x_t \mid \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)\mathbf{I})$$

coefficients given by β

$$\alpha_t := 1 - \beta_t$$
$$\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

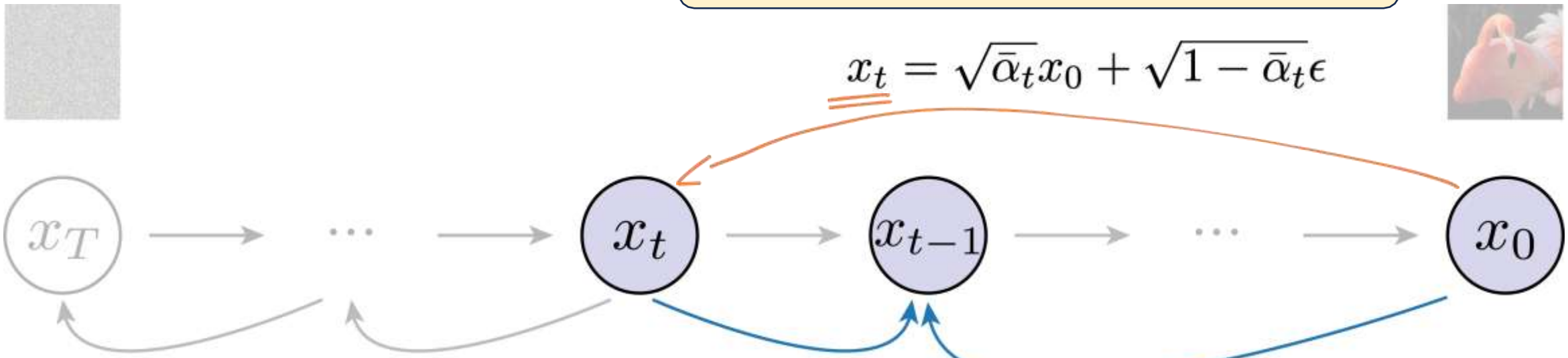
Reverse Process



- our target
- but unknown

Reverse Process

tl; dr:
- outcome of the dependency graph
- some linear combinations



$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_{t-1} \mid \tilde{\mu}(x_t, x_0), \tilde{\beta}_t \mathbf{I})$$

$$\tilde{\mu}_t(x_t, x_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t$$

mean

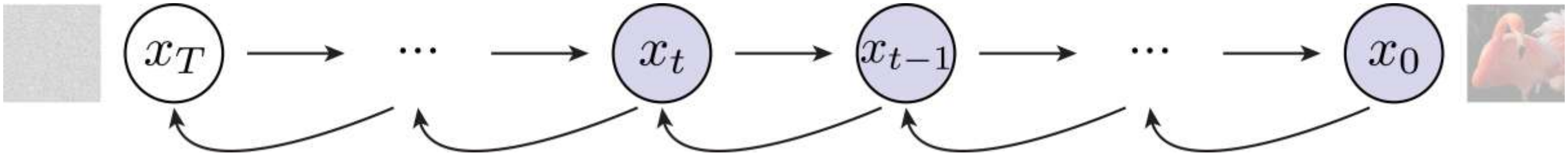
$$= \frac{1}{\sqrt{\bar{\alpha}_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

"noise"

$$\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

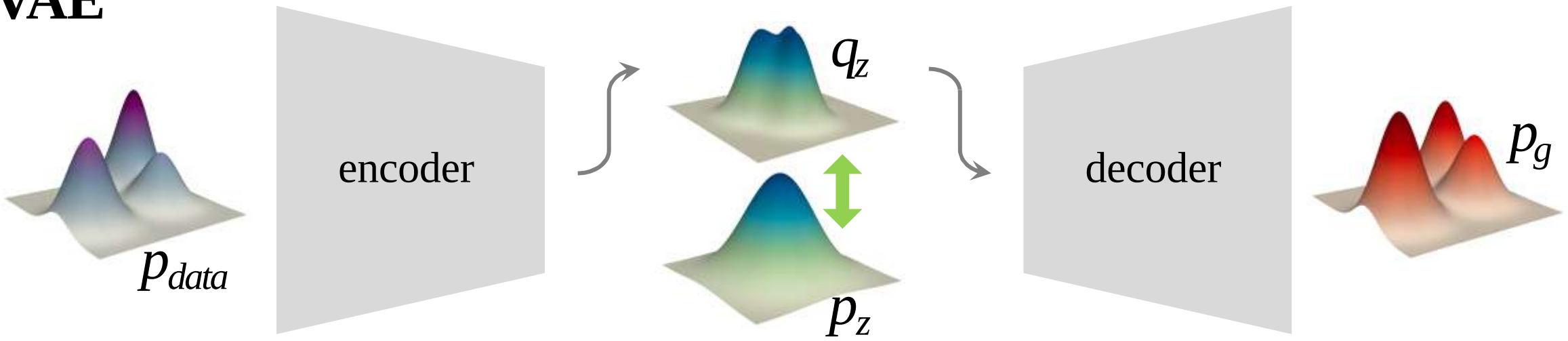
var

Training Objective

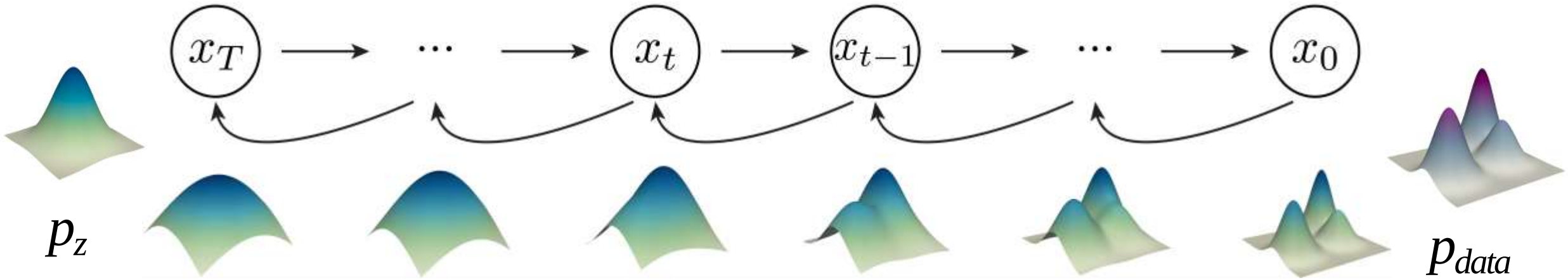


$$
\begin{aligned}
\mathcal{L}_{\text{VLB}} &:= \mathcal{L}_T + \mathcal{L}_{T-1} + \ldots + \mathcal{L}_0 \\
\mathcal{L}_T &:= D_{\text{KL}}\Big(q(x_T \mid x_0) \,\|\, p_\theta(x_T)\Big) \\
\mathcal{L}_{t-1} &:= D_{\text{KL}}\Big(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\Big) \\
\mathcal{L}_0 &:= -\log p_\theta(x_0 \mid x_1)
\end{aligned}
$$

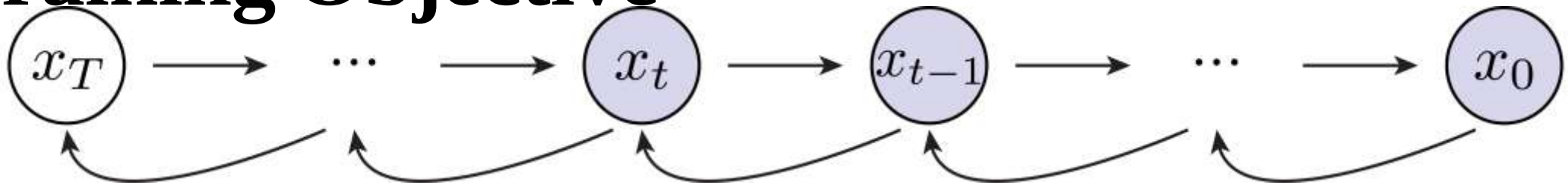
VAE



DM/Flow

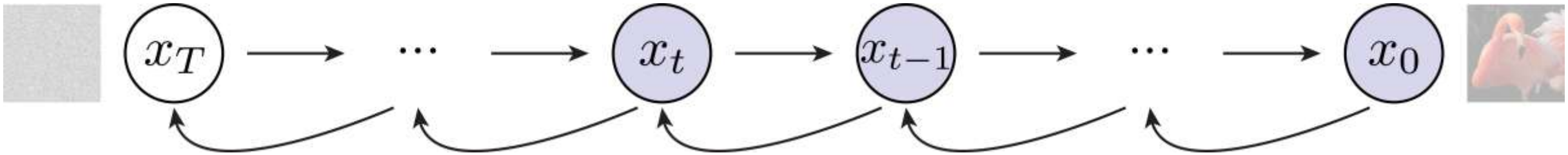


Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

Training Objective



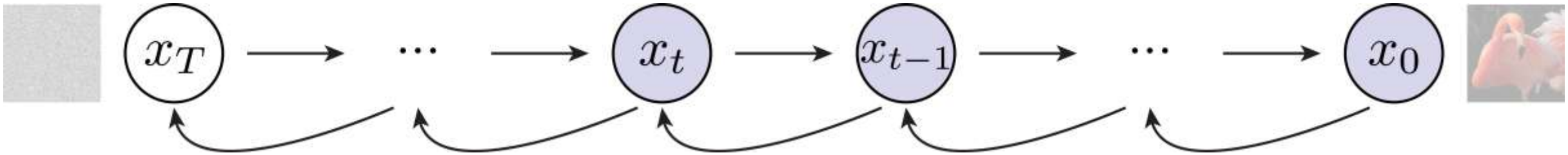
$$\mathcal{L} = \mathbb{E}_{x_0, t, \epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

over p_{data}

over $[1, T]$

over $\mathcal{N}(0, \mathbf{I})$

Training Objective



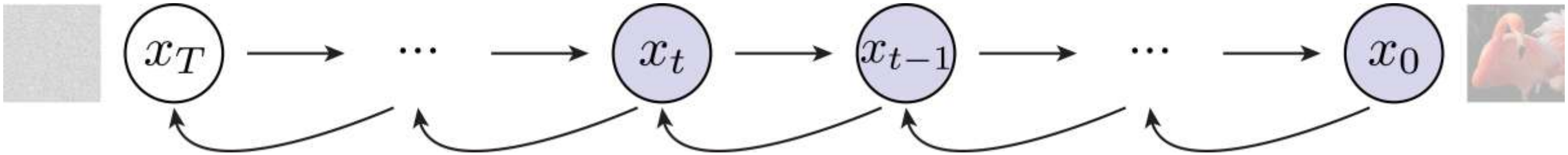
$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

set as 1 **(critical)**

Objective	IS	FID
L, learned diagonal $\boldsymbol{\Sigma}$	–	–
L, fixed isotropic $\boldsymbol{\Sigma}$	7.67 ± 0.13	13.51
$\|\tilde{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}_\theta\|^2$ (L_{simple})	$\mathbf{9.46 \pm 0.11}$	**3.17**

[Ho et al. 2020]; see more in [Salimans & Ho, 2022]

Training Objective



$$\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\left[w_t \|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

network to predict noise

conditioned on noise level **(critical)**

Noise Conditional Network

How to represent $p_\theta(x_{t-1} \mid x_t)$

- network input: x_t
- network output: μ and σ of a distribution

- parametrize μ by: $\epsilon_\theta(x_t, t)$

noisy image:
- condition
- network input

noise level:
- condition
- network input

Noise Conditional Network

x_t

network input

t

also network input

$\epsilon_\theta(x_t, t)$

Unet, ViT, ...
(behaves like Denoising AE)

ϵ

output: same size

Diffusion algorithm annotated:

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}_{x_t}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

estimated μ

sampling from estimated distribution

Diffusion algorithm annotated:

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

tl; dr: noising and denoising

- Turns out to be extremely simple
- Being “simple and effective” moves the needle

Example: Unconditional Generation on CIFAR-10

noise
x_T

generated
x_0

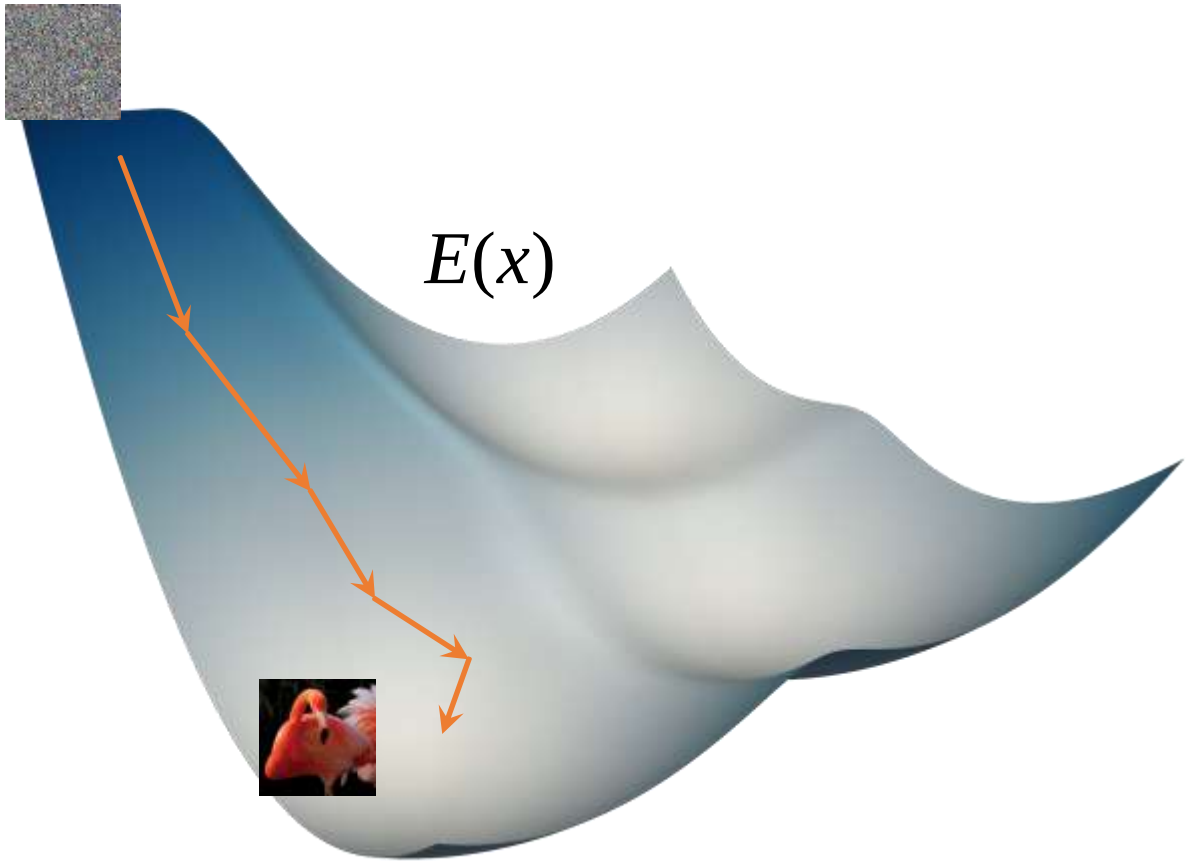
[Ho et al, 2020]

Example: shared intermediate latents

[Ho et al, 2020]

Energy-based Models

- Define a scalar function, called “energy”.
- At inference time, find x that minimizes energy

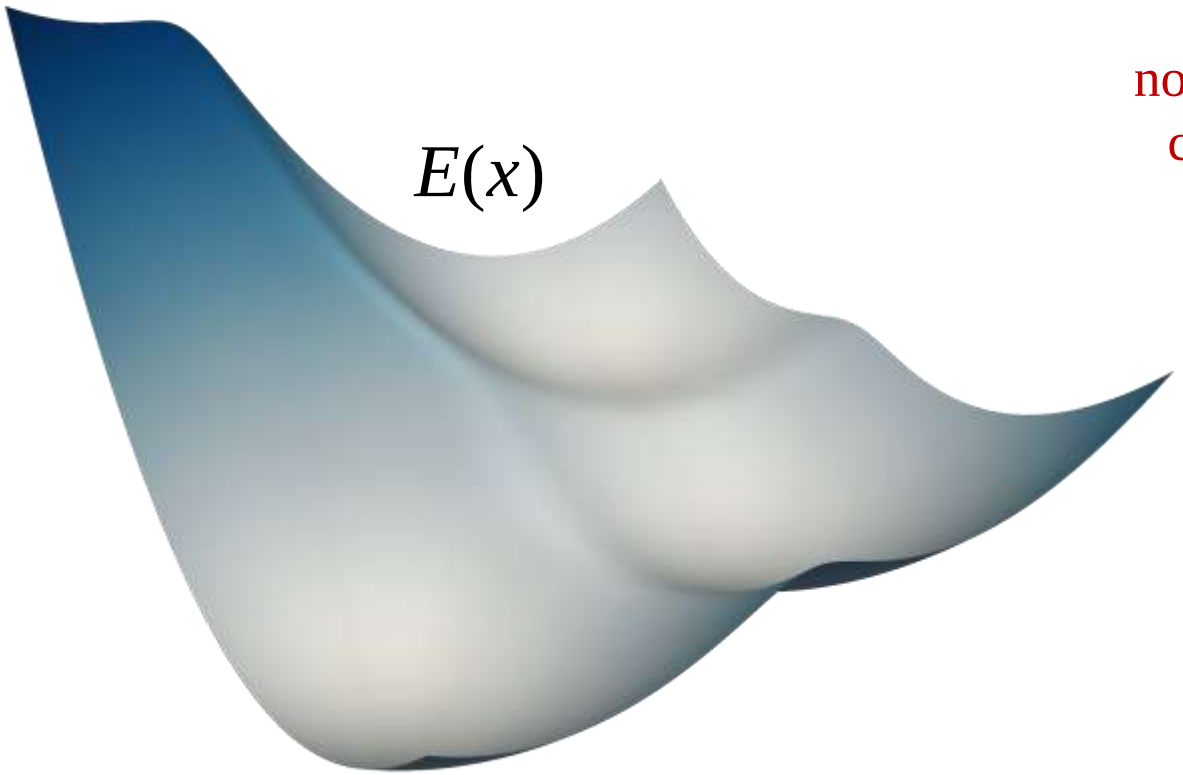


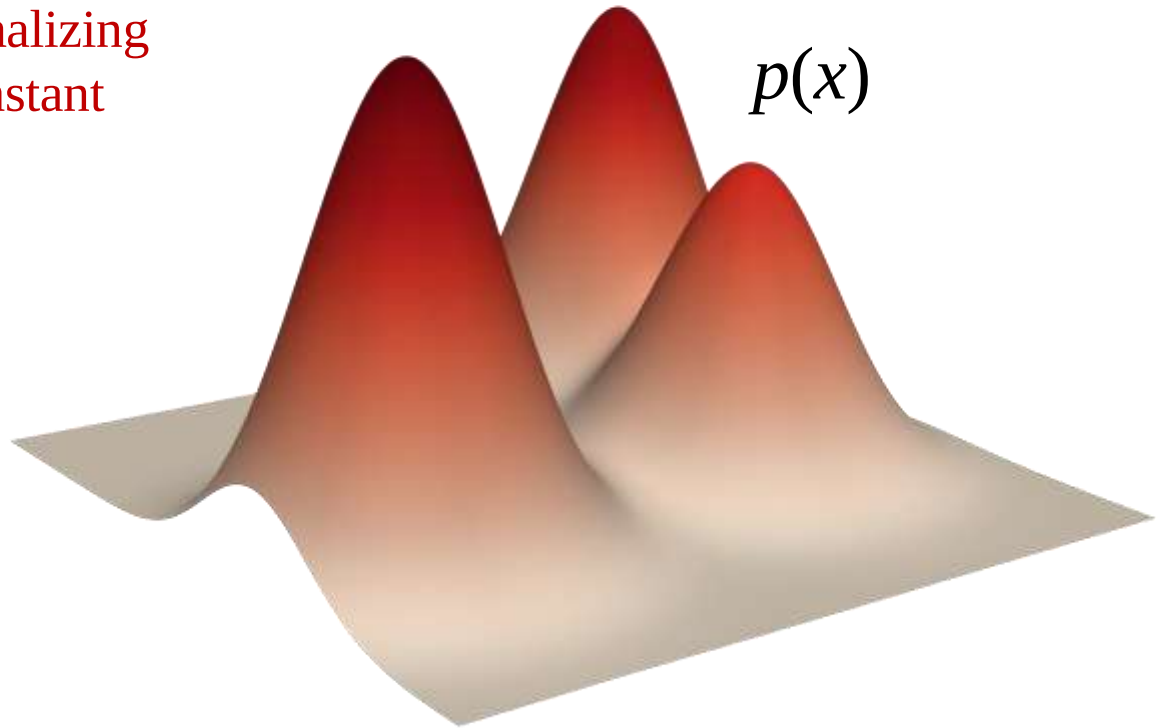
Energy-based Models

- We can use an energy to model a probability distribution

$$p(x) = \frac{\exp(-E(x))}{Z}$$

normalizing constant



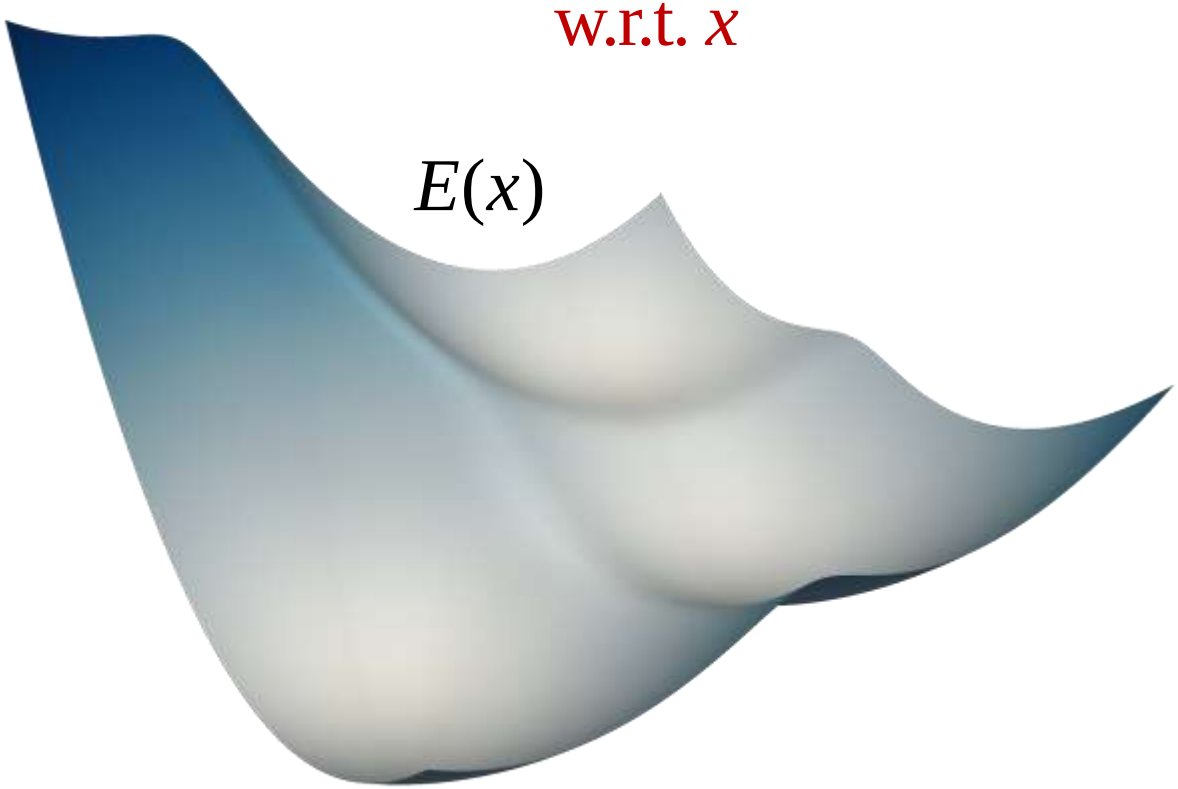


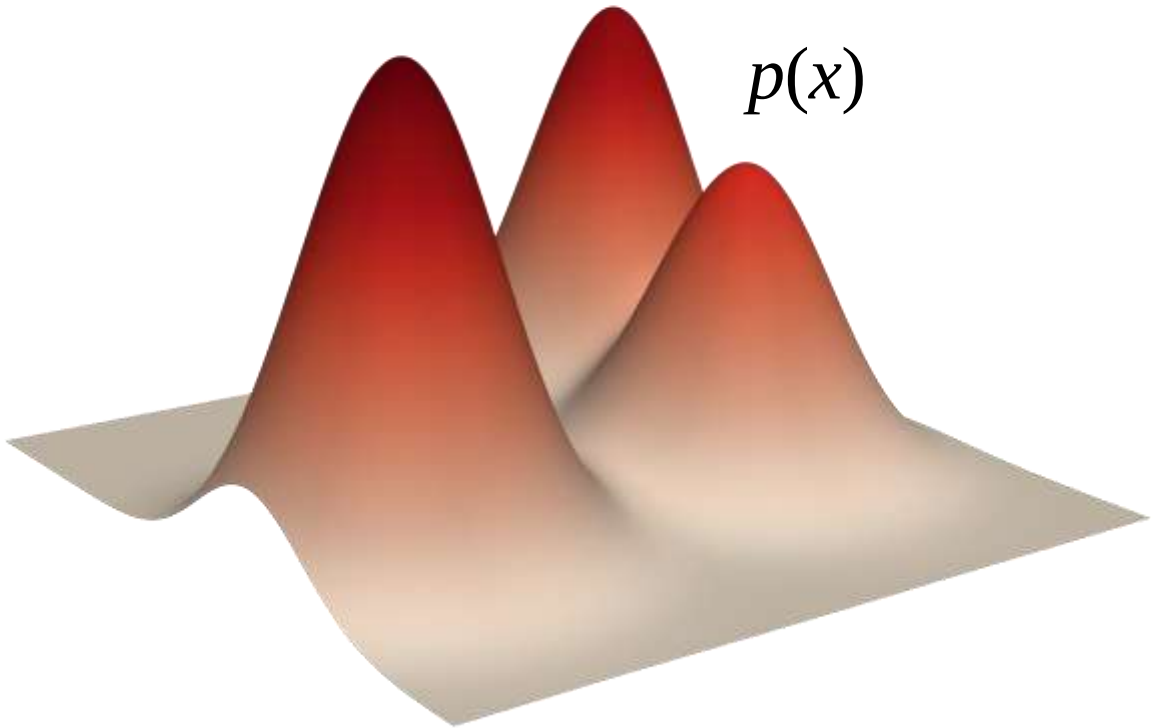
Energy-based Models

- **“Score function”**: gradient of log-probability.

