Frontiers in Generative Modeling

March 11

Tailin Wu, Westlake University

Website: ai4s.lab.westlake.edu.cn/course

Image from: OpenAI

Class 1: A bird-eye view of deep learning

Tasks

- Classification/ regression
- Simulation
- Inverse design/ inverse problem
- Control/planning

×

Neural architecture

- Multilayer perceptron
- Graph Neural Networks
- Convolutional Neural Networks
- Transformers

×

Learning paradigm

- Supervised learning
- Generative modeling
- Foundation models
- Reinforcement learning
- Evolutionary and multi-objective optimization

Application (AI & Science)

- Robotics
- Games (e.g., Go, atari)
- Autonomous Driving
- PDEs
- Life science
- Materials science

Class 2: Deep learning fundamentals

1. Principle 1: Model a hard transformation by composing simple transformations:
   - Multilayer Perceptron (MLP)
   - Backpropagation

2. Principle 2: Directly optimizing the final objective using maximum likelihood and information theory:
   - Maximum likelihood: MSE, uncertainty estimation
   - Information: cross-entropy, Information Bottleneck

3. Optimization
   - Adam: combining momentum and per-dimension magnitude
   - SAM (sharpness-aware minimization): $\max_{\boldsymbol{\epsilon} \in N_{\boldsymbol{\theta}}} \ell(\boldsymbol{\theta} + \boldsymbol{\epsilon})$ finds flat and robust minima
   - Federative learning: improves the data privacy by only sharing client models

Generative modeling

Images and shapes generated by diffusion models:

By DallE 2

By MeshDiffusion [1]

[1] Liu, Zhen, et al. "Meshdiffusion: Score-based generative 3d mesh modeling." *ICLR 2023*

Generative modeling

Robotic policy by diffusion models [1]

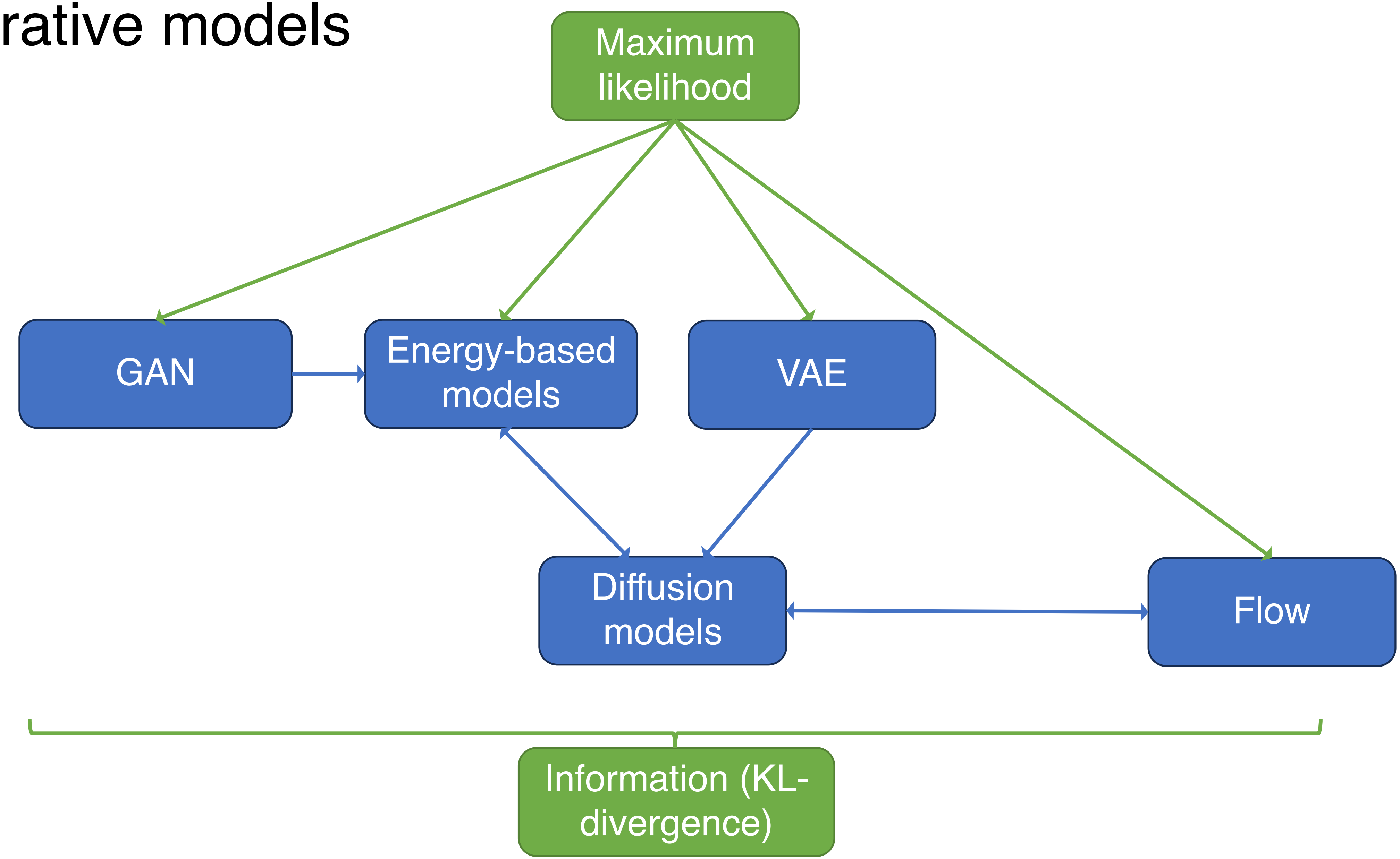
Text to video generation by Sora [2]

