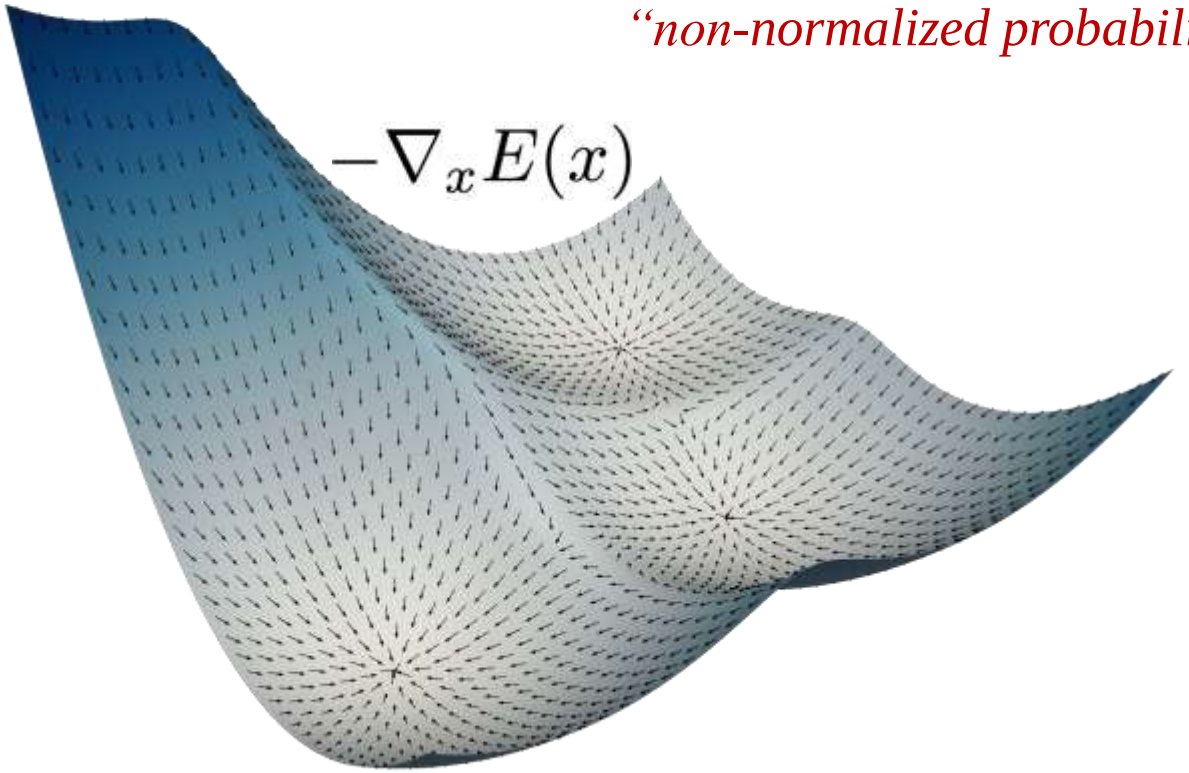
Recap.

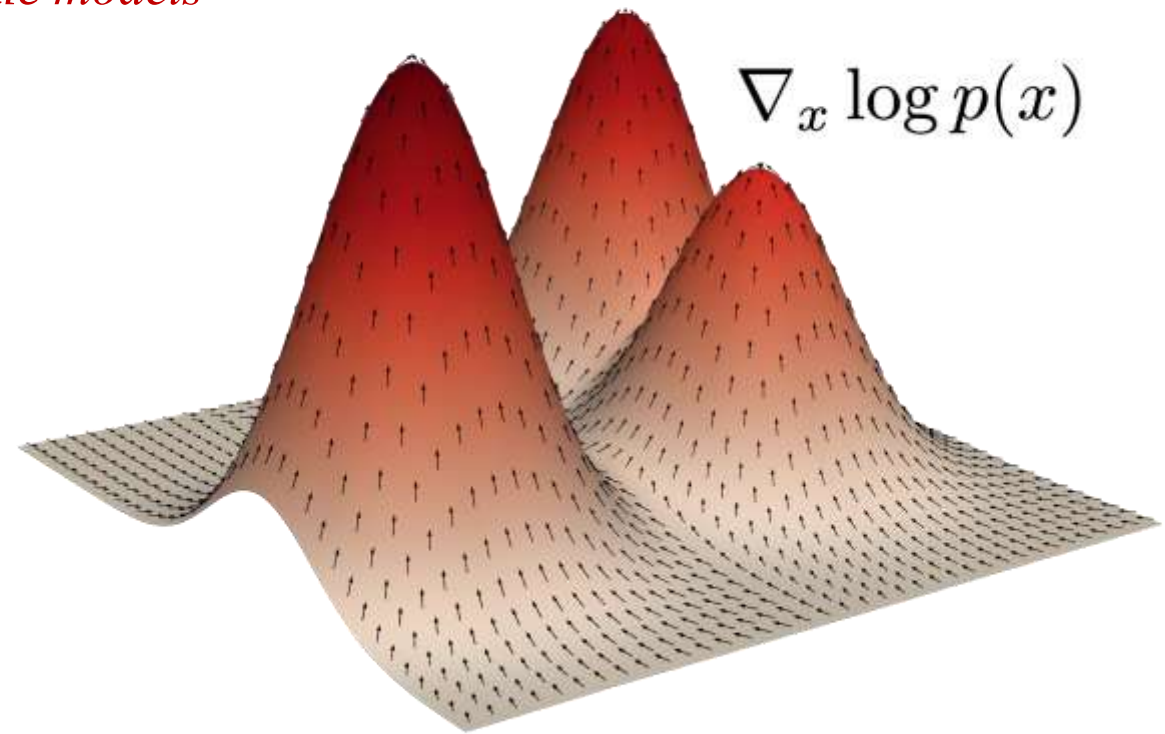
Energy-based Models

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

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

“non-normalized probabilistic models”





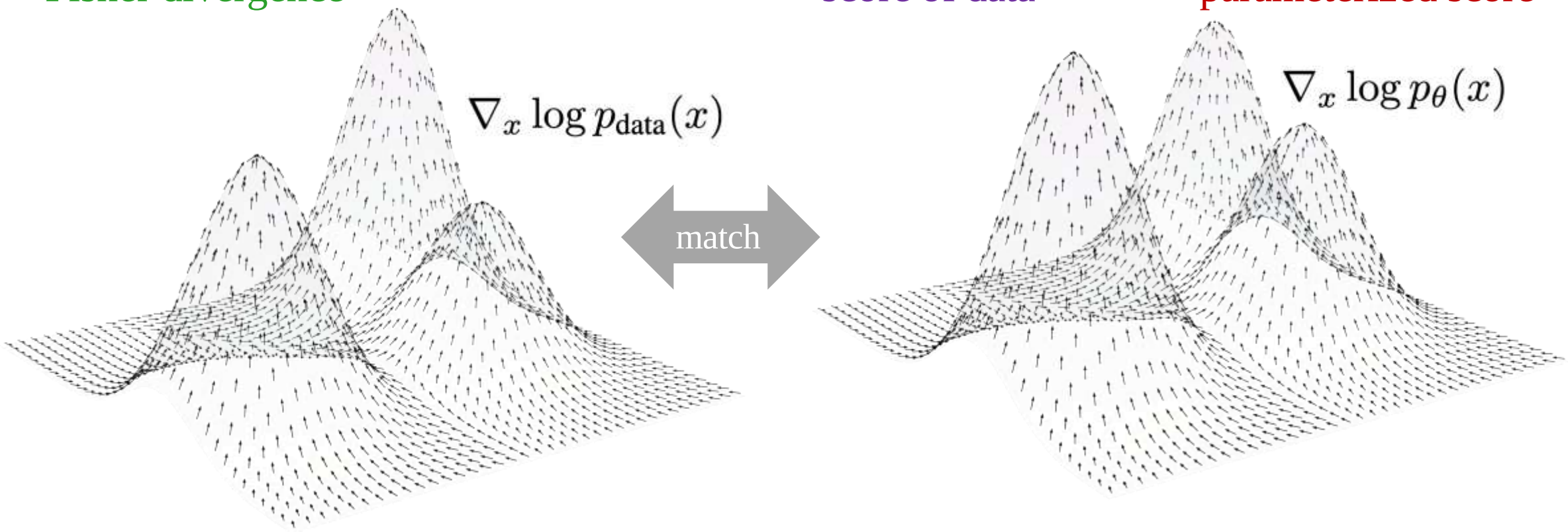
*only visualize directions

Score Matching

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

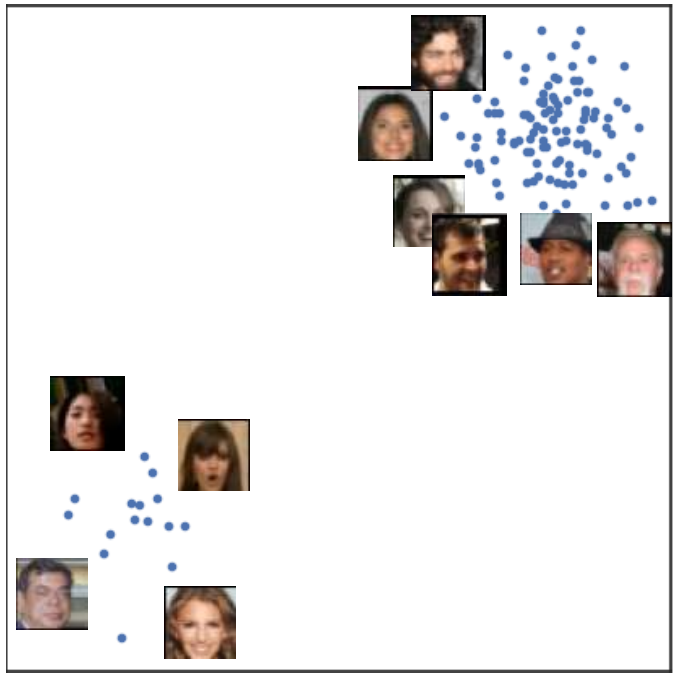
$$D_F(p_{\text{data}}(x) \parallel p_\theta(x)) = \mathbb{E}_{p_{\text{data}}(x)}\left[\frac{1}{2}\|\nabla_x \log p_{\text{data}}(x) - \nabla_x \log p_\theta(x)\|^2\right]$$

Fisher divergence | score of data | parameterized score



*only visualize directions

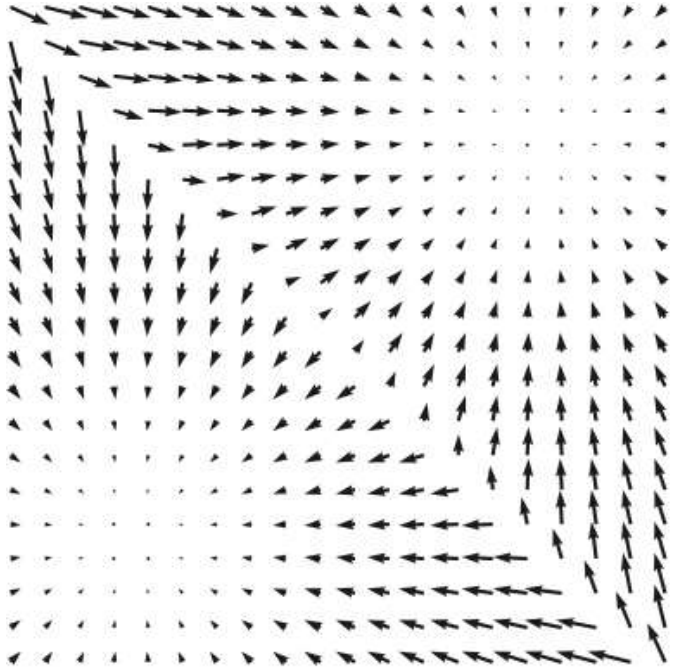
Score-based generative modeling



Data samples

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

score matching



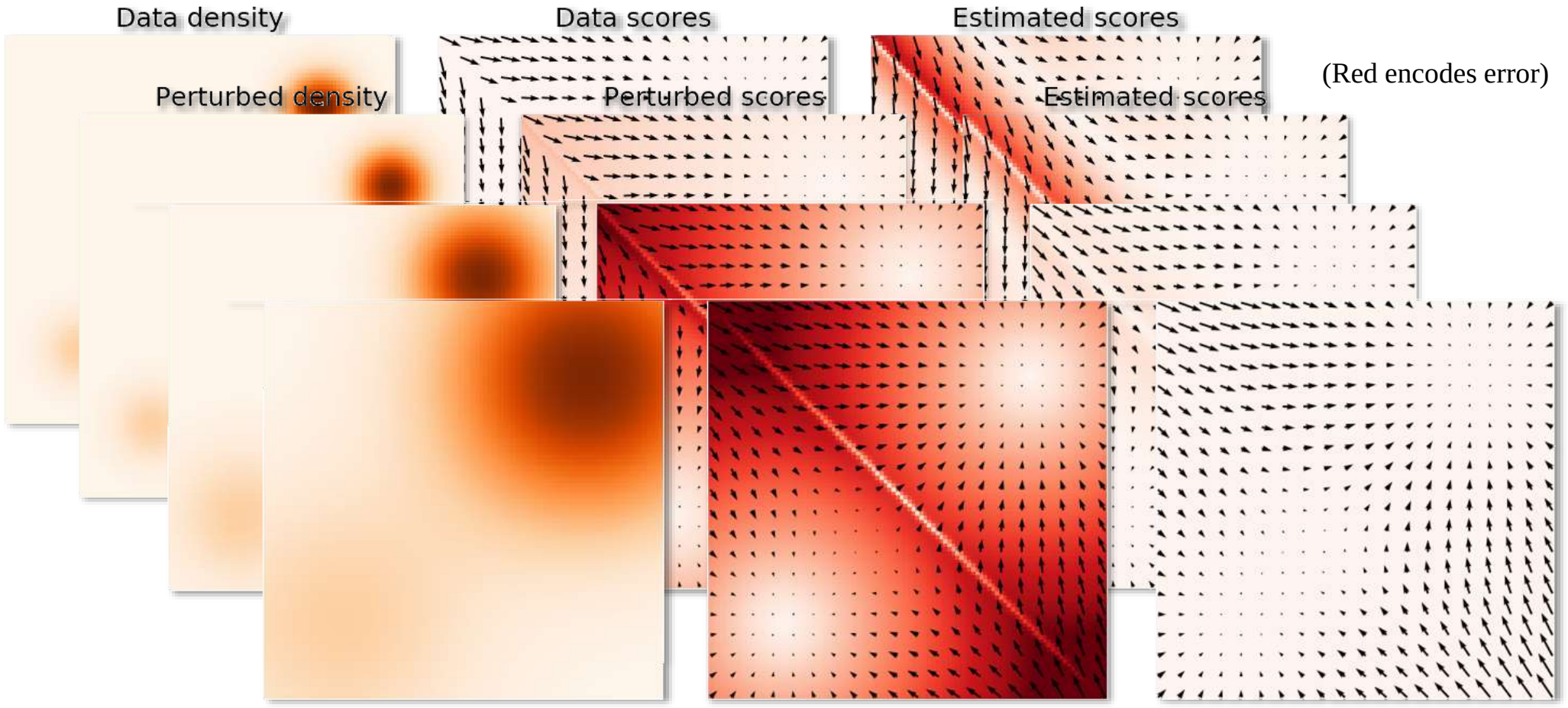
Scores

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

Langevin dynamics

New samples

Trading off Data Quality and Estimation Accuracy



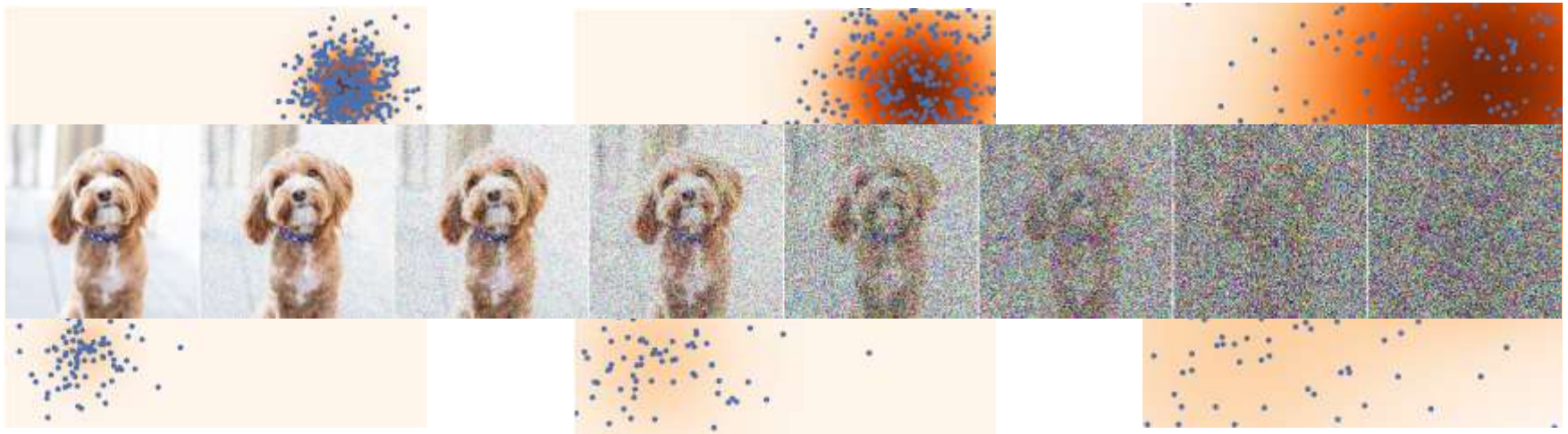
(Red encodes error)

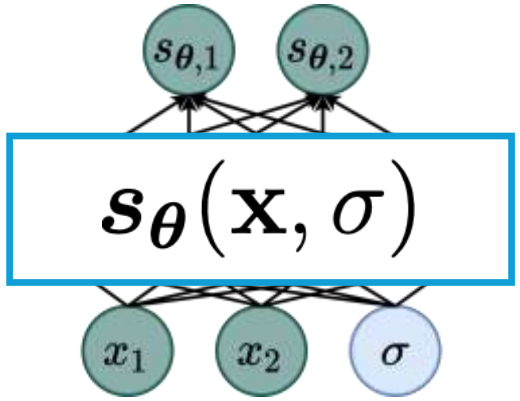
Worse data quality!

Better score estimation!

Using multiple noise levels

Data





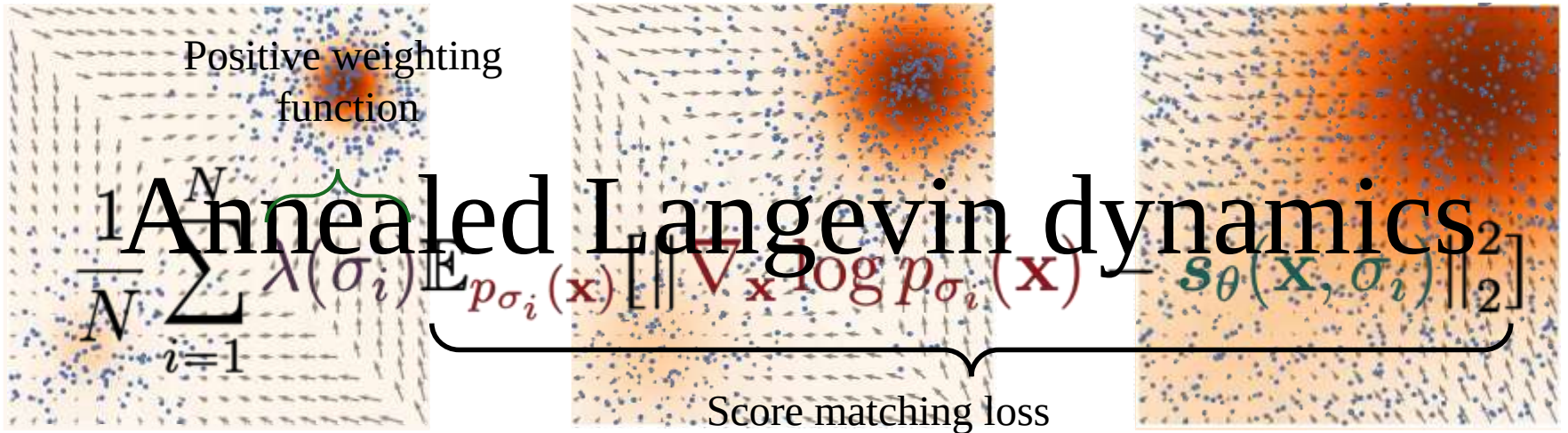
Noise Conditional Score Model

Positive weighting function

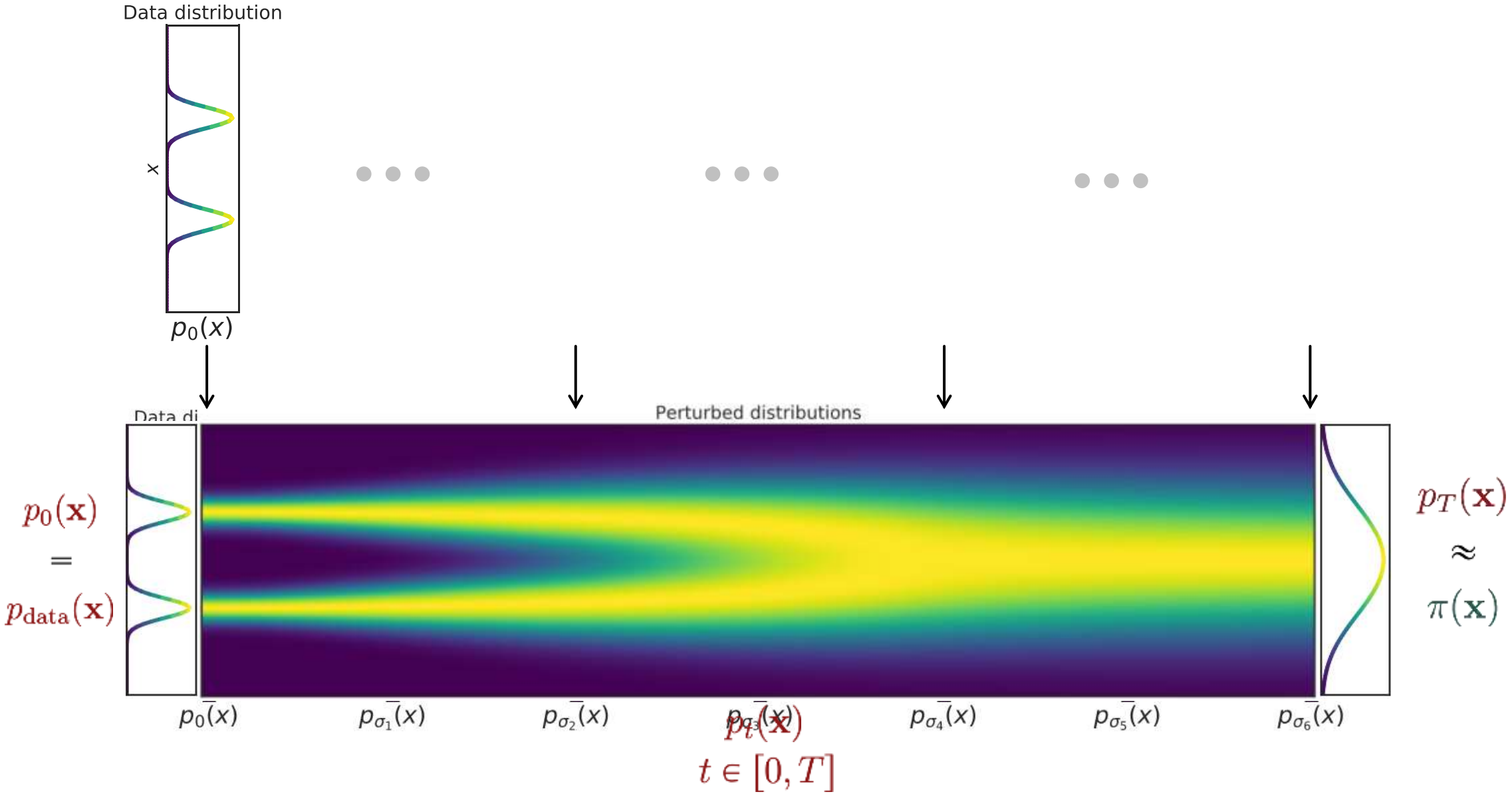
$$\frac{1}{N}\sum_{i=1}^{N}\lambda(\sigma_i)\mathbb{E}_{p_{\sigma_i}(\mathbf{x})}[\|\nabla_{\mathbf{x}}\log p_{\sigma_i}(\mathbf{x}) - s_\theta(\mathbf{x},\sigma_i)\|_2^2]$$

Annealed Langevin dynamics

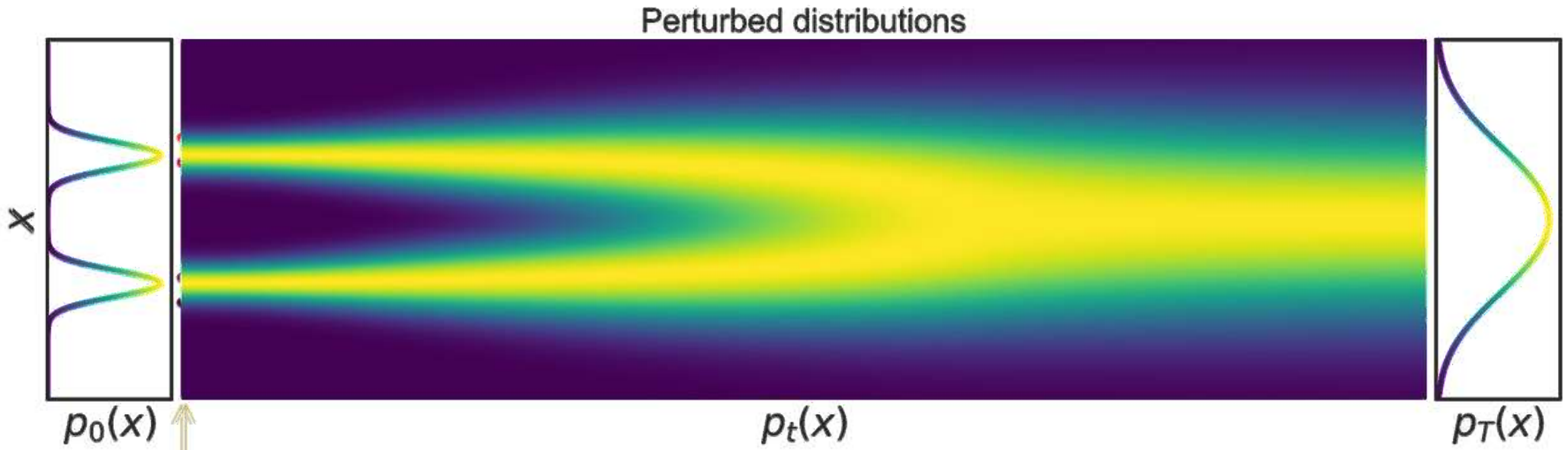
Score matching loss



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)

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

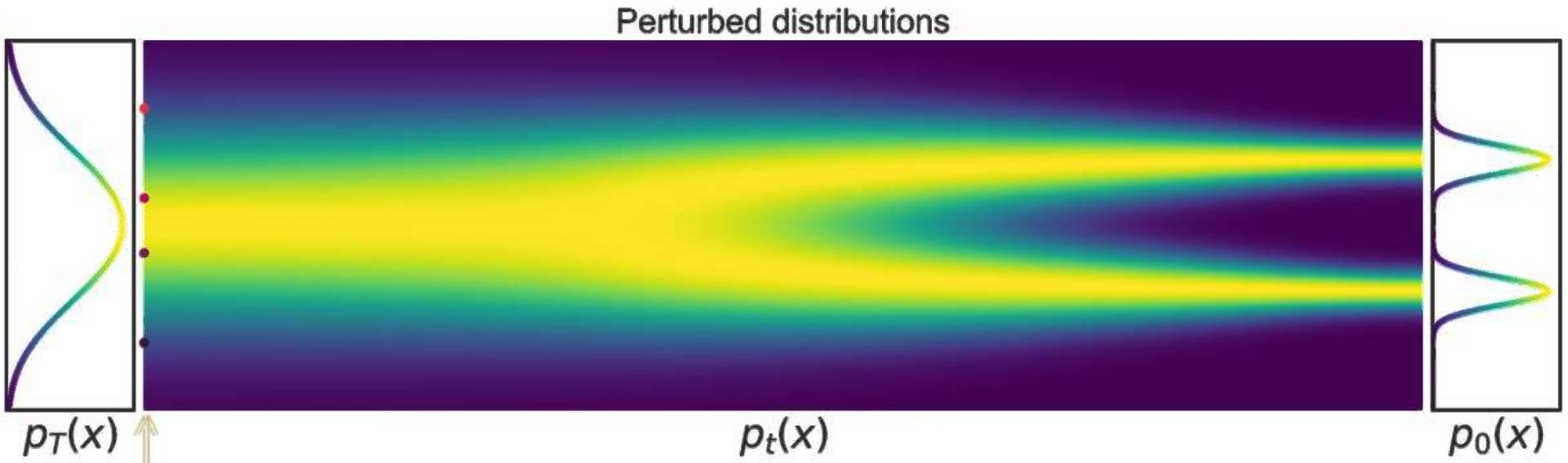
Deterministic drift — Infinitesimal noise

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)

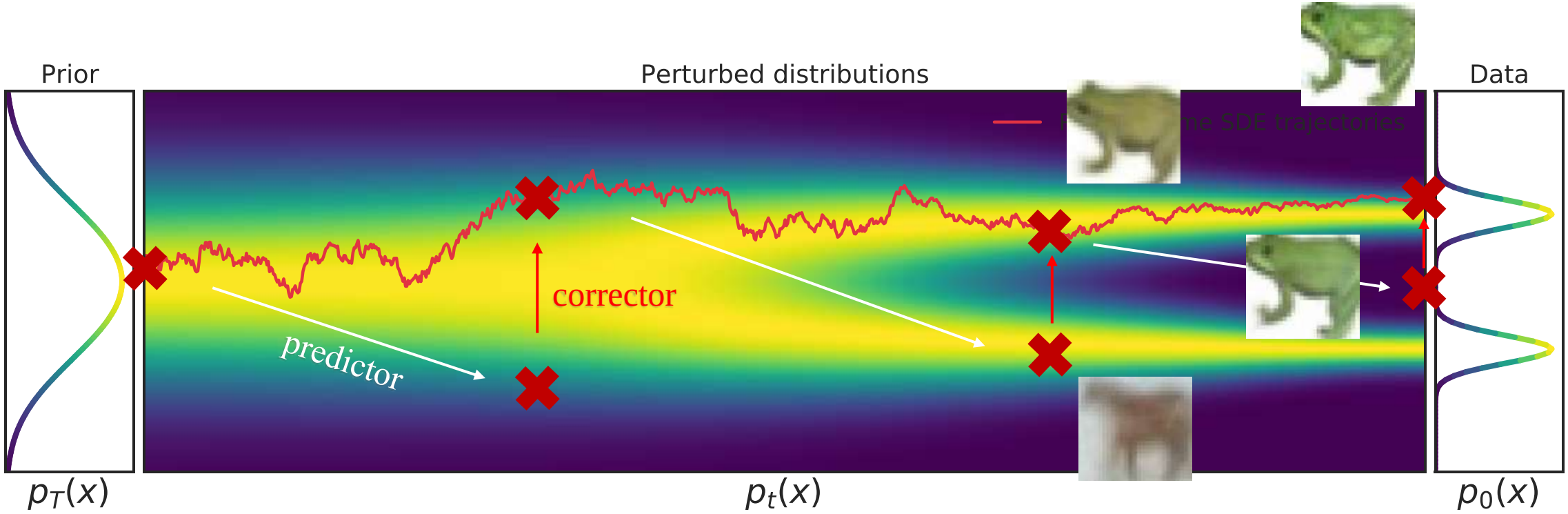
$$\mathrm{d}\mathbf{x}_t = -\sigma(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)\,\mathrm{d}t + \sigma(t)\,\mathrm{d}\bar{\mathbf{w}}_t$$

Infinitesimal noise in the reverse time direction

Score function!