$$\nabla_x \log p(x) = -\nabla_x E(x) - \nabla_x \log Z$$

w.r.t. x =0



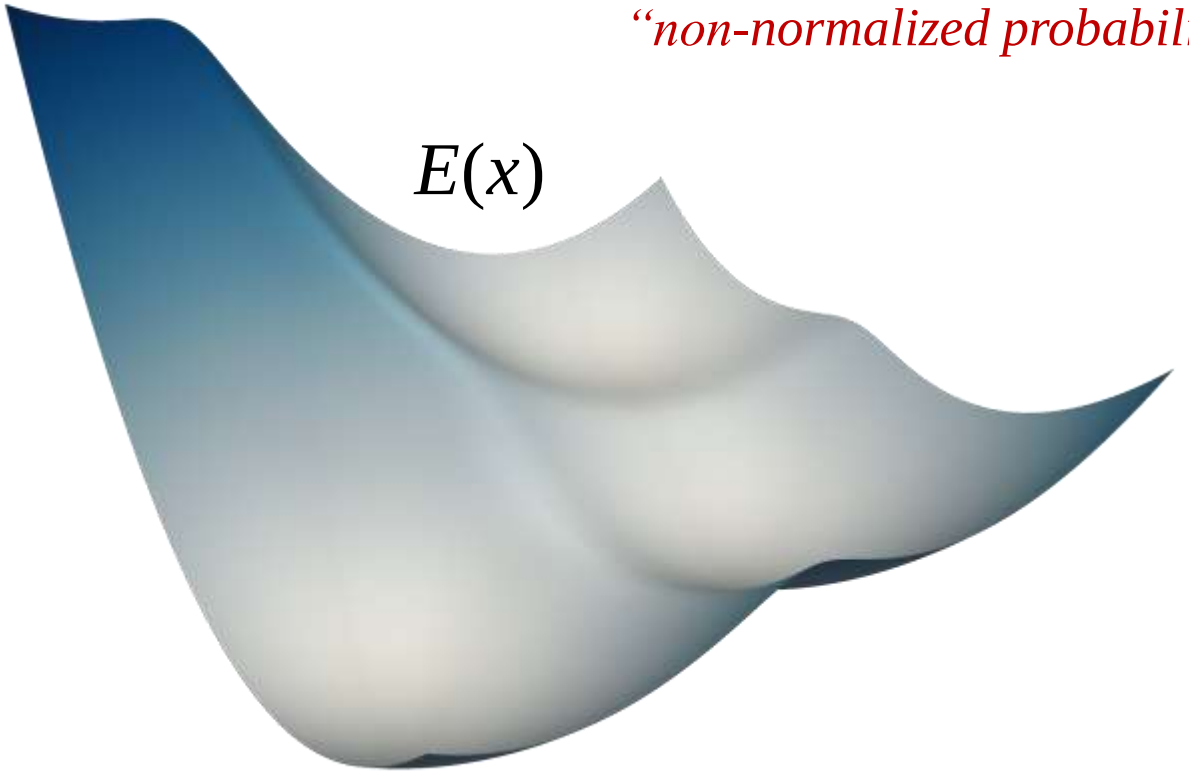


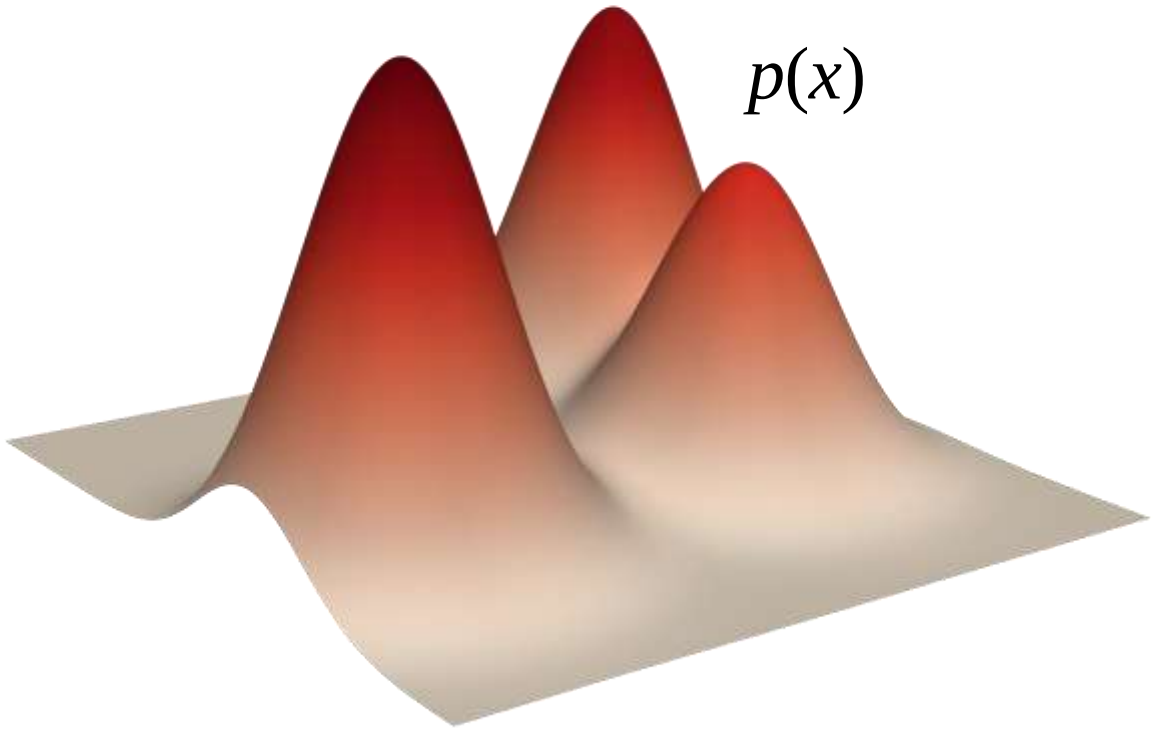
Energy-based Models

- **“Score function”**: gradient of log-probability

$$\nabla_x \log p(x) = -\nabla_x E(x)$$

“non-normalized probabilistic models”



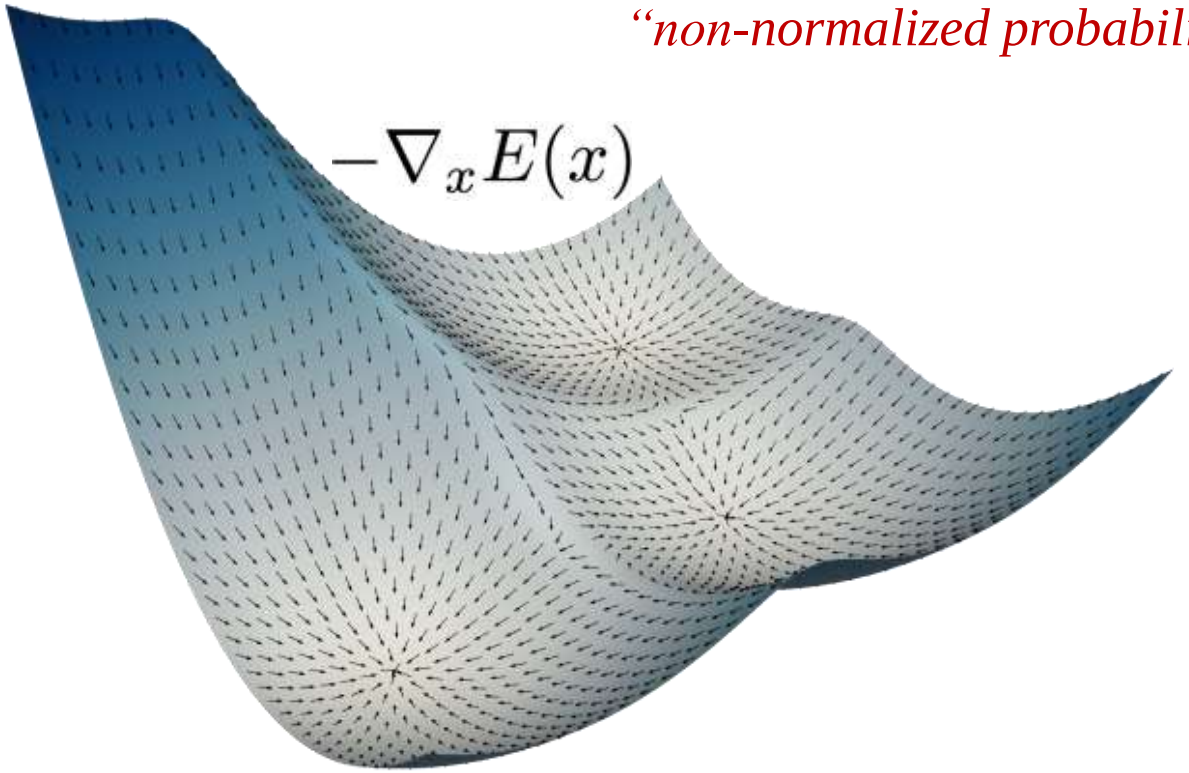


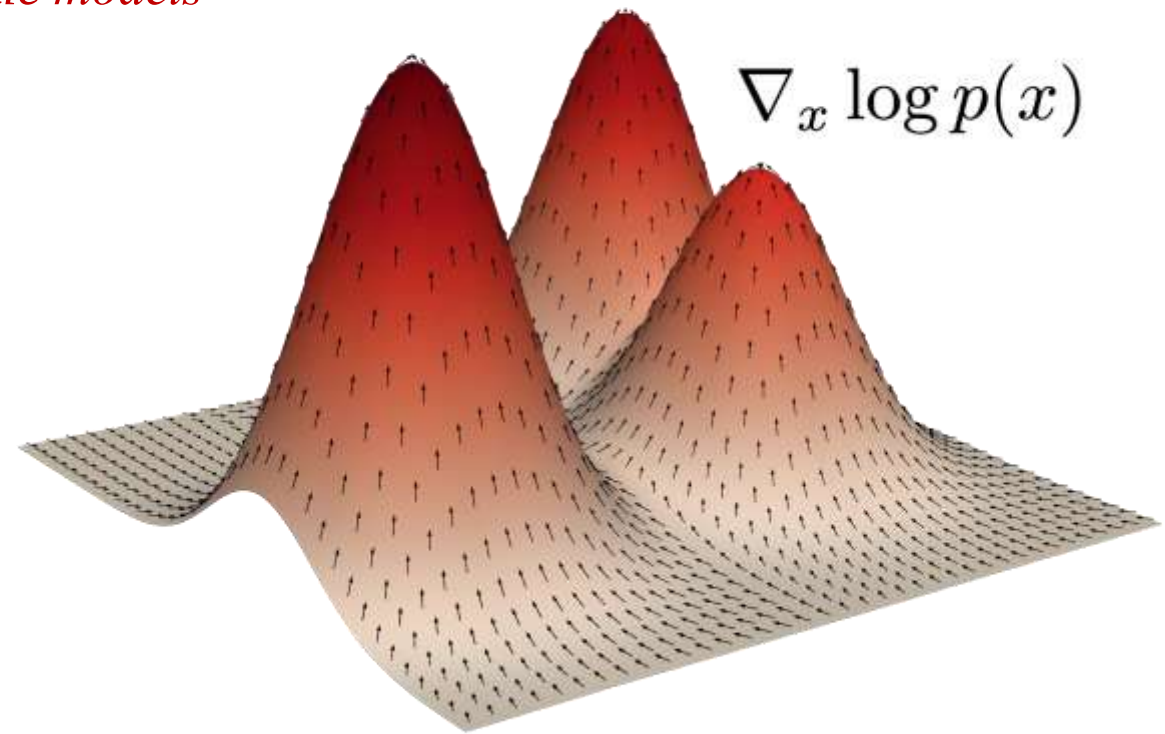
Energy-based Models

- **“Score function”**: gradient of log-probability

$$\nabla_x \log p(x) = -\nabla_x E(x)$$

“non-normalized probabilistic models”





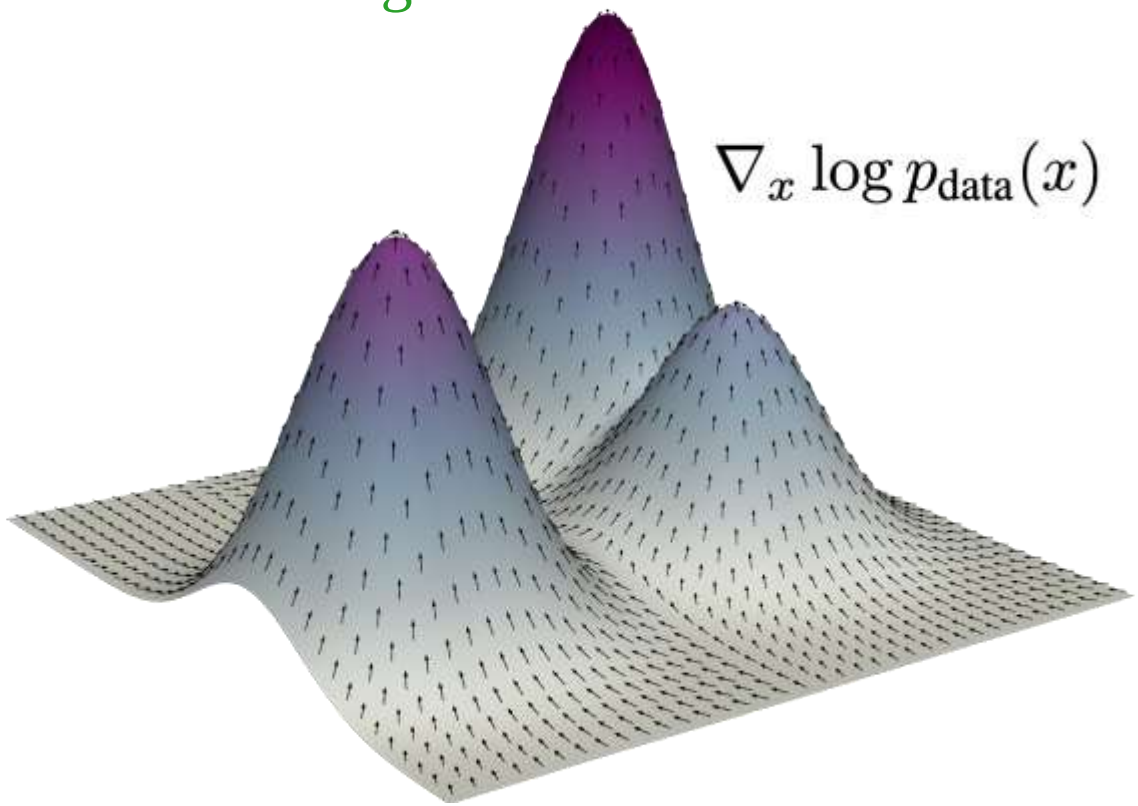
*only visualize

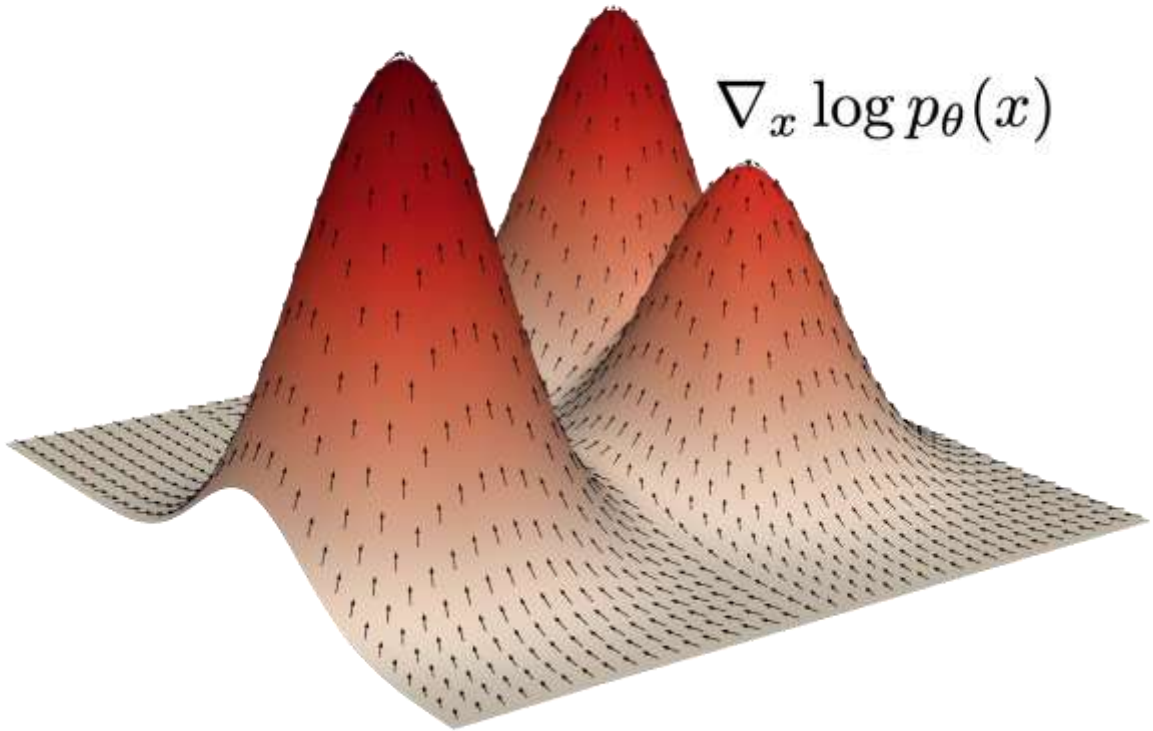
Score Matching

- Instead of parametrizing *p*, we can parametrize the score

$$D_F(p_{\text{data}}(x) \parallel p_\theta(x)) = \mathbb{E}_{p_{\text{data}}(x)}\left[\frac{1}{2}\|\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\theta(x)\|^2\right]$$

Fisher divergence — score of data — parameterized score





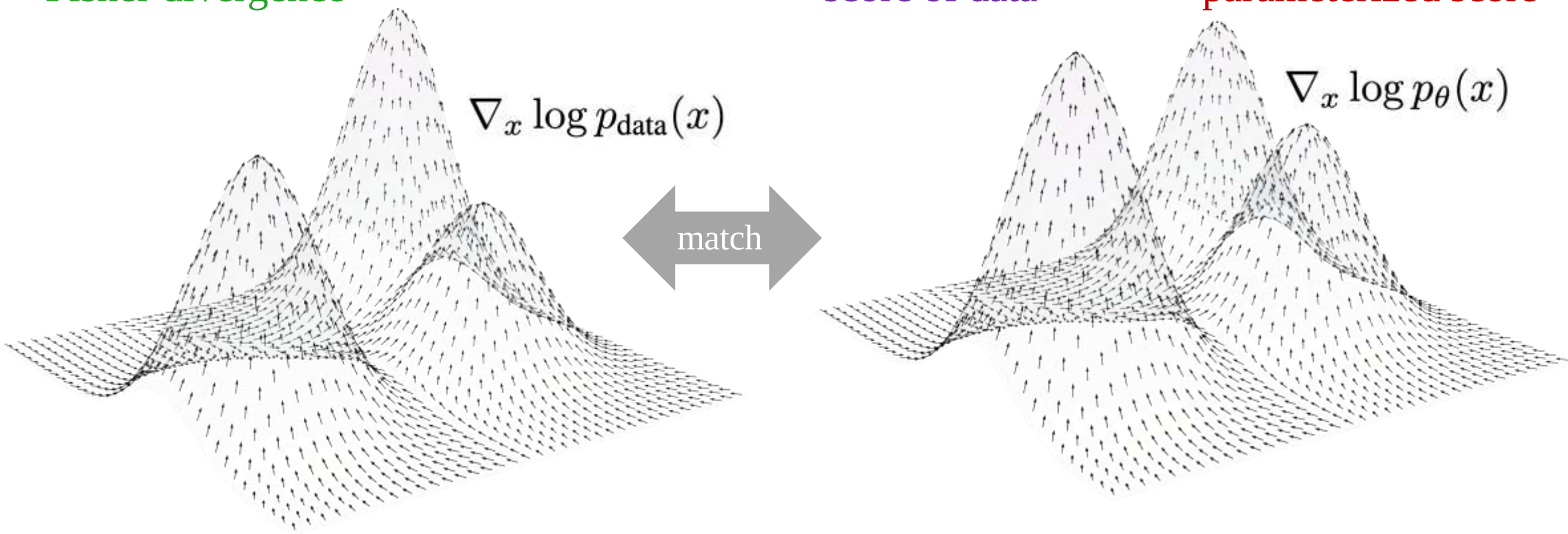
*only visualize

Score Matching

- Instead of parametrizing *p*, we can parametrize the score

$$D_F(p_{\text{data}}(x) \parallel p_\theta(x)) = \mathbb{E}_{p_{\text{data}}(x)}\left[\frac{1}{2}\|\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\theta(x)\|^2\right]$$

Fisher divergence — score of data — parameterized score

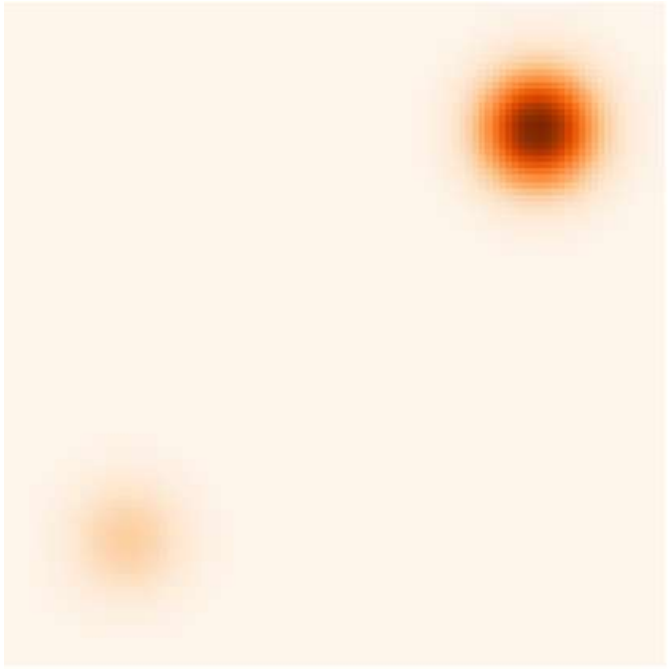


*only visualize

Score estimation by training score-based models

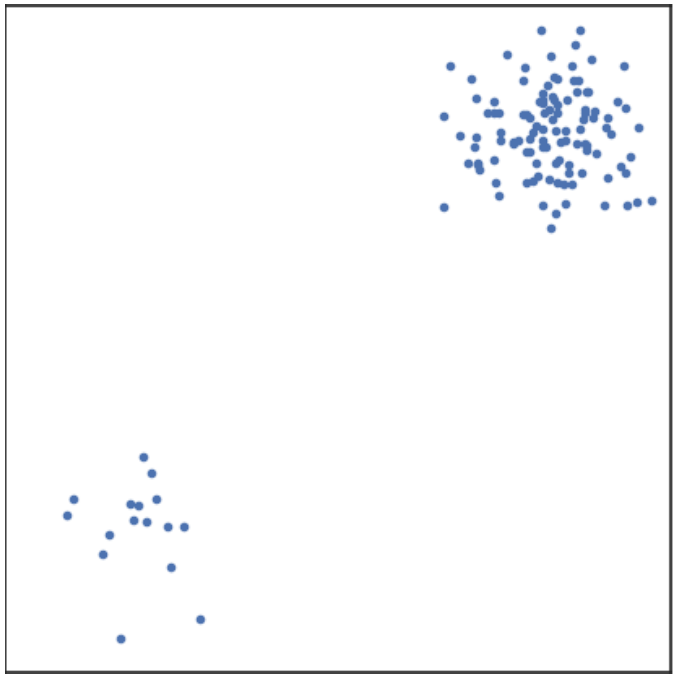
Probability density

$$p_{\text{data}}(\mathbf{x})$$



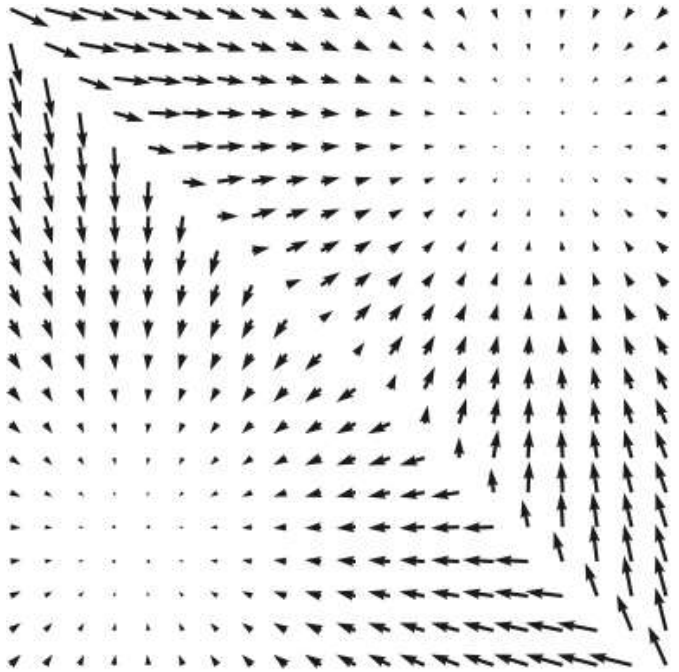
i.i.d. samples

$$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$$



Score function

$$\boldsymbol{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$$



Denoising Score Matching

- with noised data $\tilde{x} := x + \epsilon$, it can be proven: [Vincent, 2011]

$$D_F(q(\tilde{x}) \parallel p_\theta(\tilde{x})) = \mathbb{E}_{q(x,\tilde{x})}\left[\frac{1}{2}\left\|\nabla_{\tilde{x}} \log q(\tilde{x} \mid x) - \nabla_{\tilde{x}} \log p_\theta(\tilde{x})\right\|^2\right] + \text{constant}$$

Fisher divergence of noised data

joint distribution

score of conditional

parameterized score

See: Vincent, “A Connection Between Score Matching and Denoising Autoencoders”, Neural Computation, 2011

Denoising Score Matching

- with noised data $\tilde{x} := x + \epsilon$, it can be proven: [Vincent, 2011]

$$D_F(q(\tilde{x}) \parallel p_\theta(\tilde{x})) = \mathbb{E}_{q(x,\tilde{x})}\left[\frac{1}{2}\left\|\nabla_{\tilde{x}} \log q(\tilde{x} \mid x) - \nabla_{\tilde{x}} \log p_\theta(\tilde{x})\right\|^2\right] + \text{constant}$$

a network to predict (negative) noise

Gaussian noise:

$$= \frac{1}{\sigma^2}(x - \tilde{x})$$

$$-\epsilon$$

See: Vincent, “A Connection Between Score Matching and Denoising Autoencoders”, Neural Computation, 2011

Langevin Dynamics

- Given a score function, we can sample *x* from *p* by iterating:

$$x_t \leftarrow x_{t-1} + \frac{\sigma^2}{2}\nabla_x \log p_\theta(x_{t-1}) + \sigma z_t$$

step size

score function
(don't need to know *p*)

$z_t \sim \mathcal{N}(0, \mathbf{I})$

(neg) gradient of energy

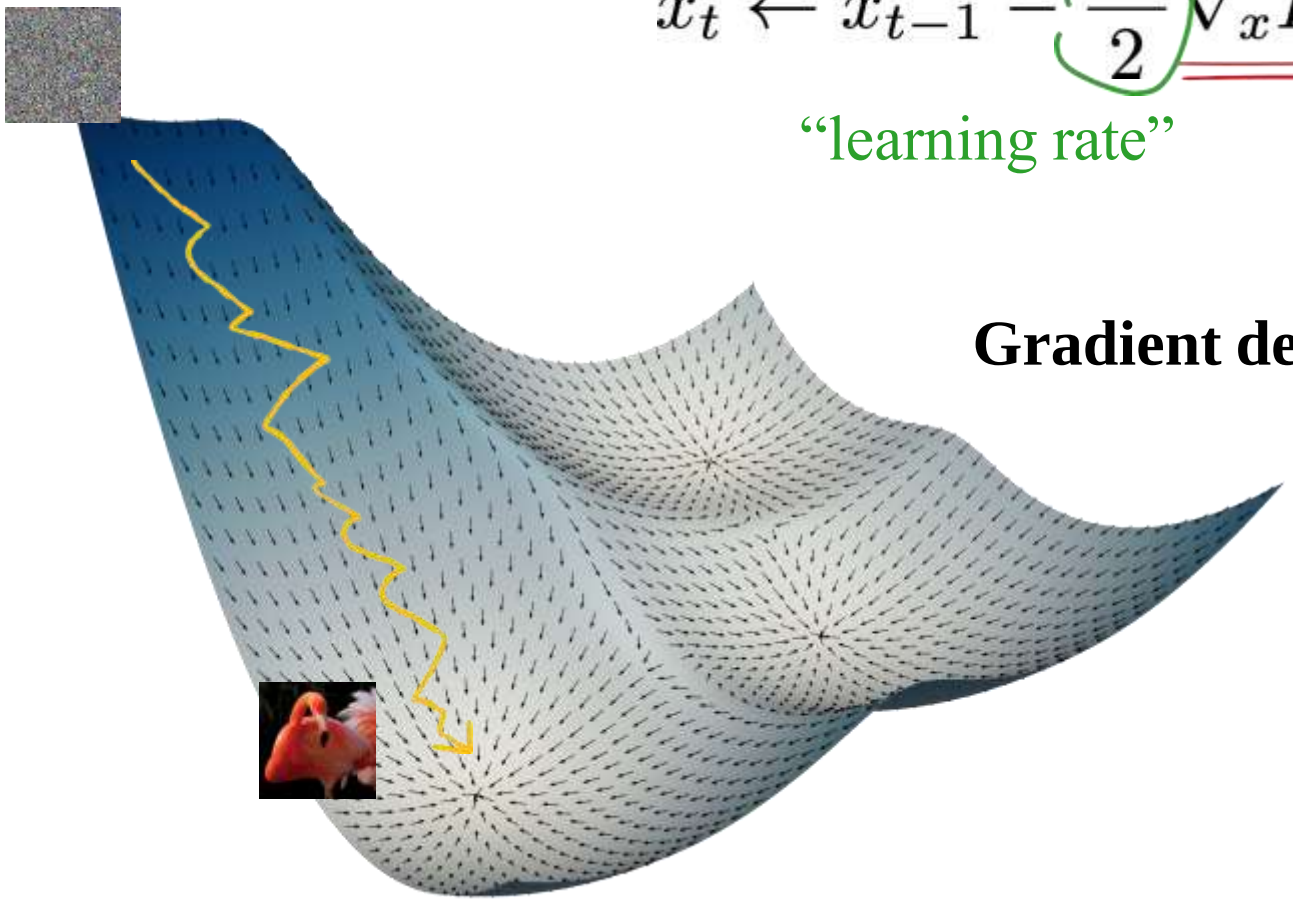
$$-\nabla_x E_\theta(x_{t-1})$$

Langevin Dynamics

- Given a score function, we can sample *x* from *p* by iterating:

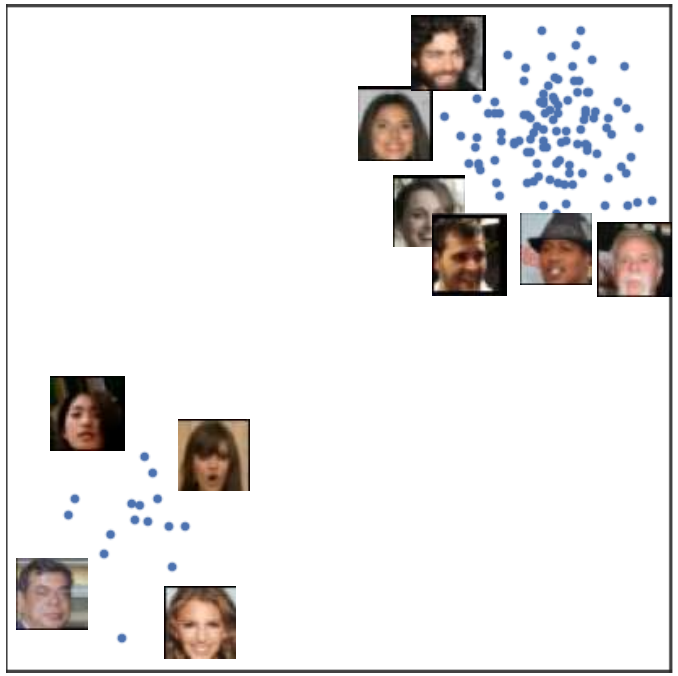
$$x_t \leftarrow x_{t-1} - \frac{\sigma^2}{2}\nabla_x E_\theta(x_{t-1}) + \sigma z_t$$

"learning rate" gradient perturbation



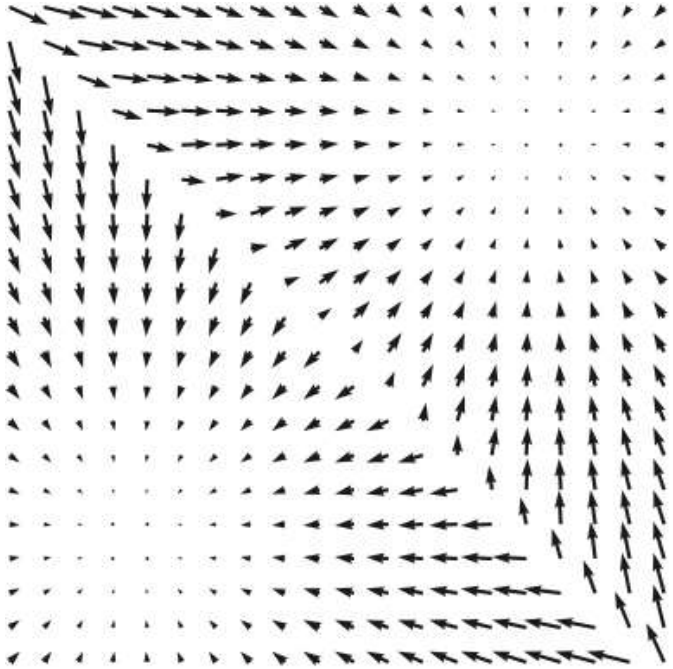
Gradient decent in the energy landscape

Score-based generative modeling



Data samples

Score Matching



Scores

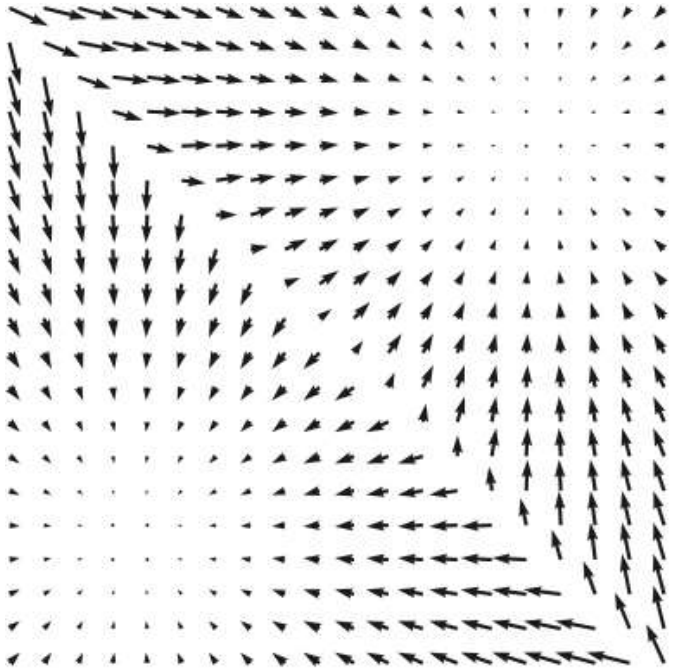
?

New samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$$

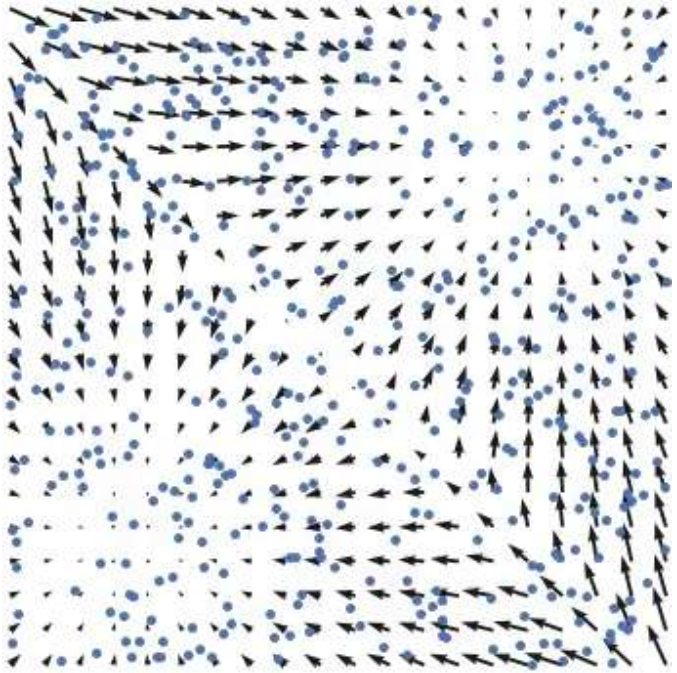
$$\boldsymbol{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$$

From scores to samples: Langevin MCMC



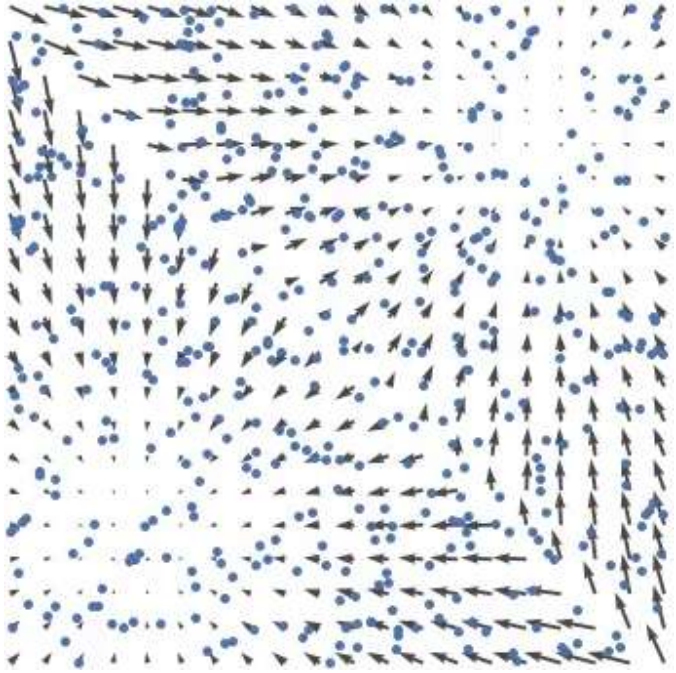
Scores

$$\mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x})$$



Follow the scores

$$\tilde{\mathbf{x}}_{t+1} \leftarrow \tilde{\mathbf{x}}_t + \frac{\epsilon}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_t)$$

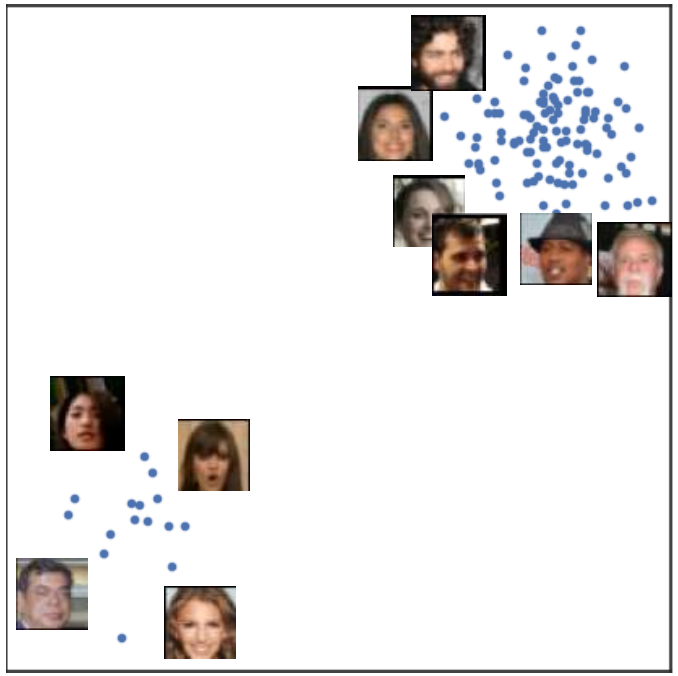


Follow noisy scores:
Langevin MCMC

$$\mathbf{z}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\tilde{\mathbf{x}}_{t+1} \leftarrow \tilde{\mathbf{x}}_t + \frac{\epsilon}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_t) + \sqrt{\epsilon}\,\mathbf{z}_t$$

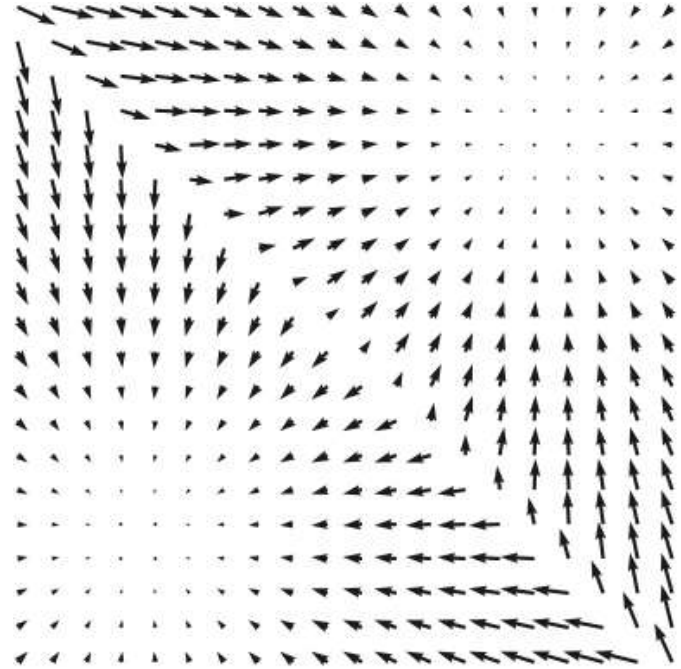
Score-based generative modeling



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\} \sim p_{\text{data}}(\mathbf{x})$$

score matching



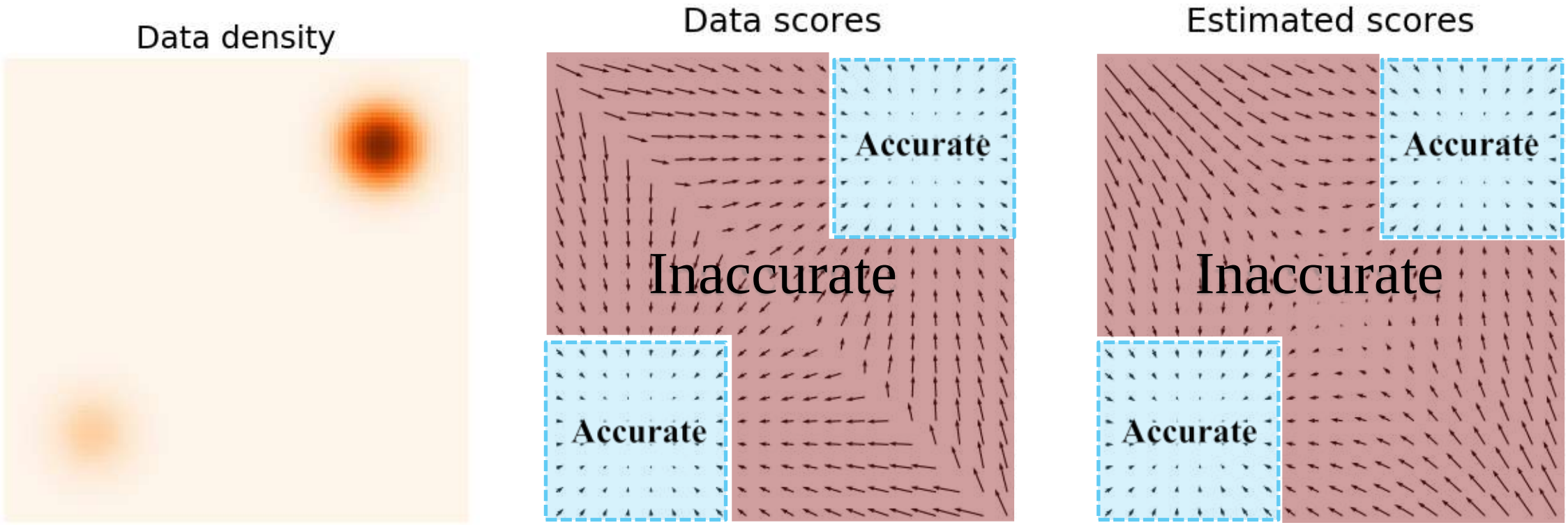
Scores

$$\boldsymbol{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$$

Langevin dynamics

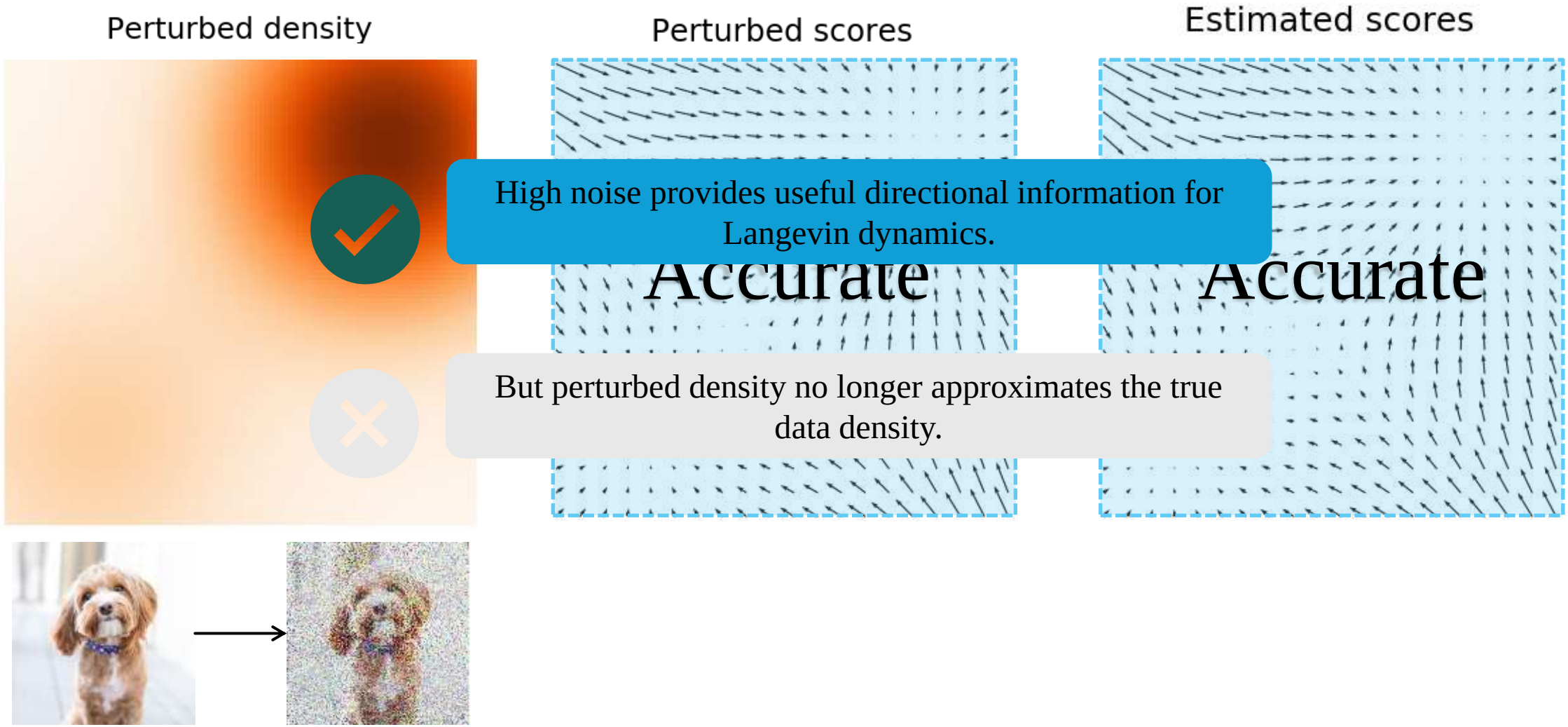
New samples

Challenge in low data density regions



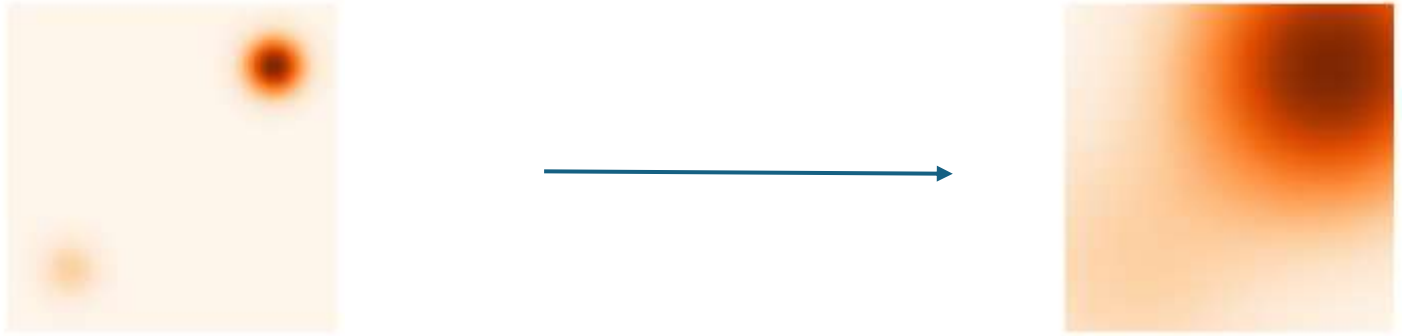
Song and Ermon. “Generative Modeling by Estimating Gradients of the Data Distribution.” NeurIPS 2019.

Improving score estimation by adding noise



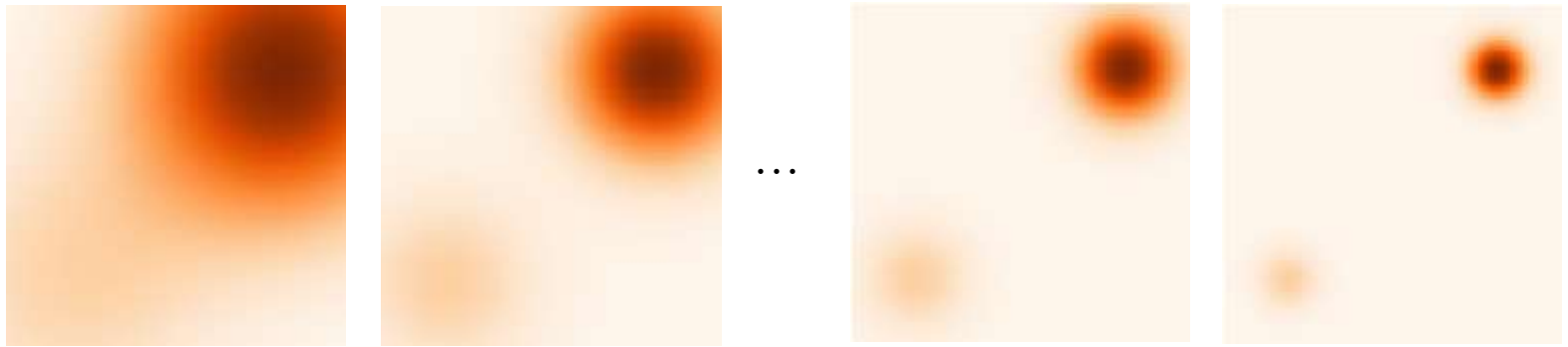
Multi-scale Noise Perturbation

- How much noise to add?

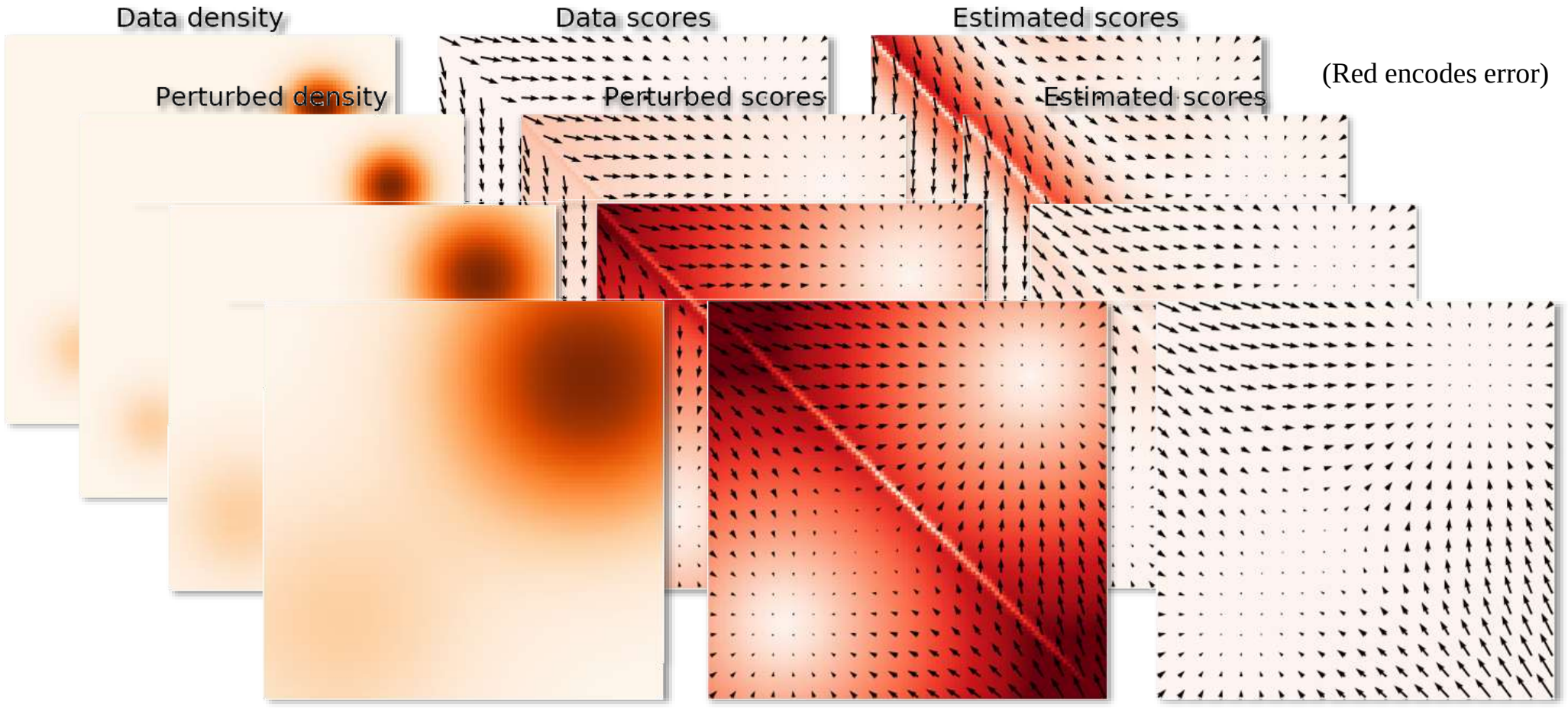


- Multi-scale noise perturbations.

$$\sigma_1 > \sigma_2 > \cdots > \sigma_{L-1} > \sigma_L$$



Trading off Data Quality and Estimation Accuracy

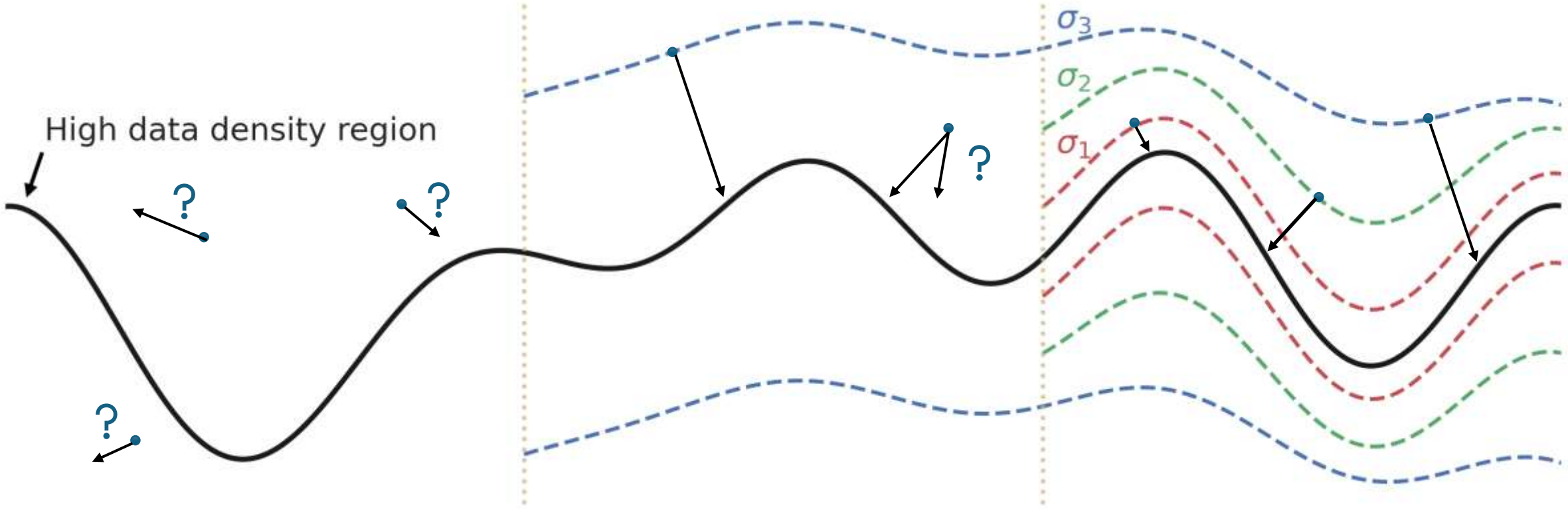


(Red encodes error)

Worse data quality!

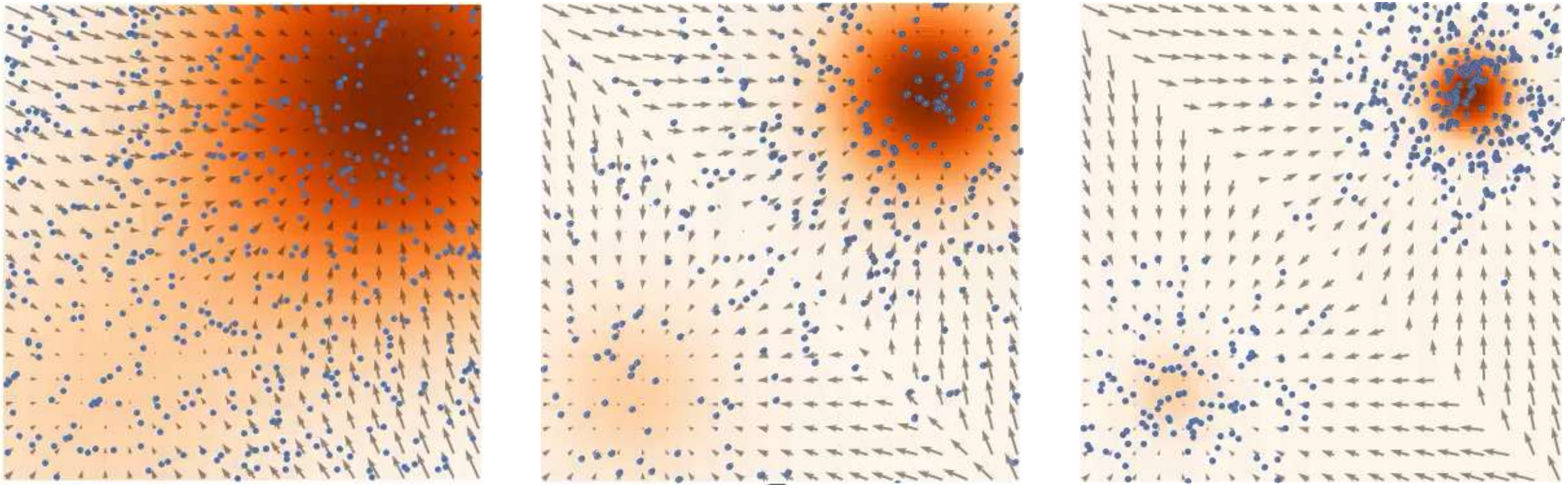
Better score estimation!