[1] Fu, Zipeng, Tony Z. Zhao, and Chelsea Finn. "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation." *arXiv preprint arXiv:2401.02117* (2024).
[2] OpenAI team. “Video generation models as world simulators”, 2024

Outline

- Generative models
  - VAE
  - GAN
  - Energy-based models
  - Diffusion models
  - Flows
- Application of diffusion models
  - Image, video, and shape generation
  - Simulation
  - Inverse design/inverse problem
  - Control/planning

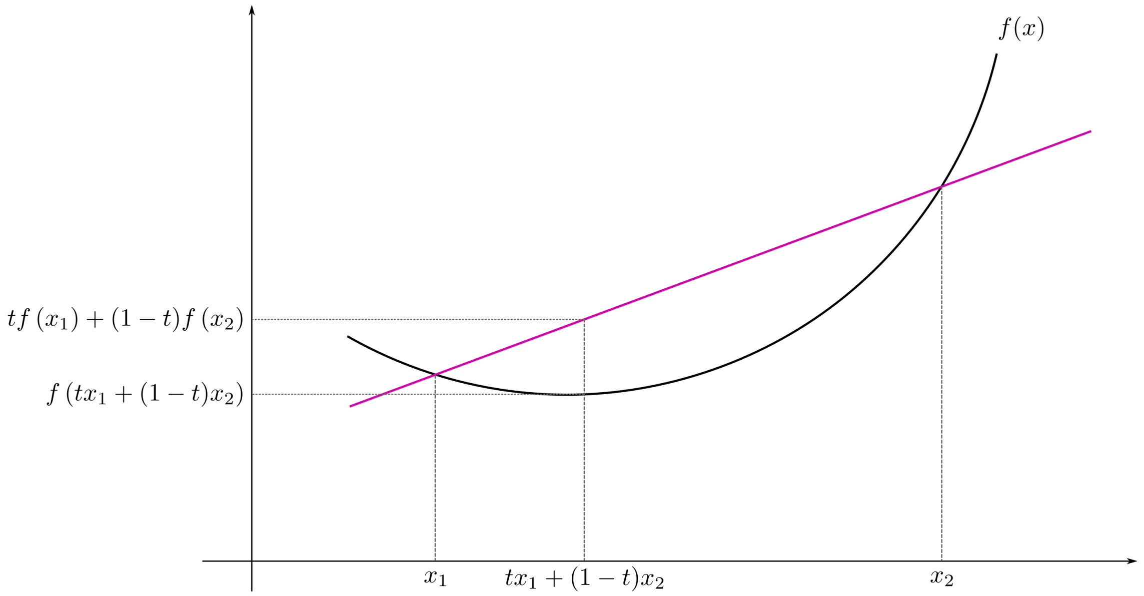
Generative models



Preliminary: Jensen's inequality

For a convex function $f(x)$, given $t \in [0,1]$, we have:

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$



More generally, let X be a random variable, we have

f convex: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$

f concave: $f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]$

Equality holds when $f(x)$ is a linear function, or the variable X is a constant

Preliminary: KL divergence

KL divergence measures how one probability distribution P is different from a second, reference probability distribution Q.

$$\mathbb{D}_{KL}(P||Q) = \mathbb{E}_{x \sim P(x)}\left[\log \frac{P(x)}{Q(x)}\right]$$

P: data; Q: a model (typically)

$\mathbb{D}_{KL}(P||Q)$: The average difference of the number of bits required for encoding samples of P using a code optimized for Q rather than one optimized for P

Properties:

1. $\mathbb{D}_{KL}(P||Q) \geq 0$
2. When $P(x)$ and $Q(x)$ are identical, $\mathbb{D}_{KL}(P||Q) = 0$

Preliminary: Proving non-negative of KL divergence

Proving $\mathbb{D}_{KL}(P||Q) \geq 0$:

$$\mathbb{D}_{KL}(P||Q)$$
$$= \mathbb{E}_{x\sim P(x)}\left[-\log \frac{Q(\boldsymbol{x})}{P(\boldsymbol{x})}\right]$$
$$\geq -\log \mathbb{E}_{x\sim P(x)}\left[\frac{Q(\boldsymbol{x})}{P(\boldsymbol{x})}\right]$$
$$= -\log \int P(x)\frac{Q(\boldsymbol{x})}{P(\boldsymbol{x})}dx$$
$$= 0$$

f convex: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$

$-\log(x)$ is a convex function

Equality holds when $\frac{Q(\boldsymbol{x})}{P(\boldsymbol{x})}$ is a **constant**, i.e., $P(\boldsymbol{x}) \equiv Q(\boldsymbol{x})$.

Generative models

What makes a generative model?