Predictor-Corrector sampling methods

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



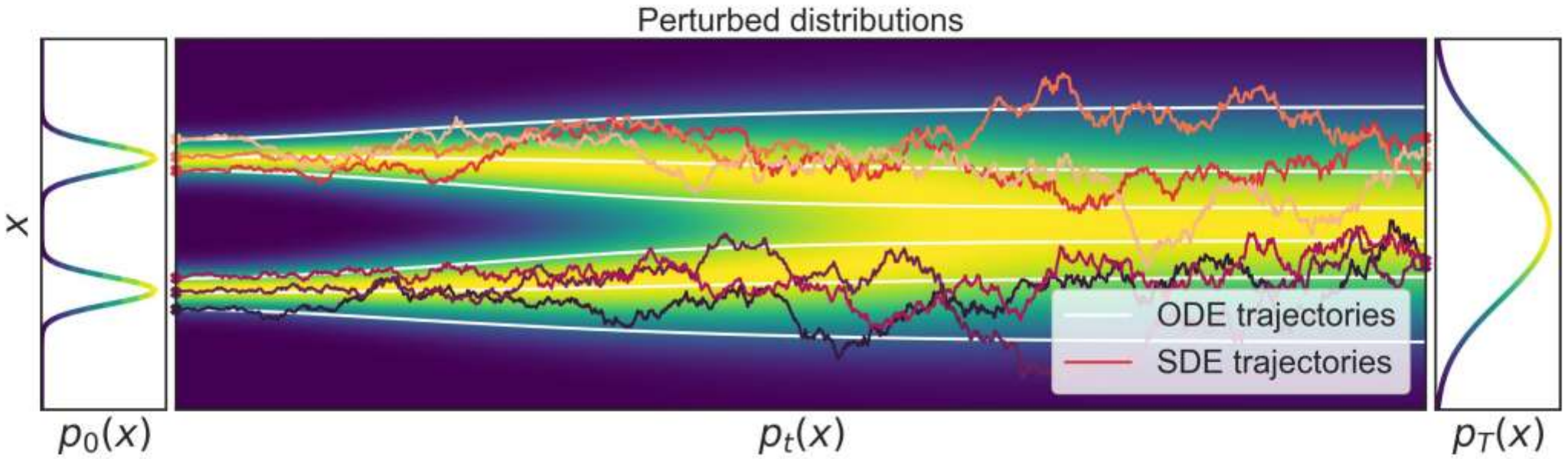
Predictor-Corrector sampling methods

- 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

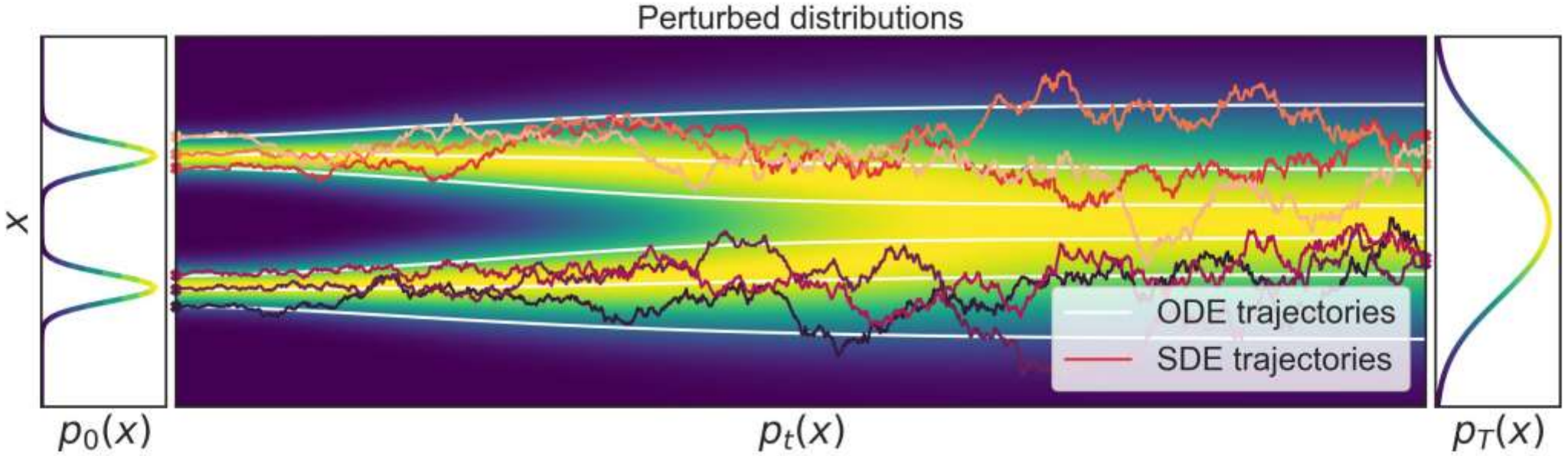


SDE **Ordinary differential equation (ODE)**

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

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

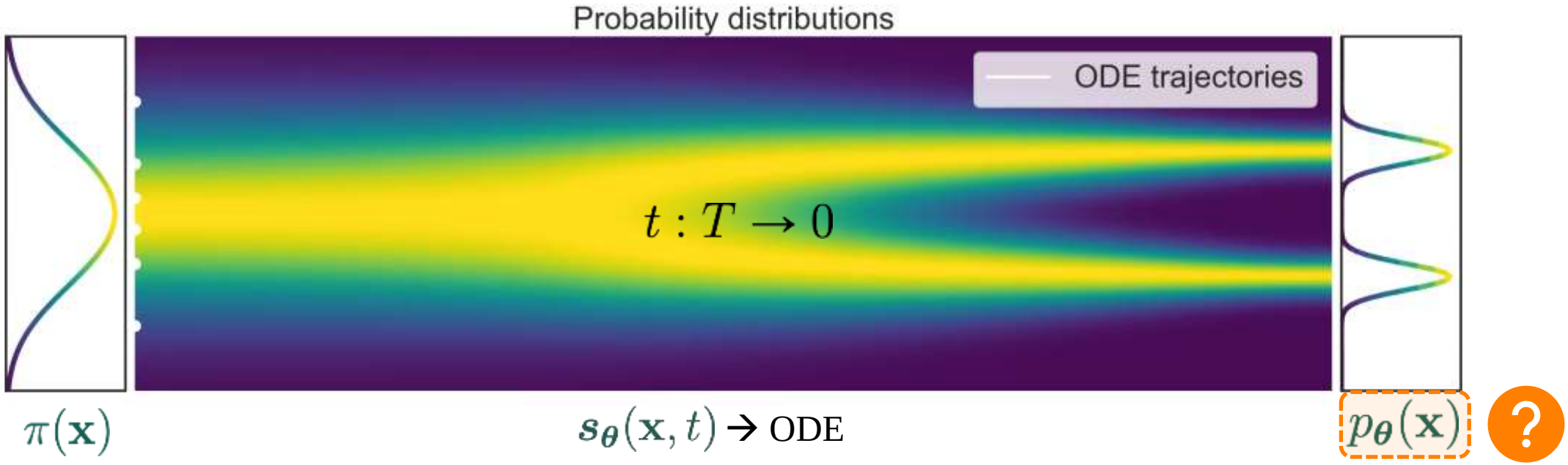
Converting the SDE to an ODE



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)

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

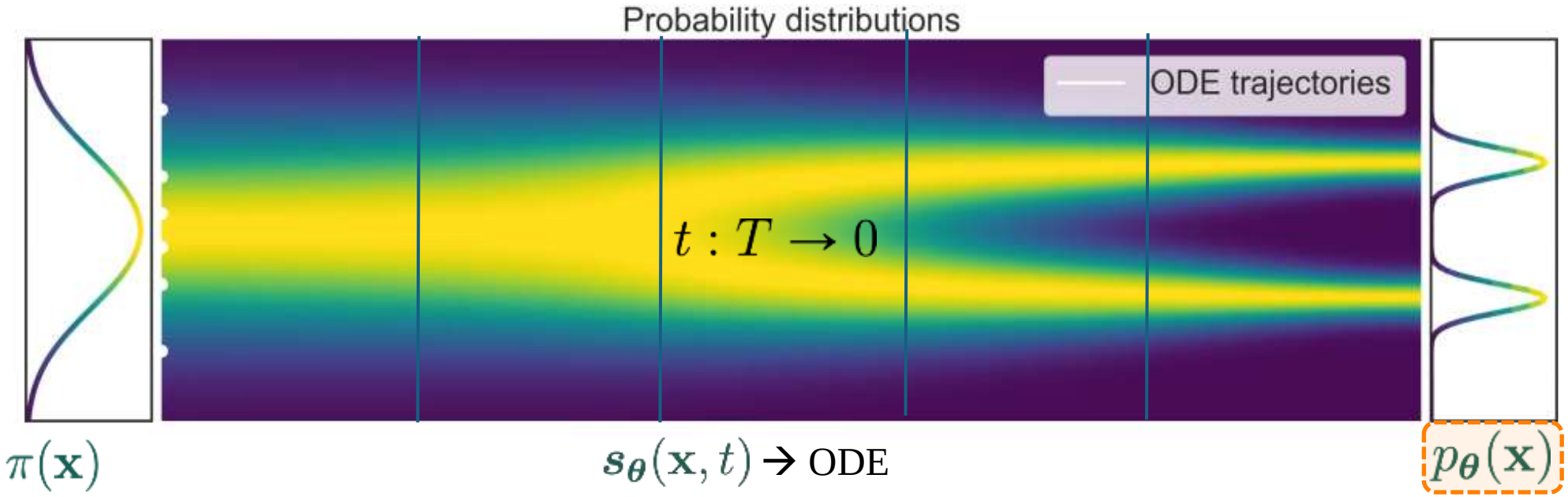
ODE solver

Computable in polynomial time

Unbiased estimator

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

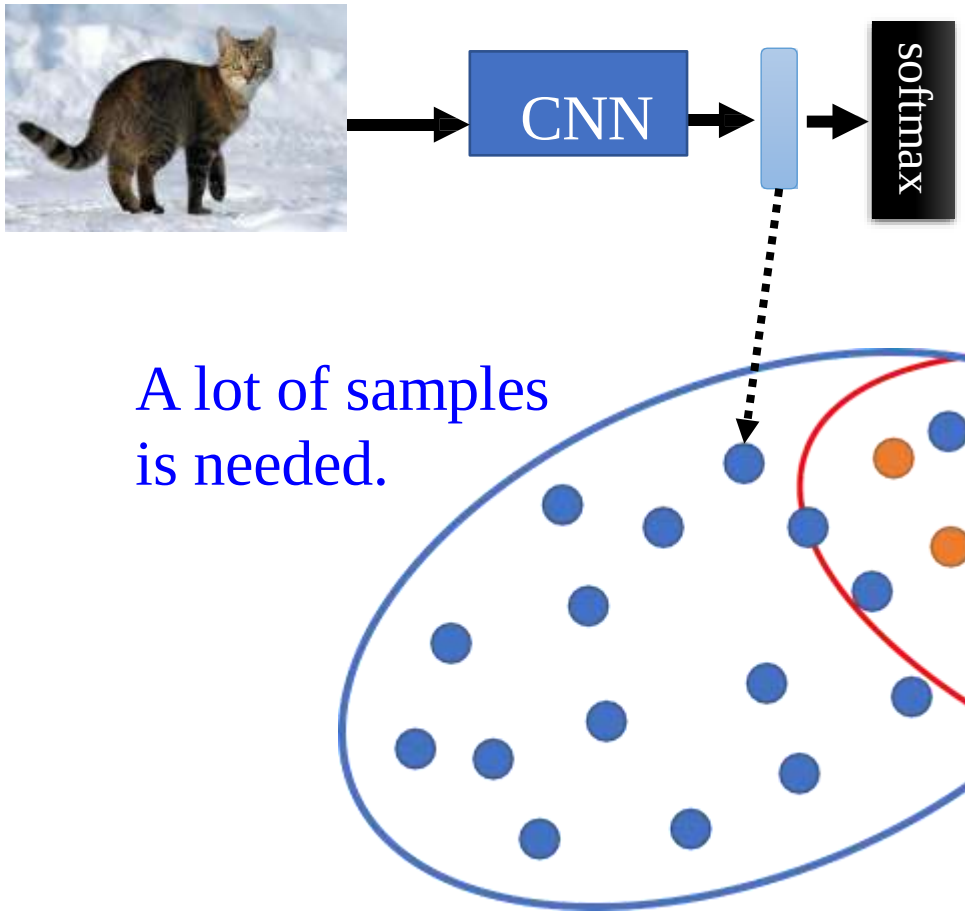
Accelerated sampling



- 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

Fréchet Inception Distance (FID)

https://arxiv.org/abs/1706.08500



red points: real images

blue points: generated images

FID = Fréchet distance between the two **Gaussians**

Smaller is better

Accelerated sampling

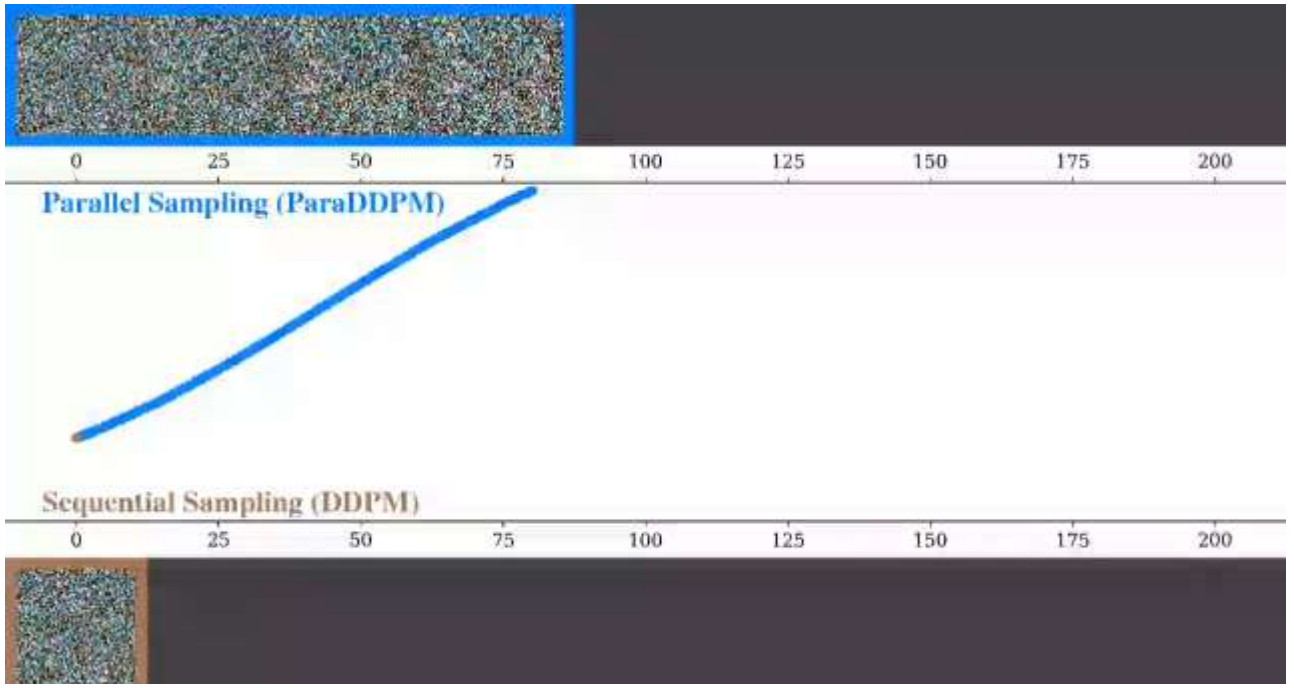
		CIFAR10 (32 × 32)					CelebA (64 × 64)				
	S	10	20	50	100	1000	10	20	50	100	1000
η	0.0	**13.36**	**6.84**	**4.67**	**4.16**	4.04	**17.33**	**13.73**	**9.17**	**6.53**	3.51
	0.2	14.04	7.11	4.77	4.25	4.09	17.66	14.11	9.51	6.79	3.64
	0.5	16.66	8.35	5.25	4.46	4.29	19.86	16.06	11.01	8.09	4.28
	1.0	41.07	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**

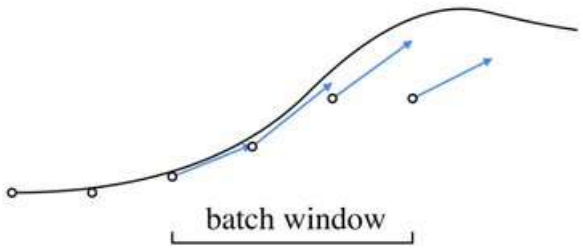
- Numerical methods + ODE formulation to accelerate sampling
- DDIM [Song and Ermon, 2021]:
  - Coarsely discretize the time axis, take big steps
  - Corresponds to exponential integrator (semi-linear ODE) [Lu et al, 2022; Zhang and Chen, 2022]
  - 10x-50x speedups, comparable sample quality

Compared to DDPM, DDIM is able to:

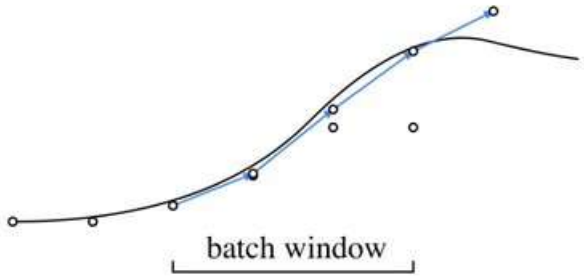
- Generate higher-quality samples using a much fewer number of steps.
- Have “consistency” property since the generative process is deterministic, meaning that multiple samples conditioned on the same latent variable should have similar high-level features.
- Because of the consistency, DDIM can do semantically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

Parallel ODE solving

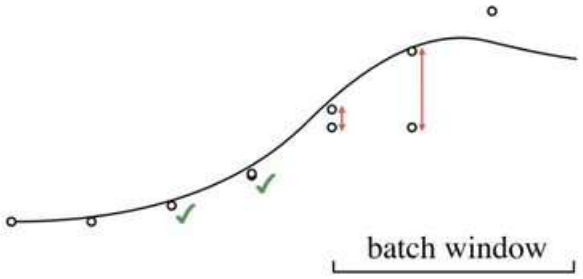




(a) Compute the drift of $x_{t:t+p}^{k}$ on a batch window of size $p = 4$, in parallel

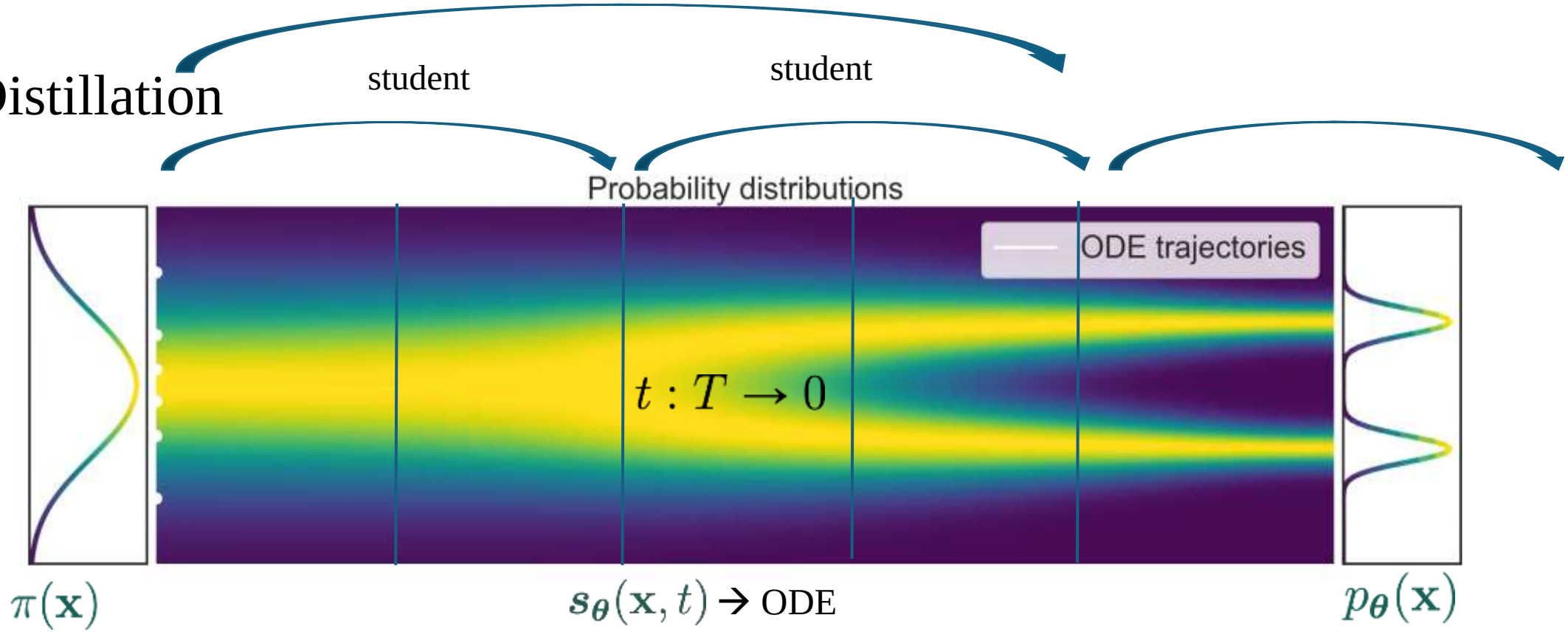


(b) Update the values to $x_{t:t+p}^{k+1}$ using the cumulative drift of points in the window



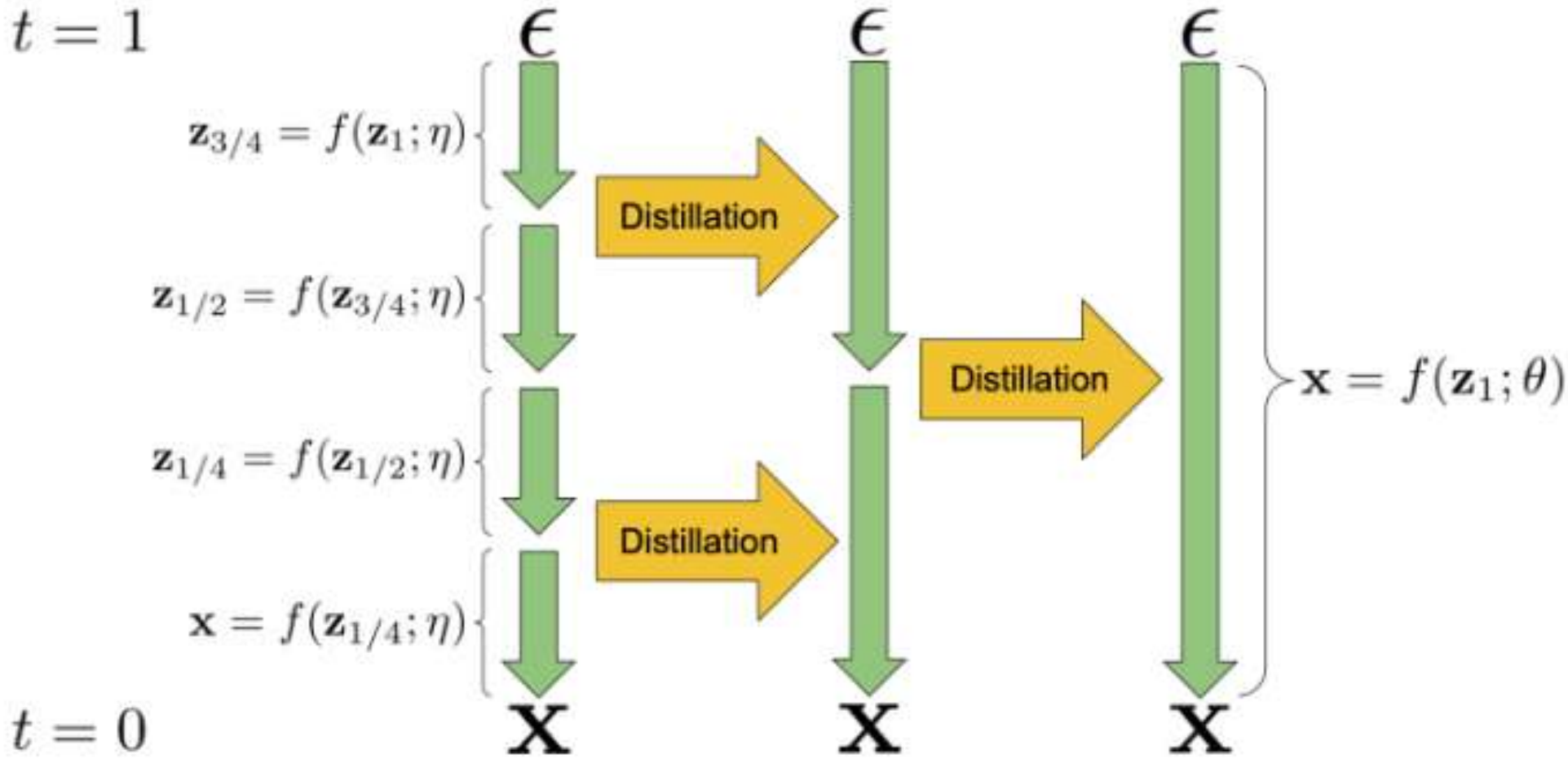
(c) Determine how far to slide the window forward, based on the error $\|x_i^{k+1} - x_i^k\|^2$.

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

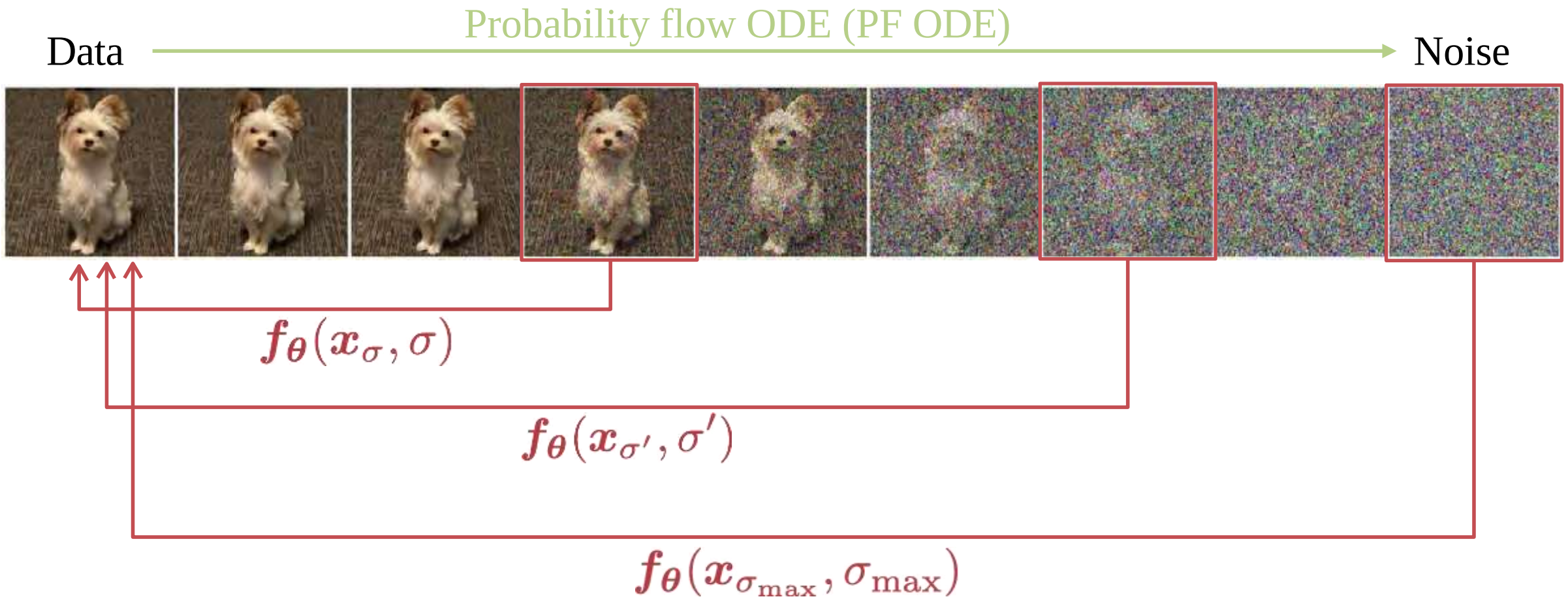
Require: Trained teacher model $\hat{\mathbf{x}}_\eta(\mathbf{z}_t)$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N
for K iterations **do**
  $\theta \leftarrow \eta$ ▷ Init student from teacher
  while not converged **do**
    $\mathbf{x} \sim \mathcal{D}$
    $t = i/N, \;\; i \sim Cat[1, 2, \ldots, N]$
    $\epsilon \sim N(0, I)$
    $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
    `# 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

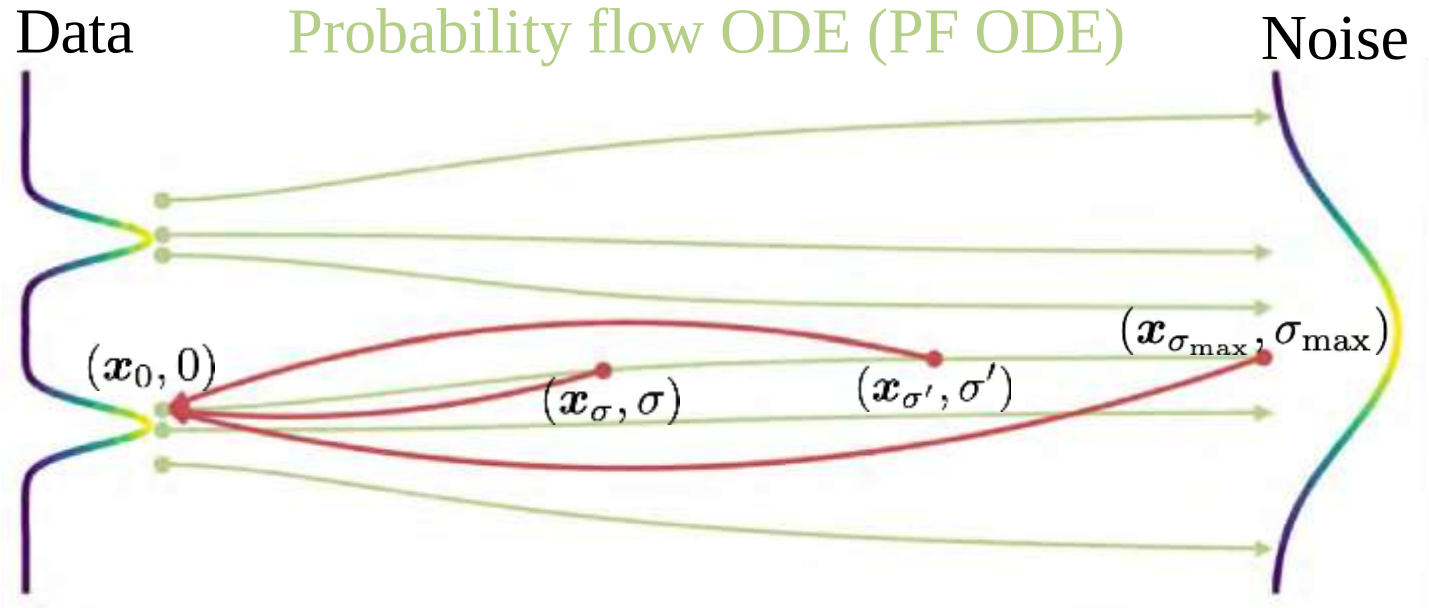


?