Using multiple noise scales

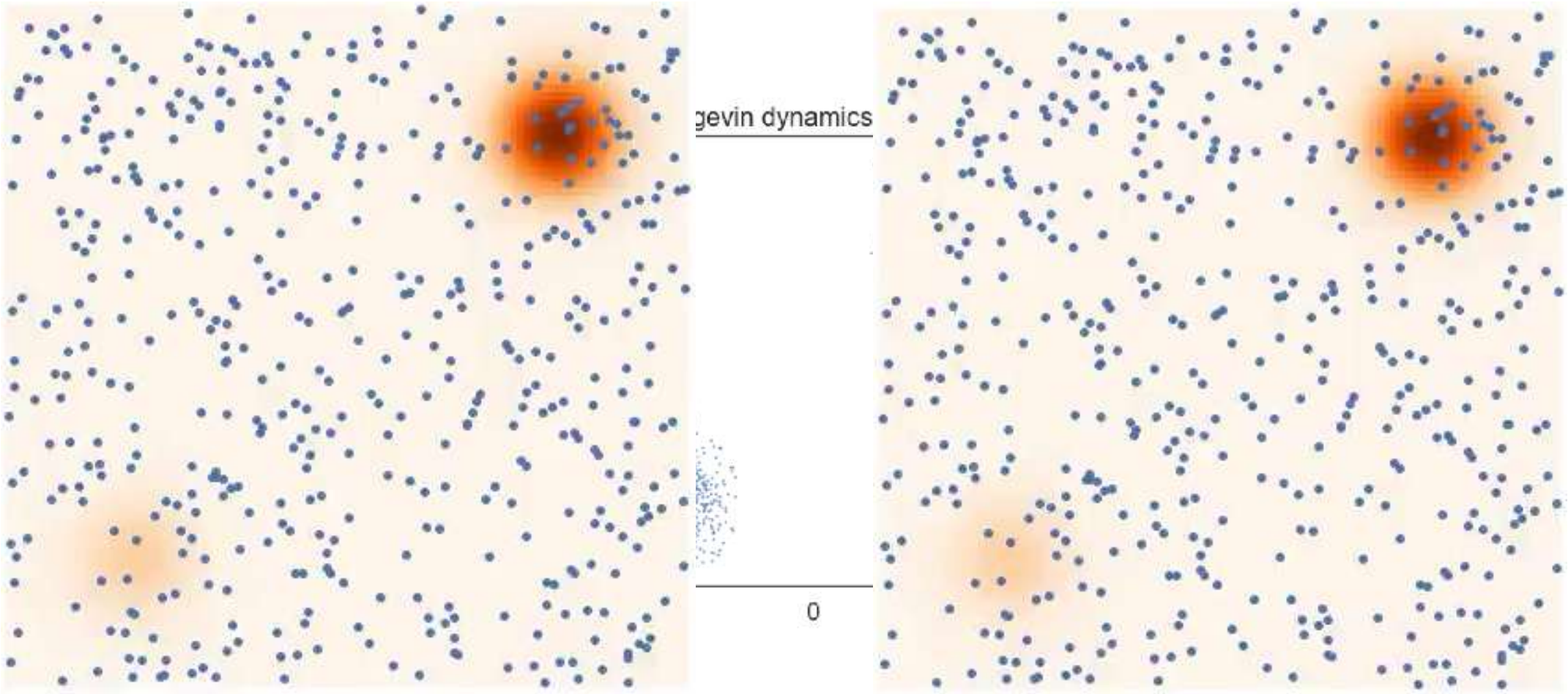


Annealed Langevin Dynamics: Joint Scores to Samples

- Sample using $\sigma_1, \sigma_2, \cdots, \sigma_L$ sequentially with Langevin dynamics.
- Anneal down the noise level.
- Samples used as initialization for the next level.



Comparison to the vanilla Langevin dynamics

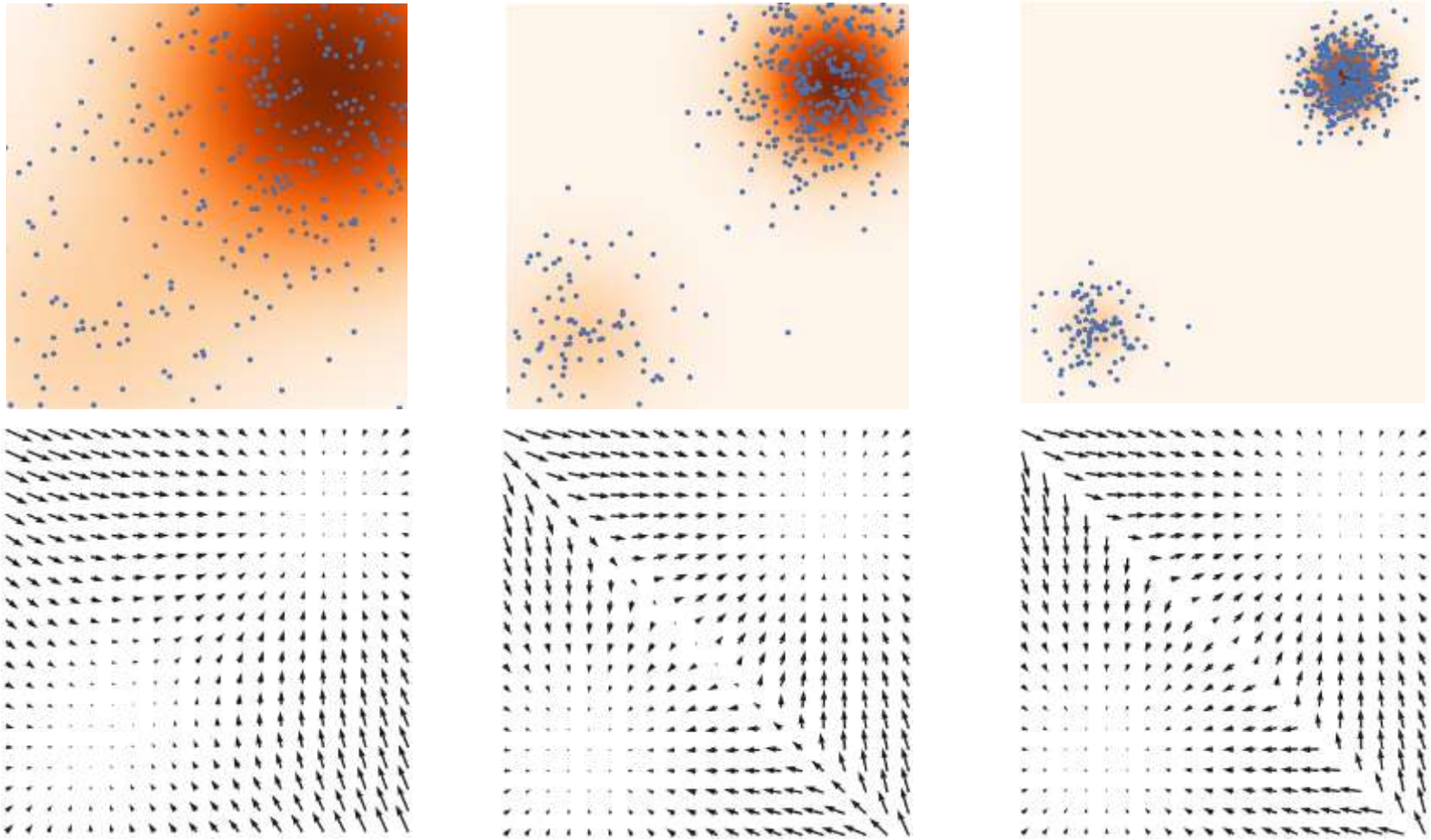


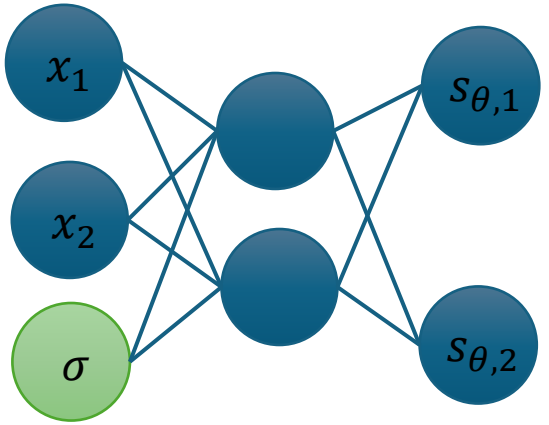
Langevin dynamics

Annealed Langevin dynamics

Joint Score Estimation via Noise Conditional Score Networks

σ_1 σ_2 σ_3



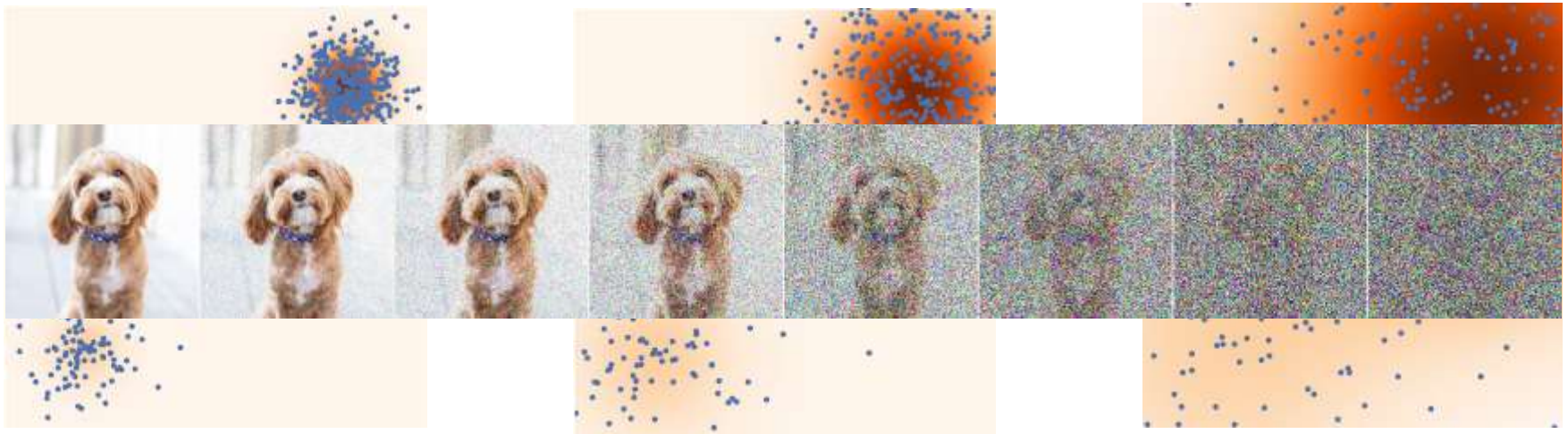


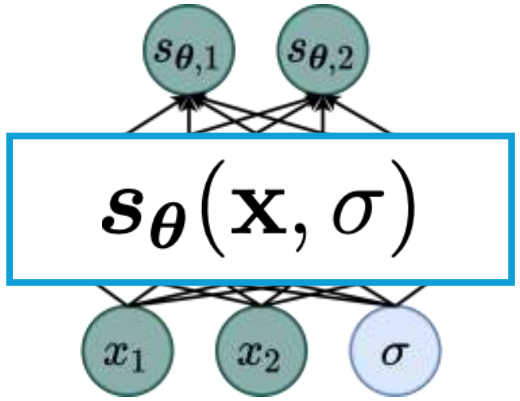
Noise Conditional Score Network (NCSN)

Using multiple noise levels

$p_{\sigma_1}(\mathbf{x}) < p_{\sigma_2}(\mathbf{x}) < p_{\sigma_3}(\mathbf{x})$

Data





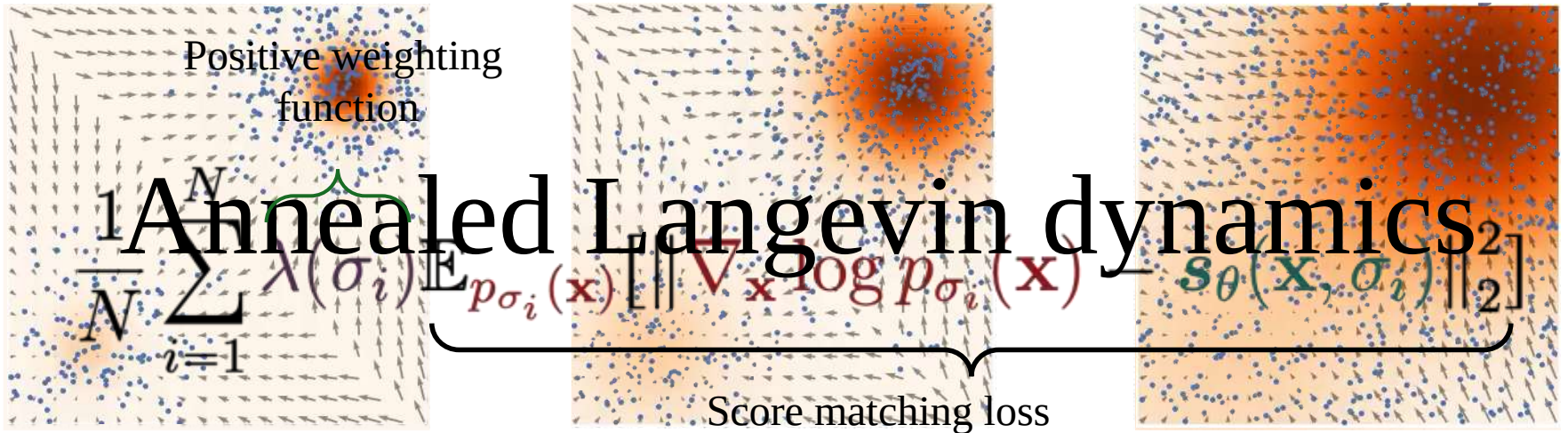
Noise Conditional Score Model

Positive weighting function

$$\frac{1}{N}\sum_{i=1}^{N}\lambda(\sigma_i)\mathbb{E}_{p_{\sigma_i}(\mathbf{x})}[\|\nabla_{\mathbf{x}}\log p_{\sigma_i}(\mathbf{x}) - s_\theta(\mathbf{x}, \sigma_i)\|_2^2]$$

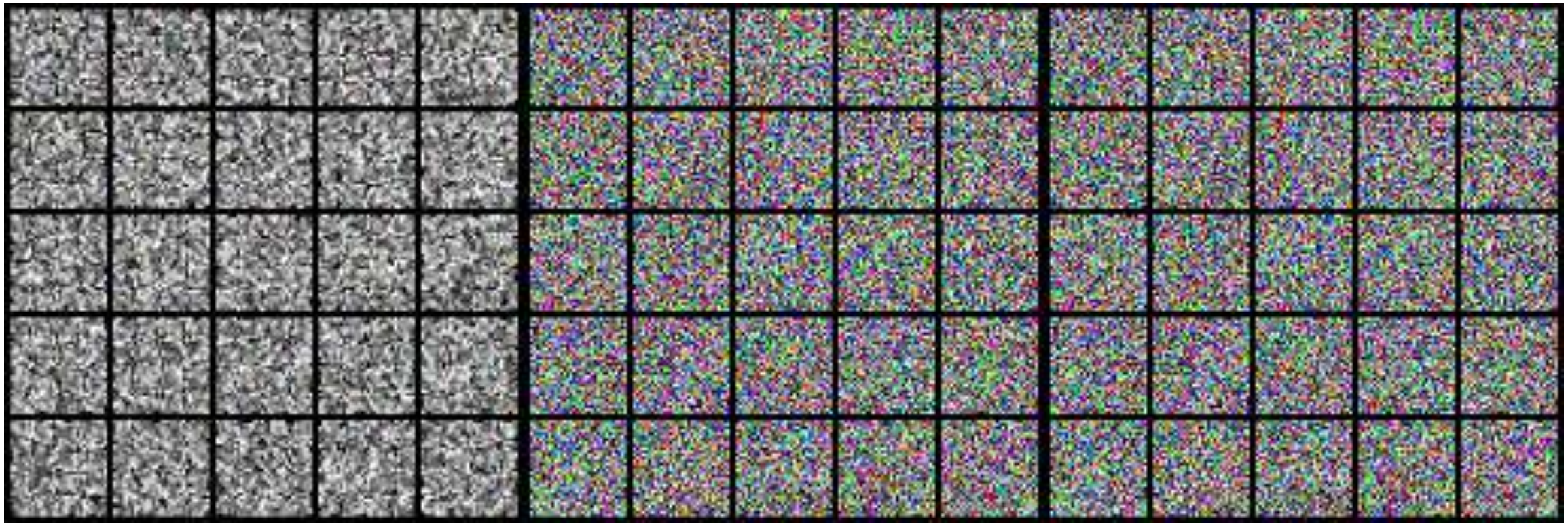
Score matching loss

Annealed Langevin dynamics

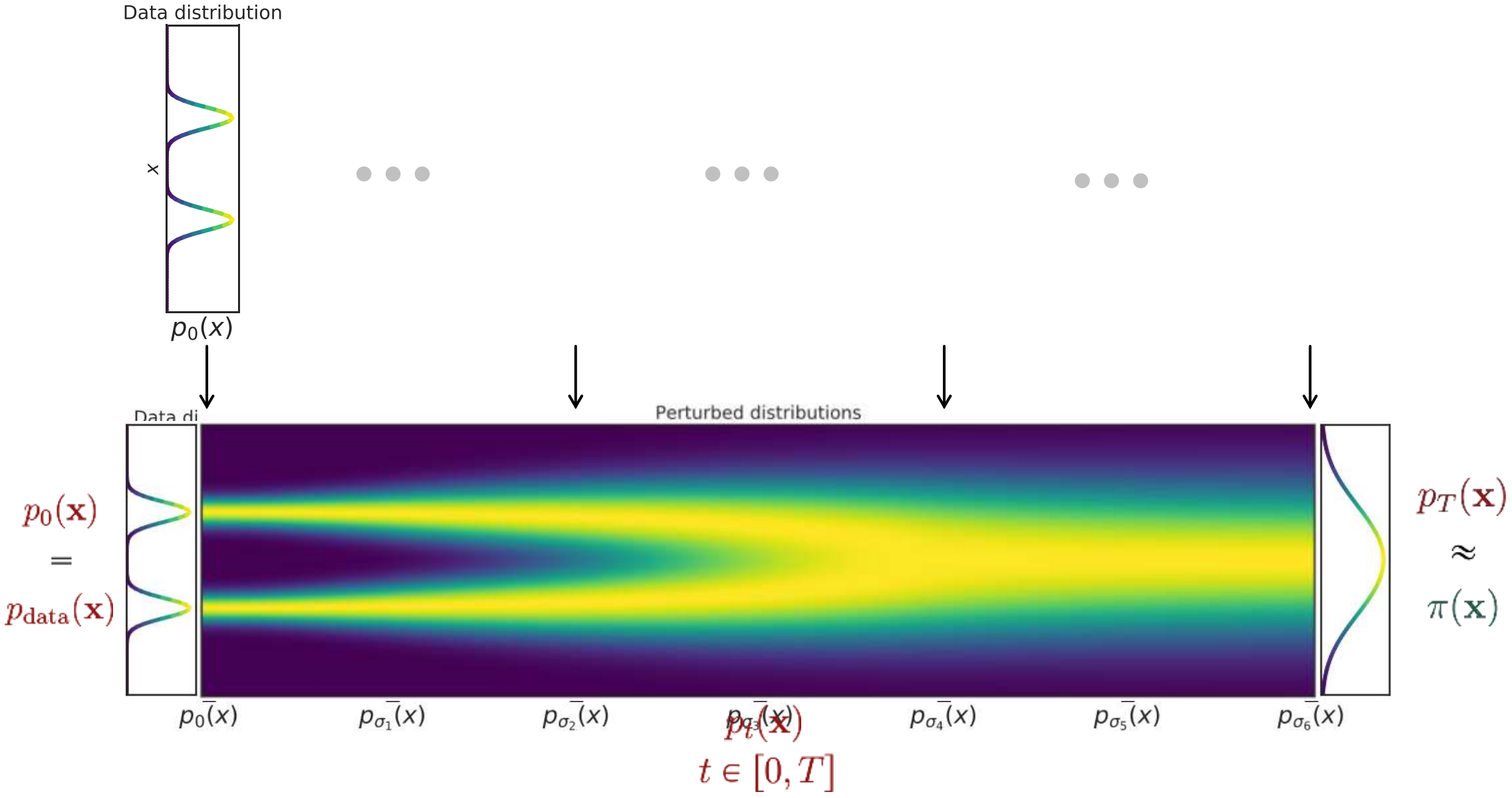


$s_\theta(\mathbf{x}, \sigma_1)$ $s_\theta(\mathbf{x}, \sigma_2)$ $s_\theta(\mathbf{x}, \sigma_3)$

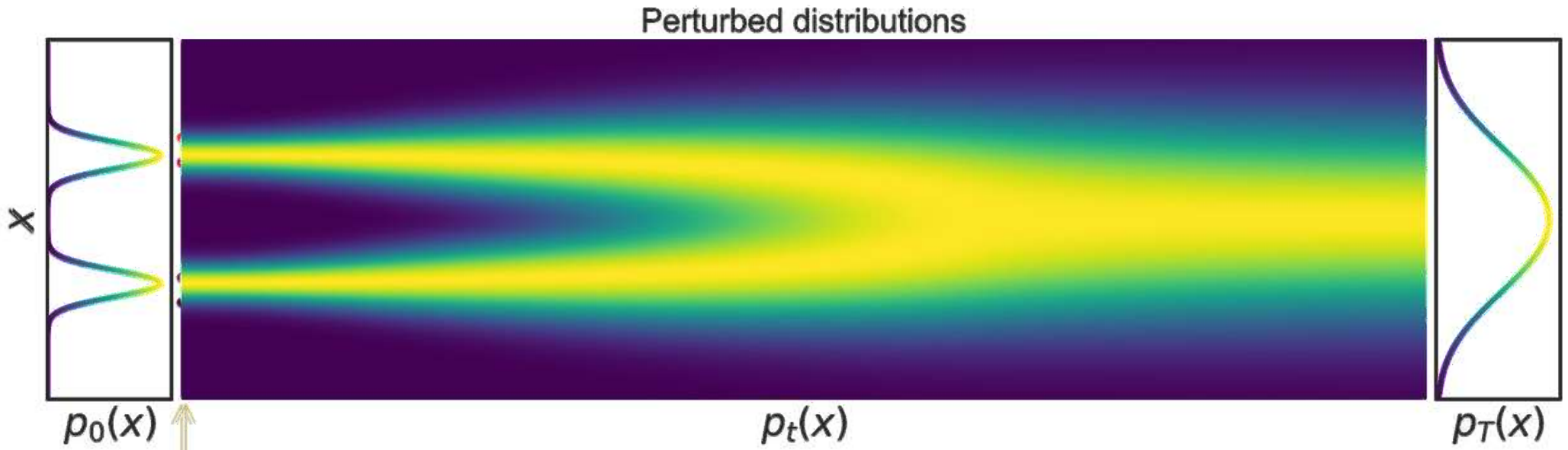
Experiments: Sampling



Infinite noise levels



Perturbing data with stochastic processes



Stochastic process

$\{\mathbf{x}_t\}_{t\in[0,T]}$

Probability densities

$\{p_t(\mathbf{x})\}_{t\in[0,T]}$

Stochastic differential equation (SDE)

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

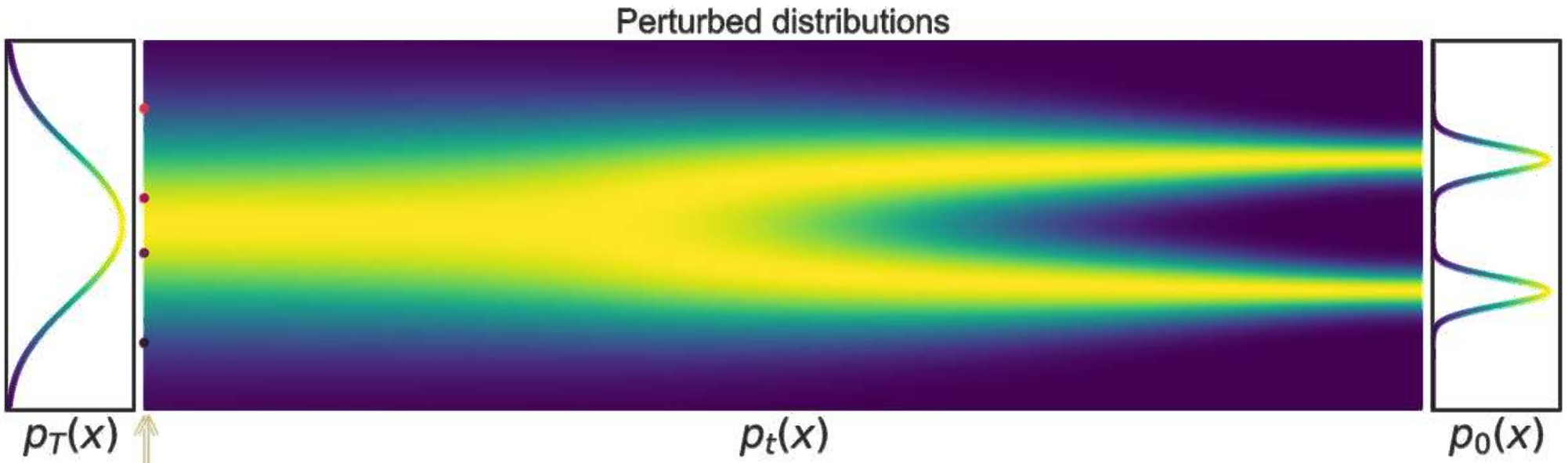
Deterministic drift — Infinitesimal noise

WLOG: Toy SDE

$$\mathrm{d}\mathbf{x}_t = \sigma(t)\,\mathrm{d}\mathbf{w}_t$$

$$p_T(\mathbf{x}) \approx \pi(\mathbf{x})$$

Generation via reverse stochastic processes



$\pi(\mathbf{x}) \approx p_T(\mathbf{x})$

Forward SDE (t: 0→T)

$$\mathrm{d}\mathbf{x}_t = \sigma(t)\,\mathrm{d}\mathbf{w}_t$$

Reverse SDE (t: T→0)

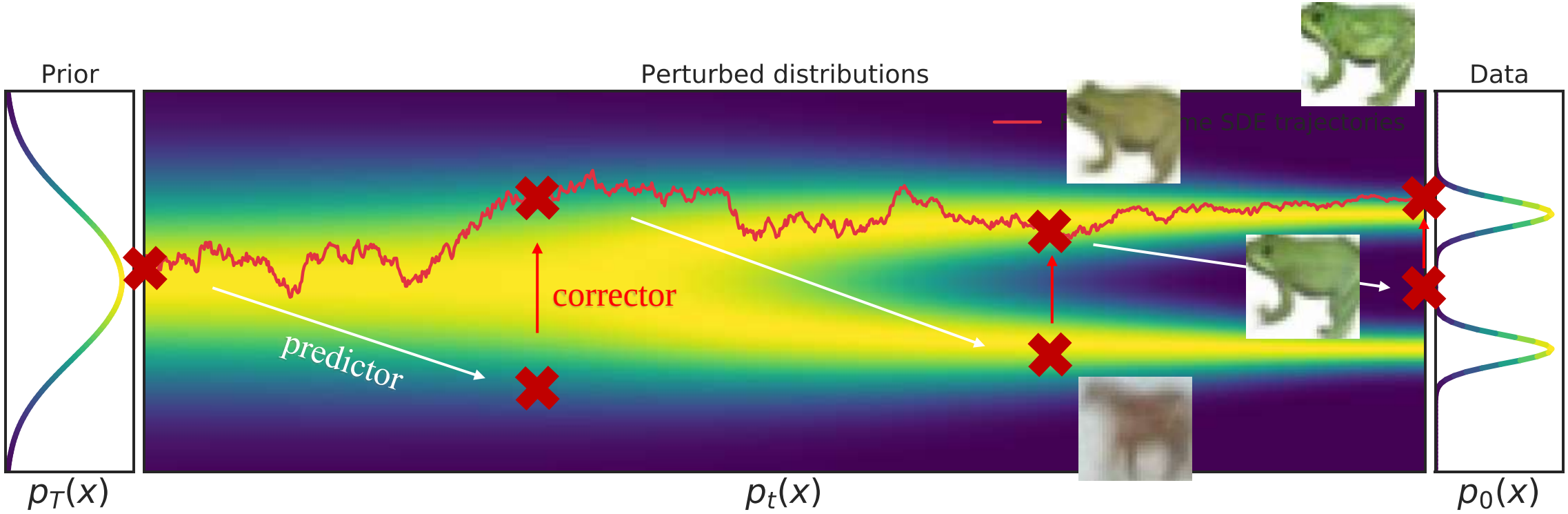
$$\mathrm{d}\mathbf{x}_t = -\sigma(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\,\mathrm{d}t + \sigma(t)\,\mathrm{d}\bar{\mathbf{w}}_t$$

Infinitesimal noise in the reverse time direction

Score function!