- image
- video
- graph
- time series
- natural language
- …

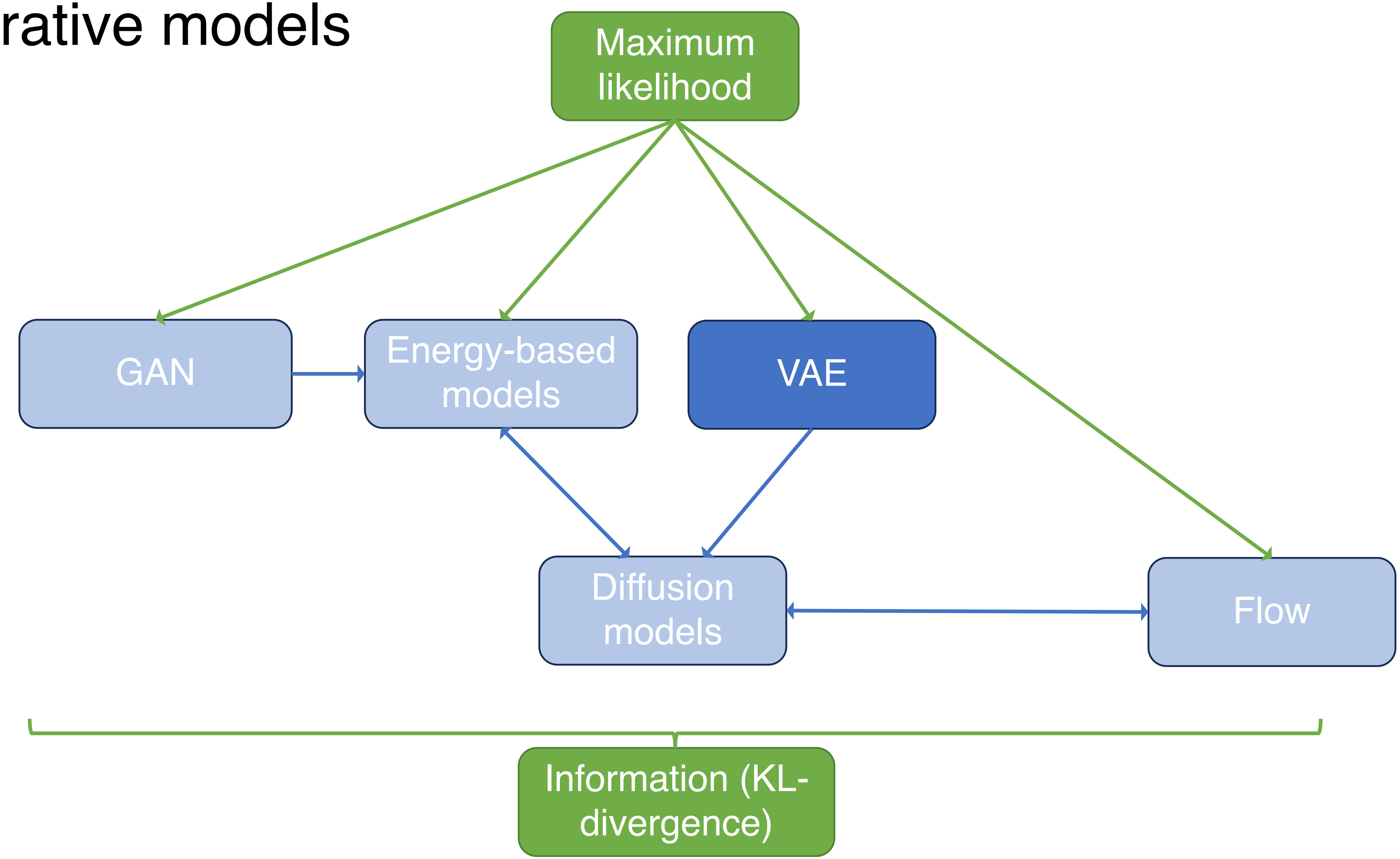
input X $\Longrightarrow$ $p_\theta(X)$?

Probability model

Given many examples of the input X, learn a probability model $p_\theta(X)$ that can **sample** new instances of X that conform to the data distribution.

- Sampling (required)
- Computing the probability of a sample (optional)