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**.

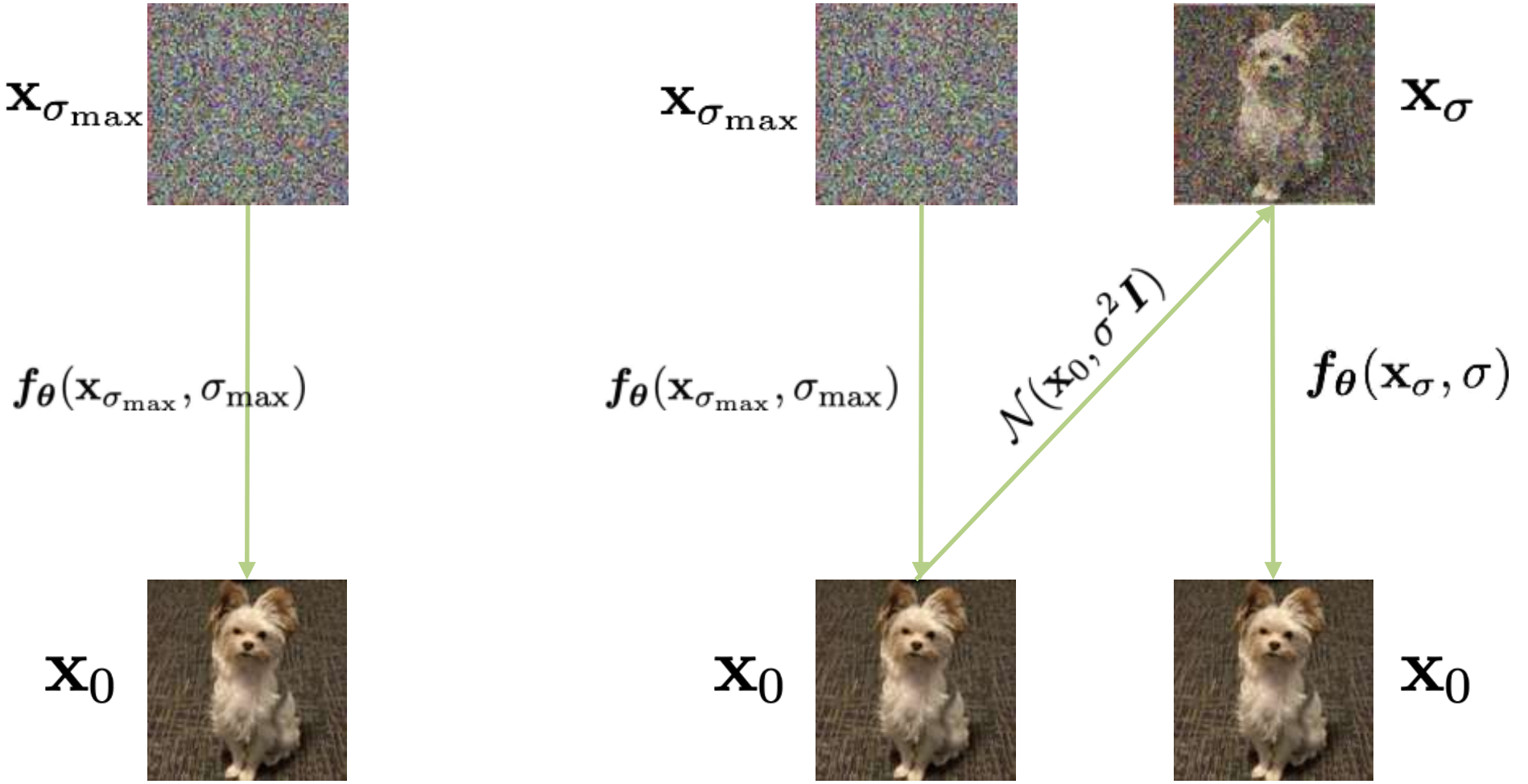
$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_\sigma, \sigma) = \boldsymbol{x}_0$$

Enforced via learning

Self-consistency

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

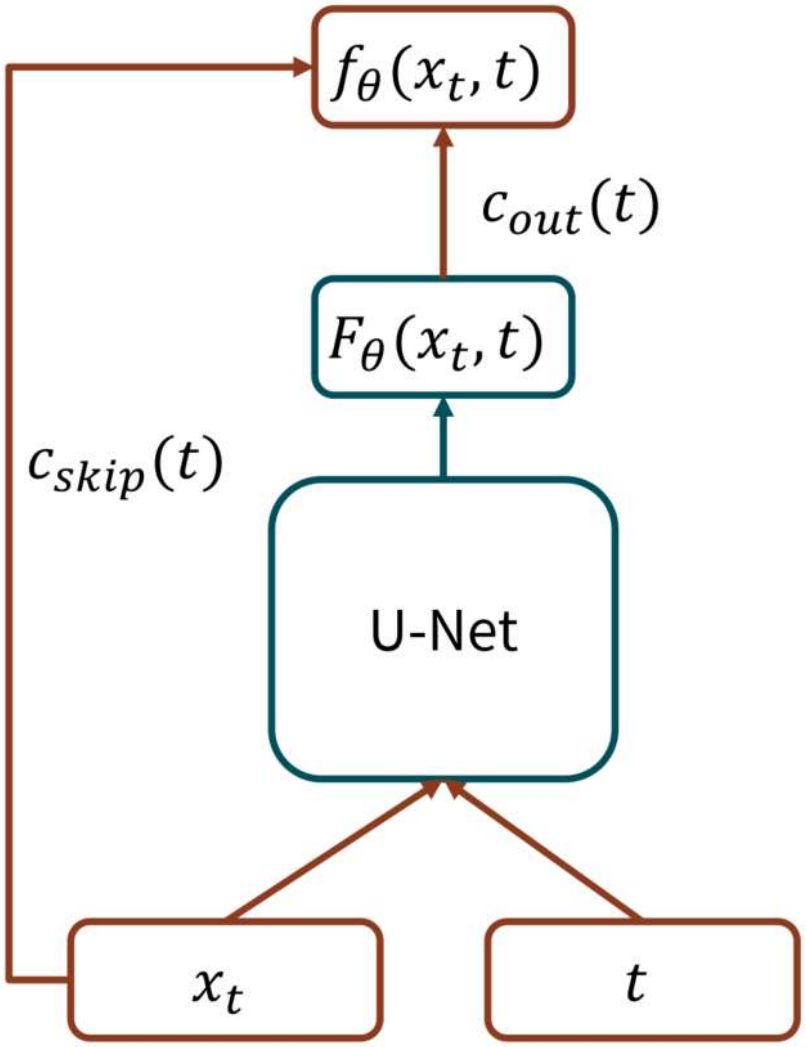
Sampling from consistency models



- Trading compute for quality
- Zero-shot image editing

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

Enforcing the boundary condition



- Skip connections for enforcing the boundary condition:

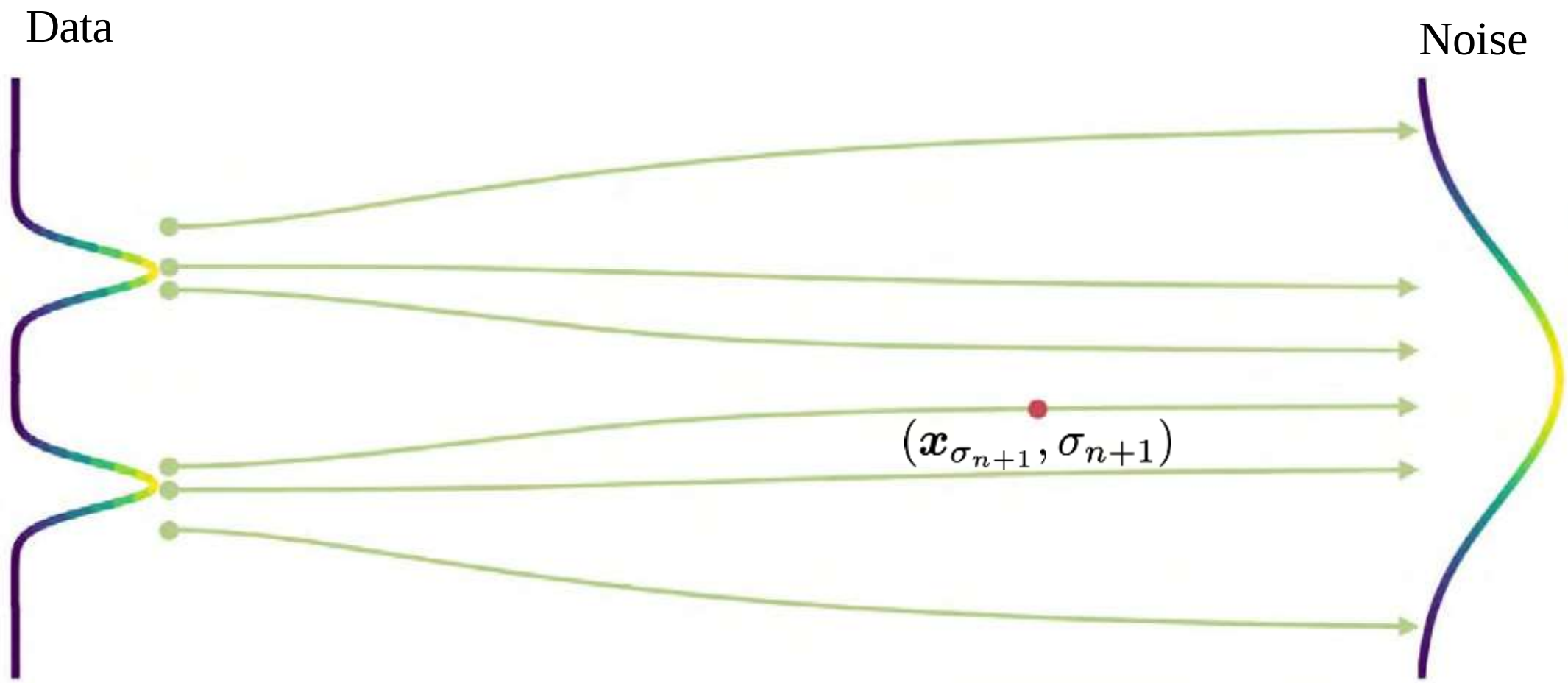
$$\boldsymbol{f_\theta}(\mathbf{x}, t) = c_{\text{skip}}(t)\mathbf{x} + c_{\text{out}}(t)\boldsymbol{F_\theta}(\mathbf{x}, t)$$

$$c_{\text{skip}}(0) = 1$$
$$c_{\text{out}}(0) = 0$$

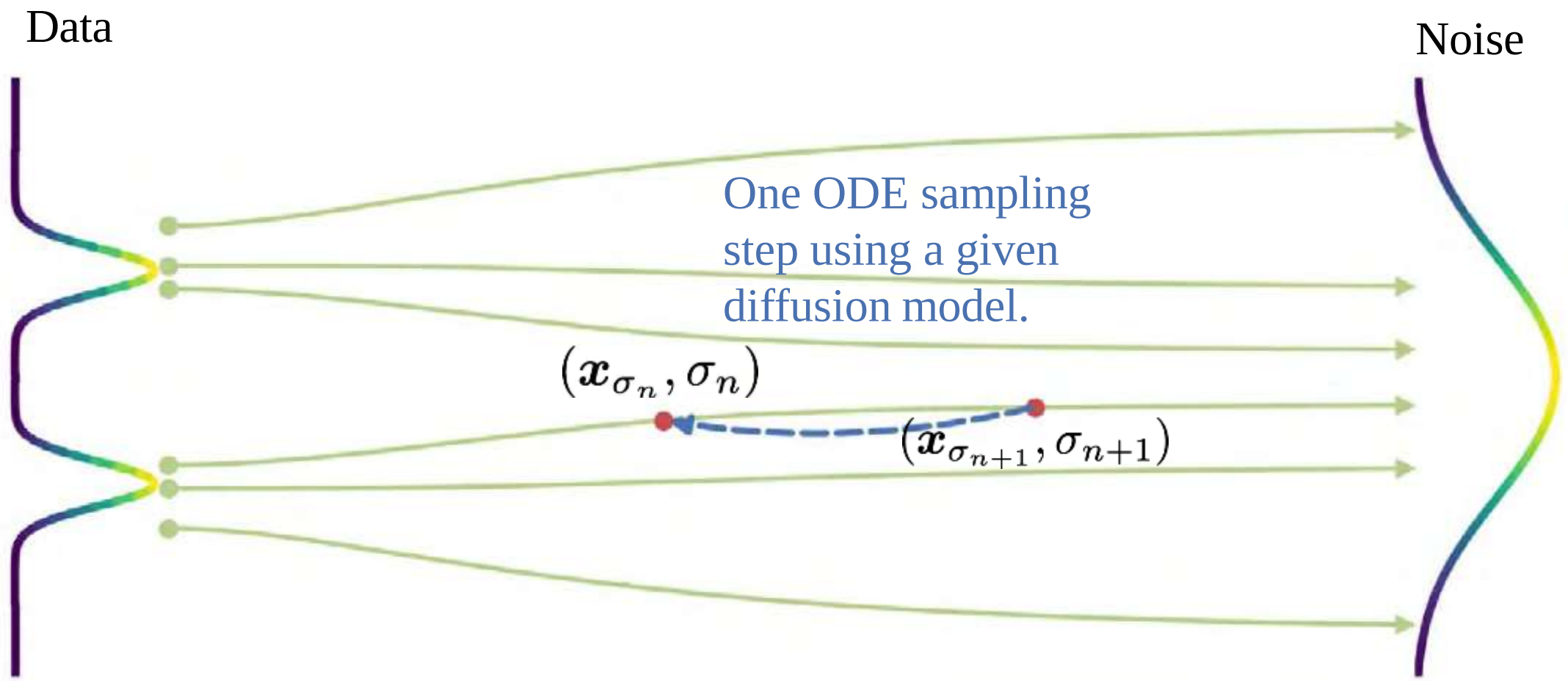
- The denoising/score network in diffusion models often has a similar parameterization (cf., EDM, v-prediction, etc.)

Karras, Tero, et al. "Elucidating the design space of diffusion-based generative models." *arXiv preprint arXiv:2206.00364* (2022).

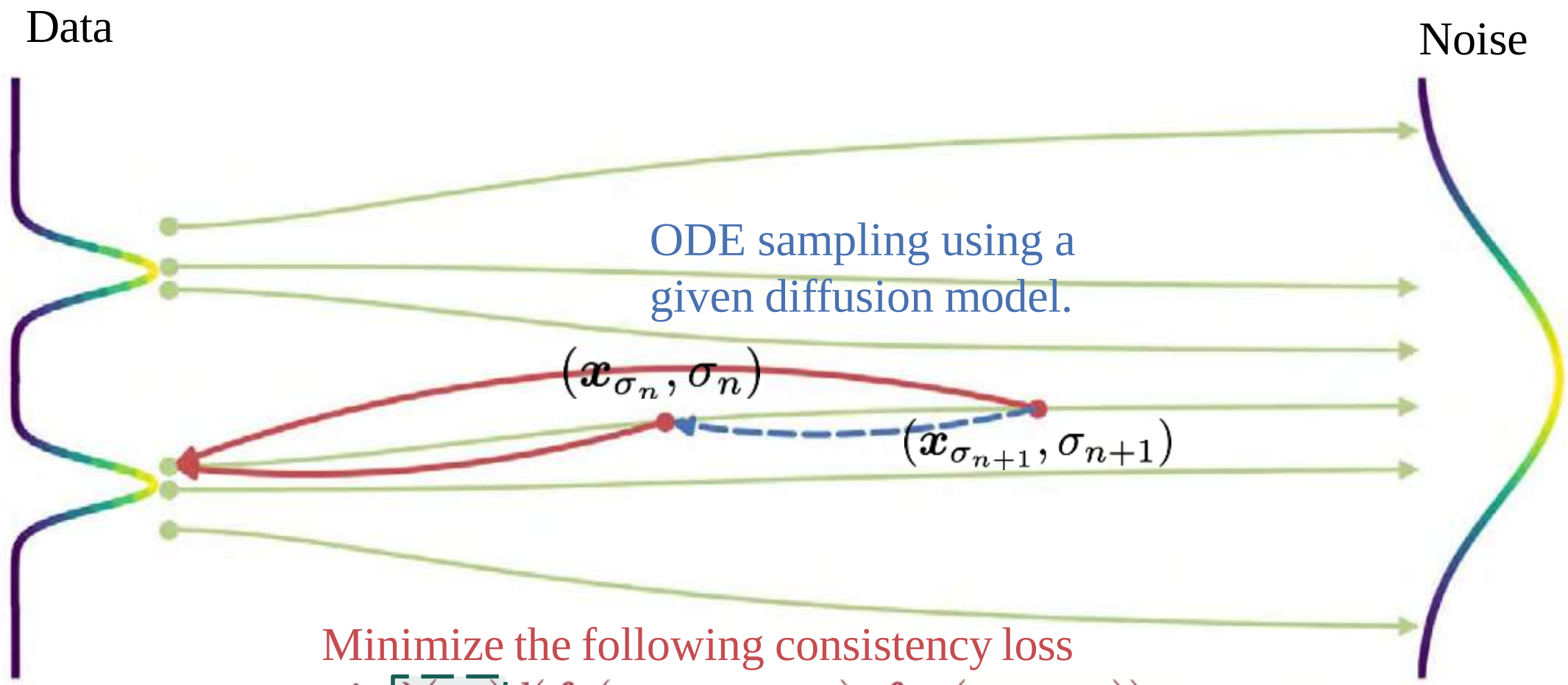
Training consistency models via distillation



Training consistency models via distillation



Training consistency models via distillation



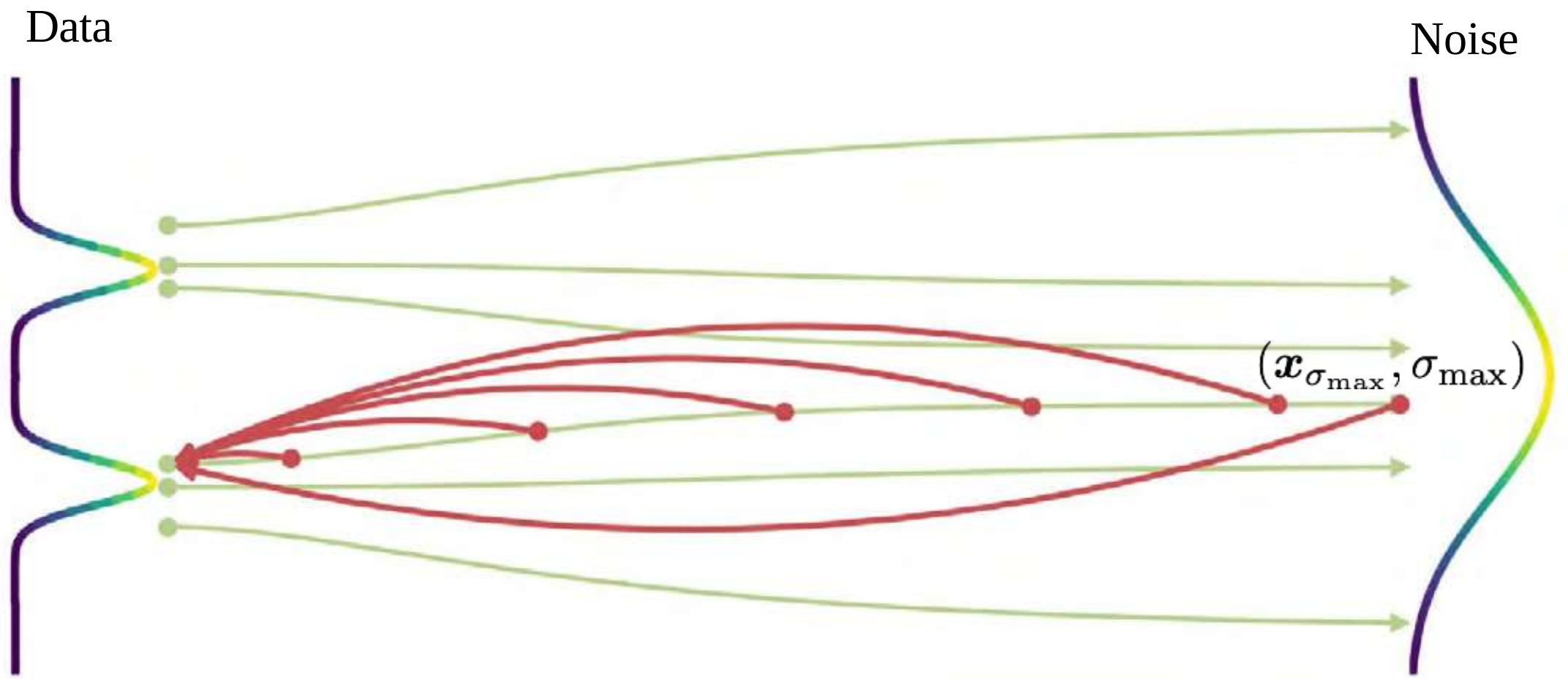
Minimize the following consistency loss

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

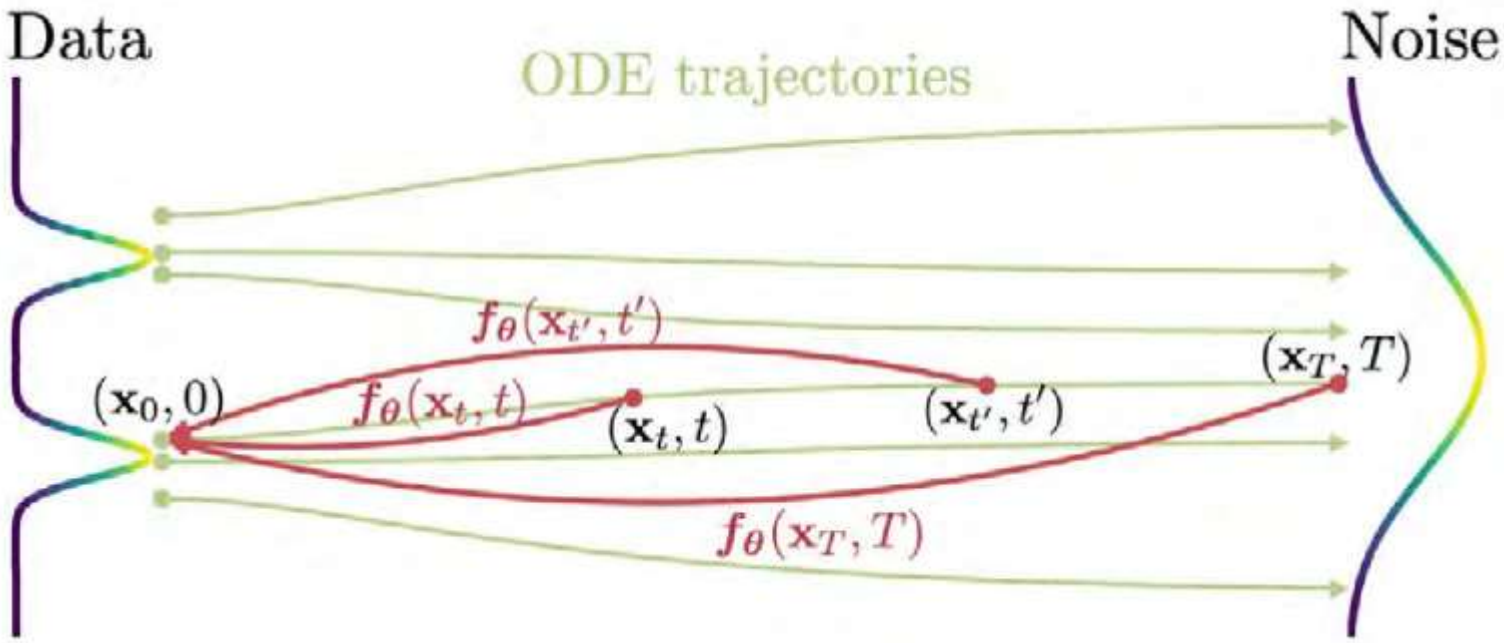
Weighting function

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

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

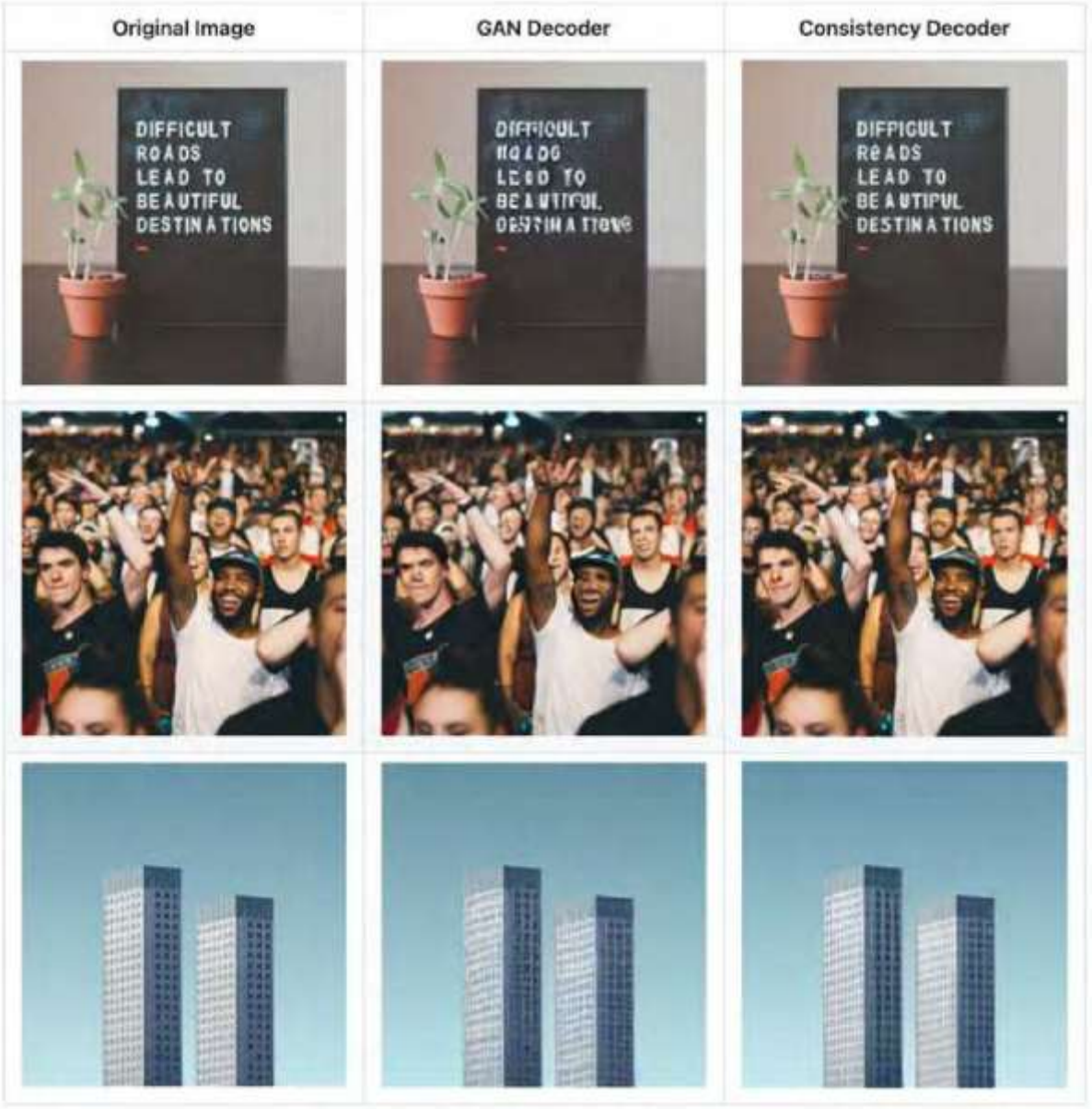
Consistency models distilled from diffusion models

EDM
FID = 3.57
NFE = 79

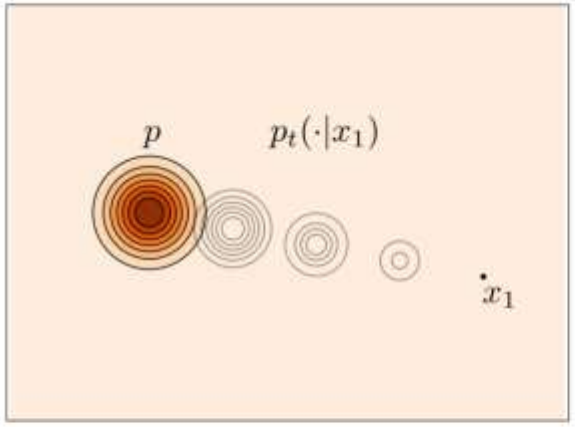
One step
FID = 7.80
NFE = 1

Two steps
FID = 5.22
NFE = 2

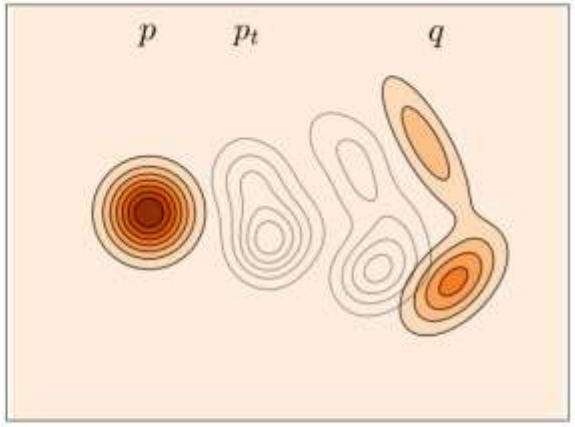
Consistency decoder in DALLE 3



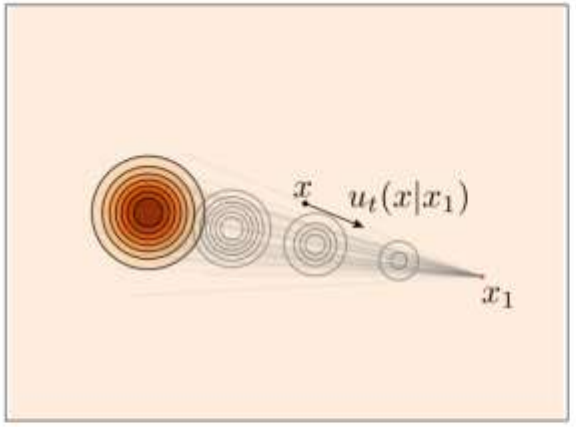
Flow Matching



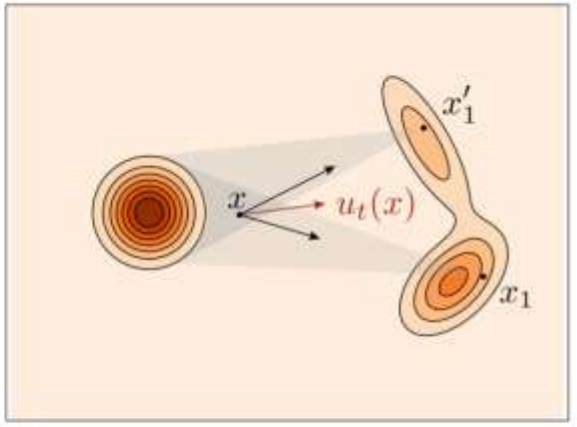
(a) Conditional probability path $p_t(x|x_1)$.