Predictor-Corrector sampling methods

- Predictor-Corrector sampling.
  - **Predictor:** Numerical SDE solver
  - **Corrector:** Score-based MCMC



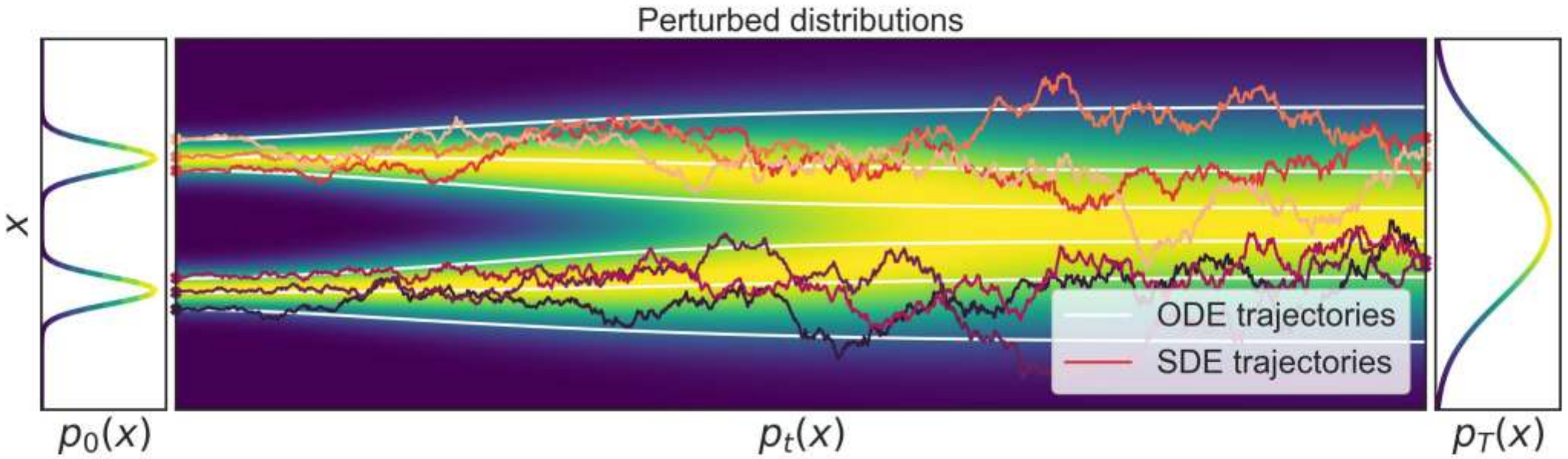
Predictor-Corrector sampling methods

- Predictor-Corrector sampling.
  - **Predictor:** Numerical SDE solver
  - **Corrector:** Score-based MCMC

Algorithm 3 PC sampling (VP SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $i = N - 1$ **to** 0 **do**
  $\mathbf{x}'_i \leftarrow (2 - \sqrt{1 - \beta_{i+1}})\mathbf{x}_{i+1} + \beta_{i+1}\mathbf{s}_{\boldsymbol{\theta}*}(\mathbf{x}_{i+1}, i + 1)$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}'_i + \sqrt{\beta_{i+1}}\mathbf{z}$ — Predictor
  for $j = 1$ **to** M **do** — Corrector
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}*}(\mathbf{x}_i, i) + \sqrt{2\epsilon_i}\mathbf{z}$
return $\mathbf{x}_0$

Converting the SDE to an ODE

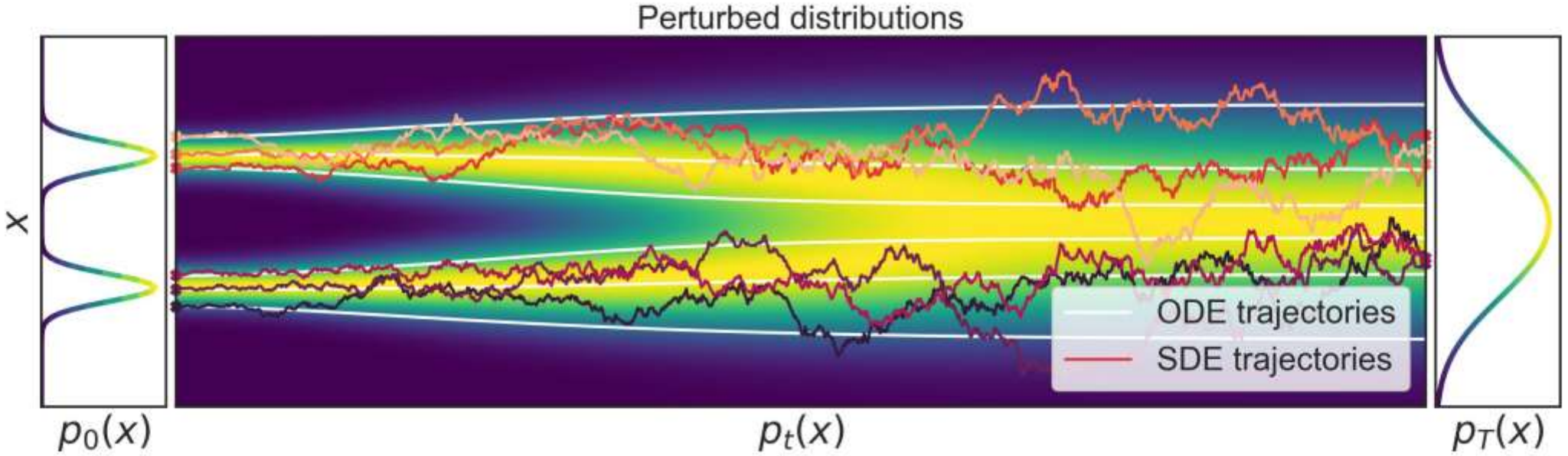


SDE **Ordinary differential equation (ODE)**

$$\mathrm{d}\mathbf{x}_t = \sigma(t)\,\mathrm{d}\mathbf{w}_t \quad \Longleftrightarrow \quad \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = -\frac{1}{2}\sigma(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$$

Score function
$\approx s_\theta(\mathbf{x}, t)$

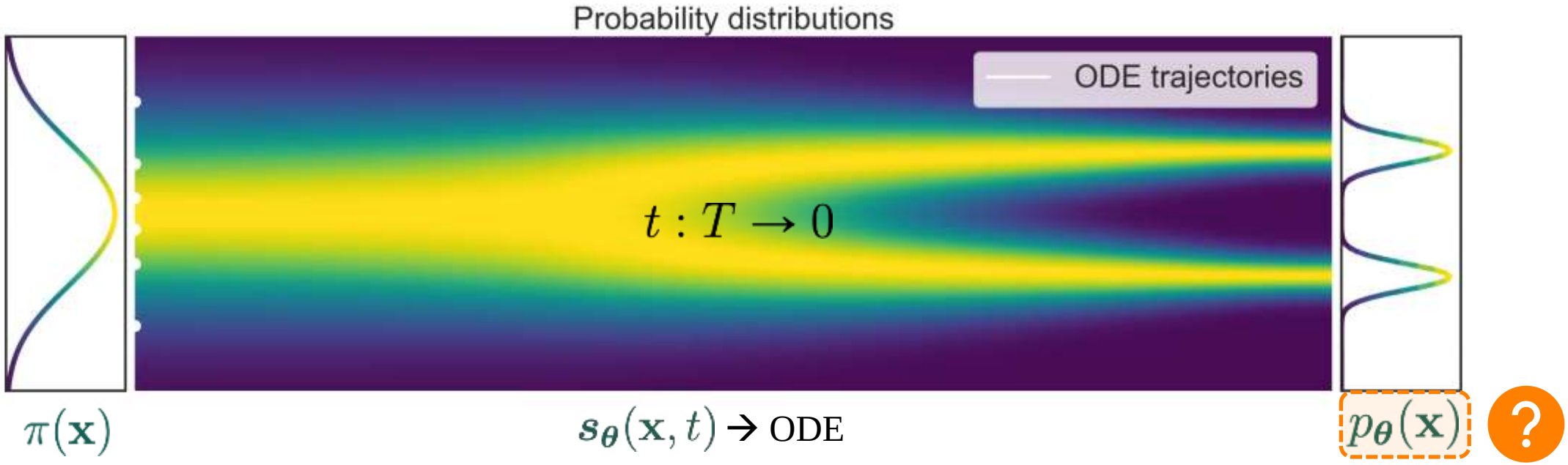
Converting the SDE to an ODE



We can think of this as a (continuous time, infinite depth) normalizing flow

1. Unique ODE solution → Invertible mapping
2. To invert, solve ODE backwards from T to 0

Evaluating the probabilities with ODEs



Computing the probability density function (change of variables formula)

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}_0) = \log \pi(\mathbf{x}_T) - \frac{1}{2}\int_0^T \sigma(t)^2 \operatorname{trace}\left(\nabla_{\mathbf{x}} \boldsymbol{s}_{\boldsymbol{\theta}}(\mathbf{x}, t)\right) \mathrm{d}t$$

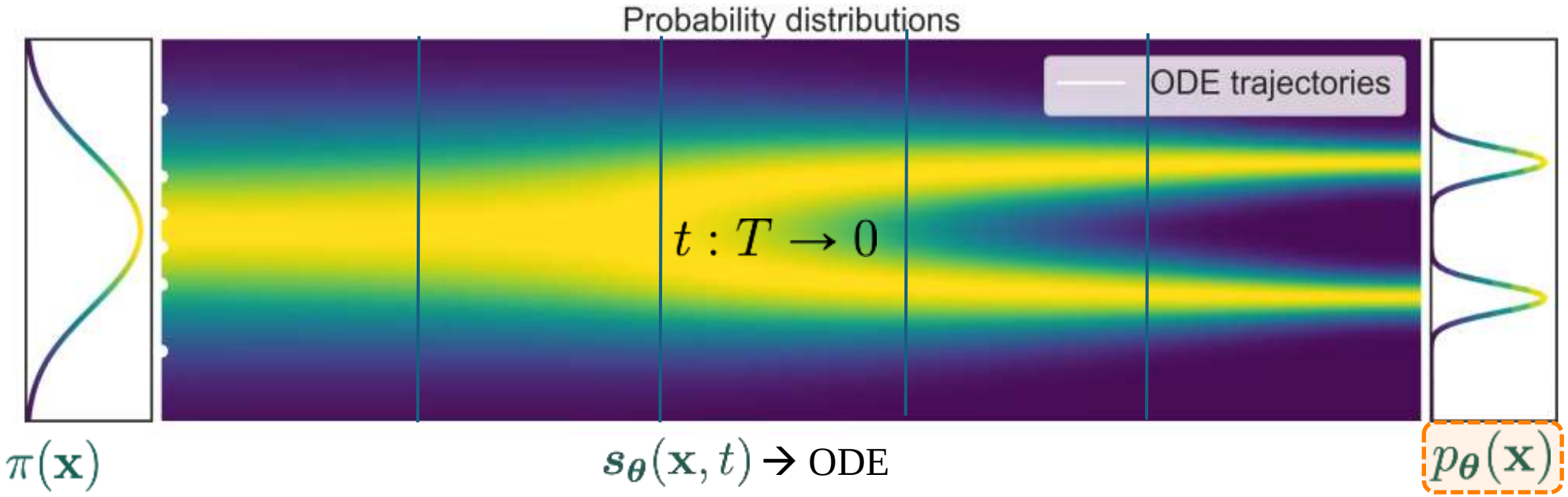
ODE solver

Computable in polynomial time

Unbiased estimator

- **It is a (continuous-time) normalizing flow model!**

Accelerated sampling



- Numerical methods + ODE formulation to accelerate sampling
- DDIM [Song and Ermon, 2021]:
  - Coarsely discretize the time axis, take big steps
  - Corresponds to exponential integrator (semi-linear ODE) [Lu et al, 2022; Zhang and Chen, 2022]
  - 10x-50x speedups, comparable sample quality

Accelerated sampling

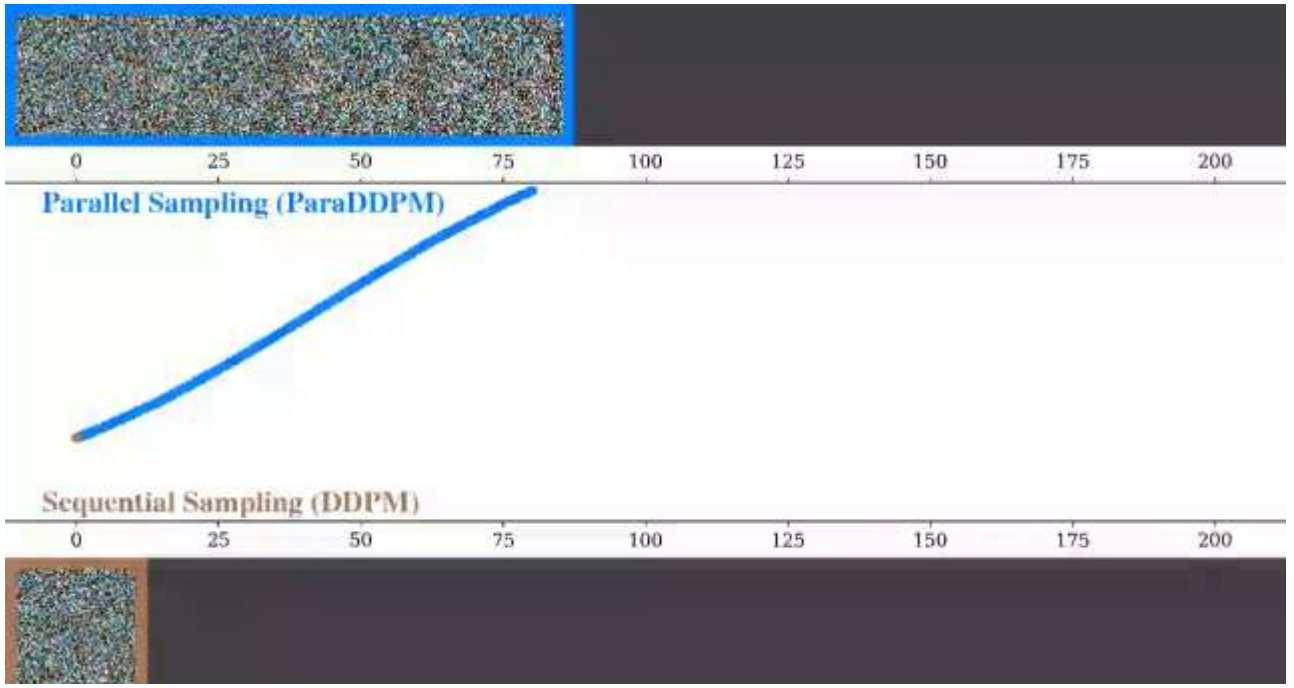
		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	18.36	8.01	5.78	4.73	33.12	26.03	18.48	13.93	5.98
	$\hat{\sigma}$	367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**

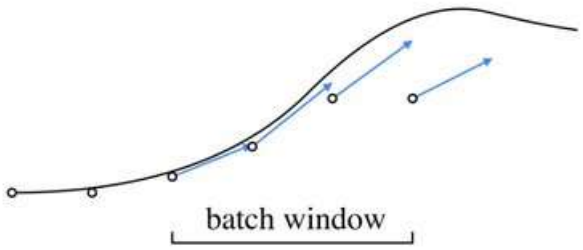
- Numerical methods + ODE formulation to accelerate sampling
- DDIM [Song and Ermon, 2021]:
  - Coarsely discretize the time axis, take big steps
  - Corresponds to exponential integrator (semi-linear ODE) [Lu et al, 2022; Zhang and Chen, 2022]
  - 10x-50x speedups, comparable sample quality

Compared to DDPM, DDIM is able to:

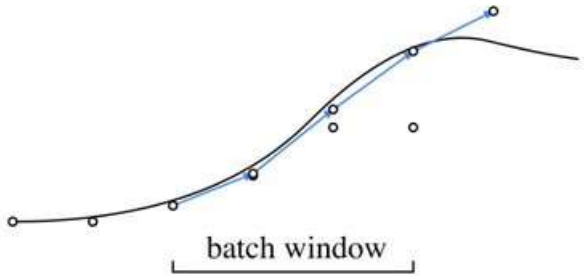
- Generate higher-quality samples using a much fewer number of steps.
- Have “consistency” property since the generative process is deterministic, meaning that multiple samples conditioned on the same latent variable should have similar high-level features.
- Because of the consistency, DDIM can do semantically meaningful interpolation in the latent variable.

Parallel ODE solving

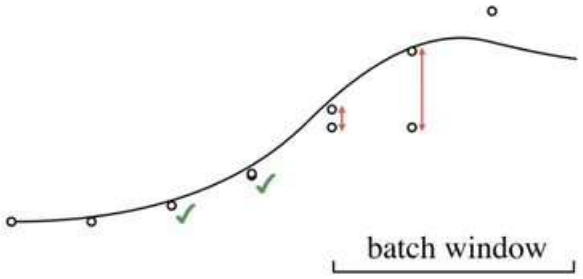




(a) Compute the drift of $x_{t:t+p}^{k}$ on a batch window of size $p = 4$, in parallel

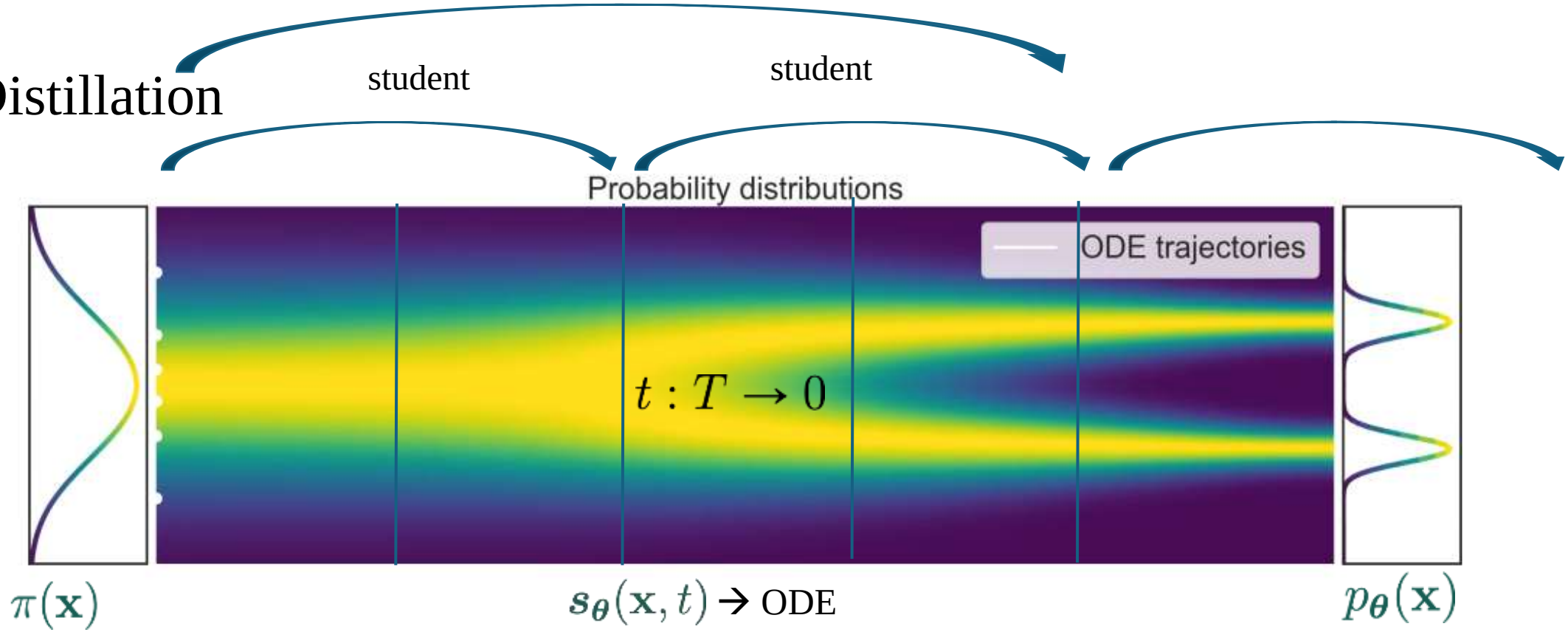


(b) Update the values to $x_{t:t+p}^{k+1}$ using the cumulative drift of points in the window



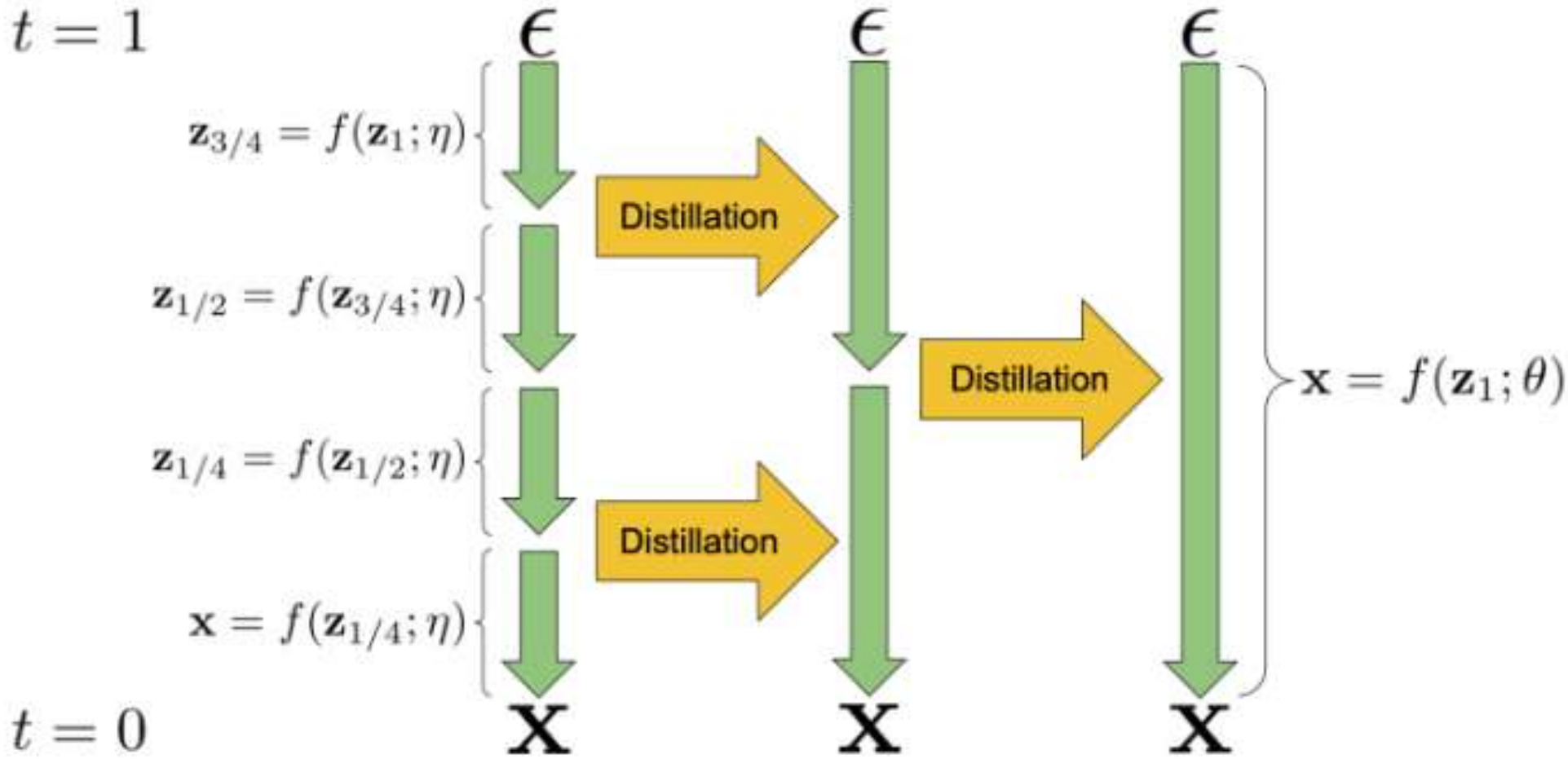
(c) Determine how far to slide the window forward, based on the error $\|x_i^{k+1} - x_i^k\|^2$.

Distillation



- Progressive distillation [Salimans, Ho 2022]
  - DDIM sampler as a teacher model
  - Student model trained to do in 1 step what DDIM achieves in 2 steps
  - Applied recursively to drastically reduce the number of steps required

Distillation



Algorithm 1 Standard diffusion training

Require: Model $\hat{\mathbf{x}}_\theta(\mathbf{z}_t)$ to be trained
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$

while not converged **do**
  $\mathbf{x} \sim \mathcal{D}$ ▷ Sample data
  $t \sim U[0, 1]$ ▷ Sample time
  $\epsilon \sim N(0, I)$ ▷ Sample noise
  $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ ▷ Add noise to data

  $\tilde{\mathbf{x}} = \mathbf{x}$ ▷ Clean data is target for $\hat{\mathbf{x}}$
  $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$ ▷ log-SNR
  $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$ ▷ Loss
  $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$ ▷ Optimization
end while

Algorithm 2 Progressive distillation

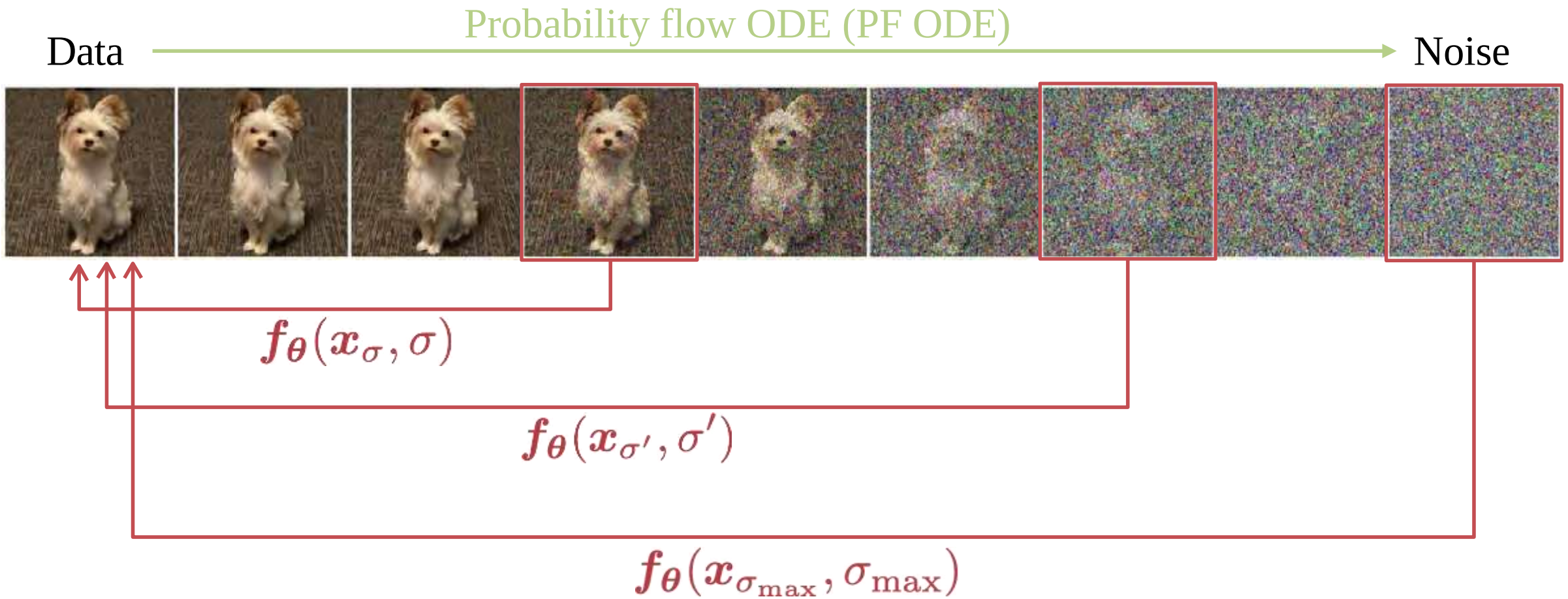
Require: Trained teacher model $\hat{\mathbf{x}}_\eta(\mathbf{z}_t)$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N
for K iterations **do**
  $\theta \leftarrow \eta$ ▷ Init student from teacher
  while not converged **do**
    $\mathbf{x} \sim \mathcal{D}$
    $t = i/N, \;\; i \sim Cat[1, 2, \ldots, N]$
    $\epsilon \sim N(0, I)$
    $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
    `# 2 steps of DDIM with teacher`
    $t' = t - 0.5/N, \;\; t'' = t - 1/N$
    $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t}(\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_\eta(\mathbf{z}_t))$
    $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}))$
    $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$ ▷ Teacher $\hat{\mathbf{x}}$ target
    $\lambda_t = \log[\alpha_t^2 / \sigma_t^2]$
    $L_\theta = w(\lambda_t) \|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$
    $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
  end while
  $\eta \leftarrow \theta$ ▷ Student becomes next teacher
  $N \leftarrow N/2$ ▷ Halve number of sampling steps
end for

On distillation of guided diffusion models

(a) 2 denoising steps (b) 4 denoising steps (c) 8 denoising steps

Meng, Rombach, Gao, Kingma, Ermon, Ho, Salimans “On distillation of guided diffusion models.” 2023.

Consistency models are designed for one-step generation

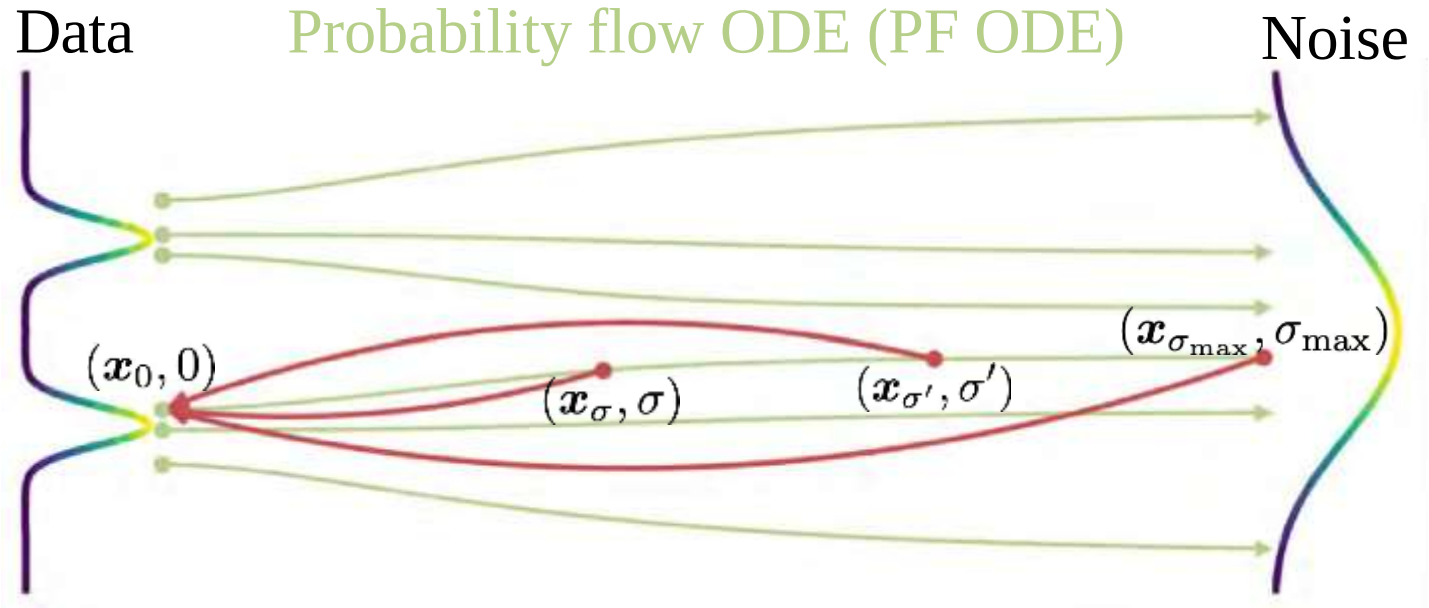


?

How does this differ from a denoiser?

Song, Dhariwal, Chen, Sutskever. Consistency Models. ICML 2023

Consistency models are designed for one-step generation



Boundary condition

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_0, 0) = \mathbf{x}_0$$

Enforced via network parameterization

Consistency models are trained to map points on any trajectory of the PF ODE to the trajectory's origin **in one step**.

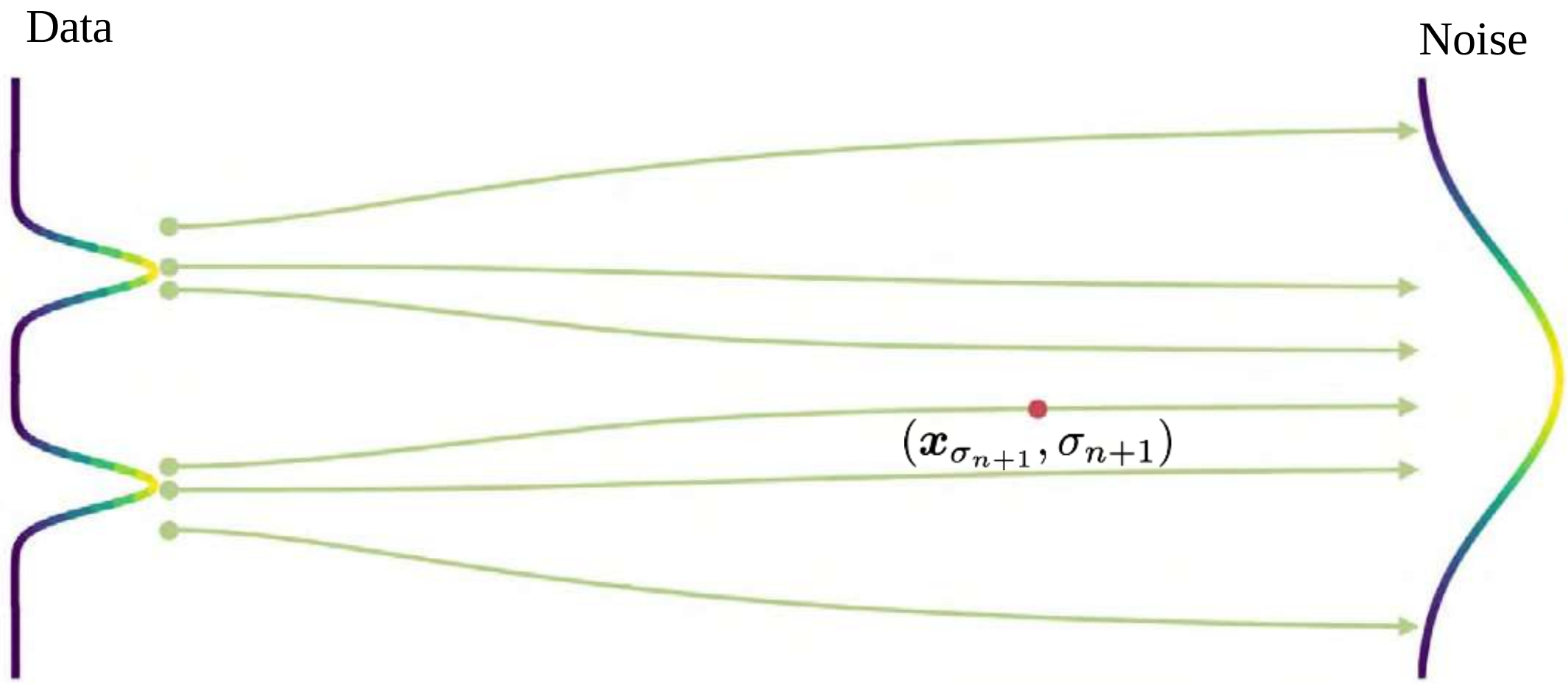
$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_\sigma, \sigma) = \boldsymbol{x}_0$$

Enforced via learning

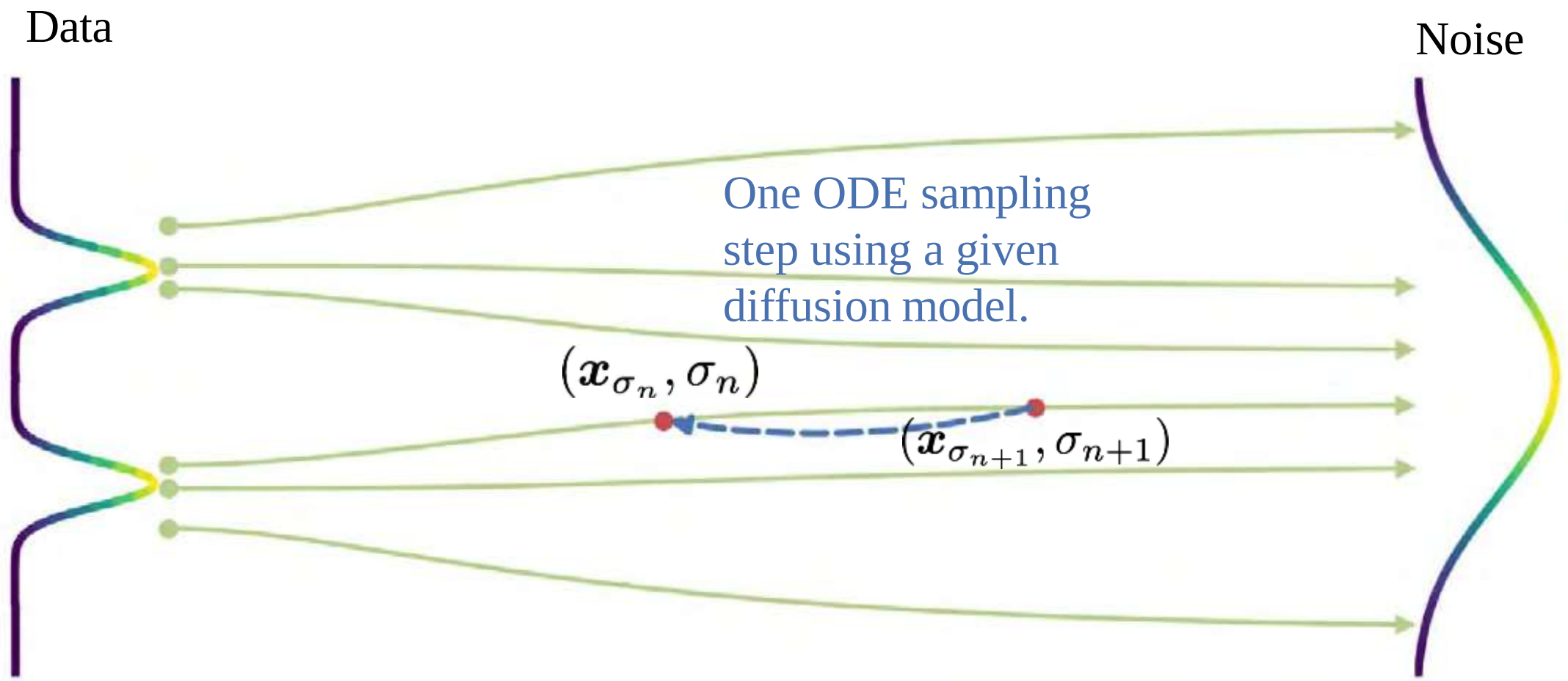
Self-consistency

$$\forall \sigma, \sigma' \in [0, \sigma_{\max}] : \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_\sigma, \sigma) = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_{\sigma'}, \sigma')$$

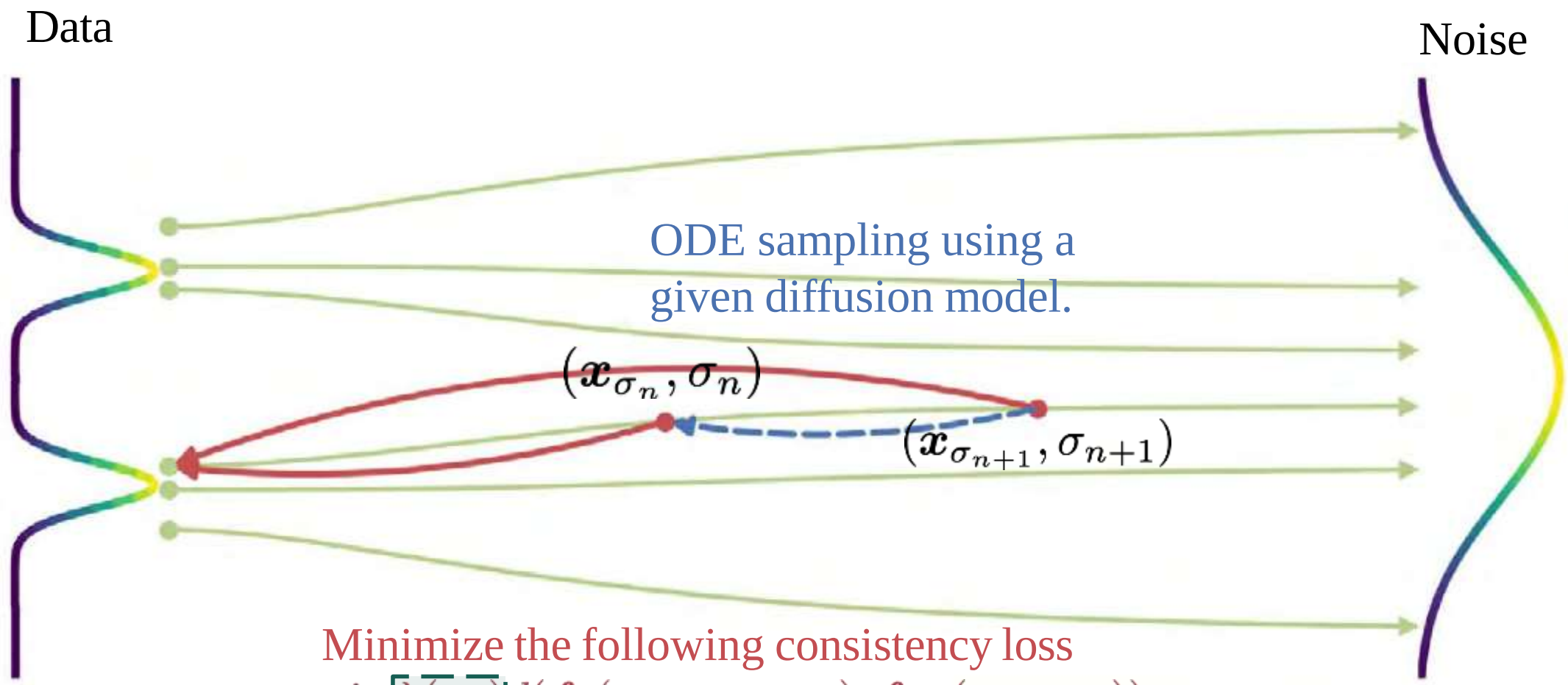
Training consistency models via distillation



Training consistency models via distillation



Training consistency models via distillation



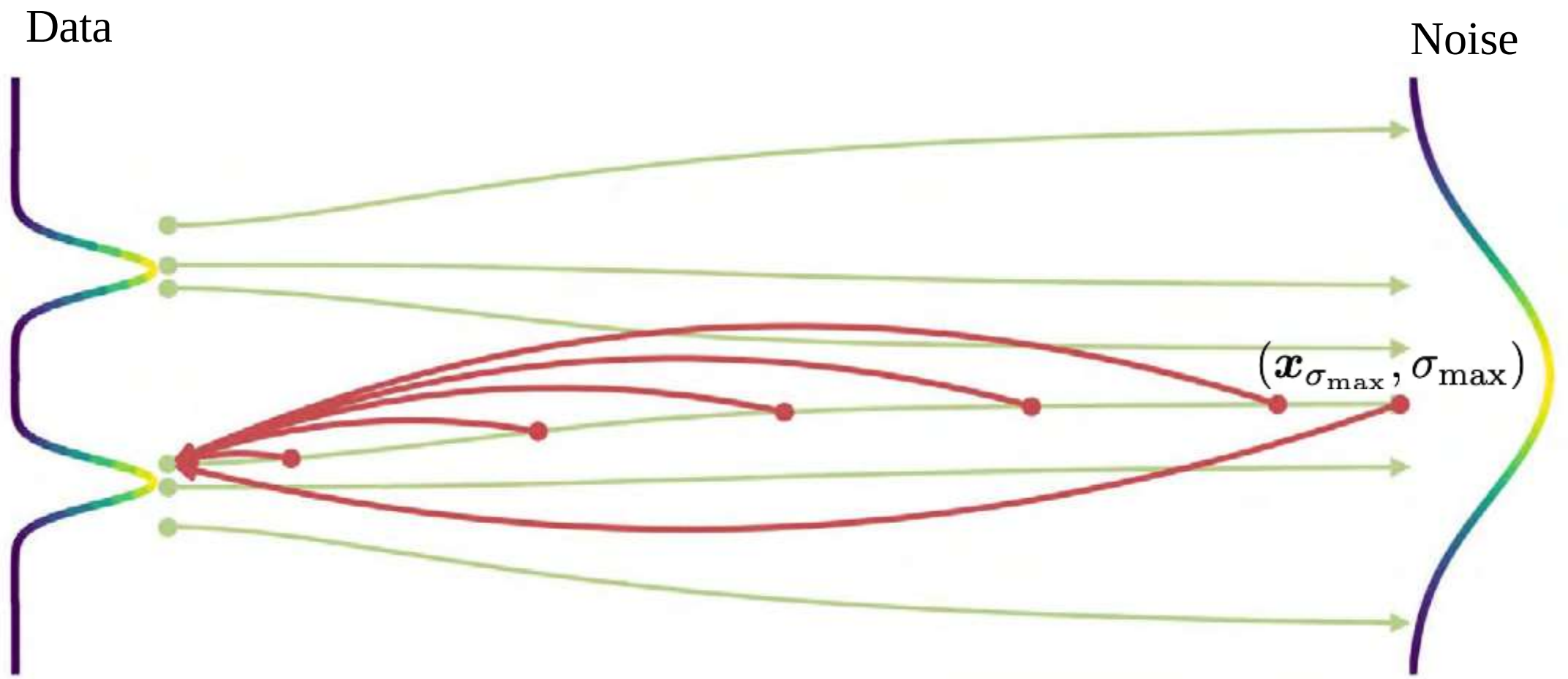
Minimize the following consistency loss

$$\min_{\boldsymbol{\theta}} \lambda(\sigma_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_{\sigma_{n+1}}, \sigma_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\boldsymbol{x}_{\sigma_n}, \sigma_n))$$

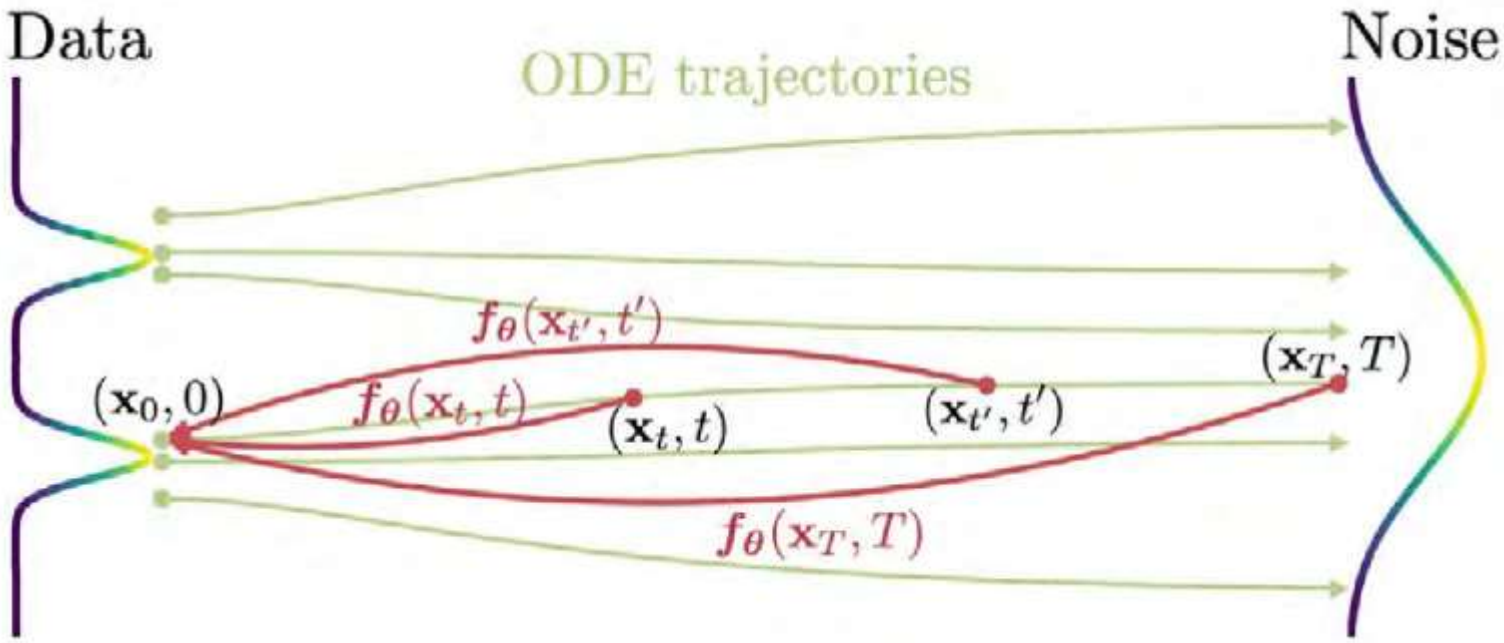
Weighting function

$$\boldsymbol{\theta}^- = \mathrm{EMA}_{\mu}(\boldsymbol{\theta})$$

Training consistency models via distillation



Enforcing self-consistency via distillation



$$\min_{\theta} \|\boldsymbol{f}_{\theta}(\mathbf{x}_{t'}, t') - \boldsymbol{f}_{\theta}(\mathbf{x}_t, t)\|_2^2$$

$\mathbf{x}_0$ ← $\mathbf{x}_t$ ← $\mathbf{x}_{t'}$ ← $\mathbf{x}_T$

ODE solver + pretrained score function

ODE solver + pretrained score function

ODE solver + pretrained score function

$$\min_{\theta} \|\boldsymbol{f}_{\theta}(\mathbf{x}_t, t) - \boldsymbol{f}_{\theta}(\mathbf{x}_0, 0)\|_2^2$$

$$\min_{\theta} \|\boldsymbol{f}_{\theta}(\mathbf{x}_T, T) - \boldsymbol{f}_{\theta}(\mathbf{x}_{t'}, t')\|_2^2$$

Consistency Model

Algorithm 2 Consistency Distillation (CD)

Input: dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate η, ODE solver $\Phi(\cdot, \cdot; \boldsymbol{\phi})$, $d(\cdot, \cdot)$, $\lambda(\cdot)$, and μ

$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$

repeat

Sample $\mathbf{x} \sim \mathcal{D}$ and $n \sim \mathcal{U}[\![1, N-1]\!]$

Sample $\mathbf{x}_{t_{n+1}} \sim \mathcal{N}(\mathbf{x}; t_{n+1}^2 \boldsymbol{I})$

$\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}} \leftarrow \mathbf{x}_{t_{n+1}} + (t_n - t_{n+1}) \Phi(\mathbf{x}_{t_{n+1}}, t_{n+1}; \boldsymbol{\phi})$

$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi}) \leftarrow$

$\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}}, t_n))$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi})$

$\boldsymbol{\theta}^- \leftarrow \text{stopgrad}(\mu \boldsymbol{\theta}^- + (1 - \mu) \boldsymbol{\theta})$

until convergence

Distill a diffusion model into a consistency model by minimizing the difference between model outputs for pairs generated out of the same trajectory. This enables a much cheaper sampling evaluation.

Consistency Model

Algorithm 3 Consistency Training (CT)

Input: dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate η, step schedule $N(\cdot)$, EMA decay rate schedule $\mu(\cdot)$, $d(\cdot,\cdot)$, and $\lambda(\cdot)$

$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$ and $k \leftarrow 0$

repeat

  Sample $\mathbf{x} \sim \mathcal{D}$, and $n \sim \mathcal{U}[\![1, N(k)-1]\!]$

  Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

  $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) \leftarrow$

$$\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x} + t_{n+1}\mathbf{z}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\mathbf{x} + t_n \mathbf{z}, t_n))$$

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)$

  $\boldsymbol{\theta}^- \leftarrow \text{stopgrad}(\mu(k)\boldsymbol{\theta}^- + (1-\mu(k))\boldsymbol{\theta})$

  $k \leftarrow k+1$

until convergence

Train a consistency model independently.

Consistency models distilled from diffusion models

EDM
FID = 2.44
NFE = 79

One step
FID = 6.20
NFE = 1

Two steps
FID = 4.70
NFE = 2

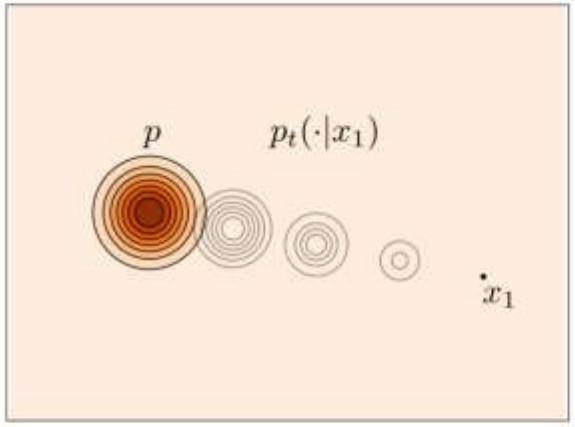
Consistency models distilled from diffusion models

EDM
FID = 3.57
NFE = 79

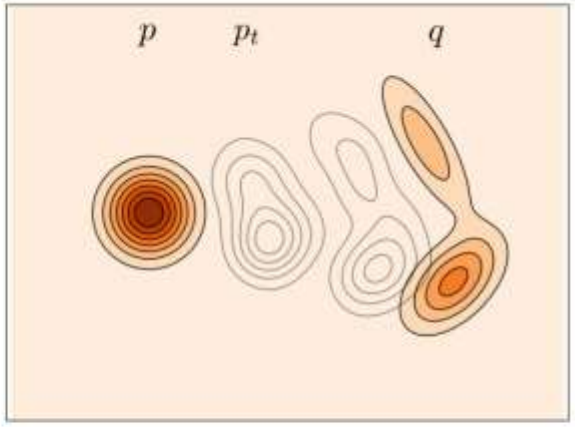
One step
FID = 7.80
NFE = 1

Two steps
FID = 5.22
NFE = 2

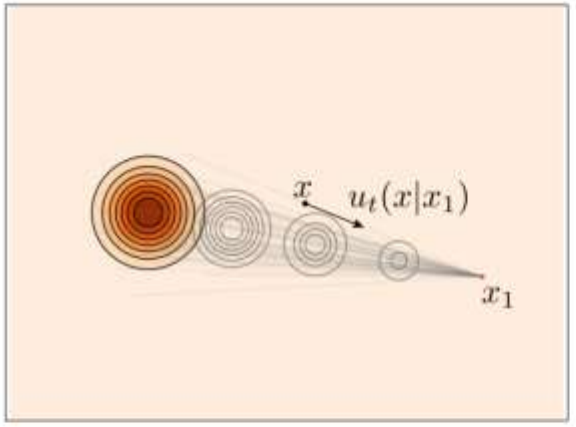
Flow Matching



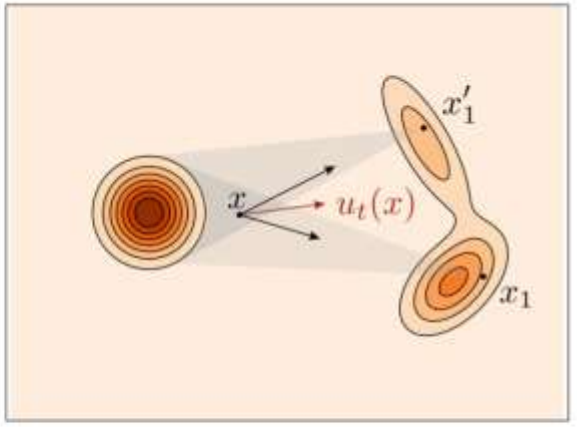
(a) Conditional probability path $p_t(x|x_1)$.



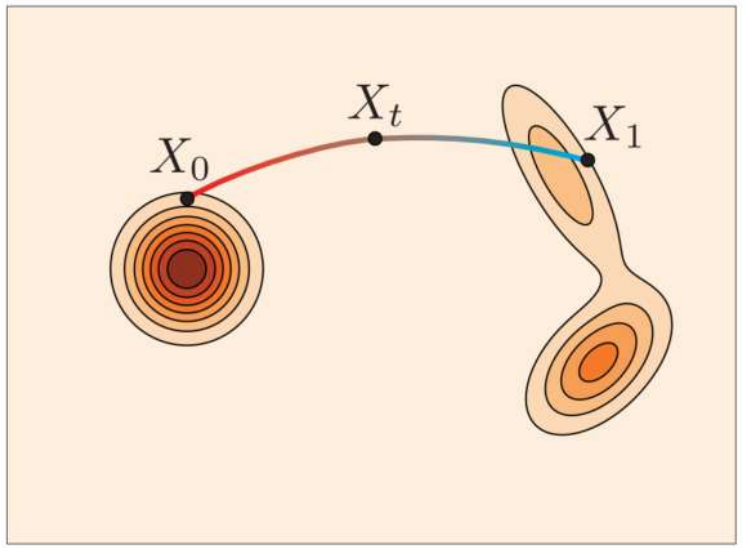
(b) (Marginal) Probability path $p_t(x)$.



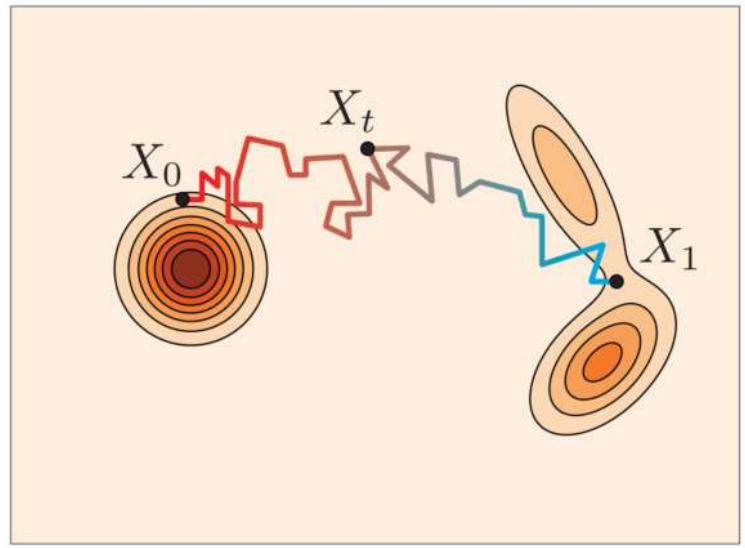
(c) Conditional velocity field $u_t(x|x_1)$.