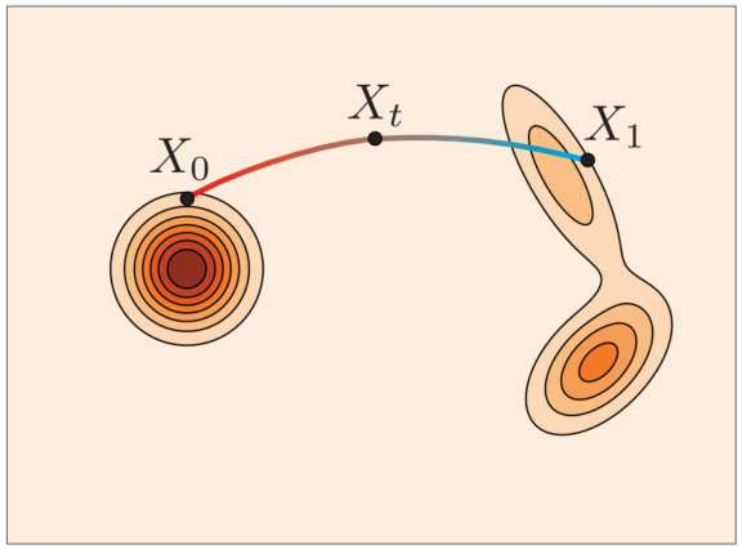
(b) (Marginal) Probability path $p_t(x)$.



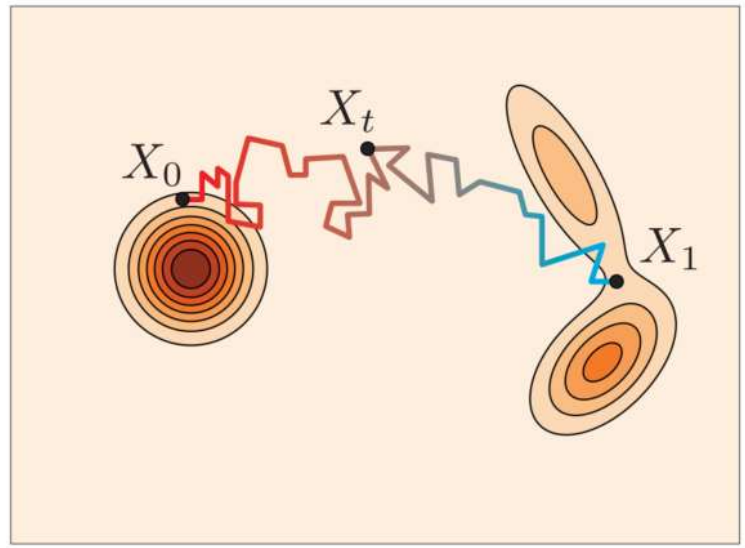
(c) Conditional velocity field $u_t(x|x_1)$.



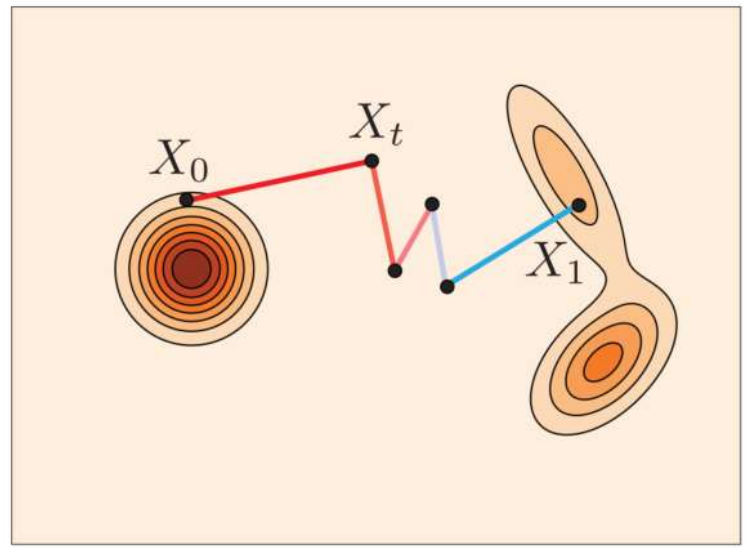
(d) (Marginal) Velocity field $u_t(x)$.



(a) Flow

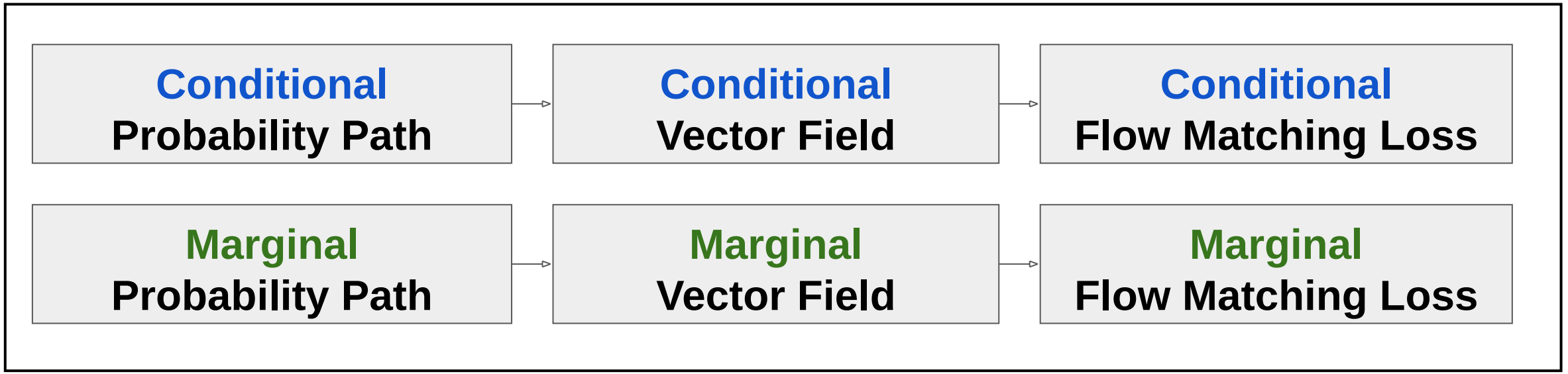


(b) Diffusion



(c) Jump

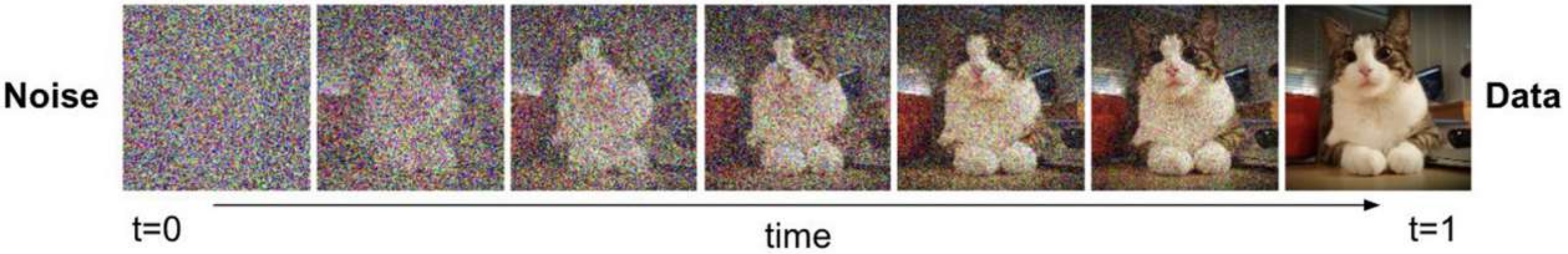
The Flow Matching Matrix

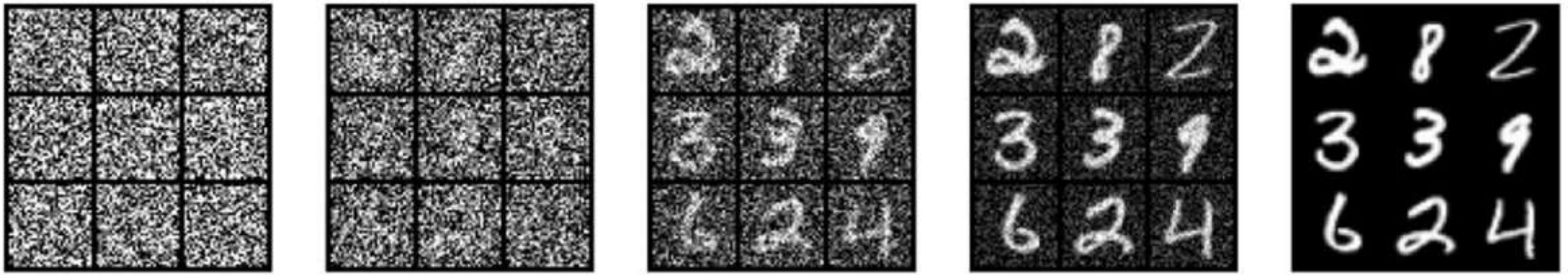


“Conditional” = “Per single data point”

“Marginal” = “Across distribution of data points”

Probability Paths: The Path from Noise to Data

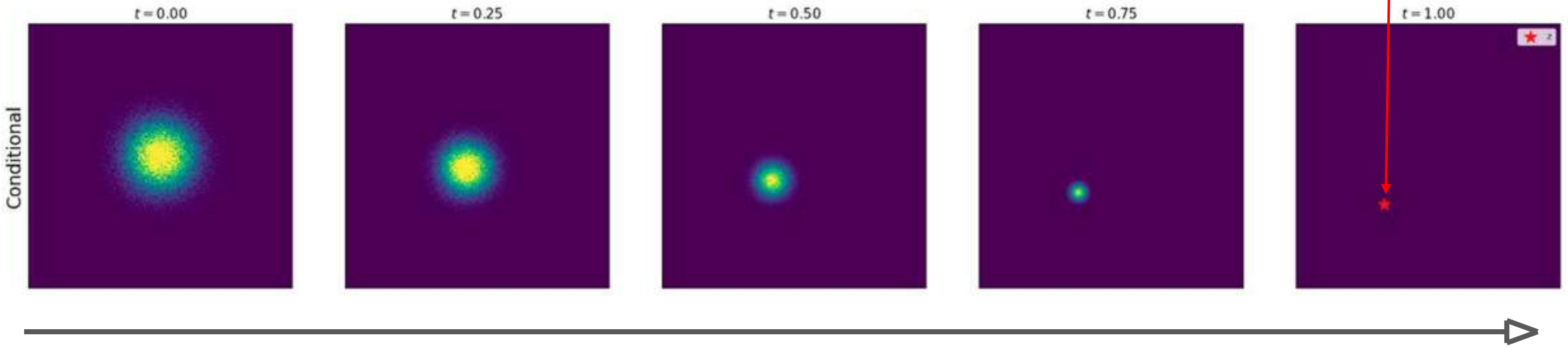




p_{init}

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

z



t=0

t=1

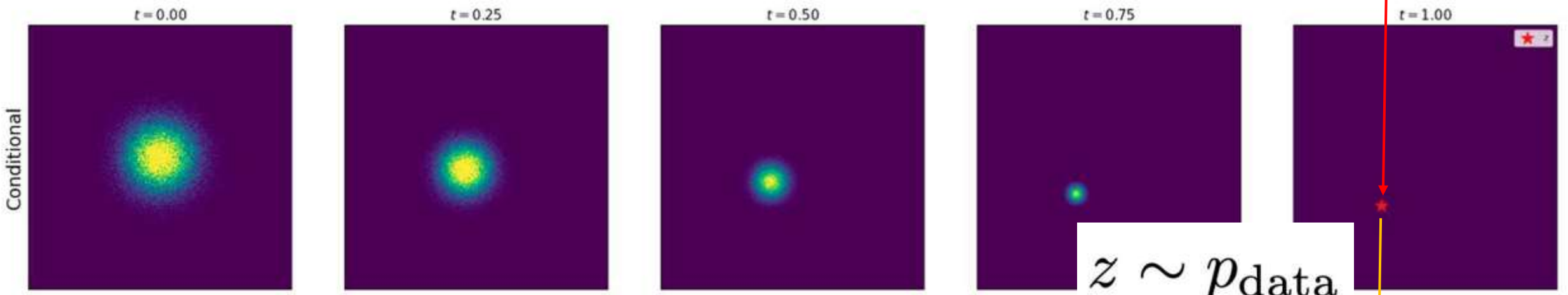
Marginal Probability Path p_t

Conditional Probability Path

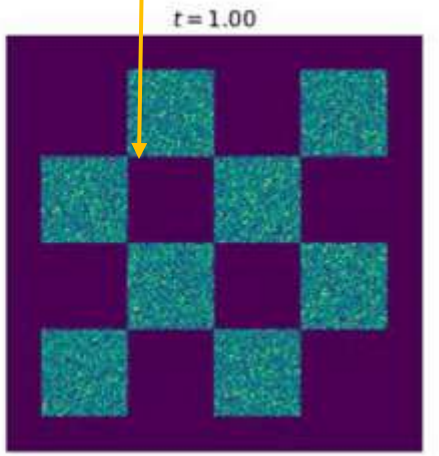
p_{init}

$p_t(\cdot|z)$

z



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

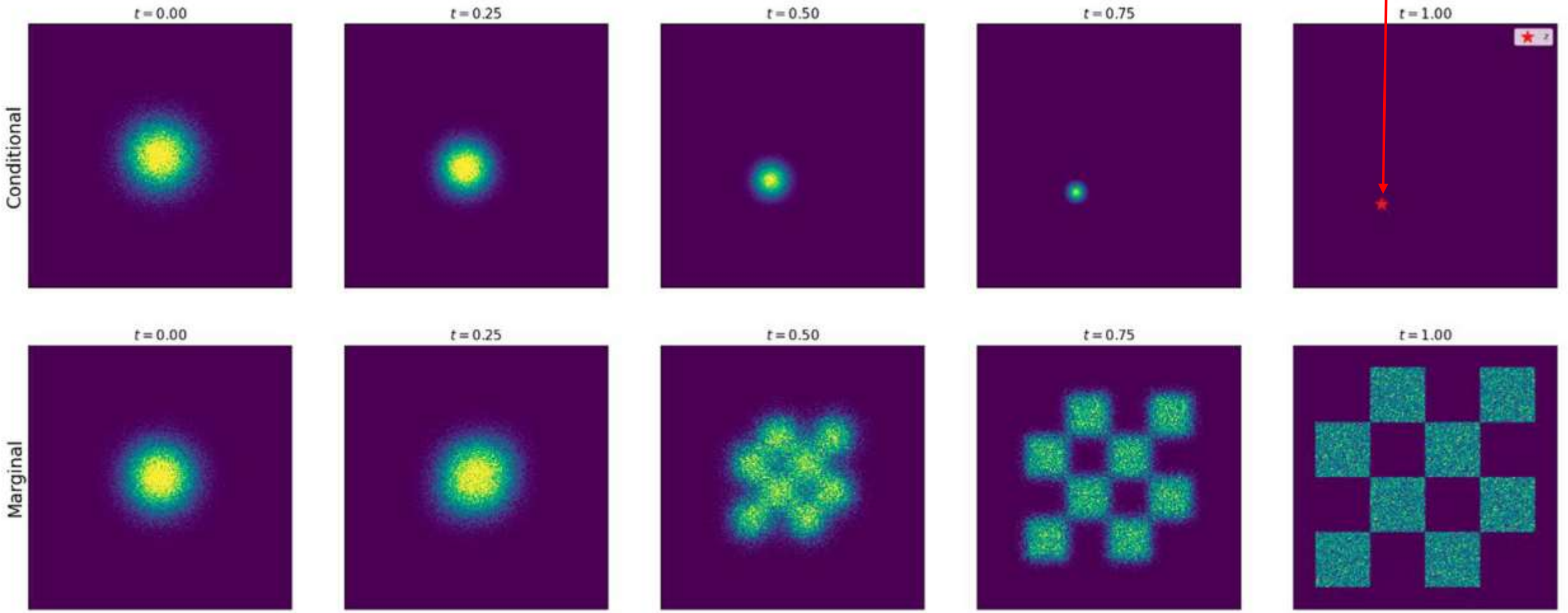


p_{data}

p_{init}

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

z



p_{init}

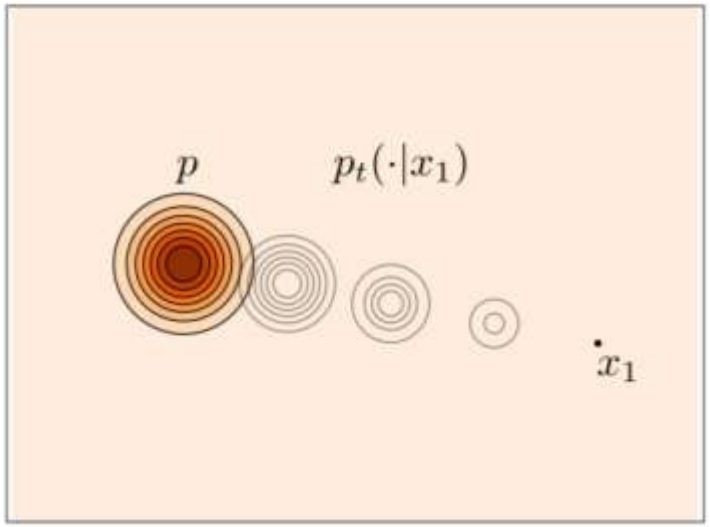
Marginal Probability Path p_t

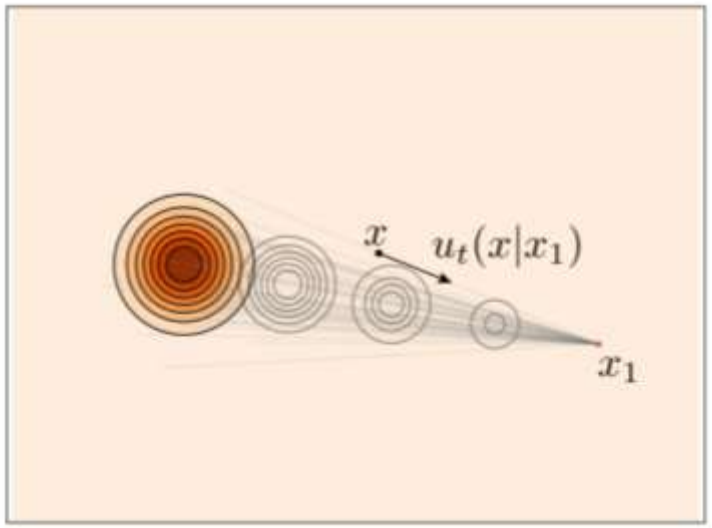
p_{data}

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





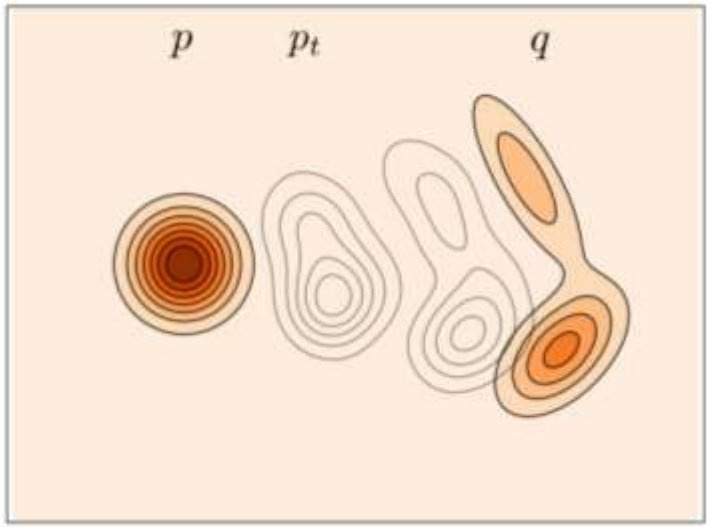
Conditional Prob. Path

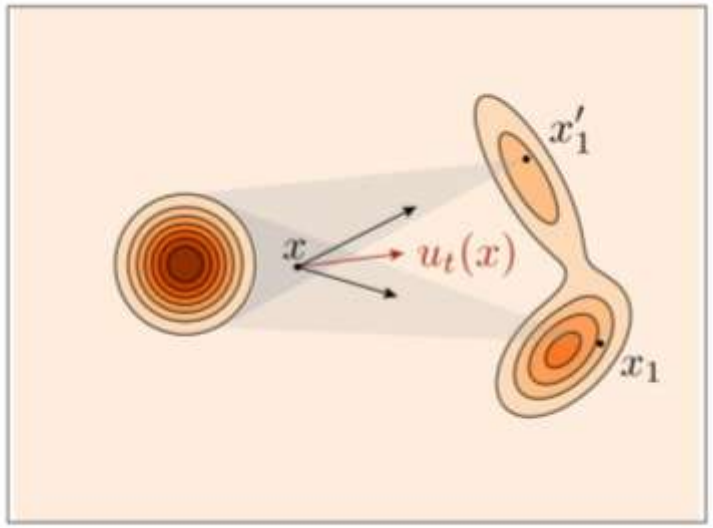
	Notation	Key property	Gaussian example
Conditional Probability Path	$p_t(\cdot\|z)$	Interpolates p_{init} and a data point z	$p_t(\cdot\|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$
Conditional Vector Field	$u_t^{\text{target}}(x\|z)$	ODE follows conditional path	$\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right)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

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

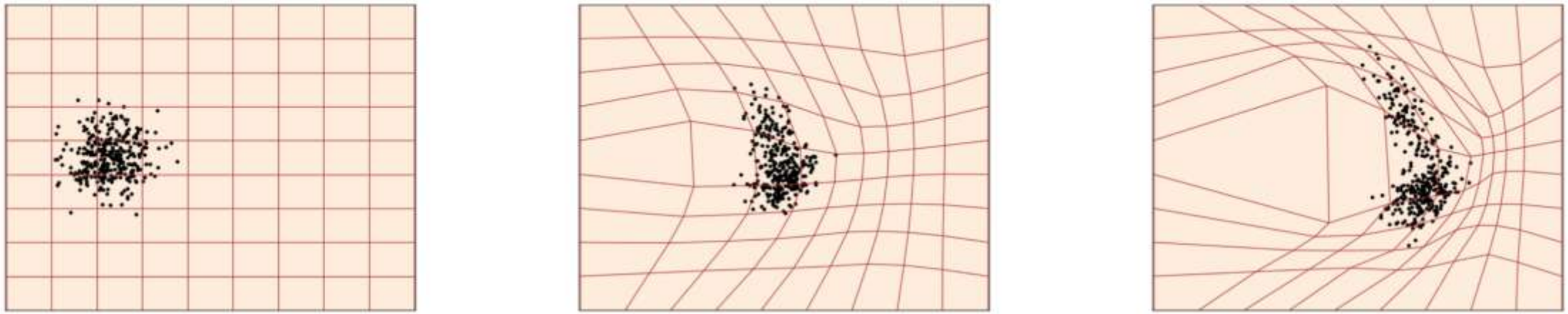


Figure 5 A flow model $X_t = \psi_t(X_0)$ is defined by a diffeomorphism $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with a brown square grid) pushing samples from a source RV X_0 (left, black points) toward some target distribution q (right). We show three different times t.

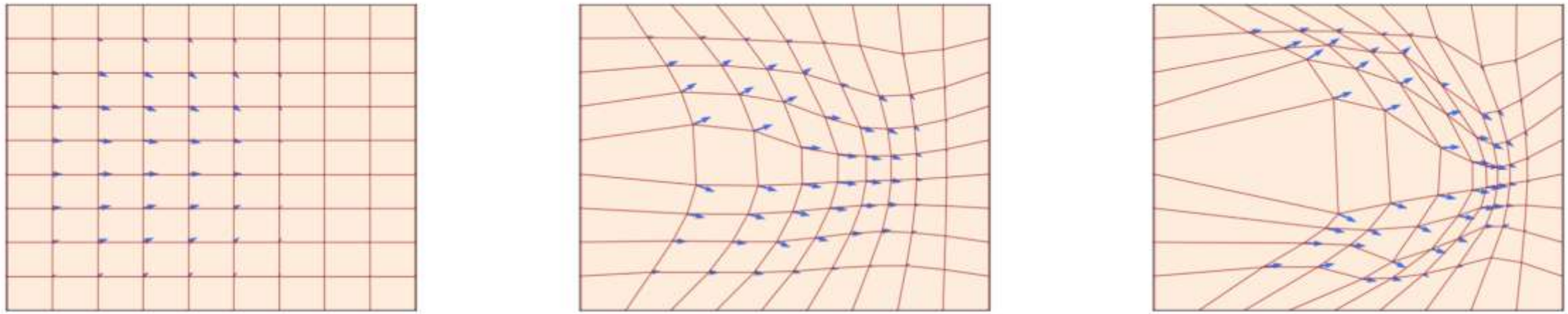


Figure 6 A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations. We show three different times t.

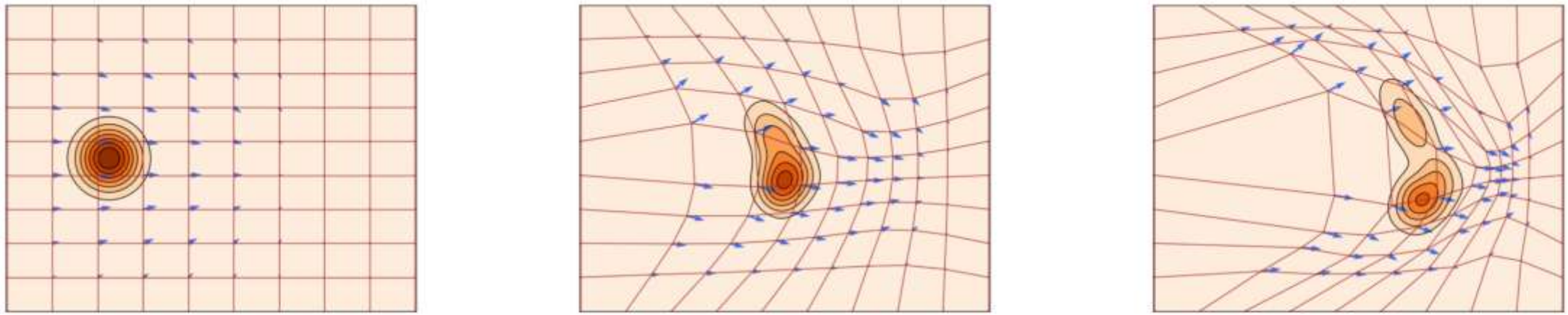
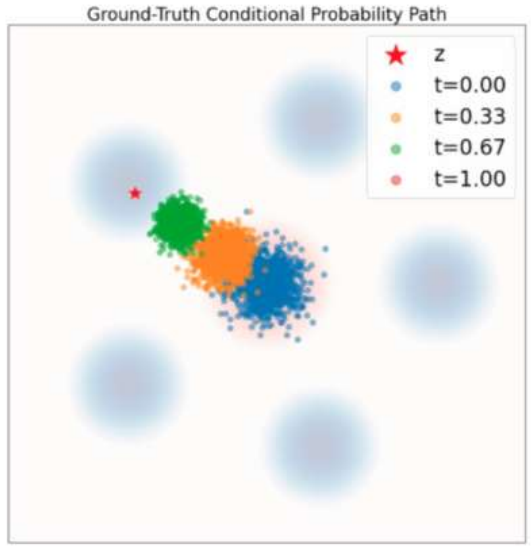
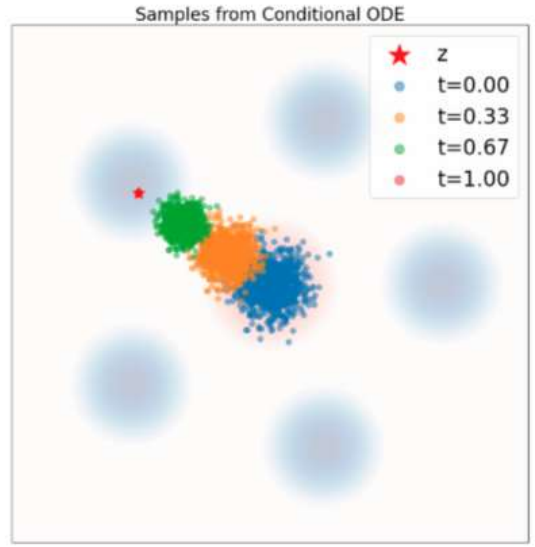
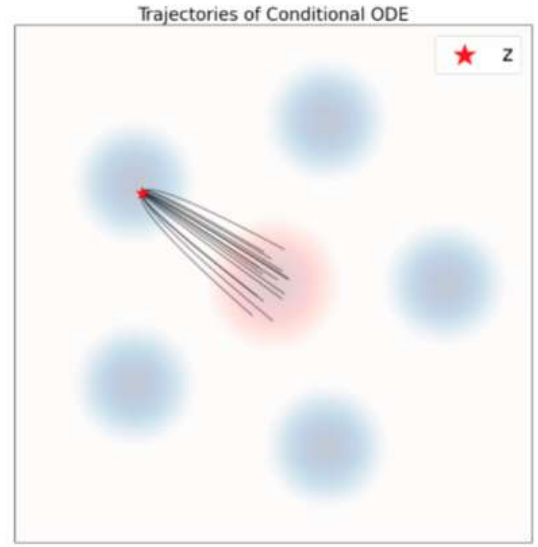


Figure 7 A velocity field u_t (in blue) *generates* a probability path p_t (PDFs shown as contours) if the flow defined by u_t (square grid) reshapes p (left) to p_t at all times $t \in [0, 1)$.