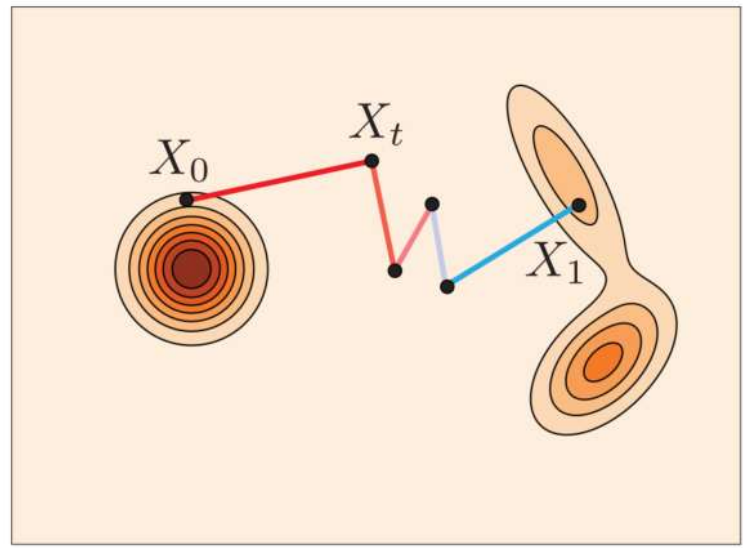
(d) (Marginal) Velocity field $u_t(x)$.



(a) Flow

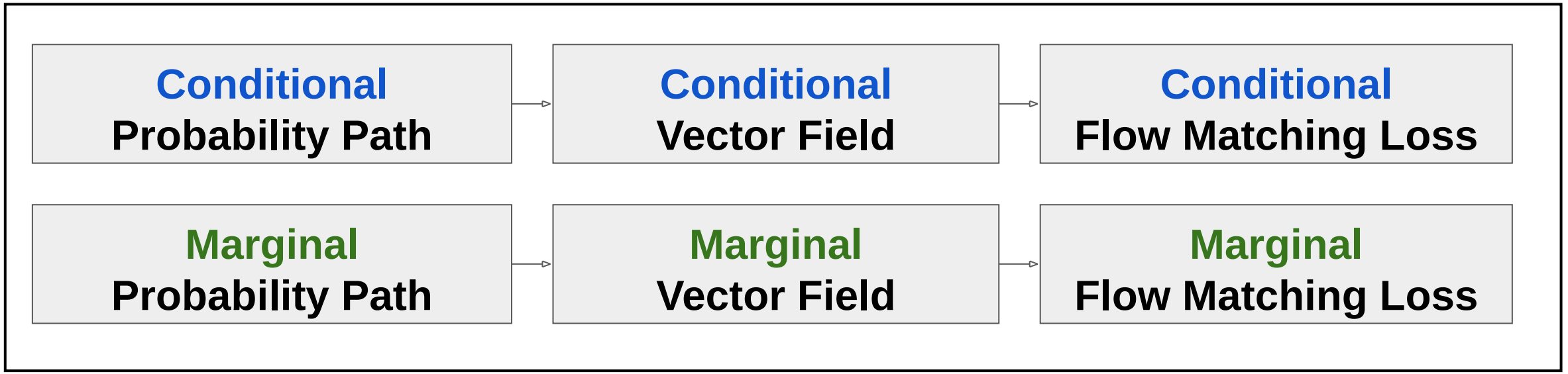


(b) Diffusion



(c) Jump

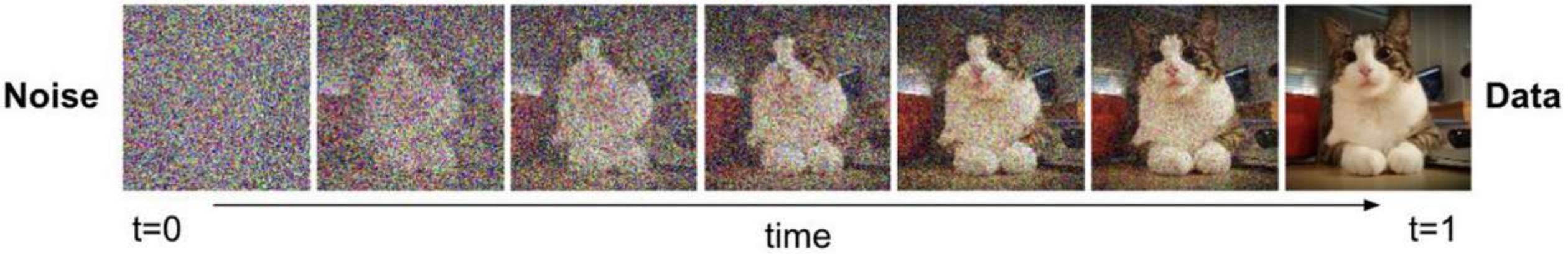
The Flow Matching Matrix

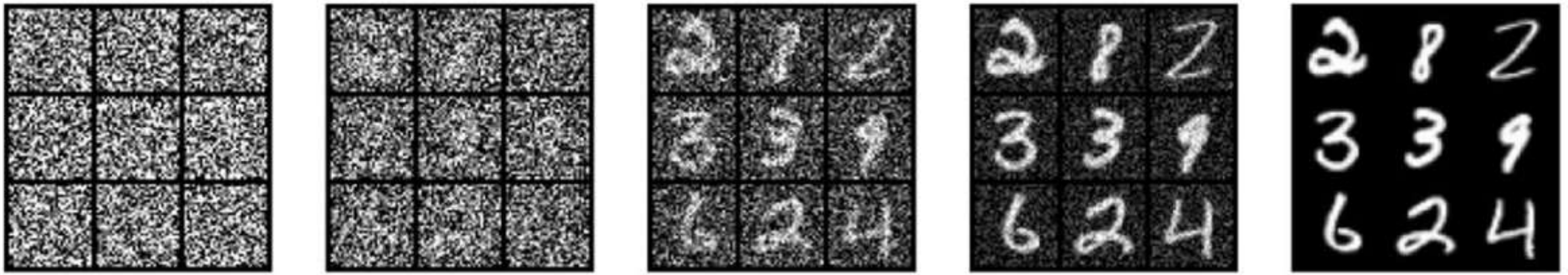


“Conditional” = “Per single data point”

“Marginal” = “Across distribution of data points”

Probability Paths: The Path from Noise to Data

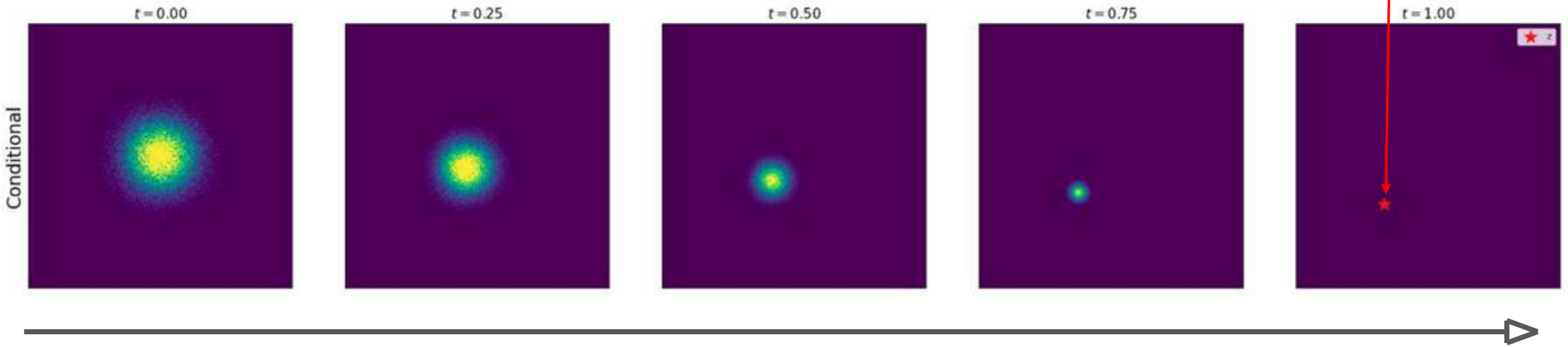




p_{init}

Conditional Probability Path $p_t(\cdot|z)$

z



t=0

t=1

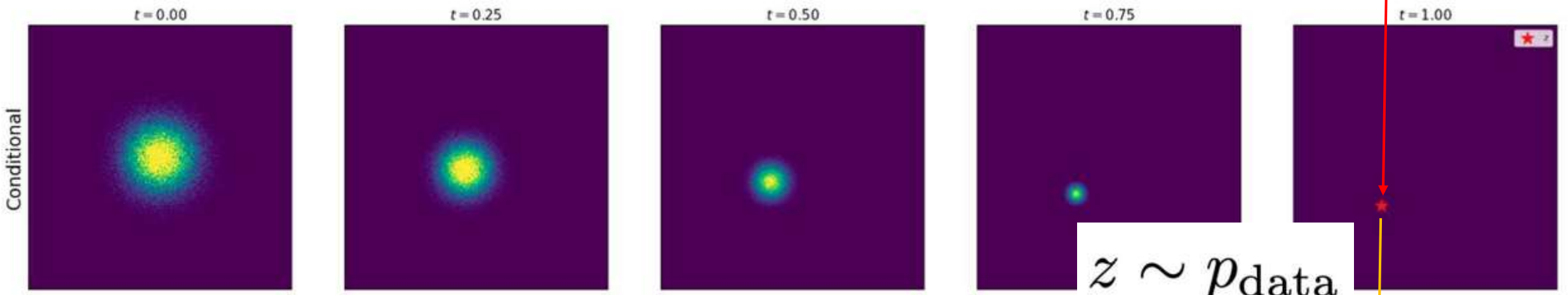
Marginal Probability Path p_t

Conditional Probability Path

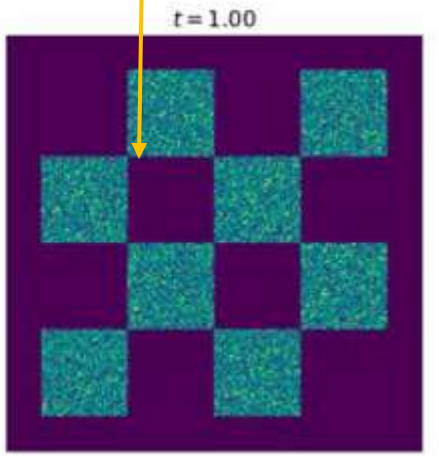
p_{init}

$p_t(\cdot|z)$

z



$z \sim p_{\text{data}}$

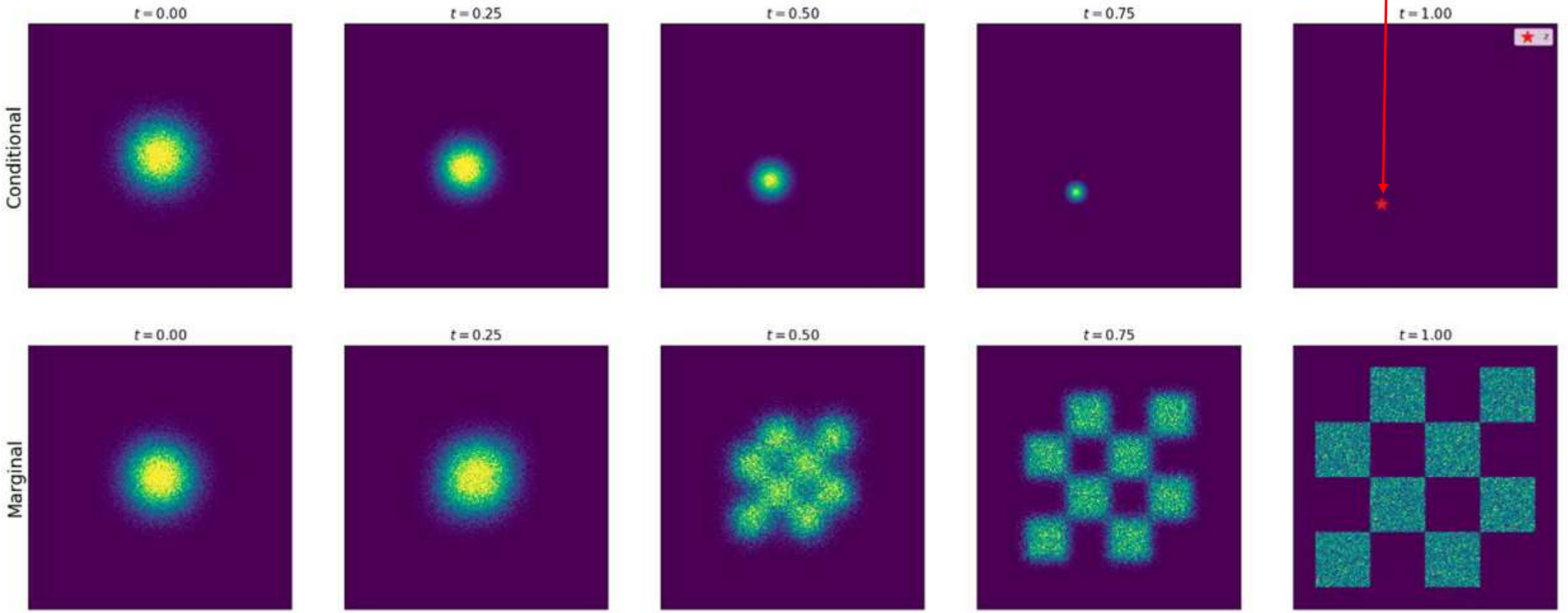


p_{data}

p_{init}

Conditional Probability Path $p_t(\cdot|z)$

z



p_{init}

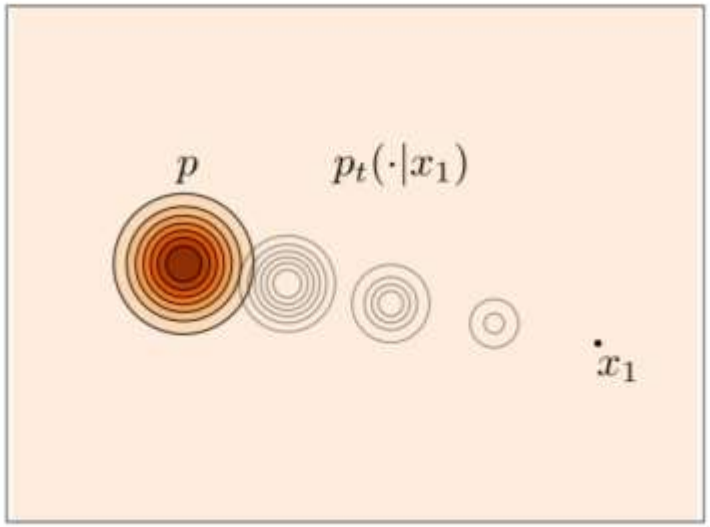
Marginal Probability Path p_t

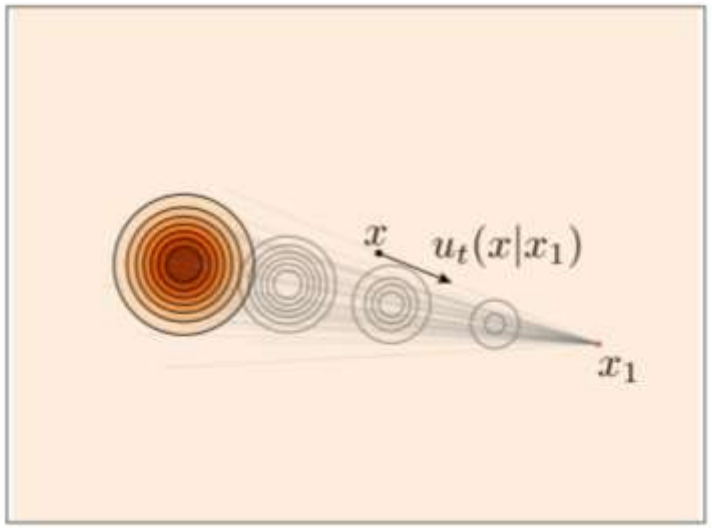
p_{data}

Conditional Prob. Path

	Notation	Key property	Gaussian example
Conditional Probability Path	$p_t(\cdot\|z)$	Interpolates p_{init} and a data point z	$p_t(\cdot\|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$

Conditional Vector Field





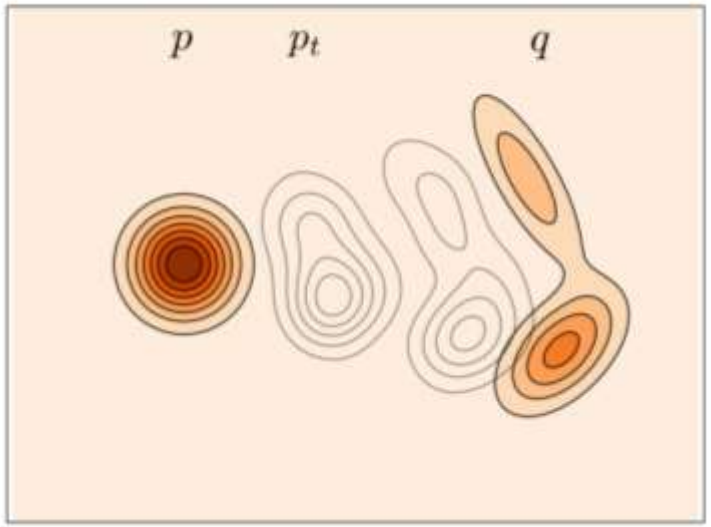
Conditional Prob. Path

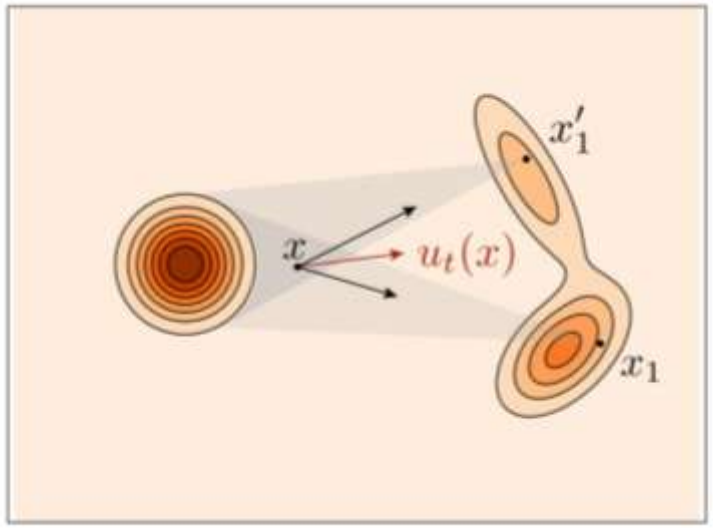
	Notation	Key property	Gaussian example
Conditional Probability Path	$p_t(\cdot\|z)$	Interpolates p_{init} and a data point z	$p_t(\cdot\|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$
Conditional Vector Field	$u_t^{\text{target}}(x\|z)$	ODE follows conditional path	$\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$
	$\nabla \log p_t(x\|z)$	Gradient of log-likelihood	$-\frac{x - \alpha_t z}{\beta_t^2}$

Marginal Prob. Path

	Notation	Key property	Formula
Marginal Probability Path	p_t	Interpolates p_{init} and p_{data}	$\int p_t(x\|z)p_{\text{data}}(z)\mathrm{d}z$

Marginal Vector Field





Marginal Prob. Path

	Notation	Key property	Formula
Marginal Probability Path	p_t	Interpolates p_{init} and p_{data}	$\int p_t(x\|z)p_{\text{data}}(z)\mathrm{d}z$
Marginal Vector Field	$u_t^{\text{target}}(x)$	ODE follows marginal path	$\int u_t^{\text{target}}(x\|z)\frac{p_t(x\|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z$
	$\nabla \log p_t(x)$	Gradient of log-likelihood of marginal path	$\int \nabla \log p_t(x\|z)\frac{p_t(x\|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z$

Example - Conditional Vector Field for Gaussian

$$u_t^{\text{target}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$$

Proof Sketch:

Step 1: By checking ODE, show that the flow of the vector field is given by

$$\psi_t^{\text{target}}(x_0|z) = \alpha_t z + \beta_t x_0$$

Step 2: If $X_0 = x_0 \sim \mathcal{N}(0, I_d)$ is random, then we know that then:

$$X_t = \psi_t(X_0|z) = \alpha_t z + \beta_t X_0 \sim \mathcal{N}(\alpha_t z, \beta_t^2 I_d) = p_t(\cdot|z)$$

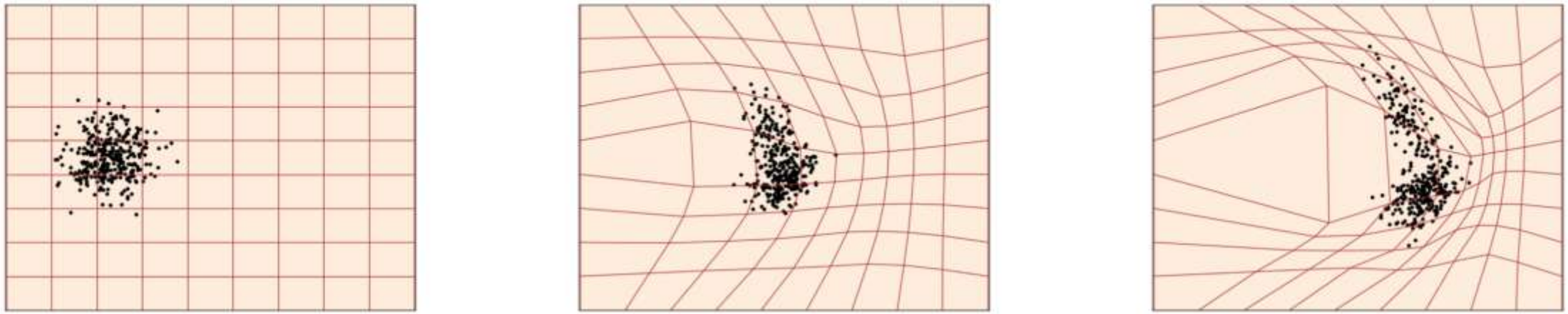


Figure 5 A flow model $X_t = \psi_t(X_0)$ is defined by a diffeomorphism $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with a brown square grid) pushing samples from a source RV X_0 (left, black points) toward some target distribution q (right). We show three different times t.

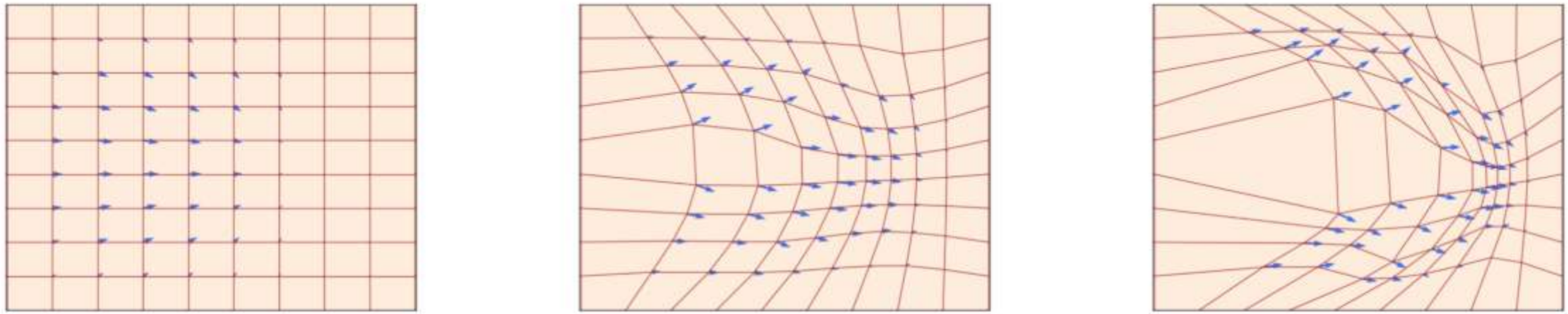


Figure 6 A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations. We show three different times t.

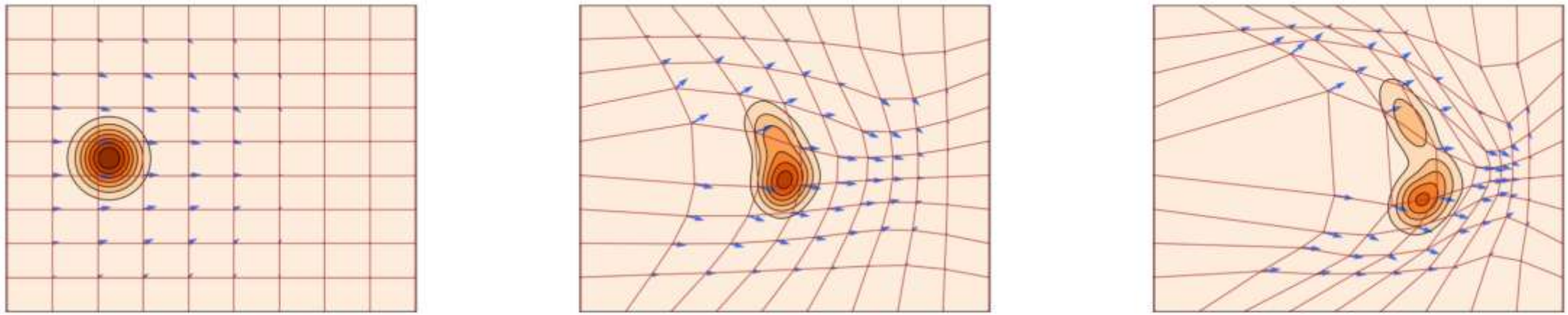
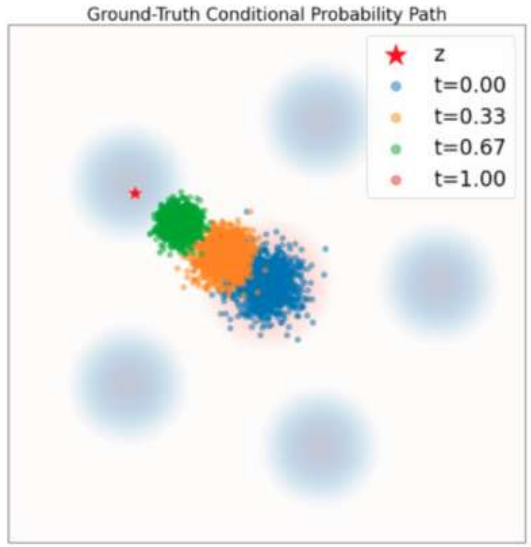
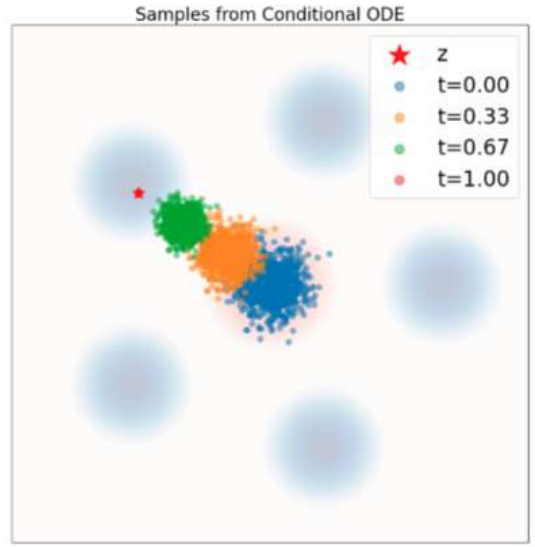
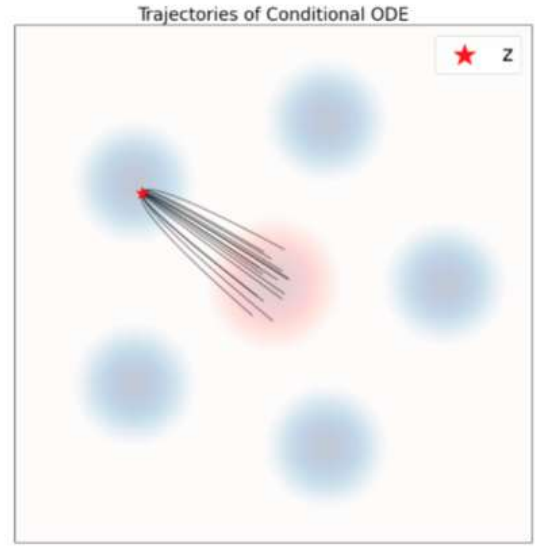


Figure 7 A velocity field u_t (in blue) *generates* a probability path p_t (PDFs shown as contours) if the flow defined by u_t (square grid) reshapes p (left) to p_t at all times $t \in [0, 1)$.