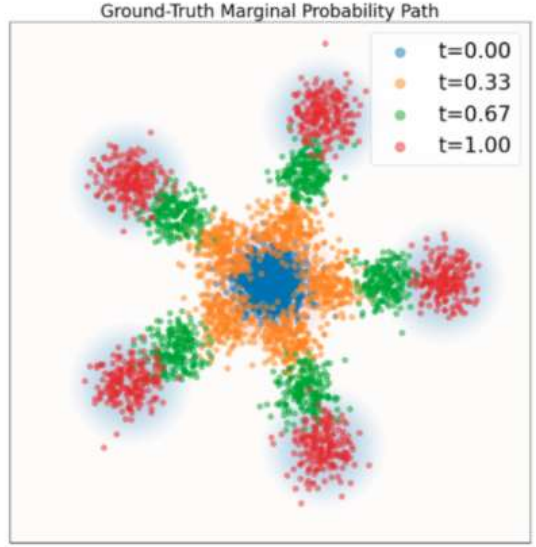
$p_t(\cdot|z)$

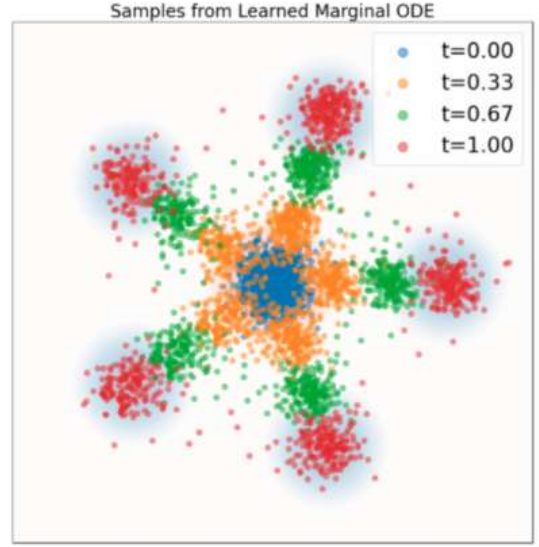

Ground-Truth Conditional Probability Path
z
t=0.00
t=0.33
t=0.67
t=1.00

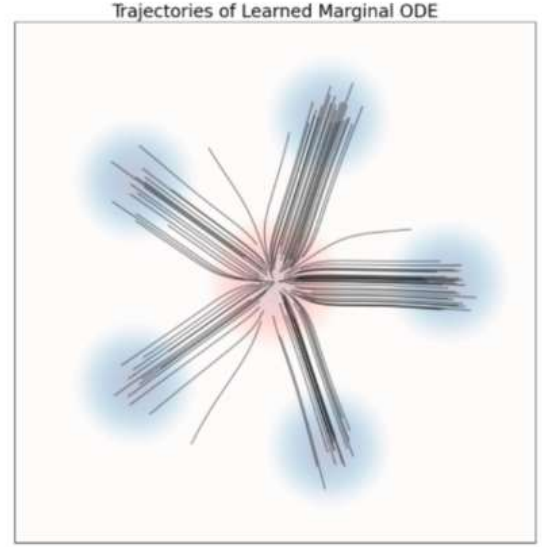

Samples from Conditional ODE
z
t=0.00
t=0.33
t=0.67
t=1.00


Trajectories of Conditional ODE
z

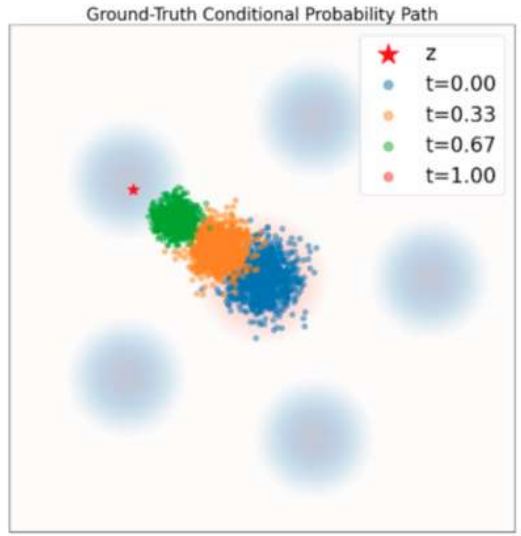
p_t

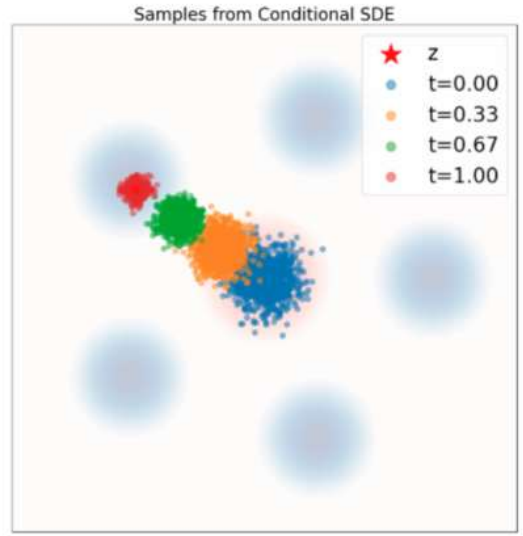

Ground-Truth Marginal Probability Path
t=0.00
t=0.33
t=0.67
t=1.00

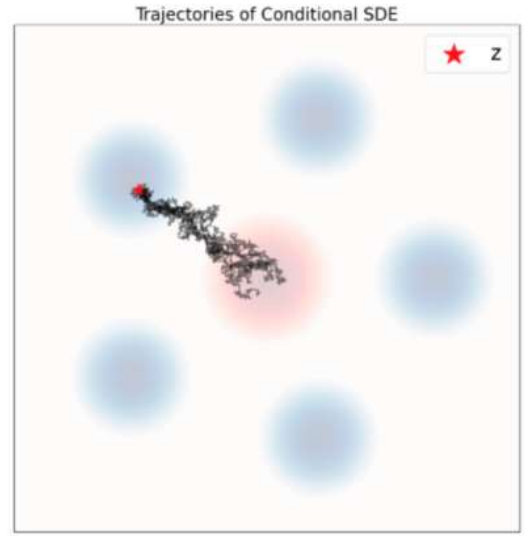

Samples from Learned Marginal ODE
t=0.00
t=0.33
t=0.67
t=1.00


Trajectories of Learned Marginal ODE

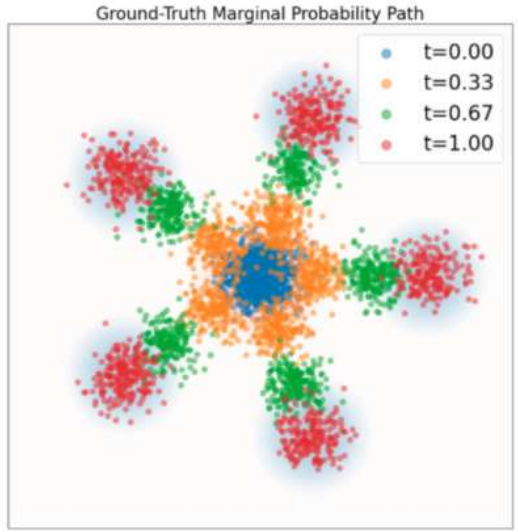
$p_t(\cdot|z)$

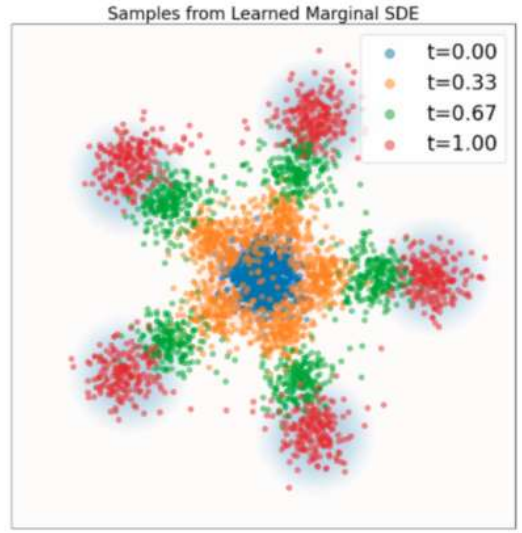

Ground-Truth Conditional Probability Path
z
t=0.00
t=0.33
t=0.67
t=1.00

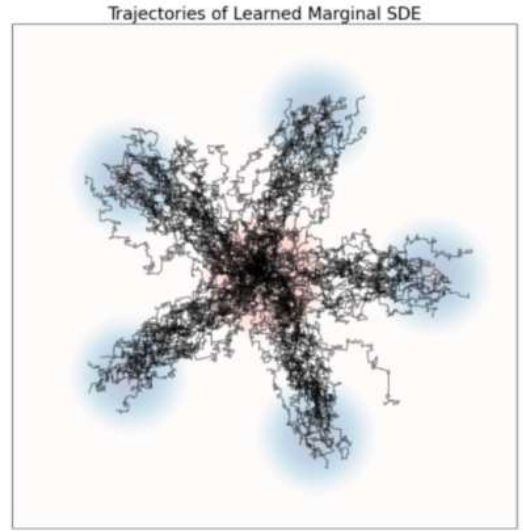

Samples from Conditional SDE
z
t=0.00
t=0.33
t=0.67
t=1.00


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

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

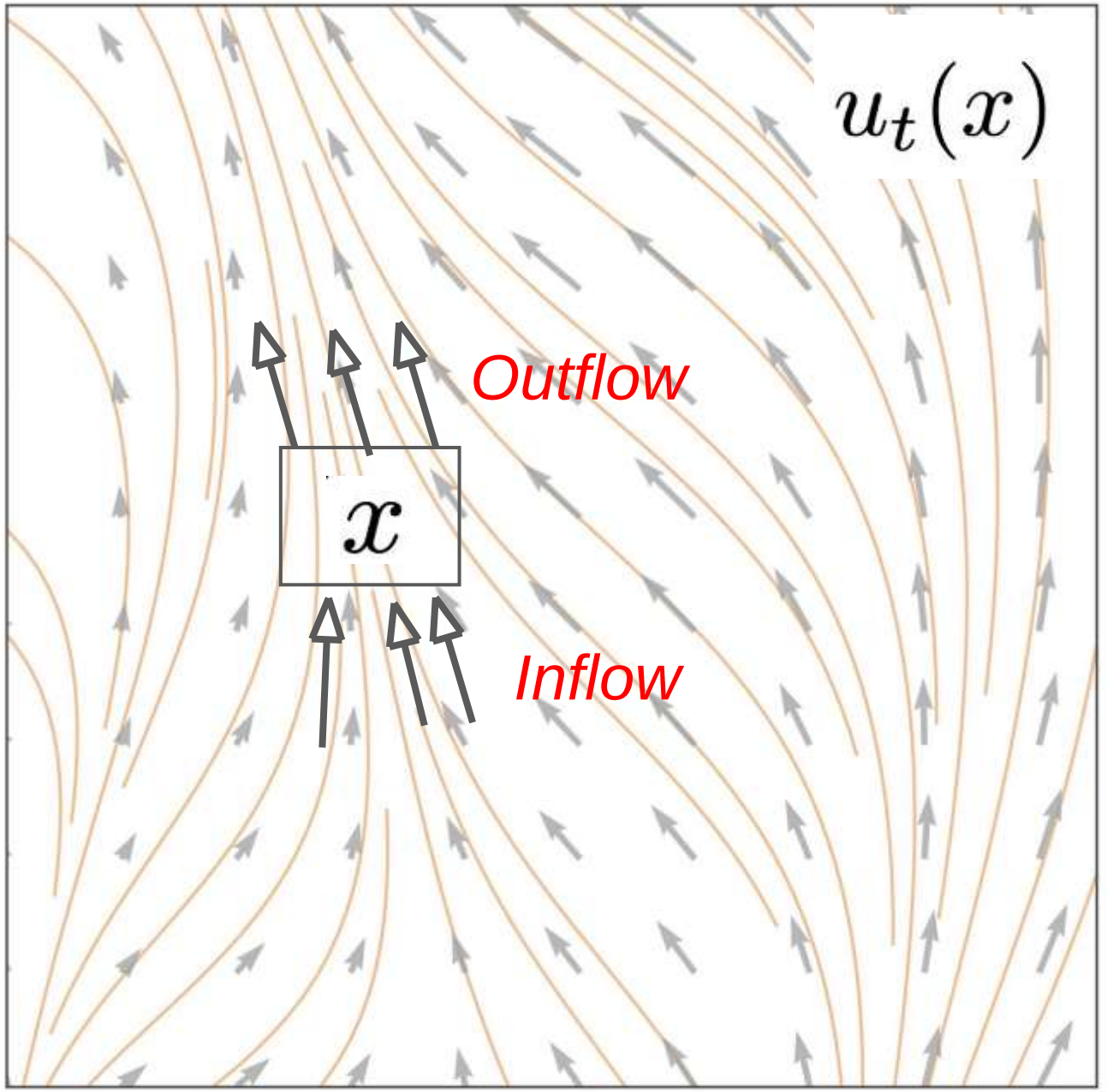
PDE holds

Continuity Equation

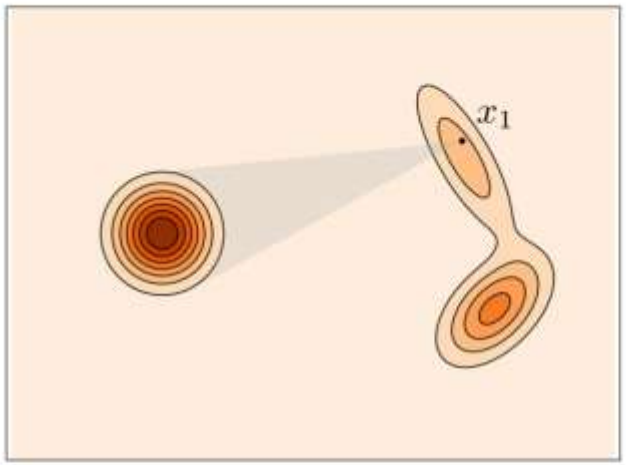
$$\frac{\mathrm{d}}{\mathrm{d}t}p_t(x) = -\mathrm{div}(p_t u_t)(x)$$

Change of probability mass at x

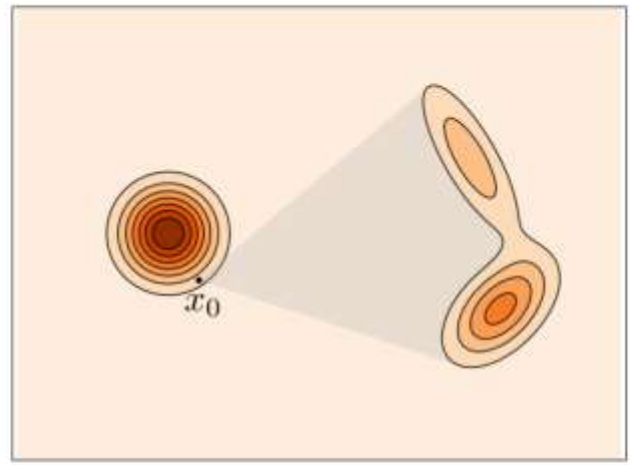
Outflow - inflow of probability mass from u



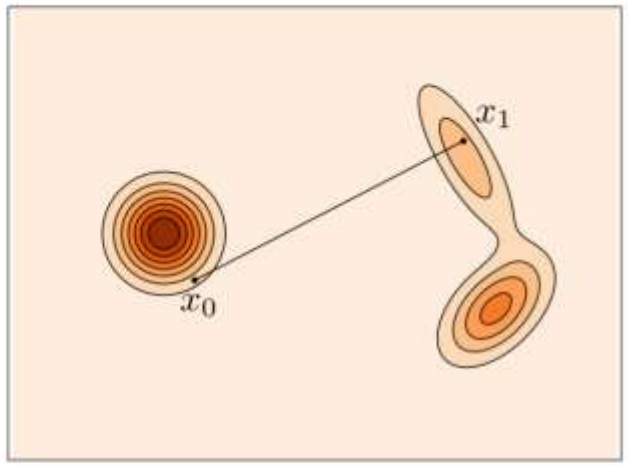
X_1 conditioning



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

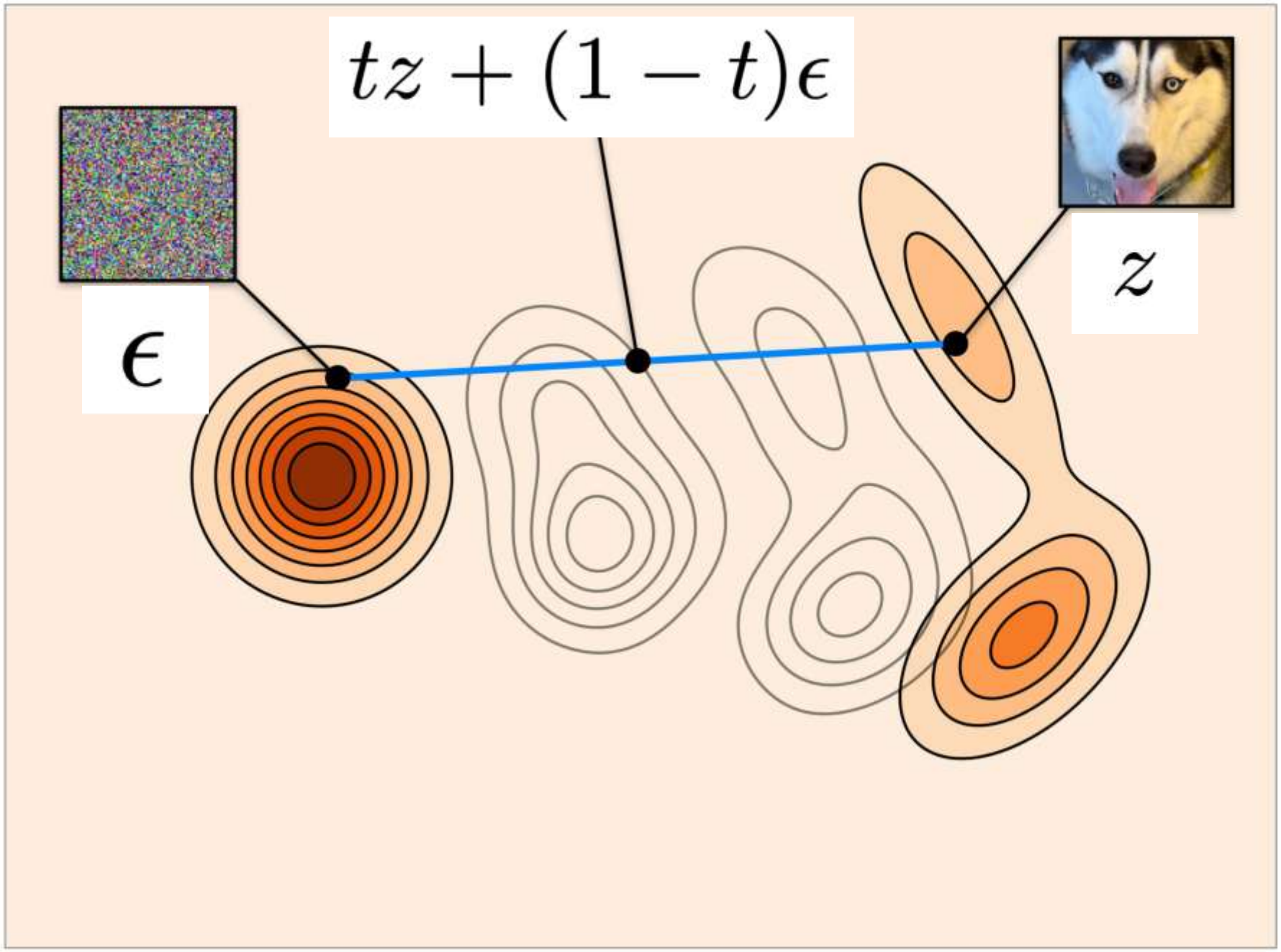


Figure credit: Yaron Lipman

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!

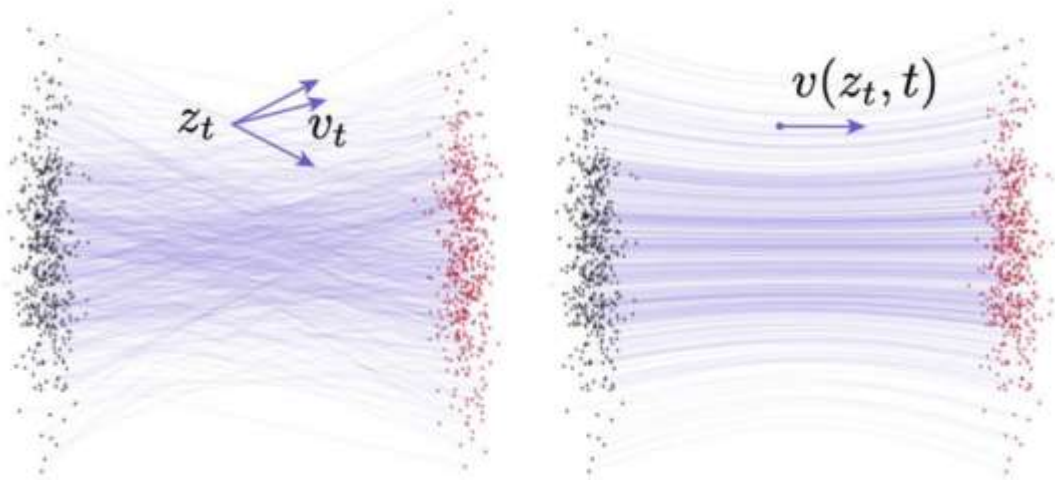


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal 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.*)

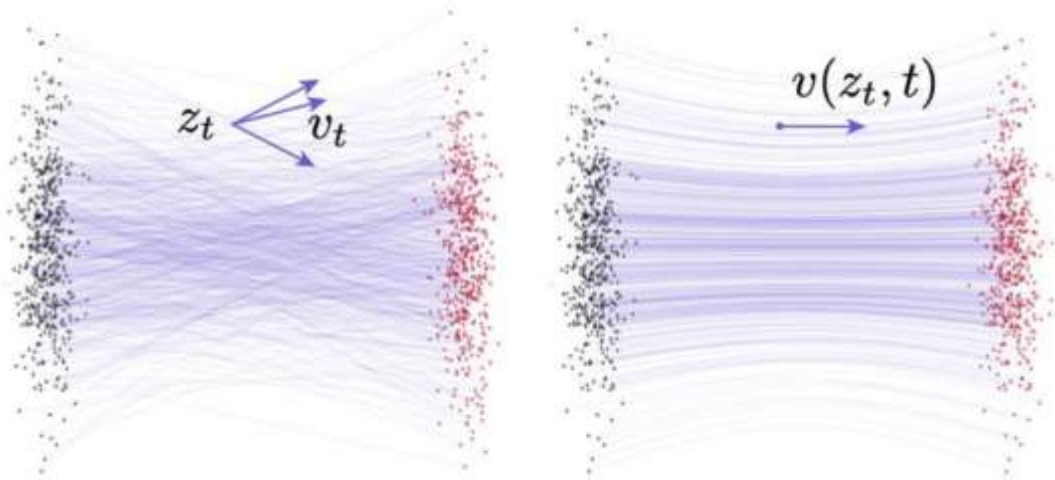
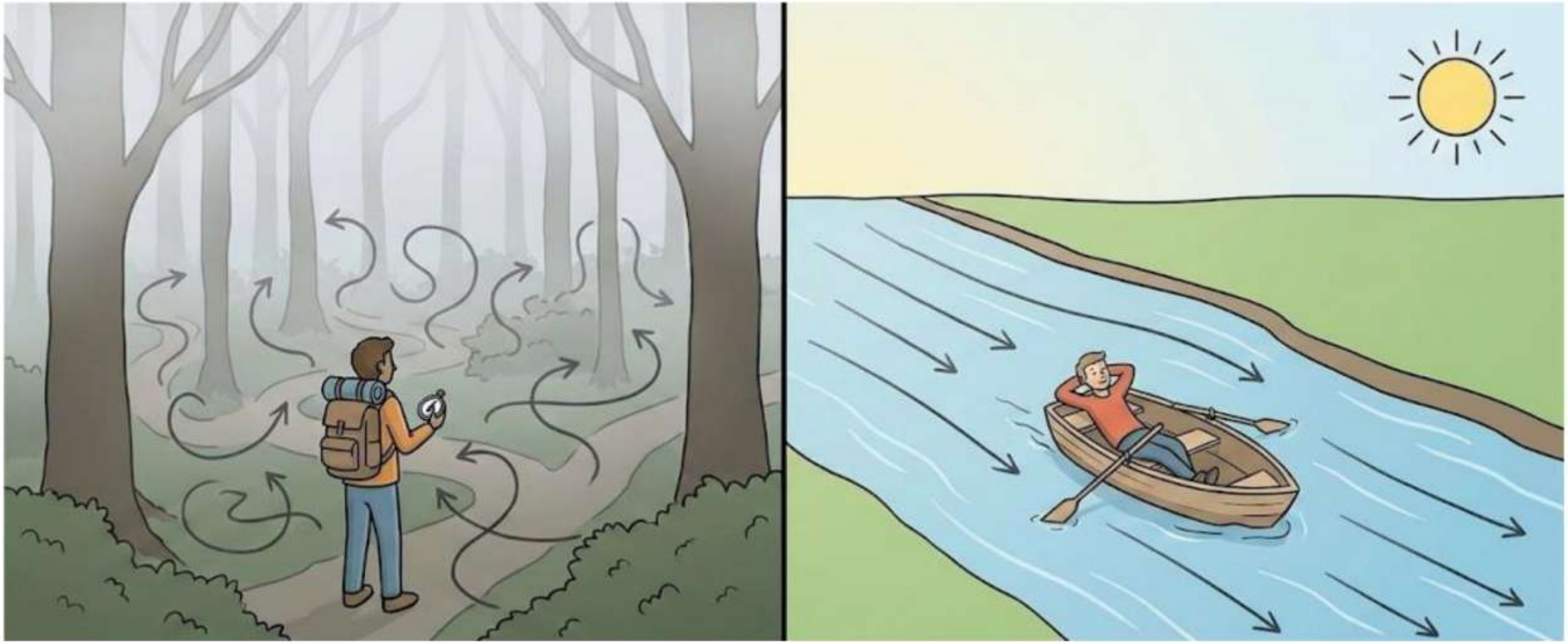


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal 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

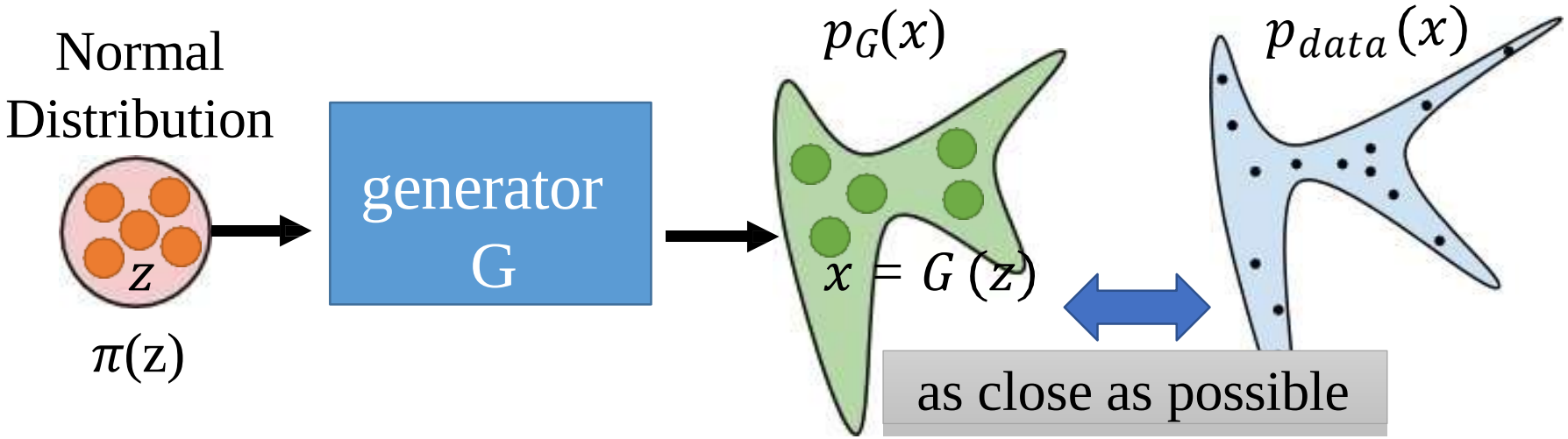
Flow Matching



Flow-Based Models

Generator

- A generator G is a network. The network defines a probability distribution p_G



$$G^* = arg \max_G \sum_{i=1}^{m} log P_G(x^i)$$

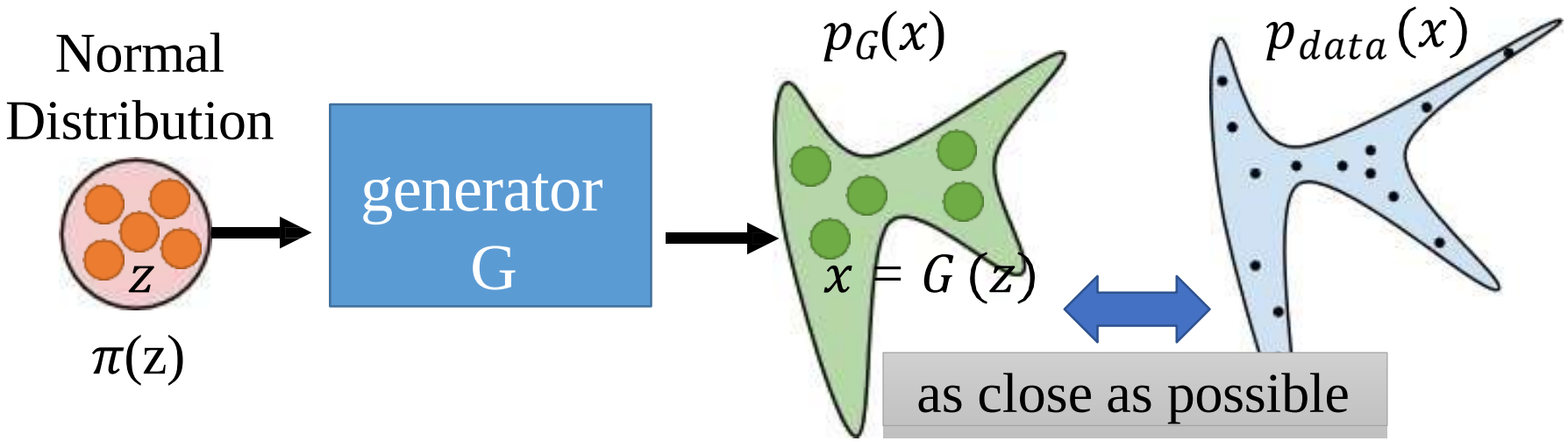
$$\approx arg \min_G KL(P_{data} || P_G)$$

$\{x^1, x^2, \ldots, x^m\}$ from $P_{data}(x)$

Ref: https://youtu.be/DMA4MrNieWo

Generator

- A generator G is a network. The network defines a probability distribution p_G



$$G^* = arg\max_G \sum_{i=1}^{m} logP_G(x^i)$$

$\{x^1, x^2, \ldots, x^m\}$ from $P_{data}(x)$

Flow-based model directly optimizes the objective function.

Math Background

Jacobian, Determinant, Change of Variable Theorem

Jacobian Matrix

$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x = f(z) \qquad z = f^{-1}(x)$$

$$J_f = \begin{bmatrix} \partial x_1 \mathrm{T} \partial z_1 & \partial x_1 \mathrm{T} \partial z_2 \\ \partial x_2 \mathrm{T} \partial z_1 & \partial x_2 \mathrm{T} \partial z_2 \end{bmatrix}$$

input

output

$$J_{f^{-1}} = \begin{bmatrix} \partial z_1 \mathrm{T} \partial x_1 & \partial z_1 \mathrm{T} \partial x_2 \\ \partial z_2 \mathrm{T} \partial x_1 & \partial z_2 \mathrm{T} \partial x_2 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\begin{bmatrix} z_1 + z_2 \\ 2z_1 \end{bmatrix} = f\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right)$$

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$\begin{bmatrix} x_2/2 \\ x_1 - x_2/2 \end{bmatrix} = f^{-1}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$$

$$J_f = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$$

$$J_{f^{-1}} = \begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix}$$

$$J_f J_{f^{-1}} = I$$

Determinant

The determinant of a **square matrix** is a **scalar** that provides information about the matrix.

- 2 X 2

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$det(A) = ad - bc$$

- 3 x 3

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$$

$$det(A) = a_1a_5a_9 + a_2a_6a_7 + a_3a_4a_8 - a_3a_5a_7 - a_2a_4a_9 - a_1a_6a_8$$

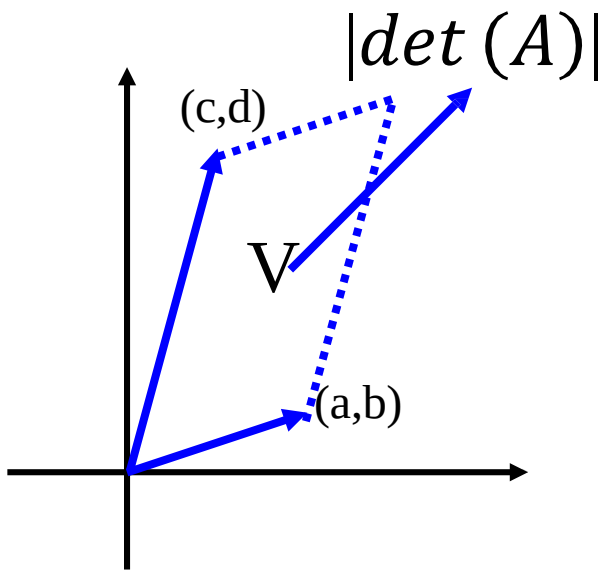
$$det(\mathrm{A}) = 1/det(A^{-1})$$

$$det(J_f) = 1/det(J_{f^{-1}})$$

Determinant

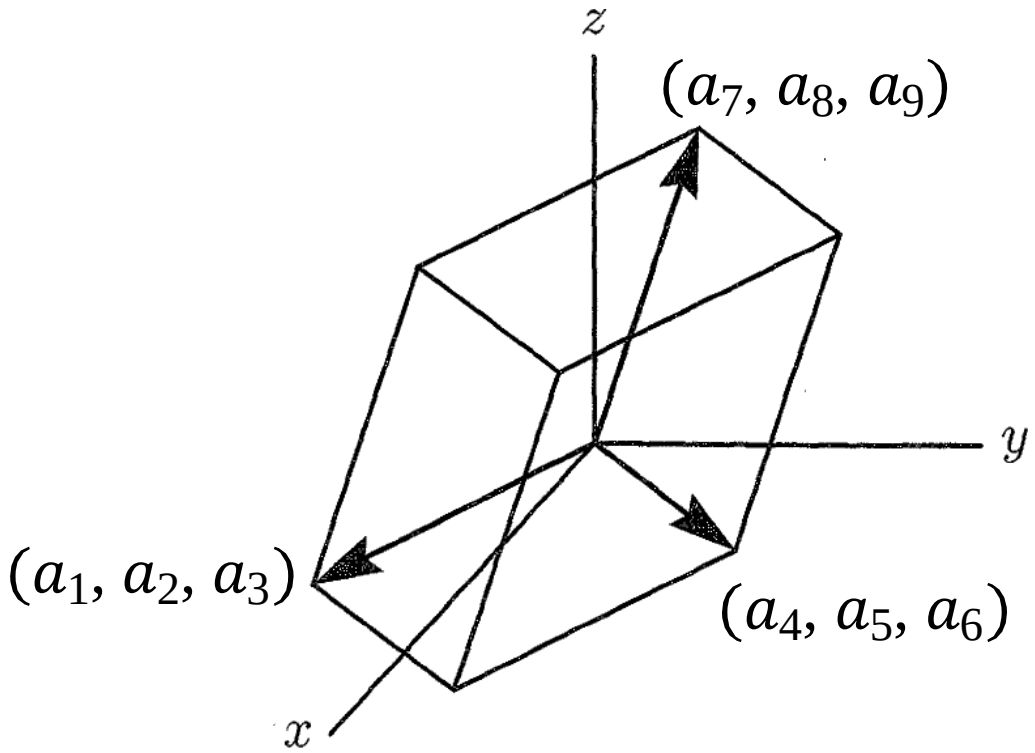
- 2 X 2

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

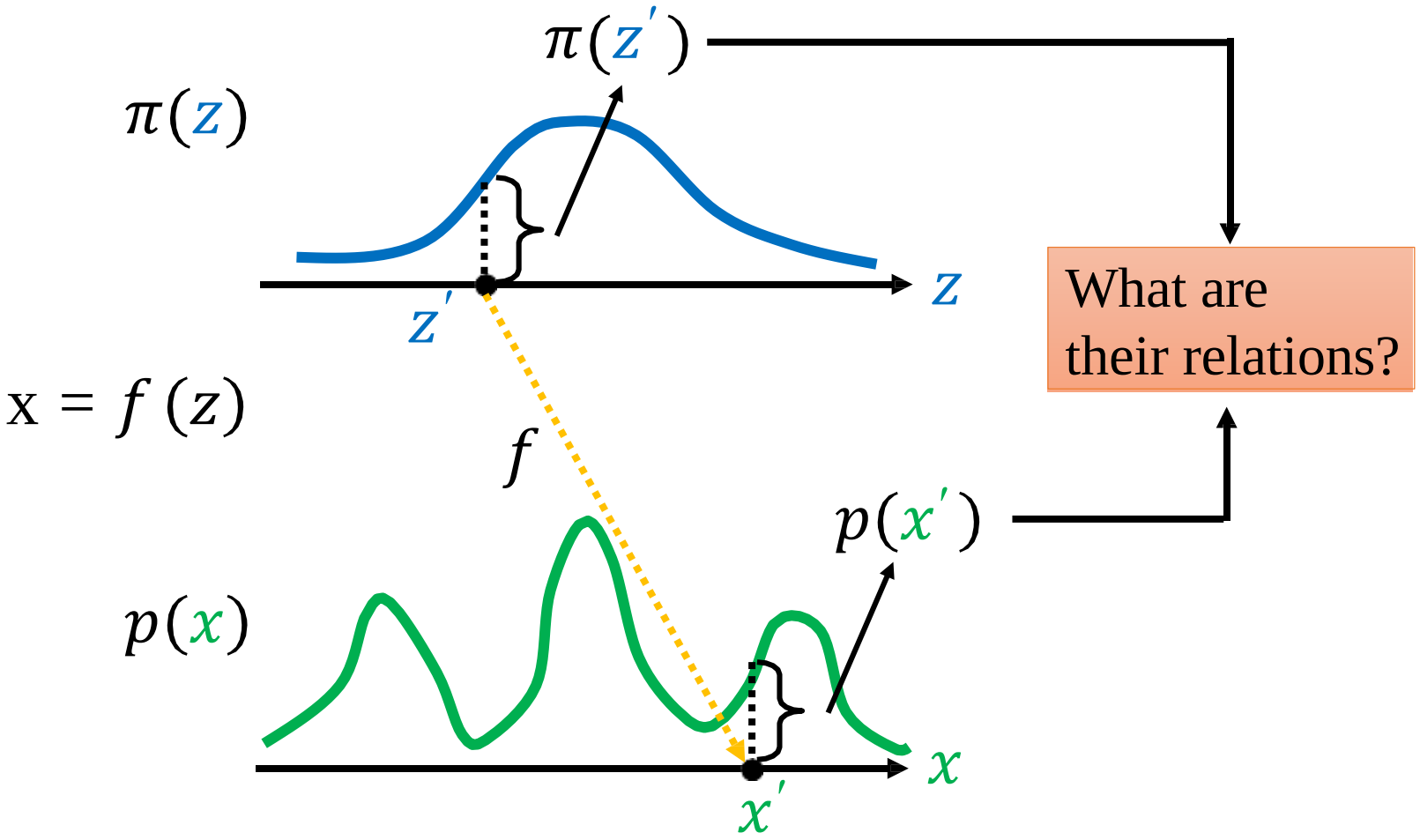


- 3 x 3

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$$

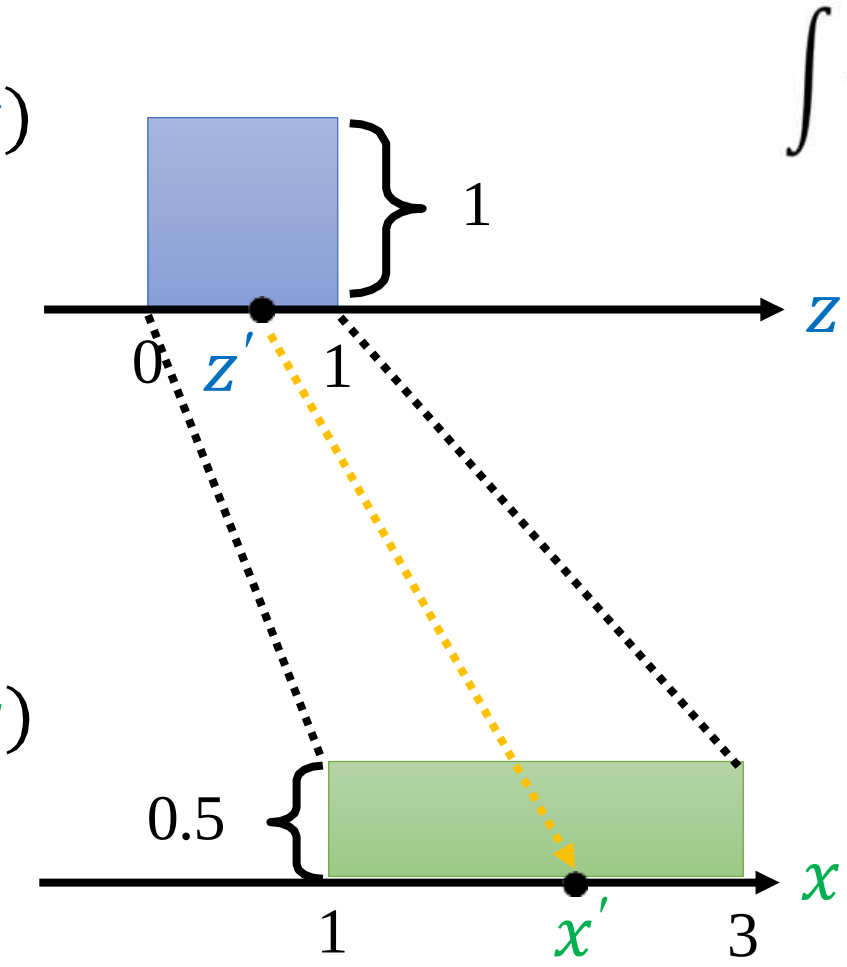


Change of Variable Theorem



Change of Variable Theorem

$$\int \pi(z)dz = 1$$

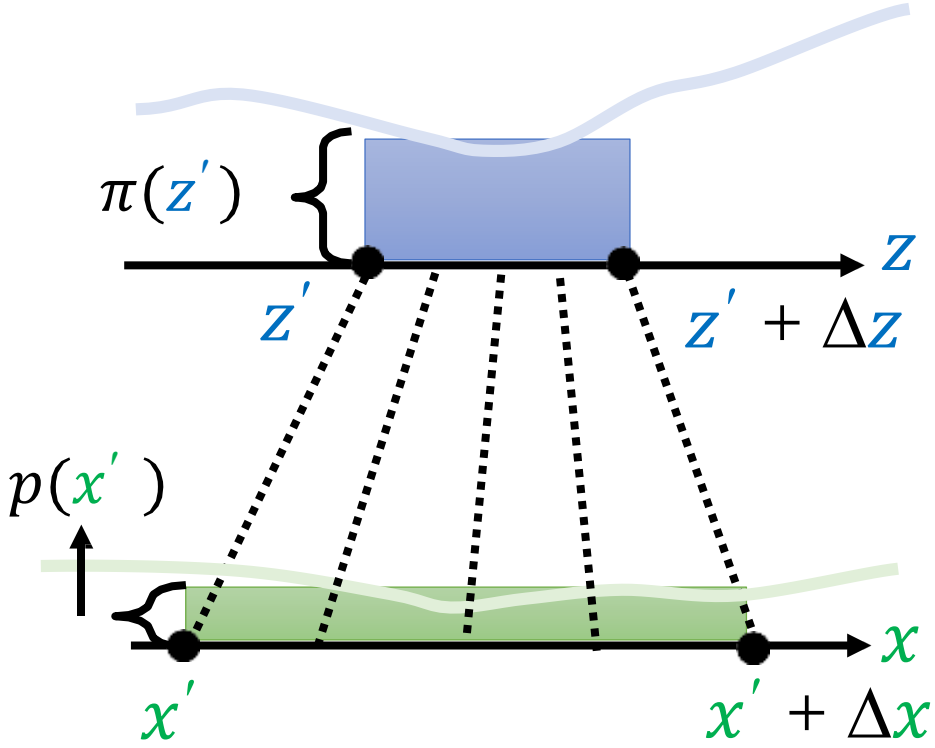


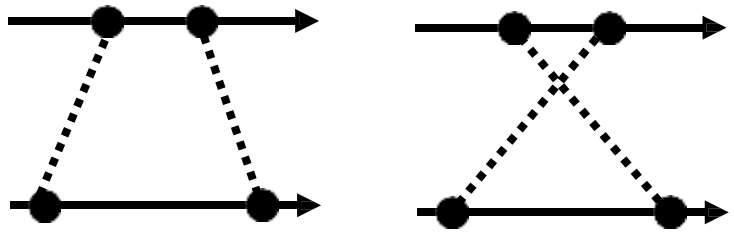
$$x = f(z)$$
$$= 2z + 1$$

$$p(x') = \frac{1}{2}\pi(z')$$

$$\int p(x)dx = 1$$

Change of Variable Theorem





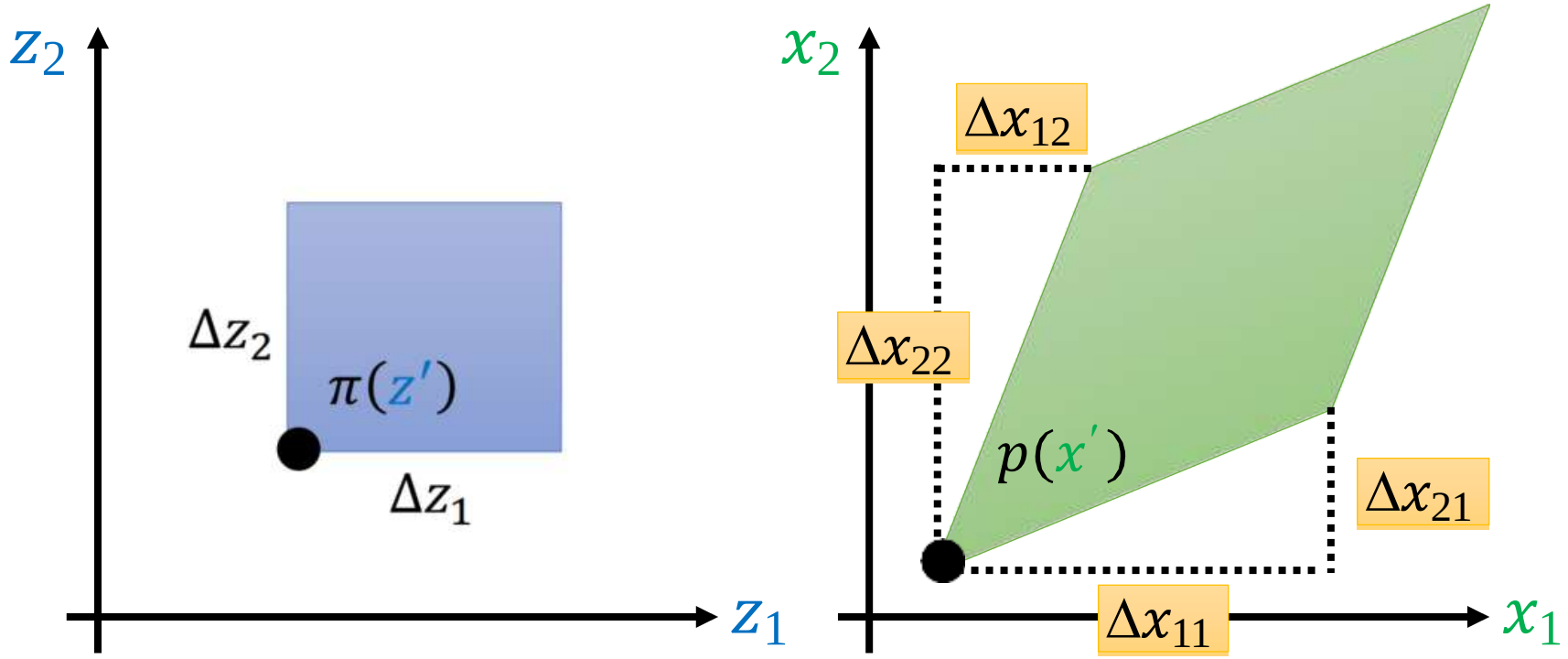
minus

$$p(x')\Delta x = \pi(z')\Delta z$$

$$p(x') = \pi(z')\frac{\Delta z}{\Delta x}$$

$$p(x') = \pi(z')\left|\frac{dz}{dx}\right|$$

Change of Variable Theorem-2D



$$p(x')\left|det\begin{bmatrix}\Delta x_{11} & \Delta x_{21}\\ \Delta x_{12} & \Delta x_{22}\end{bmatrix}\right| = \pi(z')\Delta z_1 \Delta z_2$$

$$p(x')\left|det\begin{bmatrix}\Delta x_{11} & \Delta x_{21}\\ \Delta x_{12} & \Delta x_{22}\end{bmatrix}\right| = \pi(z')\Delta z_1 \Delta z_2 \qquad x = f(z)$$

$$p(x')\left|\frac{1}{\Delta z_1 \Delta z_2} det\begin{bmatrix}\Delta x_{11} & \Delta x_{21}\\ \Delta x_{12} & \Delta x_{22}\end{bmatrix}\right| = \pi(z')$$

$$p(x')\left|det\begin{bmatrix}\Delta x_{11}/\Delta z_1 & \Delta x_{21}/\Delta z_1\\ \Delta x_{12}/\Delta z_2 & \Delta x_{22}/\Delta z_2\end{bmatrix}\right| = \pi(z')$$

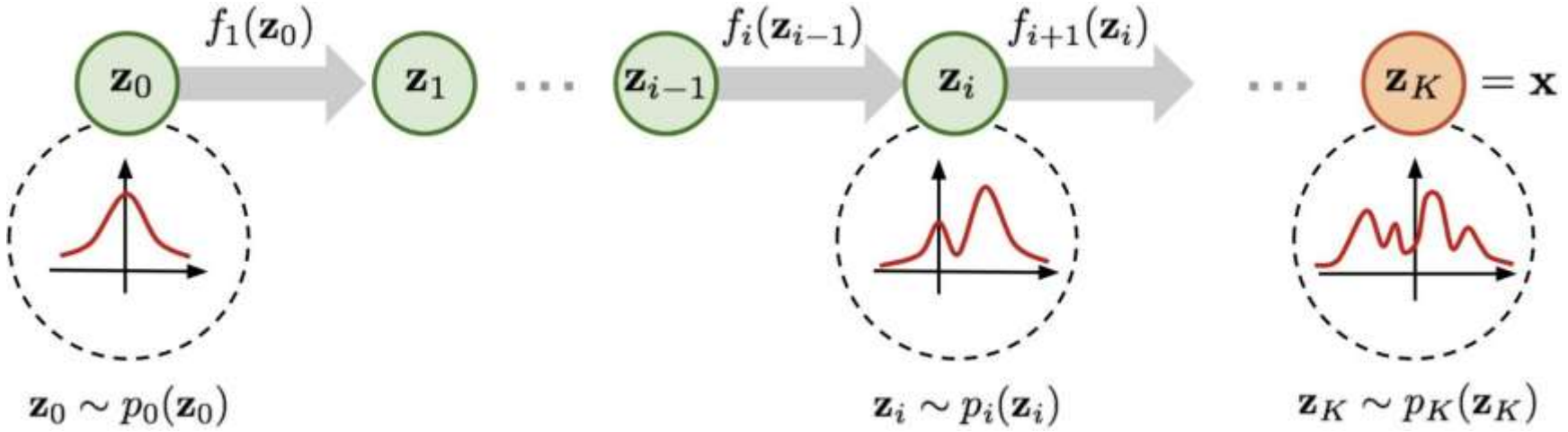
$$p(x')\left|det\begin{bmatrix}\partial x_1/\partial z_1 & \partial x_2/\partial z_1\\ \partial x_1/\partial z_2 & \partial x_2/\partial z_2\end{bmatrix}\right| = \pi(z')$$

$$p(x')\left|det\begin{bmatrix}\partial x_1/\partial z_1 & \partial x_1/\partial z_2\\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2\end{bmatrix}\right| = \pi(z')$$

$$p(x')\left|det(J_f)\right| = \pi(z')$$

$$p(x') = \pi(z')\left|det(J_{f^{-1}})\right|$$

$$p(x') = \pi(z')\left|\frac{1}{det(J_f)}\right|$$

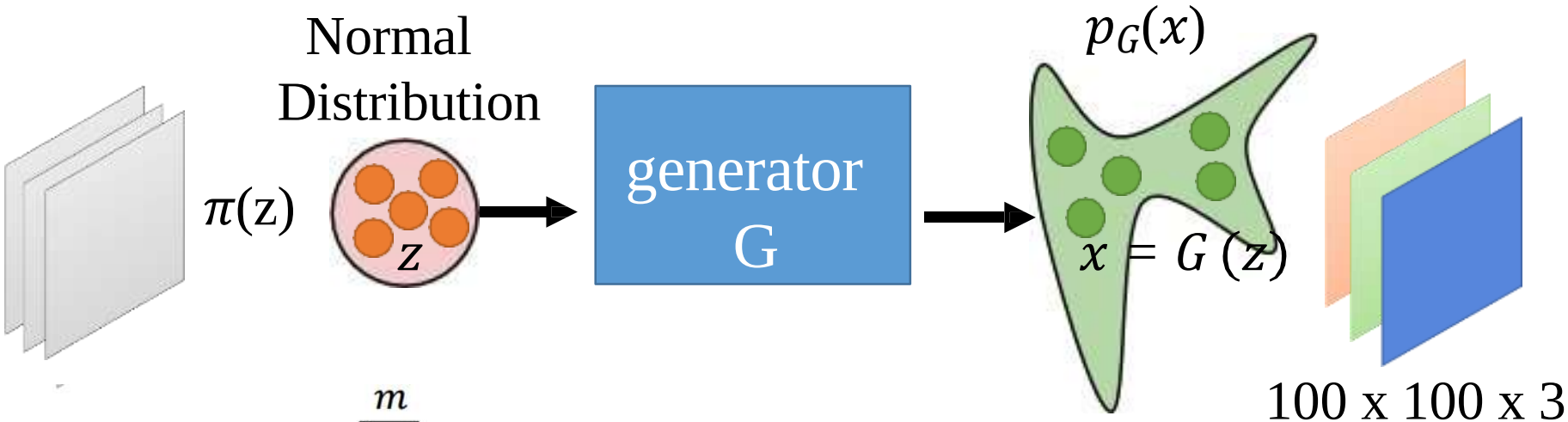


Formal Explanation

Flow-based Model

$$p(x')\left|det\left(J_f\right)\right| = \pi\left(z'\right)$$

$$p(x') = \pi\left(z'\right)\left|det\left(J_{f^{-1}}\right)\right|$$



$$G^* = arg\max_G \sum_{i=1}^{m} log p_G\left(x^i\right)$$

G has limitation

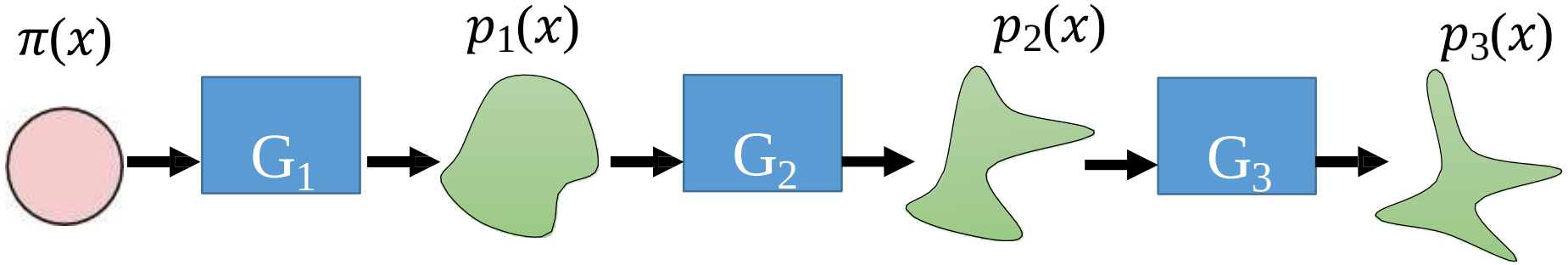
$$p_G\left(x^i\right) = \pi\left(z^i\right)\left|det\left(J_{G^{-1}}\right)\right|$$

$$z^i = G^{-1}\left(x^i\right)$$

- You can compute $det\left(J_G\right)$
- You know G^{-1}

$$log p_G\left(x^i\right) = log\pi\left(G^{-1}\left(x^i\right)\right) + log\left|det\left(J_{G^{-1}}\right)\right|$$

Multiple Generators



$$p_1(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right) \qquad z^i = G_1^{-1}\left(\cdots G_K^{-1}\left(x^i\right)\right)$$

$$p_2(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\left(\left|det\left(J_{G_2^{-1}}\right)\right|\right)$$

$$\vdots \quad \vdots$$

$$p_K(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\cdots\left(\left|det\left(J_{G_K^{-1}}\right)\right|\right)$$

$$logp_K(x^i) = log\pi(z^i) + \sum_{h=1}^{K} log\left|det\left(J_{G_K^{-1}}\right)\right| \quad \text{Maximize}$$

What you actually do?

$$log p_G(x^i) = log\pi\left(G^{-1}(x^i)\right) + log\left|det\left(J_{G^{-1}}\right)\right|$$

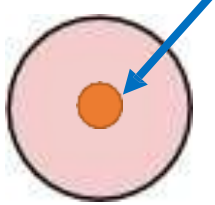
-inf

If z^i is always zero:

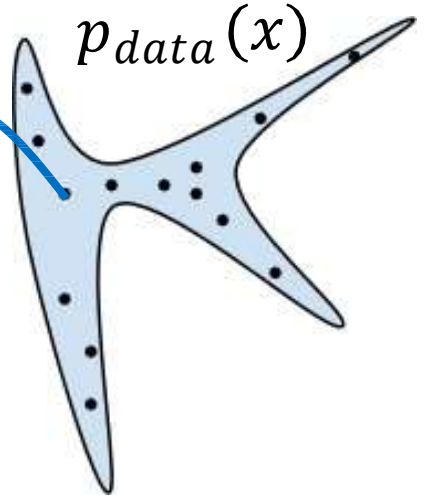
$J_{G^{-1}}$ would be zero matrix

$det(J_{G^{-1}}) = 0$

Make z^i become zero vector



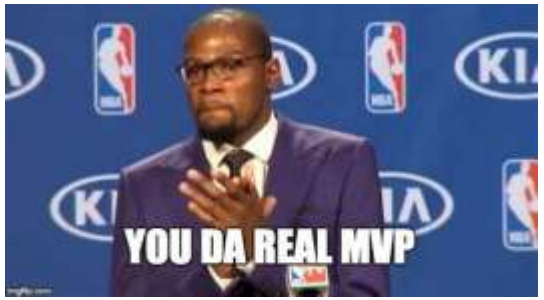
$z^i = G^{-1}(x^i)$ ← G-1 ← x^i

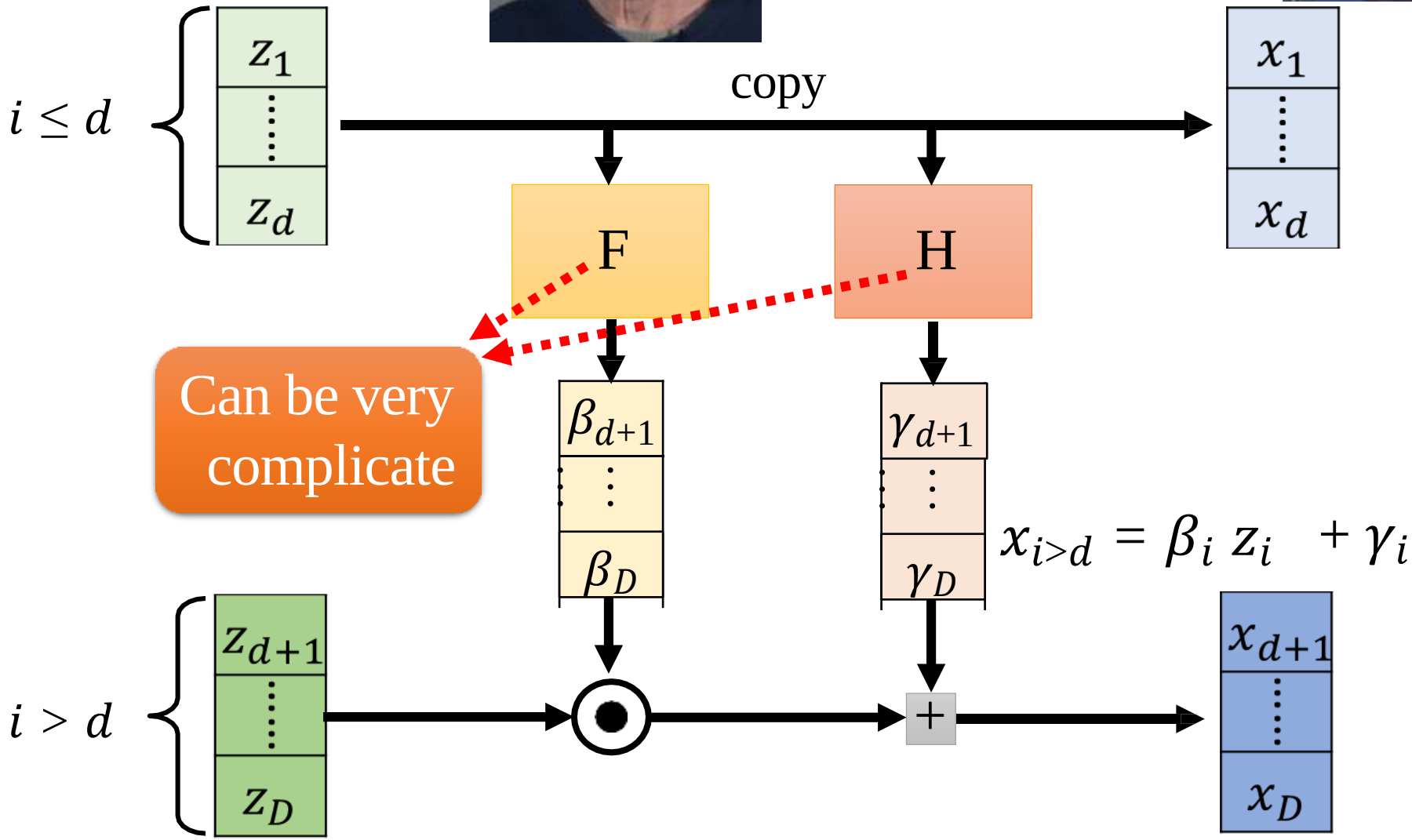


Actually, we train G^{-1} , but we use G for generation.