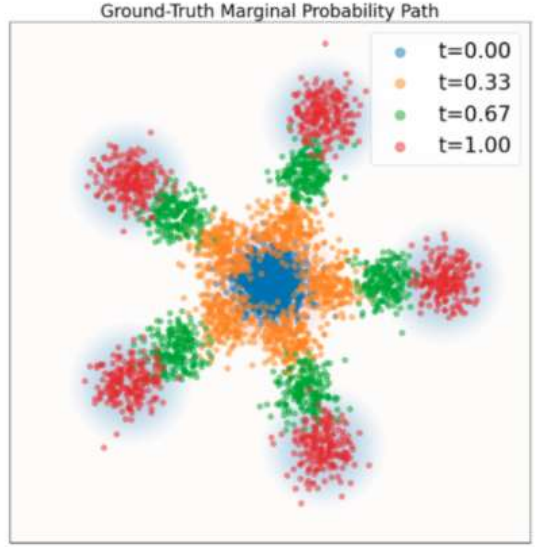
$p_t(\cdot|z)$

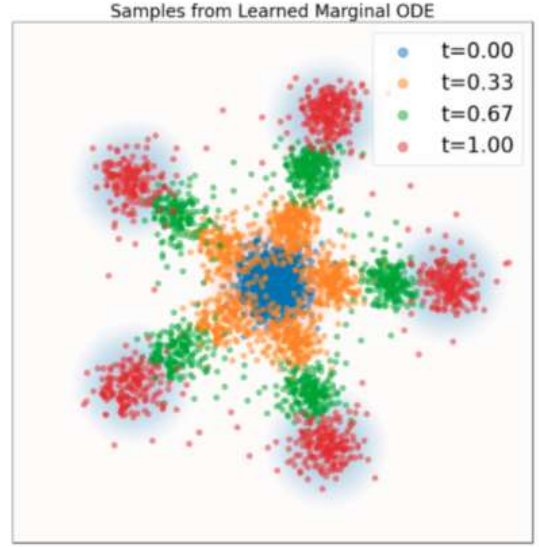

Ground-Truth Conditional Probability Path
z
t=0.00
t=0.33
t=0.67
t=1.00

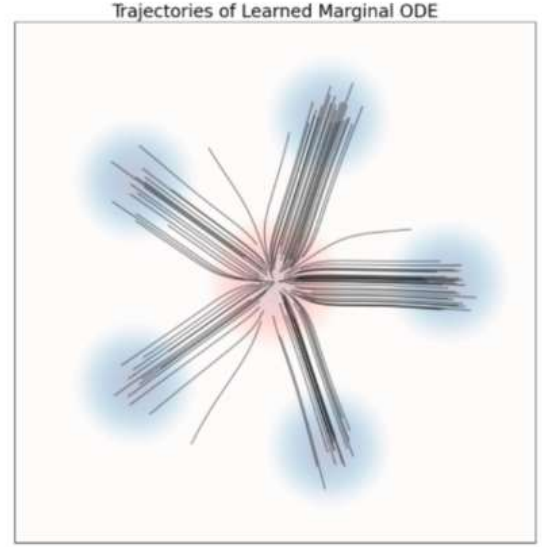

Samples from Conditional ODE
z
t=0.00
t=0.33
t=0.67
t=1.00


Trajectories of Conditional ODE
z

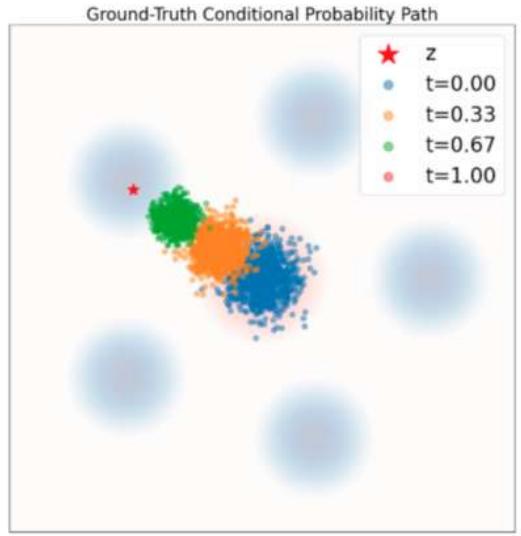
p_t

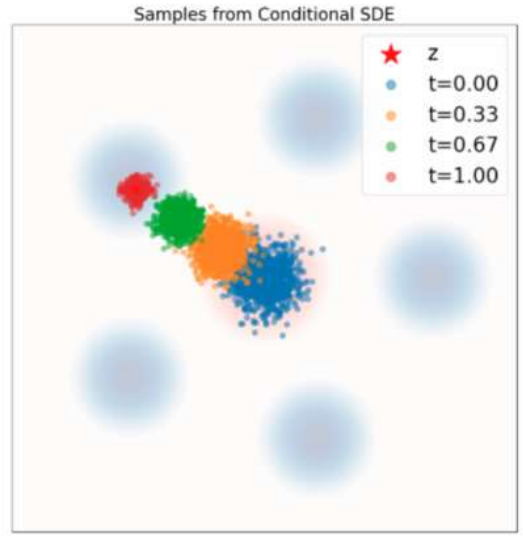

Ground-Truth Marginal Probability Path
t=0.00
t=0.33
t=0.67
t=1.00

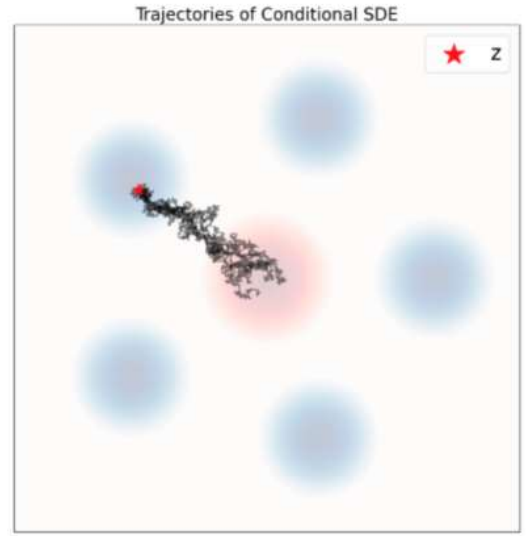

Samples from Learned Marginal ODE
t=0.00
t=0.33
t=0.67
t=1.00


Trajectories of Learned Marginal ODE

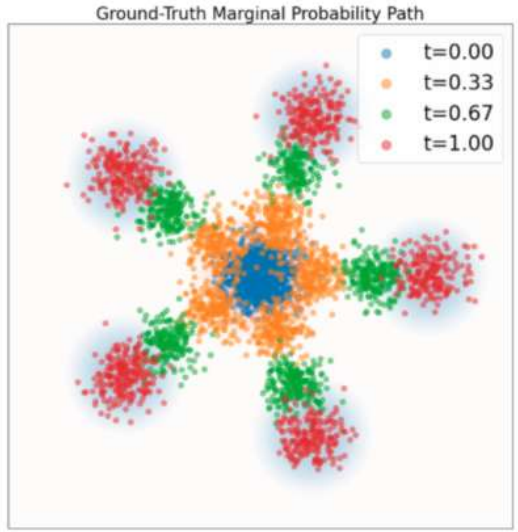
$p_t(\cdot|z)$

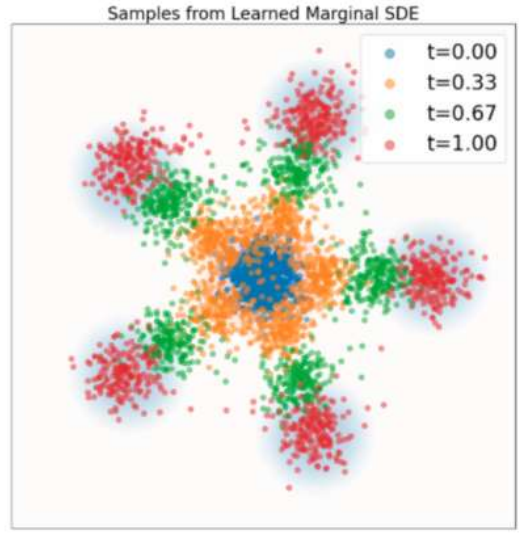

Ground-Truth Conditional Probability Path
z
t=0.00
t=0.33
t=0.67
t=1.00

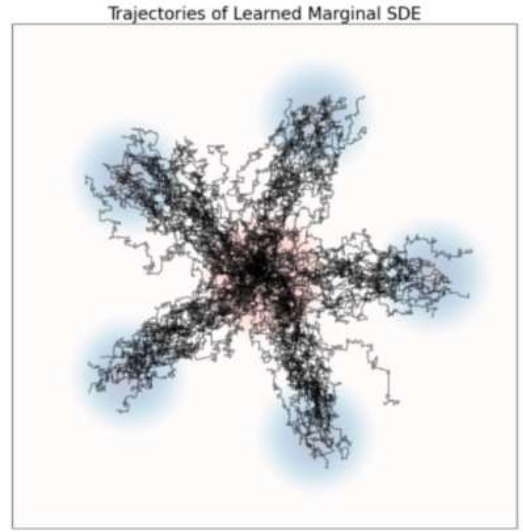

Samples from Conditional SDE
z
t=0.00
t=0.33
t=0.67
t=1.00


Trajectories of Conditional SDE
z

p_t


Ground-Truth Marginal Probability Path
t=0.00
t=0.33
t=0.67
t=1.00


Samples from Learned Marginal SDE
t=0.00
t=0.33
t=0.67
t=1.00


Trajectories of Learned Marginal SDE

Continuity Equation

Randomly initialized ODE

Given: $X_0 \sim p_{\text{init}}, \qquad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t(X_t)$

Follow probability path:

$$X_t \sim p_t \quad (0 \le t \le 1)$$

Marginals are p_t

equivalent

Continuity equation holds

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = -\mathrm{div}(p_t u_t)(x)$$

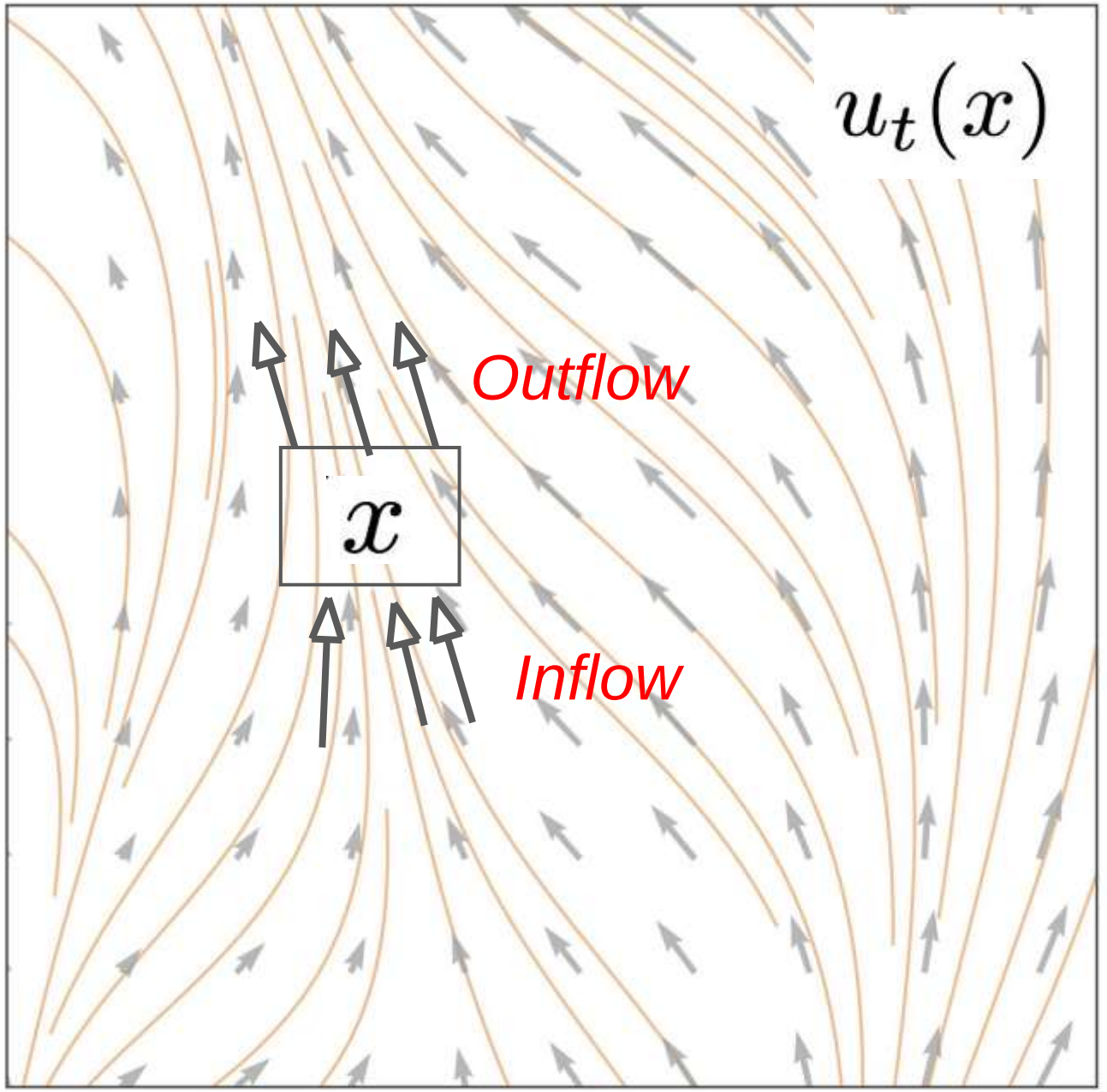
PDE holds

Continuity Equation

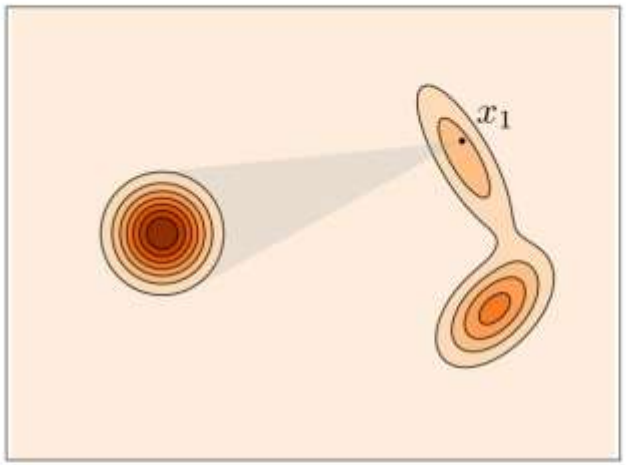
$$\frac{\mathrm{d}}{\mathrm{d}t}p_t(x) = -\mathrm{div}(p_t u_t)(x)$$

Change of probability mass at x

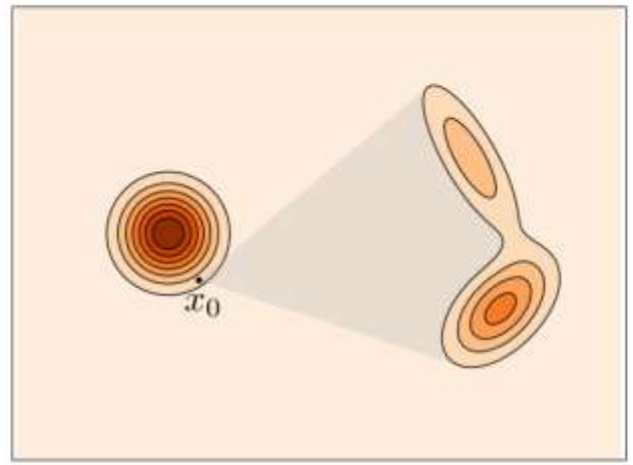
Outflow - inflow of probability mass from u



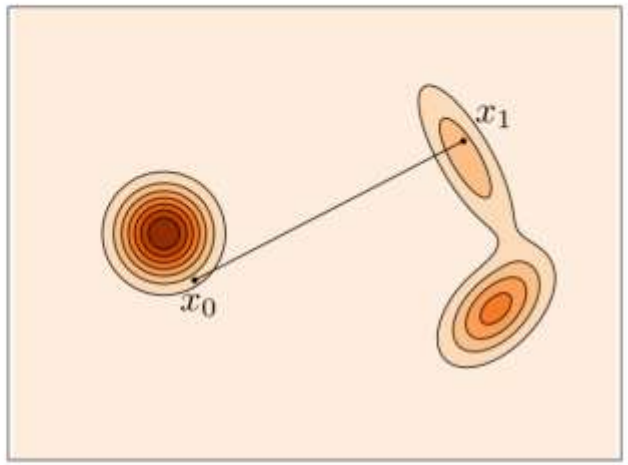
X_1 conditioning



X_0 conditioning



(X_0, X_1) conditioning



$$\psi_t(X_0|x_1) \sim p_{t|1}(\cdot|x_1) \qquad \psi_t(X_1|x_0) \sim p_{t|0}(\cdot|x_0) \qquad \psi_t(x_0, x_1) \sim p_{t|0,1}(\cdot|x_0, x_1)$$

$$p_t(x) = \int p_{t|1}(x|x_1) q(x_1) \mathrm{d}x_1 = \int p_{t|0}(x|x_0) p(x_0) \mathrm{d}x_1 = \int p_{t|0,1}(x|x_0, x_1) \pi_{0,1}(x_0, x_1) \mathrm{d}x_0 \mathrm{d}x_1$$

$$u_t(x) = \mathbb{E}\left[u_t(X_t|X_1)|X_t = x\right] = \mathbb{E}\left[u_t(X_t|X_0)|X_t = x\right] = \mathbb{E}\left[u_t(X_t|X_0, X_1)|X_t = x\right]$$

Algorithm 3 Flow Matching Training Procedure (General)

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample $x \sim p_t(\cdot|z)$
5: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2$$

6: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
7: **end for**

Conditional Flow Matching for Gaussian probability path

Prob. path $\mathcal{N}(\alpha_t z, \beta_t^2 I_d)$

Conditional VF $u_t^{\text{target}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$

Noise Sampling $x \sim p_t(\cdot|z) \quad \Leftrightarrow \quad x = \alpha_t z + \beta_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I_d)$

Plugging in Noise Sampling into CFM Loss results in:

$$L_{\text{CFM}}(\theta)$$

$$= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, x\sim p_t(\cdot|z)} \left[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2\right]$$

$$= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \epsilon\sim\mathcal{N}(0,I_d)} \left[\|u_t^\theta(\alpha_t z + \beta_t \epsilon) - u_t^{\text{target}}(\alpha_t z + \beta_t \epsilon|z)\|^2\right]$$

$$= \mathbb{E}_{t\sim\text{Unif},\, z\sim p_{\text{data}},\, \epsilon\sim\mathcal{N}(0,I_d)} \left[\|u_t^\theta(\alpha_t z + \beta_t \epsilon) - (\dot{\alpha}_t z + \dot{\beta}_t \epsilon)\|^2\right]$$

noise+data **velocity**

Straight Line Schedule

$$L_{\mathrm{CFM}}(\theta)$$

$$= \mathbb{E}_{t\sim \mathrm{Unif},\, z\sim p_{\mathrm{data}},\, \epsilon\sim\mathcal{N}(0,I_d)} \left[\| u_t^\theta(\alpha_t z + \beta_t \epsilon) - (\dot{\alpha}_t z + \dot{\beta}_t \epsilon) \|^2 \right]$$

$$= \mathbb{E}_{t\sim \mathrm{Unif},\, z\sim p_{\mathrm{data}},\, \epsilon\sim\mathcal{N}(0,I_d)} \left[\| u_t^\theta(tz + (1-t)\epsilon) - (z - \epsilon) \|^2 \right]$$

Linear interpolation of noise and data

Difference between noise and data

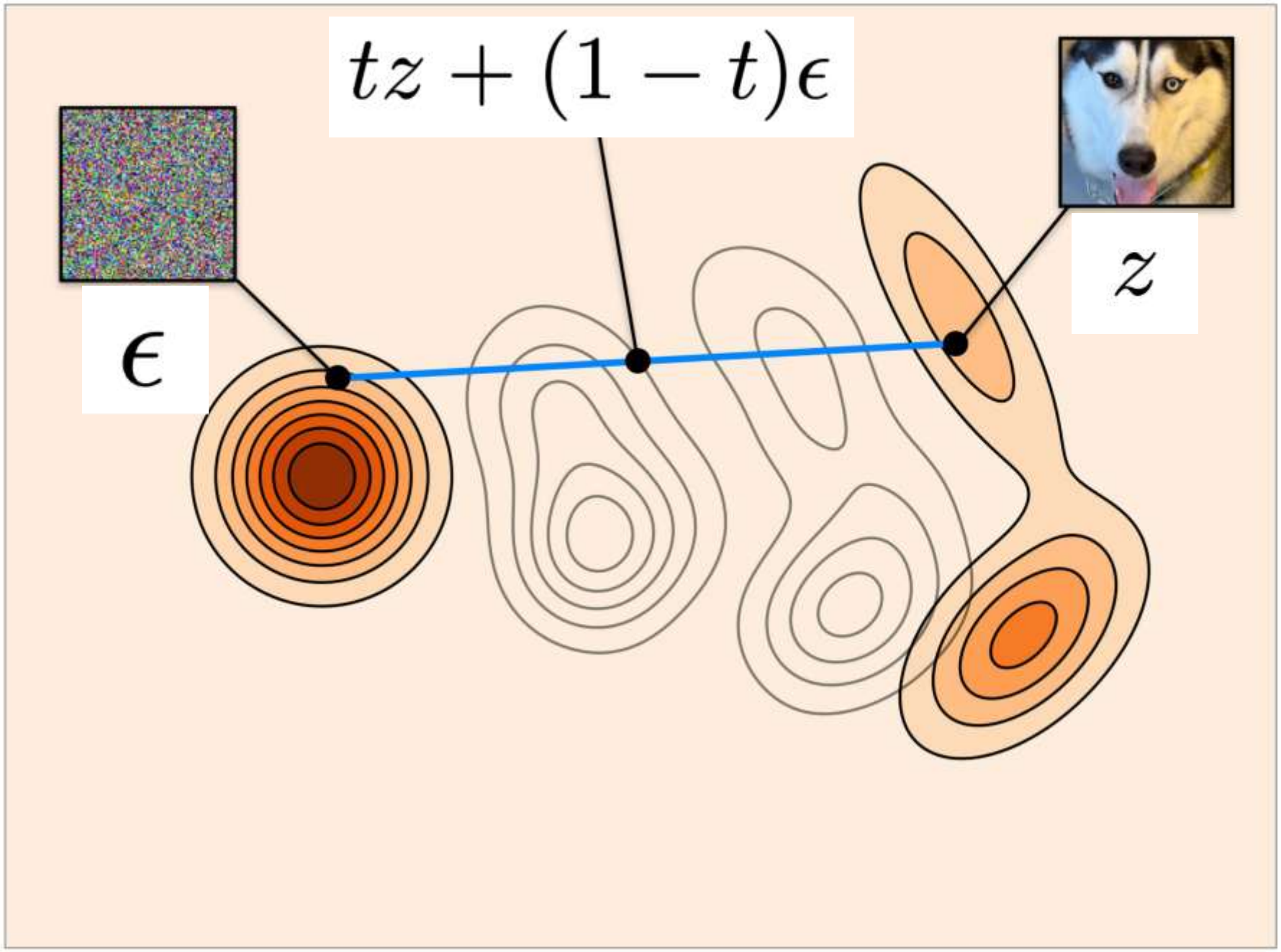


Figure credit: Yaron Lipman

Algorithm 4 Flow Matching Training for CondOT path

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set $x = tz + (1-t)\epsilon$
6: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - (z - \epsilon)\|^2$$

7: Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$.
8: **end for**

Reminder: Sampling Algorithm for Flow Model

Algorithm 1 Sampling from a Flow Model with Euler method

Require: Neural network vector field u_t^θ, number of steps n

Set $t = 0$

Set step size $h = \frac{1}{n}$

Draw a sample $X_0 \sim p_{\text{init}}$ *Random initialization!*

for $i = 1, \ldots, n-1$ **do**

  $X_{t+h} = X_t + h u_t^\theta(X_t)$

  Update $t \leftarrow t + h$

end for

return X_1 *Return final point*

The Flow Matching Matrix

Conditional Probability Path	**Conditional** Vector Field	**Conditional** Flow Matching Loss
Marginal Probability Path	**Marginal** Vector Field	**Marginal** Flow Matching Loss
Defines distributions from noise to data	**Defines training target that we want to learn**	**Loss function that we want to minimize during traing**

Conditional Prob. Path, Vector Field, and FM Loss

	Notation	Key property	Gaussian example
Conditional Probability Path	$p_t(\cdot\|z)$	Interpolates p_{init} and a data point z	$p_t(\cdot\|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$
Conditional Vector Field	$u_t^{\text{target}}(x\|z)$	ODE follows conditional path	$\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$
Conditional FM Loss	$L_{\text{CFM}}(\theta)$	Loss we minimize during training	$\mathbb{E}_{t,z,x}[\|\|u_t^\theta(x) - u_t^{\text{target}}(x\|z)\|\|^2]$

All these objects are tractable. Just analytical formulas!

Marginal Prob. Path, Vector Field, and FM Loss

	Notation	Key property	Formula
Marginal Probability Path	p_t	Interpolates p_{init} and p_{data}	$\int p_t(x\|z)p_{\text{data}}(z)\mathrm{d}z$
Marginal Vector Field	$u_t^{\text{target}}(x)$	ODE follows marginal path	$\int u_t^{\text{target}}(x\|z)\frac{p_t(x\|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z$
Marginal FM Loss	$L_{\text{FM}}(\theta)$	Implicitly minimized via cond FM loss	$\mathbb{E}_{t,z,x}[\|\|u_t^\theta(x) - u_t^{\text{target}}(x)\|\|^2]$

None of these objects are tractable. But we can still learn them!

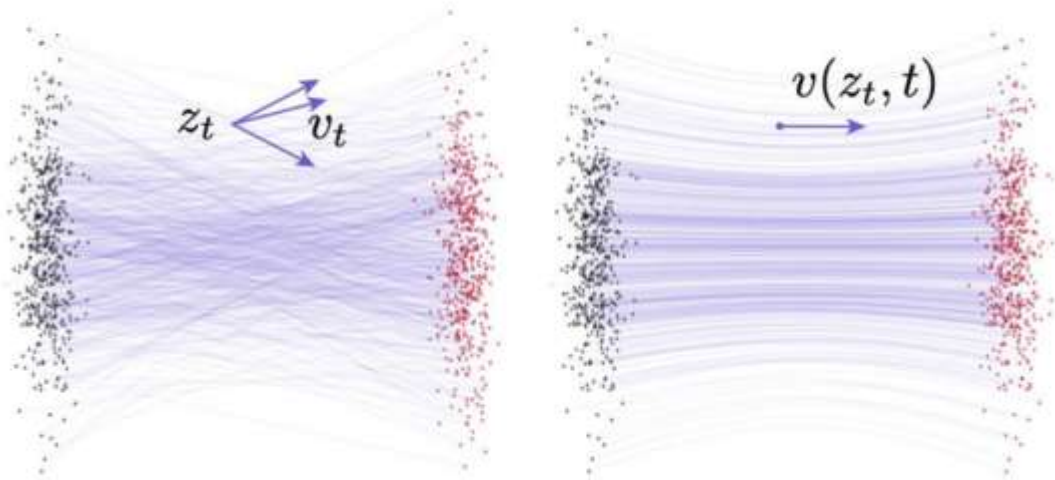


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal velocity field serves as the underlying ground-truth field for network training. All velocities shown here are essentially ***instantaneous*** velocities. Illustration follows [12]. (*Gray dots: samples from prior; red dots: samples from data.*)

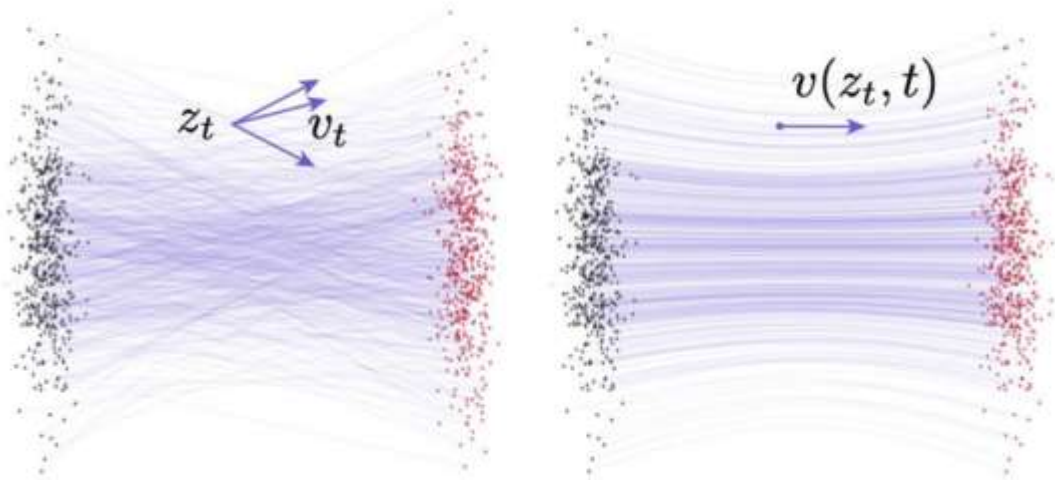


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal velocity field serves as the underlying ground-truth field for network training. All velocities shown here are essentially ***instantaneous*** velocities. Illustration follows [12]. (*Gray dots: samples from prior; red dots: samples from data.*)

Diffusion Models

Flow Matching