Coupling Layer

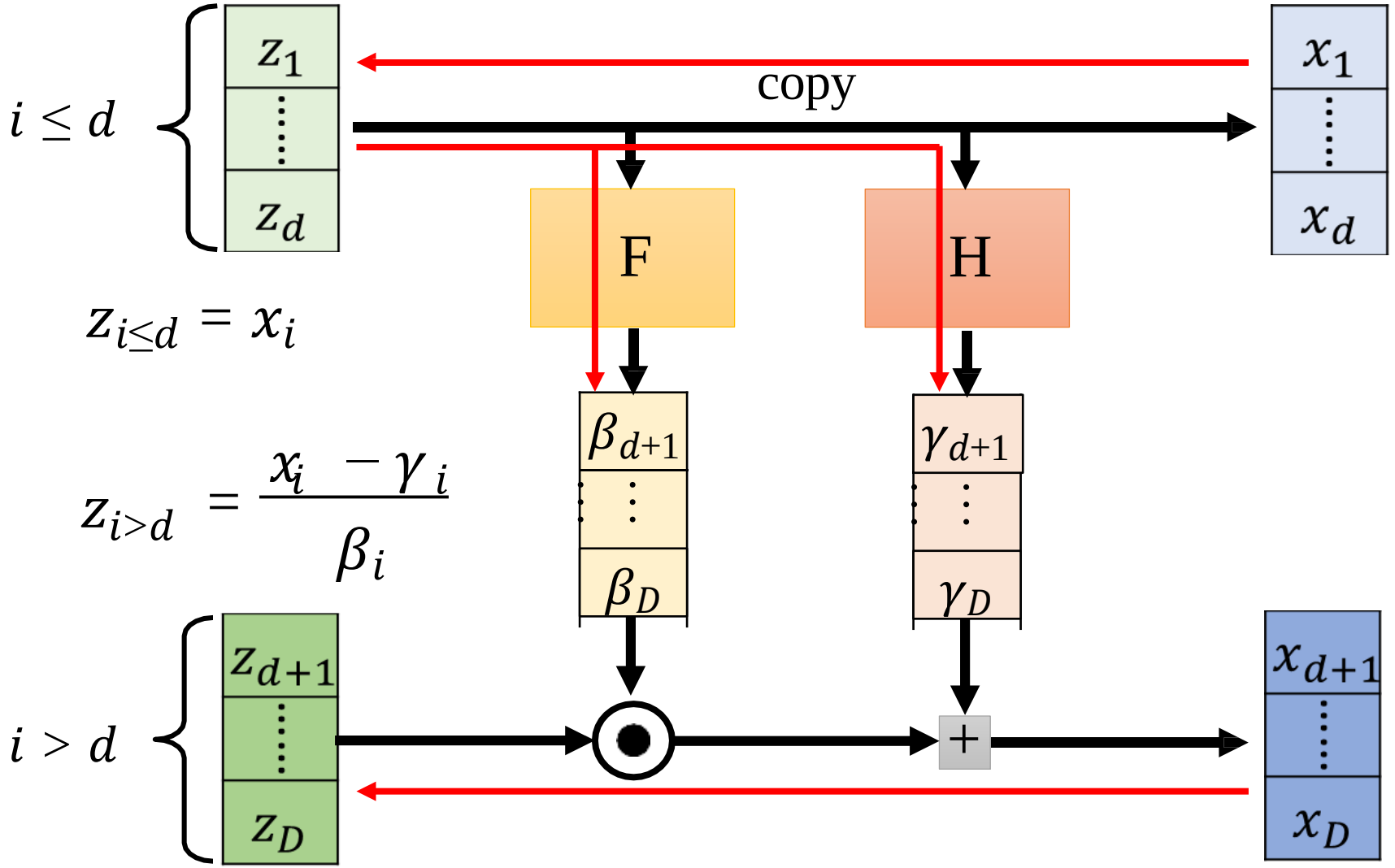
NICE
https://arxiv.org/abs/1410.8516
Real NVP
https://arxiv.org/abs/1605.08803



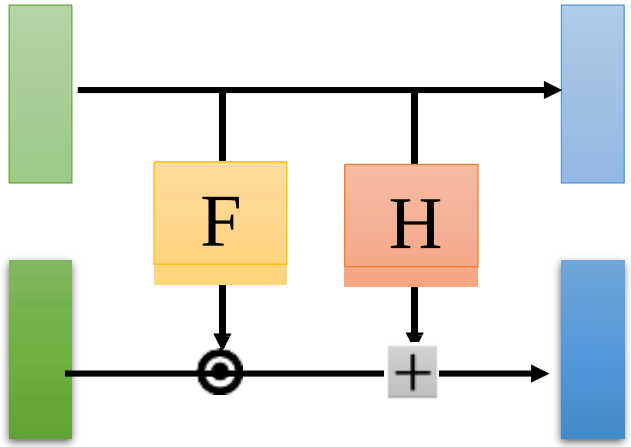


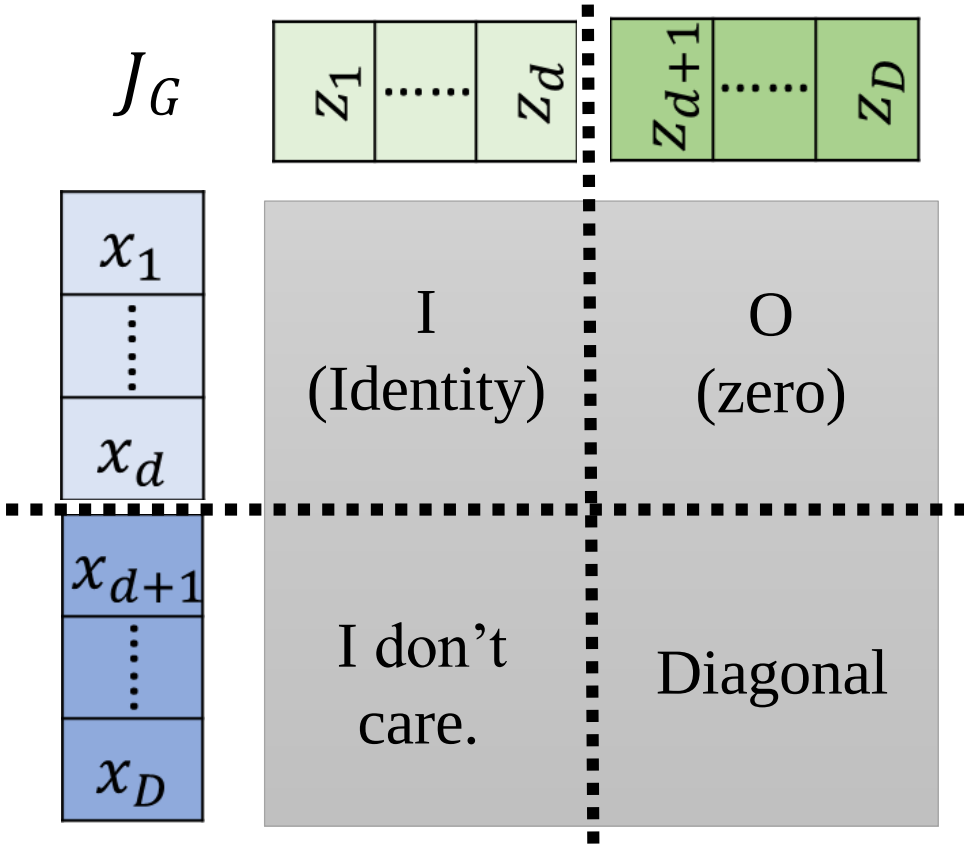
Coupling Layer

NICE
https://arxiv.org/abs/1410.8516
Real NVP
https://arxiv.org/abs/1605.08803



Coupling Layer





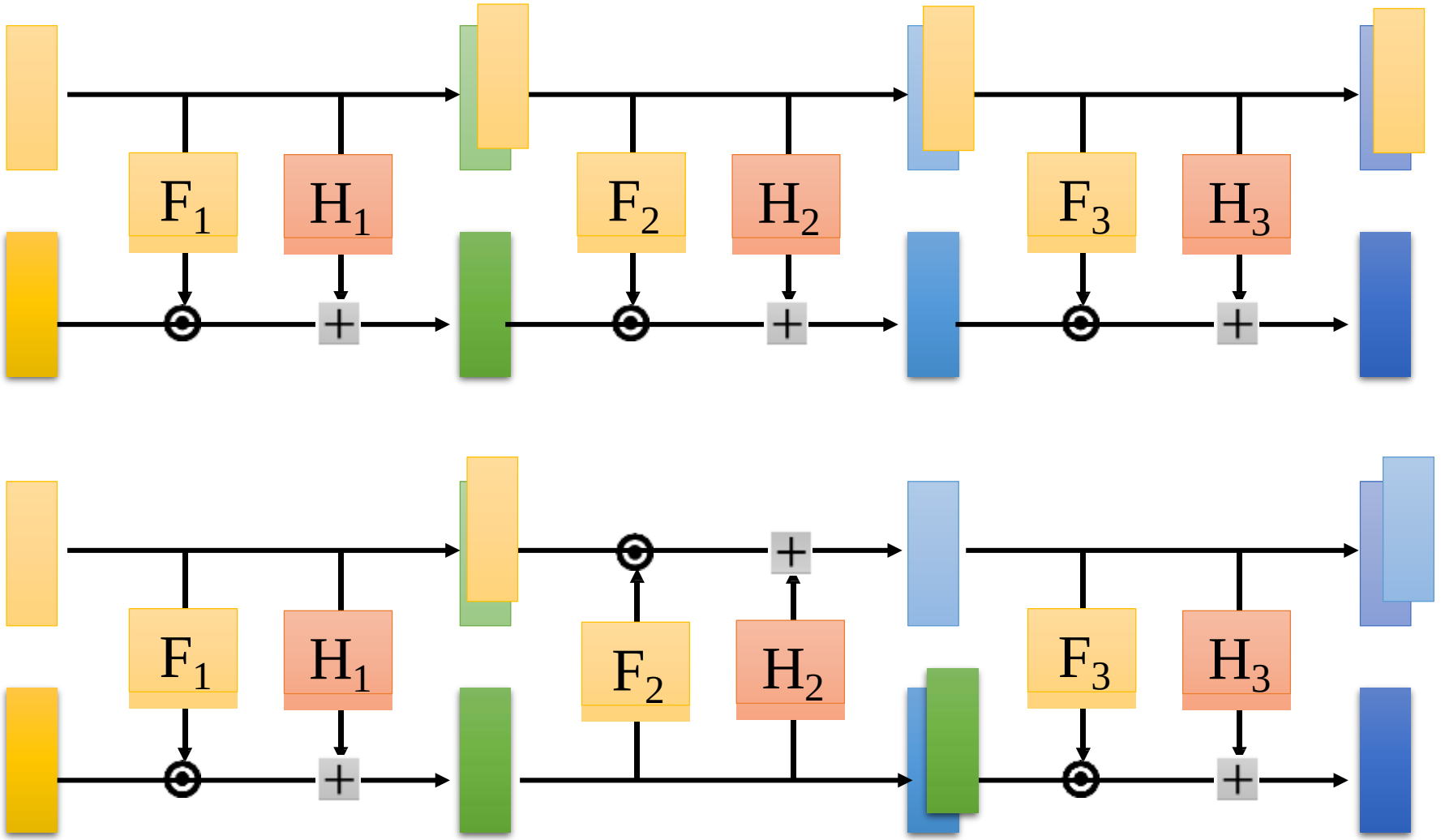
$$det(J_G)$$

$$= \frac{\partial x_{d+1}}{\partial z_{d+1}} \frac{\partial x_{d+2}}{\partial z_{d+2}} \cdots \frac{\partial x_D}{\partial z_D}$$

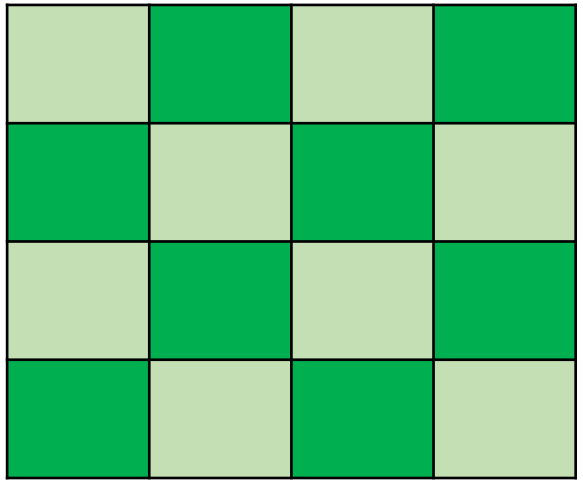
$$= \beta_{d+1}\beta_{d+2} \cdots \beta_D$$

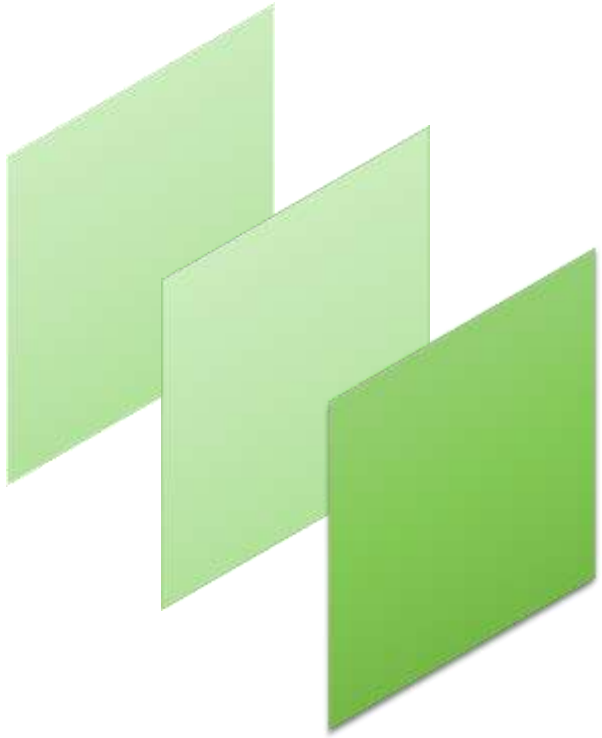
$$x_{i>d} = \beta_i \, z_i \quad + \gamma_i$$

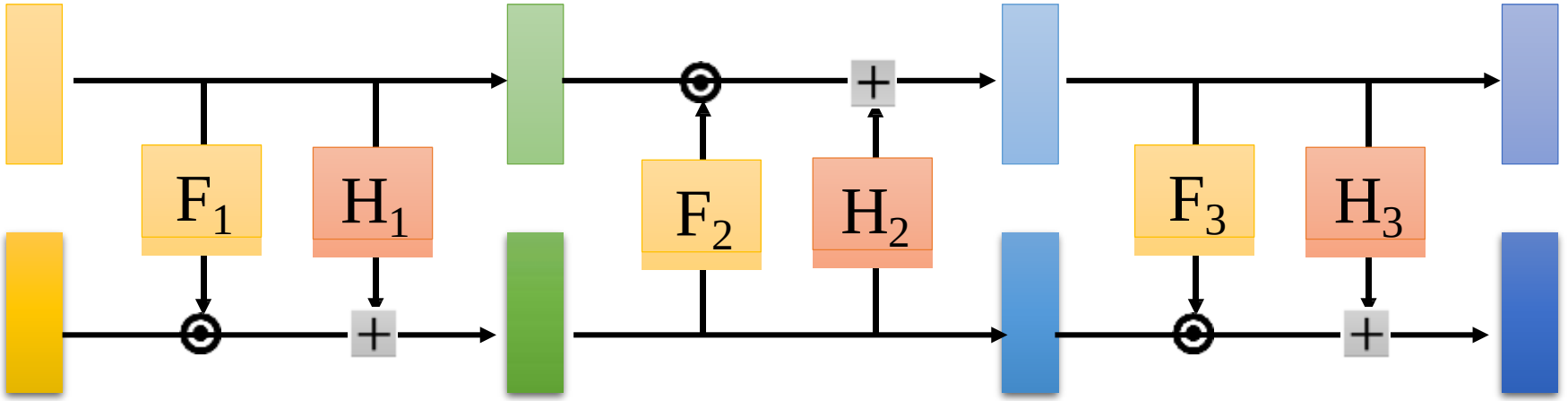
Coupling Layer - Stacking



Coupling Layer

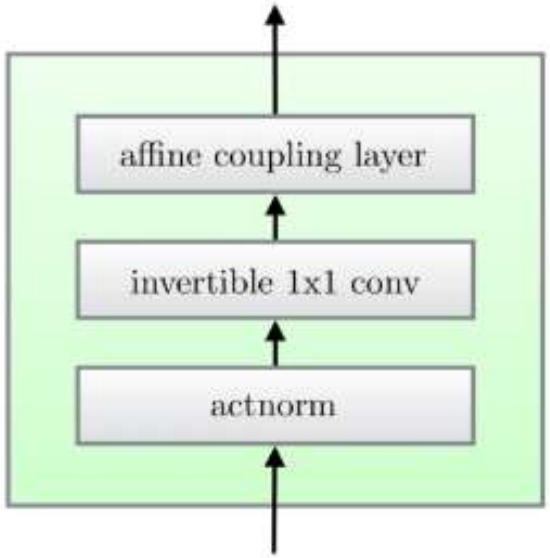






1x1 Convolution

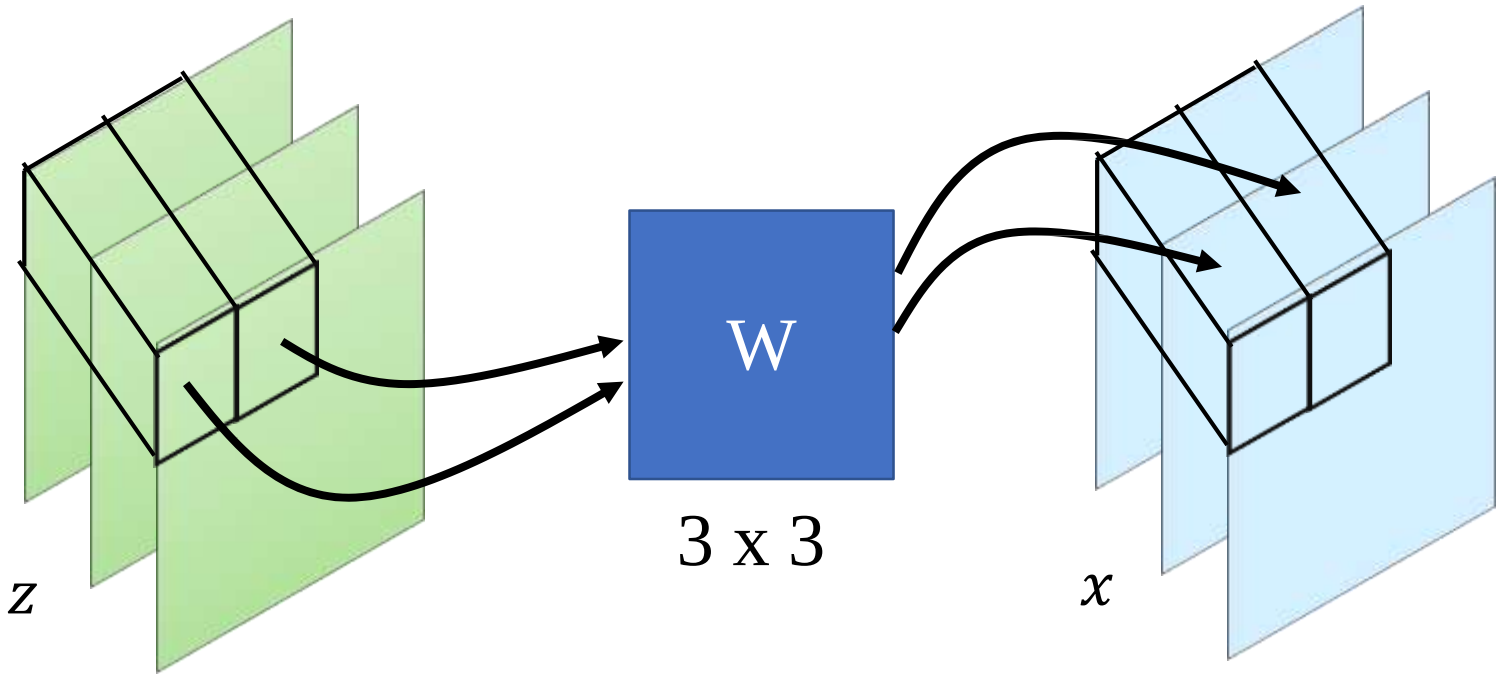
GLOW
https://arxiv.org/abs/1807.03039



Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i,j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i,j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i,j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \mathtt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \mathtt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \mathtt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \mathtt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \mathtt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \mathtt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\mathtt{sum}(\log(\lvert\mathbf{s}\rvert))$

1x1 Convolution

GLOW
https://arxiv.org/abs/1807.03039



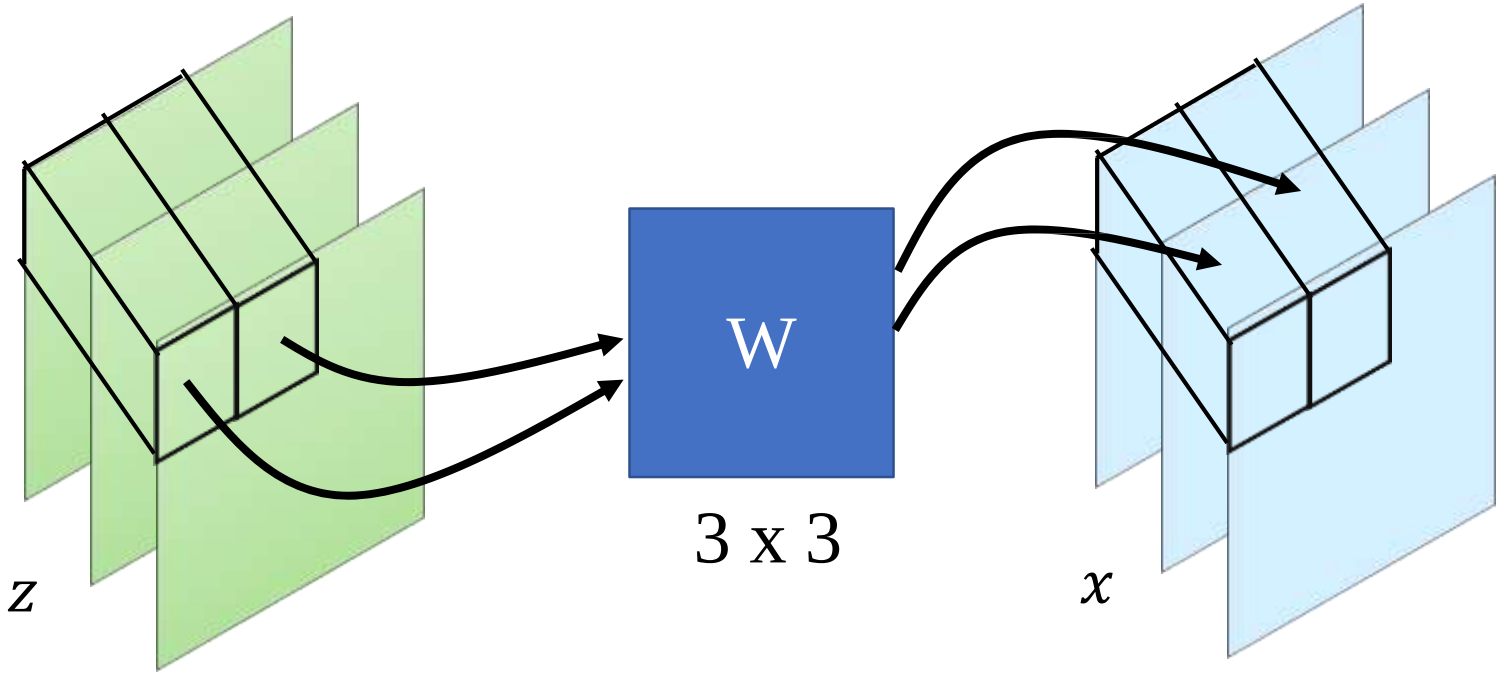
W can shuffle the **channels**.

If W is invertible, it is easy to compute W^{-1}.

3		0	0	1	1
1	=	1	0	0	2
2		0	1	0	3

1x1 Convolution

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$



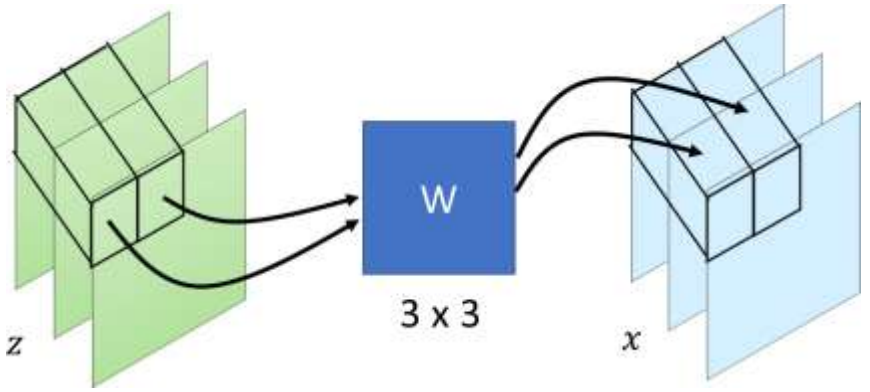
$$x = f(z) = Wz$$

$$J_f = \begin{bmatrix} \partial x_1 \mathrm{T} \partial z_1 & \partial x_1 \mathrm{T} \partial z_2 & \partial x_1 \mathrm{T} \partial z_3 \\ \partial x_2 \mathrm{T} \partial z_1 & \partial x_2 \mathrm{T} \partial z_2 & \partial x_2 \mathrm{T} \partial z_3 \\ \partial x_3 \mathrm{T} \partial z_1 & \partial x_3 \mathrm{T} \partial z_2 & \partial x_3 \mathrm{T} \partial z_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = W$$

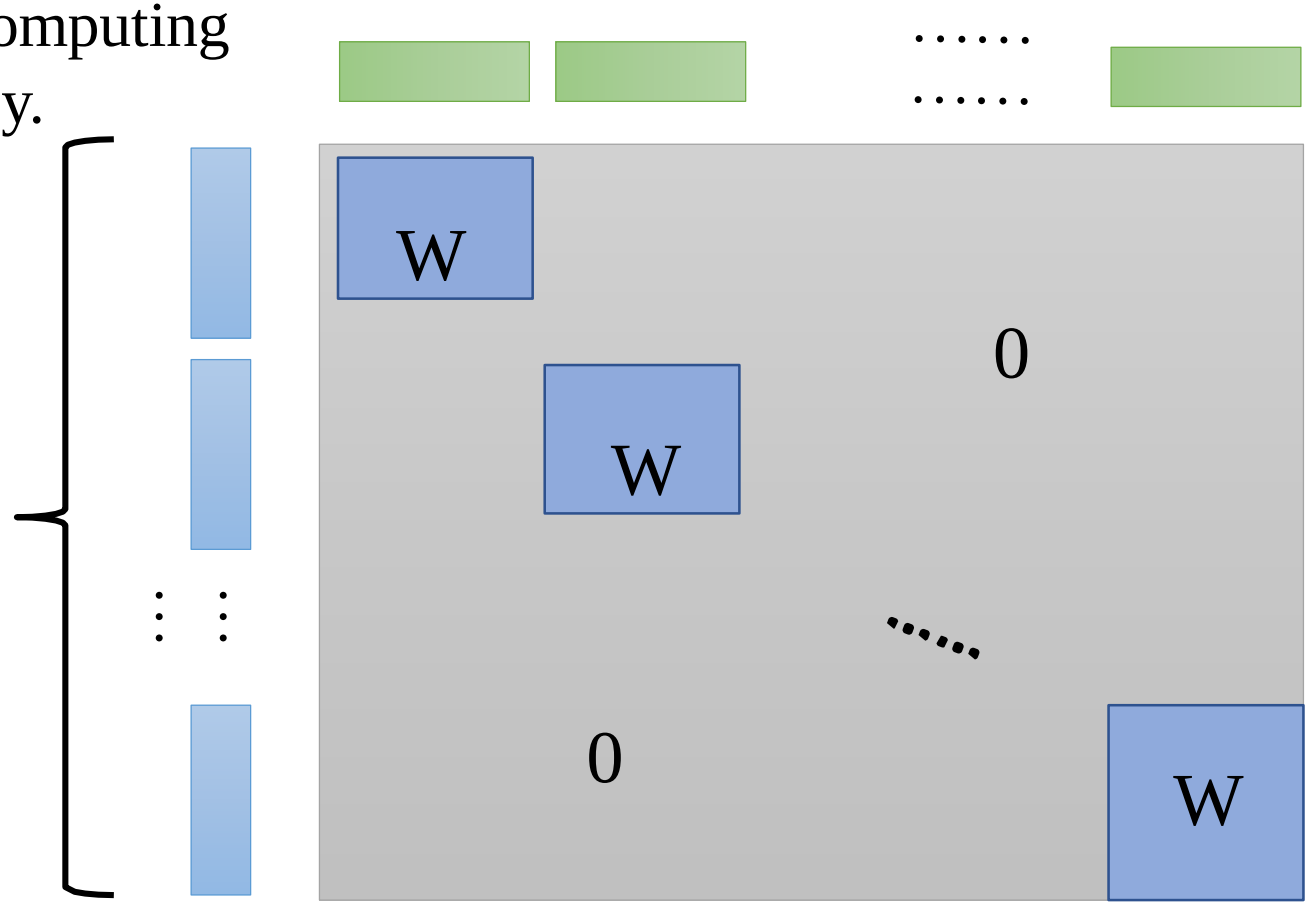
1x1 Convolution

$(det(W))^{d \times d}$

If W is 3x3, computing $det(W)$ is easy.



$d \times d$ positions



Demo of OpenAI

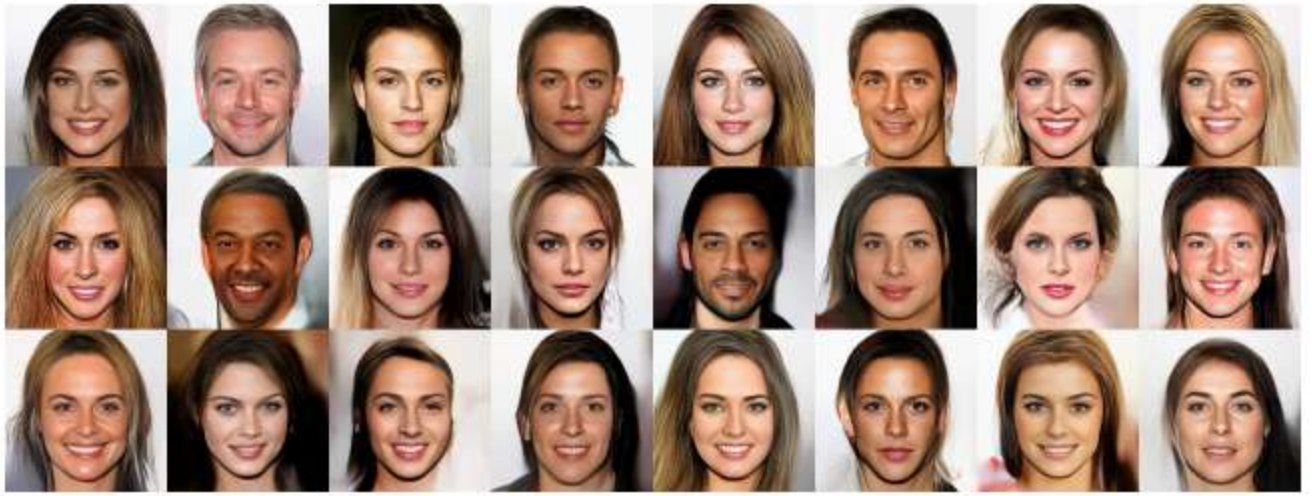
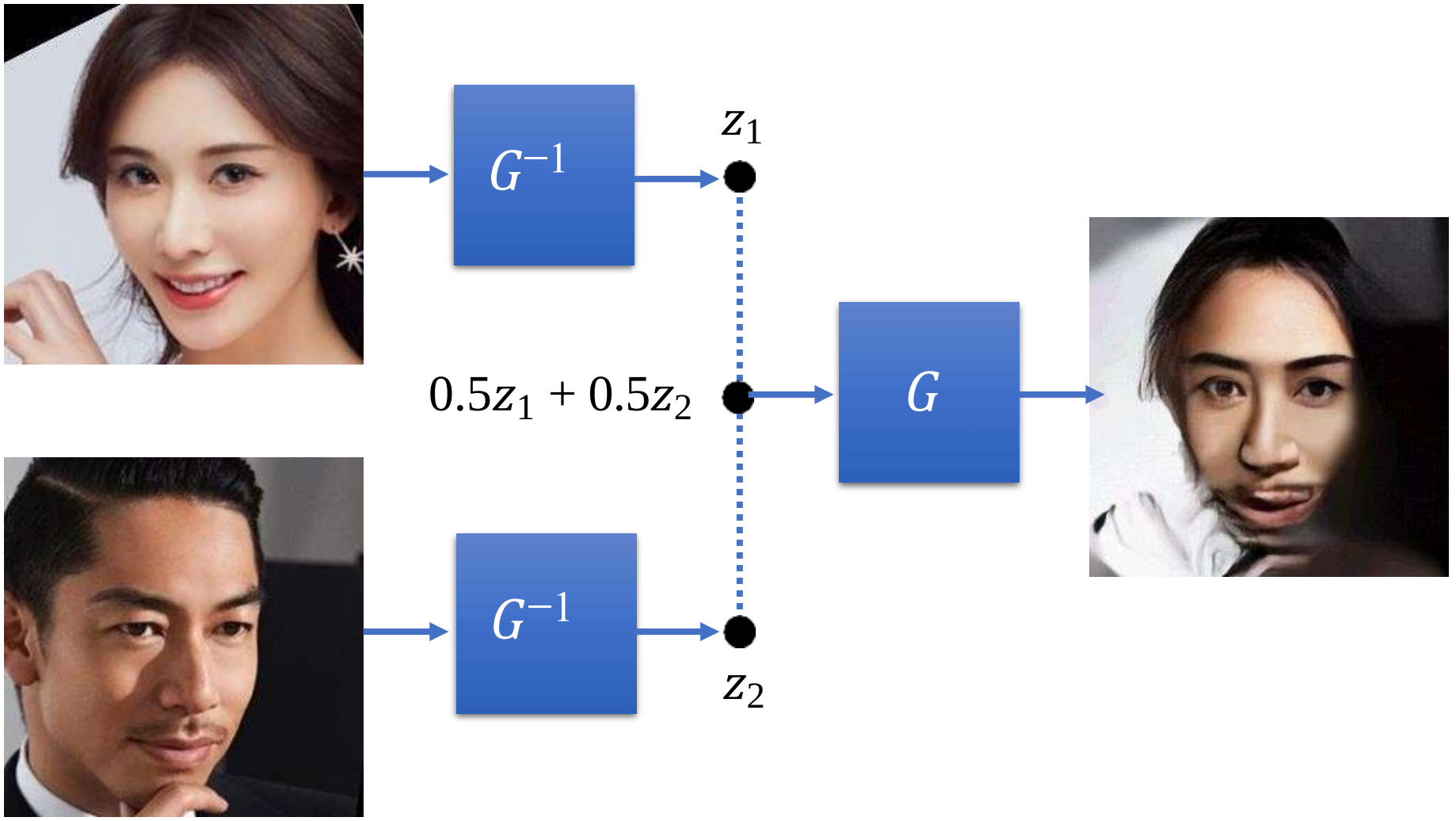


Figure 4: Random samples from the model, with temperature 0.7.

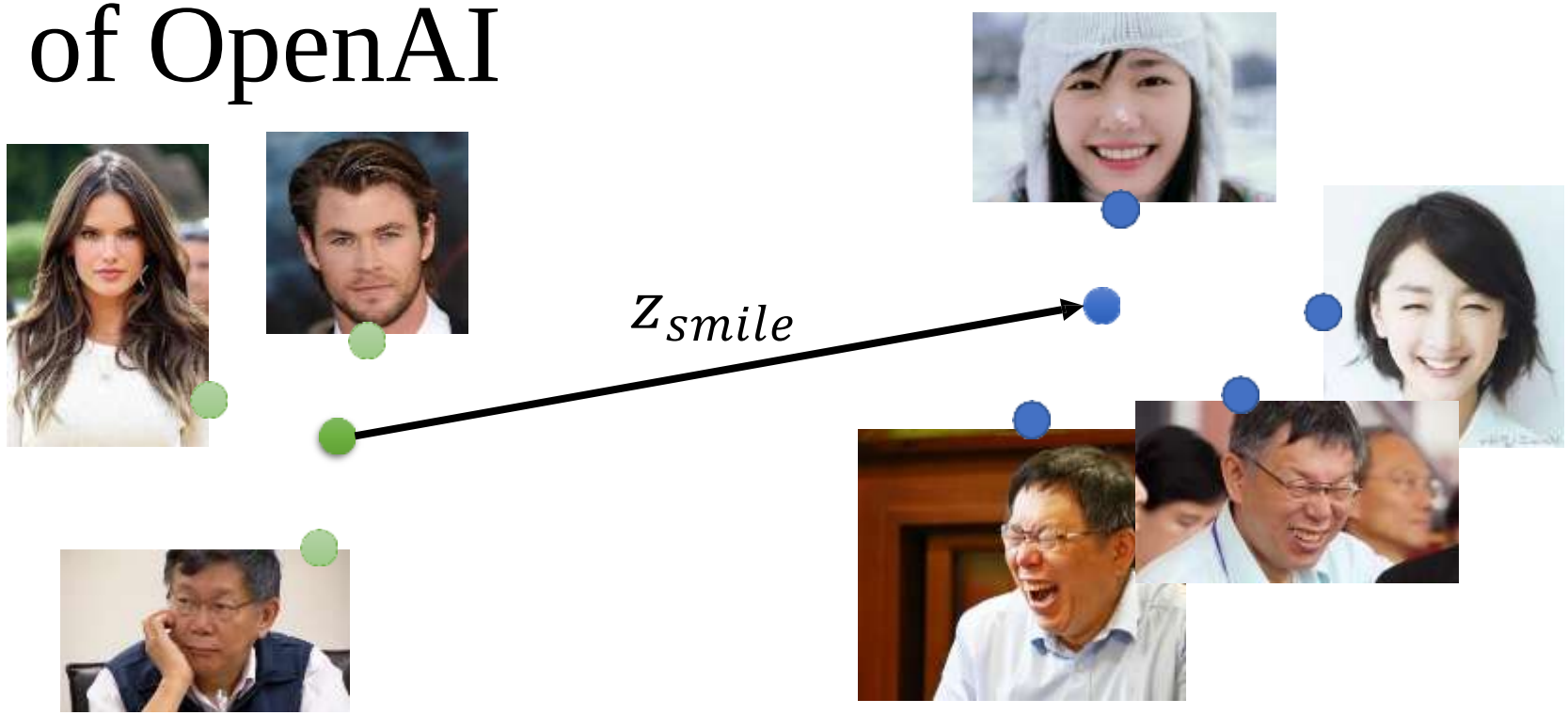
Figure 5: Linear interpolation in latent space between real images.

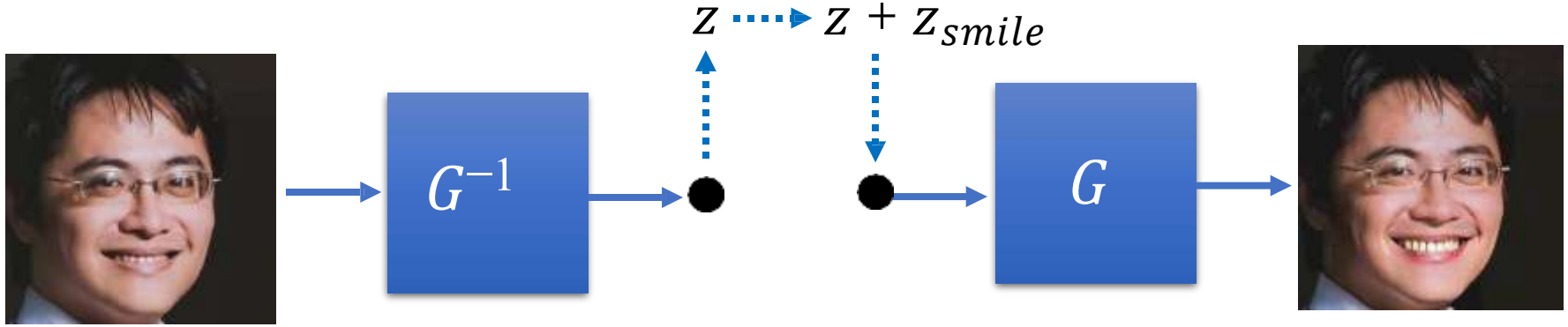
Source of image:
https://hd.stheadline.com/life/ent/realtime/1517562/

Demo of OpenAI



Demo of OpenAI





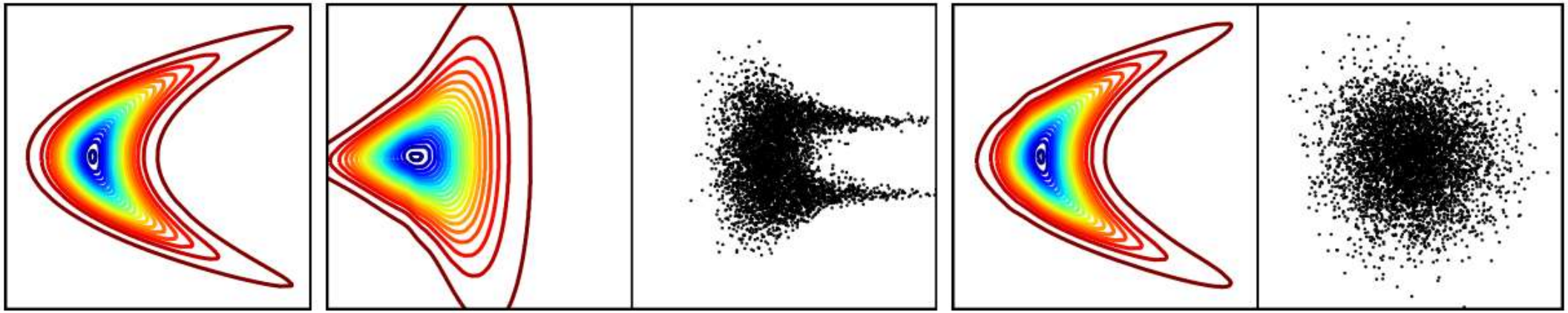
Masked Autoregressive Flow for Density Estimation

- Data generation, producing a new $\mathbf{x}$:

$$x_i \sim p(x_i|\mathbf{x}_{1:i-1}) = z_i \odot \sigma_i(\mathbf{x}_{1:i-1}) + \mu_i(\mathbf{x}_{1:i-1}), \text{ where } \mathbf{z} \sim \pi(\mathbf{z})$$

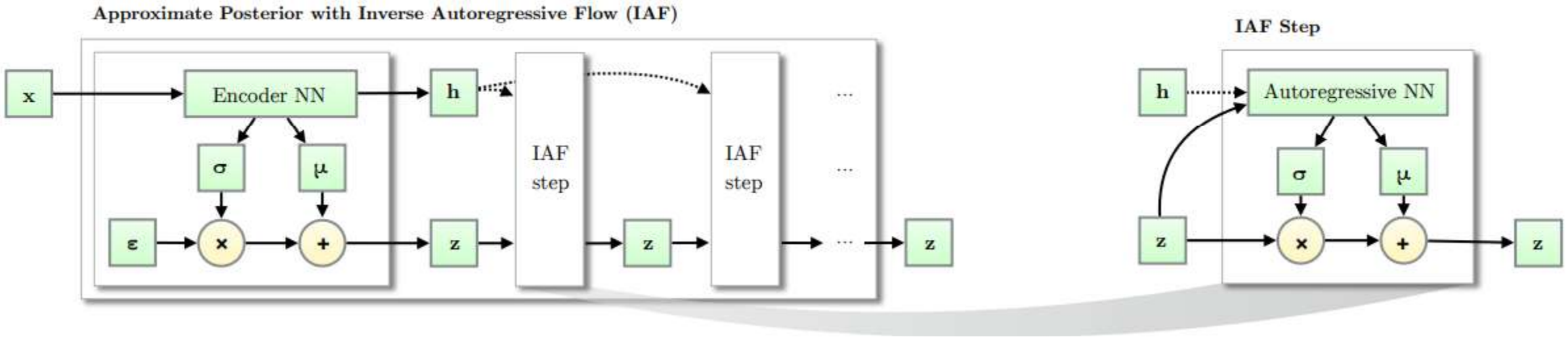
- Density estimation, given a known $\mathbf{x}$:

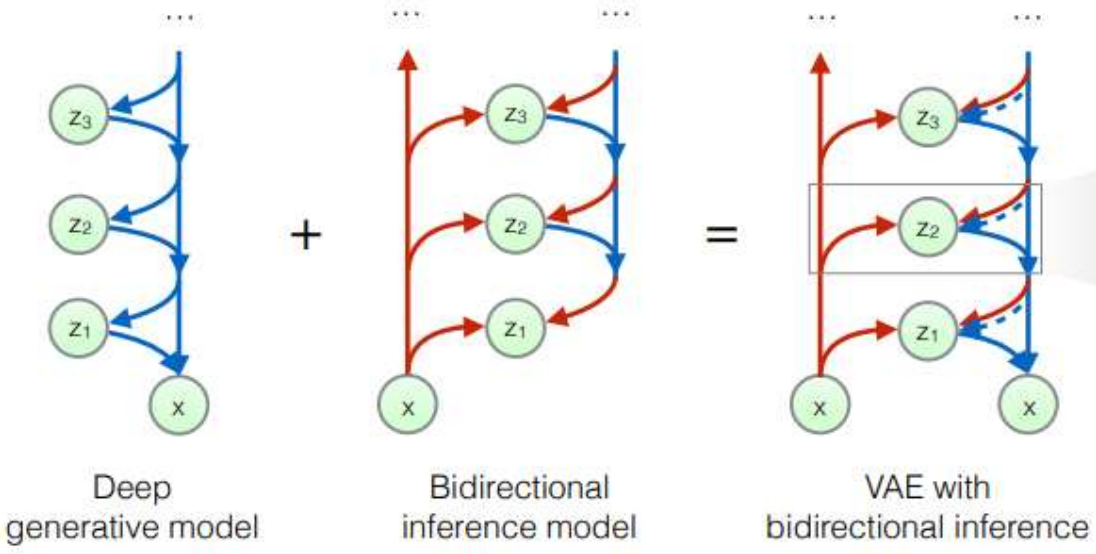
$$p(\mathbf{x}) = \prod_{i=1}^{D} p(x_i|\mathbf{x}_{1:i-1})$$



(a) Target density (b) MADE with Gaussian conditionals (c) MAF with 5 layers

Improved Variational Inference with Inverse Autoregressive Flow

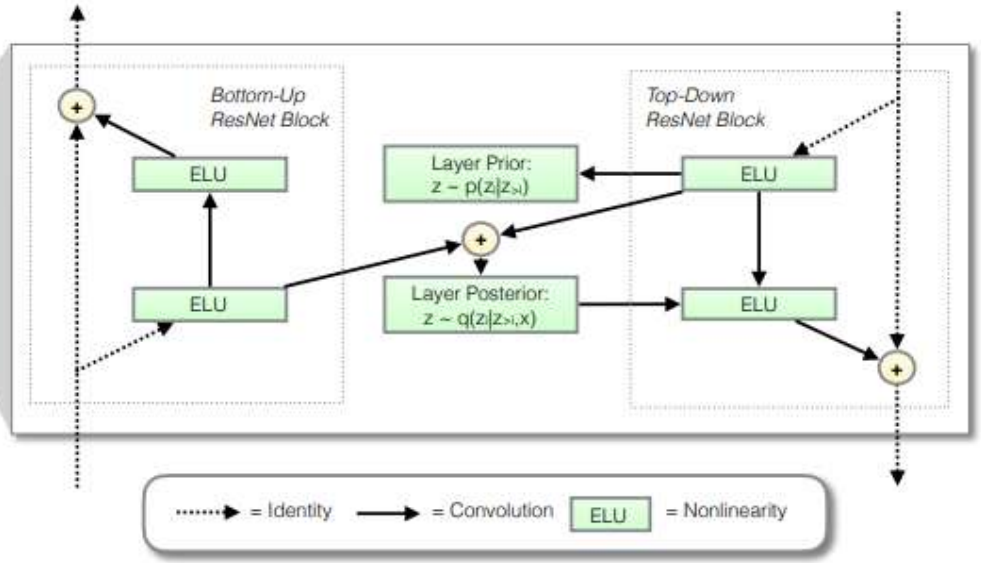




Deep generative model

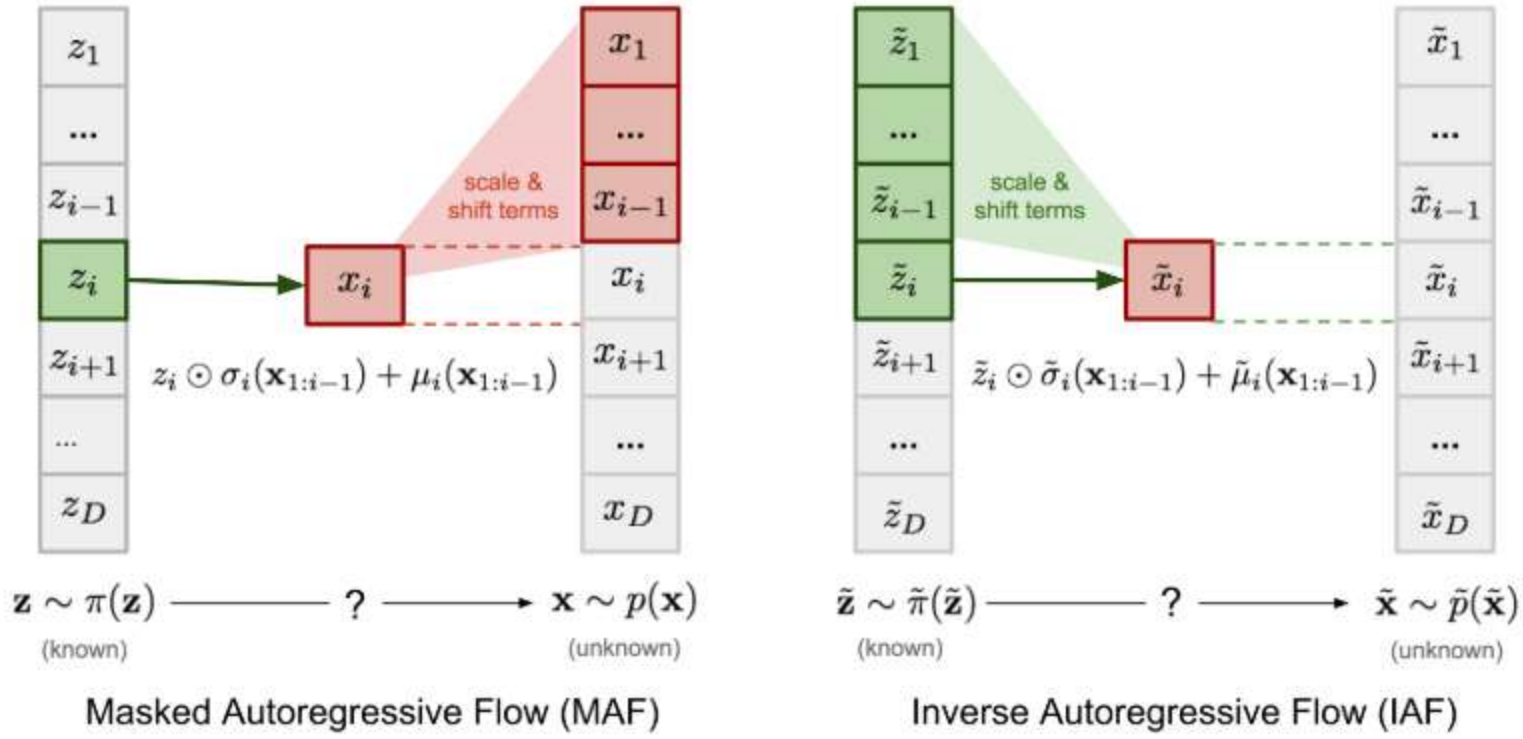
Bidirectional inference model

VAE with bidirectional inference

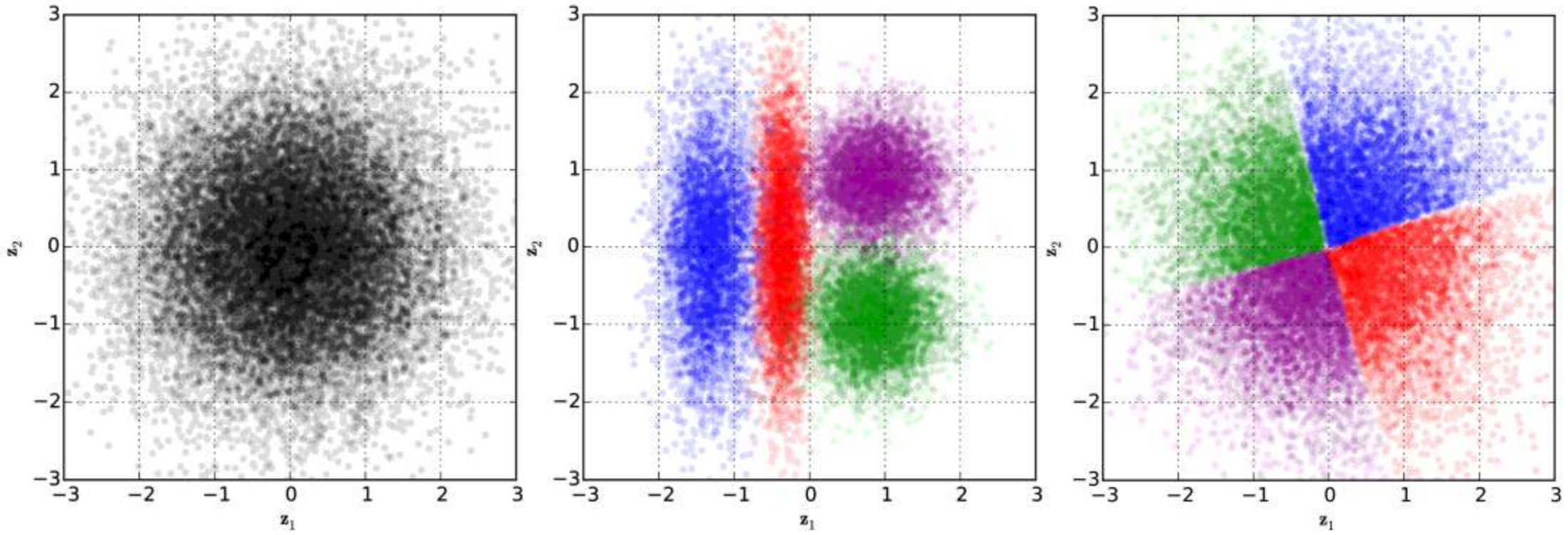


MAF VS IAF

$$z_i = \frac{x_i - \mu_i(\mathbf{x}_{1:i-1})}{\sigma_i(\mathbf{x}_{1:i-1})} = -\frac{\mu_i(\mathbf{x}_{1:i-1})}{\sigma_i(\mathbf{x}_{1:i-1})} + x_i \odot \frac{1}{\sigma_i(\mathbf{x}_{1:i-1})}$$

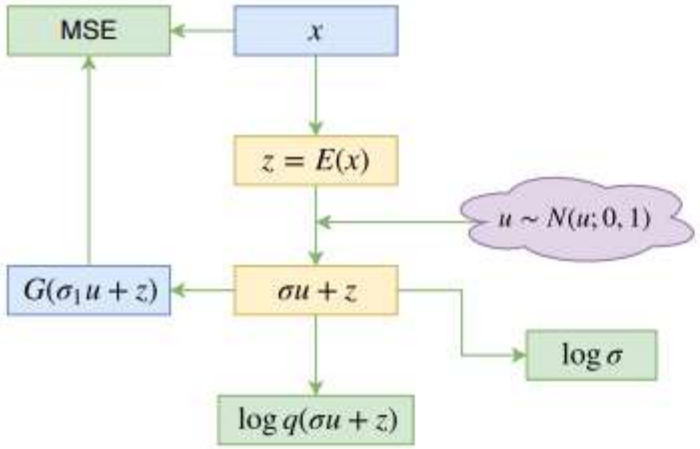


Improved Variational Inference with Inverse Autoregressive Flow

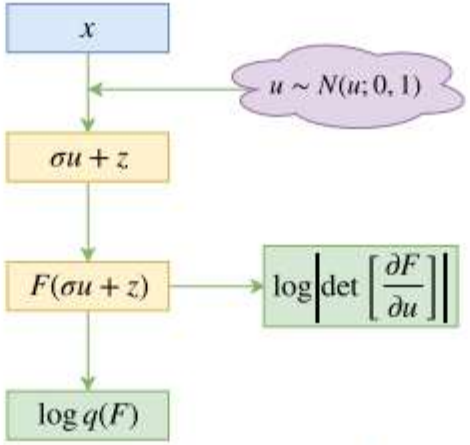


(a) Prior distribution (b) Posteriors in standard VAE (c) Posteriors in VAE with IAF

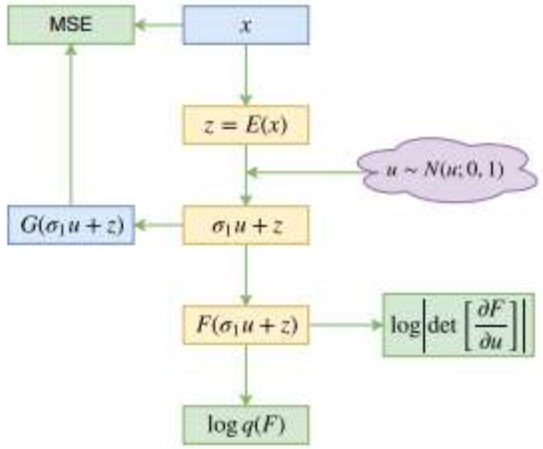
f-VAE



(a) Ordinary VAEs



(b) flow-based models

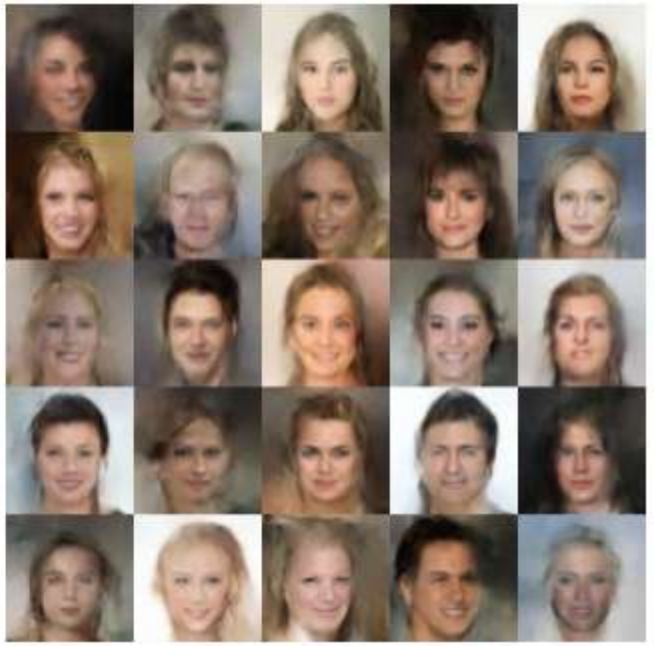


(c) f-VAEs (ours)

f-VAE

(a) Samples from VAEs

(b) Samples from Glow



(c) Samples from f-VAEs

Variational Inference with Normalizing Flows