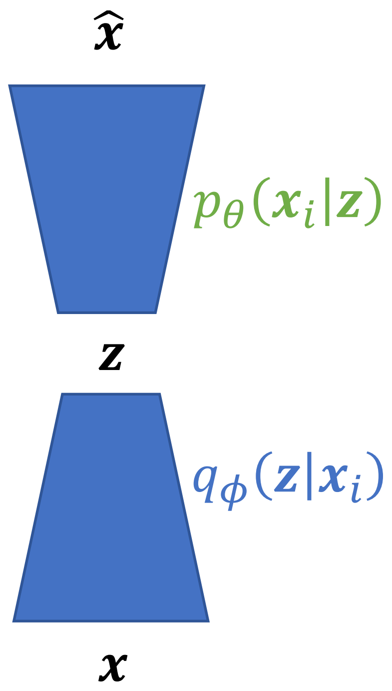
Generative models



Generative model 1: Variational autoencoder

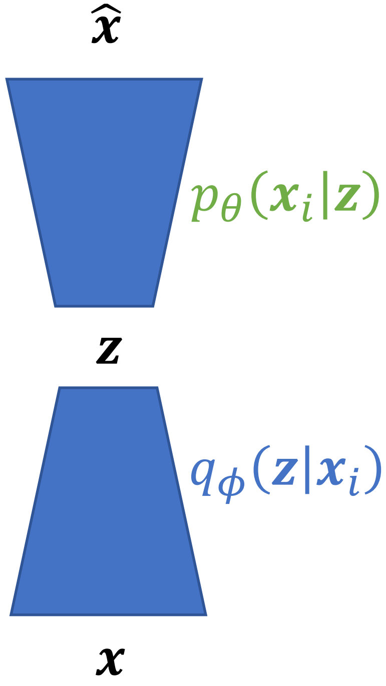
Given $\{\boldsymbol{x}_i\}_{i=1}^{N}$, learn a probability model $p_\theta(\boldsymbol{x})$ that maximizes the likelihood of data

$$
\begin{aligned}
&\log p_\theta(\boldsymbol{x}_i) \\
&\geq \mathcal{L}(\phi, \theta; \boldsymbol{x}_i) \coloneqq \log p_\theta(\boldsymbol{x}_i) - \underbrace{\mathbb{D}_{KL}\left(q_\phi(\boldsymbol{z}|\boldsymbol{x}_i) || p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)\right)}_{\geq 0} \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}[\log p_\theta(\boldsymbol{x}_i)] - \mathbb{E}_{q_\phi(z|x_i)}\left[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}{p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\left[\log \frac{p_\theta(\boldsymbol{x}_i) p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\left[\log \frac{p_\theta(\boldsymbol{z}) p_\theta(\boldsymbol{x}_i|\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}[\log p_\theta(\boldsymbol{x}_i|\boldsymbol{z})] - \mathbb{D}_{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x}_i) || p_\theta(\boldsymbol{z})]
\end{aligned}
$$



$\mathcal{L}(\phi, \theta; \boldsymbol{x}_i)$ is called Evidence Lower BOund (ELBO)

Variational autoencoder: From Jensen's inequality



Derivation using Jensen's inequality

$$\log p_\theta(\boldsymbol{x}_i)$$

$$= \log \int q_\phi(\boldsymbol{z}|\boldsymbol{x}_i) \frac{p_\theta(\boldsymbol{x}_i, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)} dz$$

$$= \log \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)} \left[\frac{p_\theta(\boldsymbol{x}_i, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\right]$$

$$\geq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)} \left[\log \frac{p_\theta(\boldsymbol{x}_i|\boldsymbol{z}) p_\theta(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}\right]$$

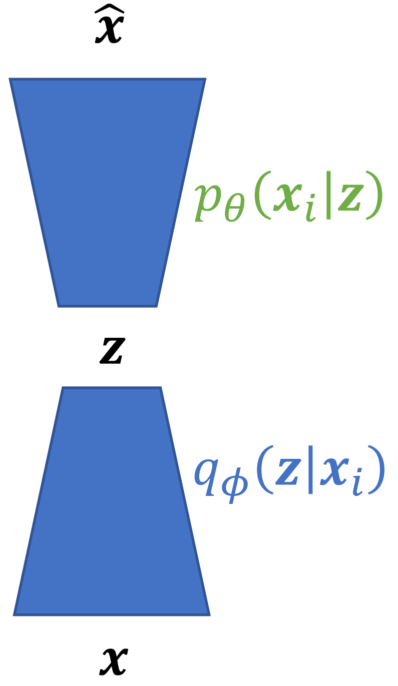
$$= \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}[\log p_\theta(\boldsymbol{x}_i|\boldsymbol{z})] - \mathbb{D}_{KL}\left[q_\phi(\boldsymbol{z}|\boldsymbol{x}_i) || p_\theta(\boldsymbol{z})\right]$$

$\boxed{f \text{ concave:} \quad f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]}$

Equality holds when $\frac{p_\theta(\boldsymbol{x}_i, \boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)} = p_\theta(\boldsymbol{x}_i) \frac{p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)}{q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}$ is a **constant** w.r.t. z, i.e., $q_\phi(\boldsymbol{z}|\boldsymbol{x}_i) \equiv p_\theta(\boldsymbol{z}|\boldsymbol{x}_i)$

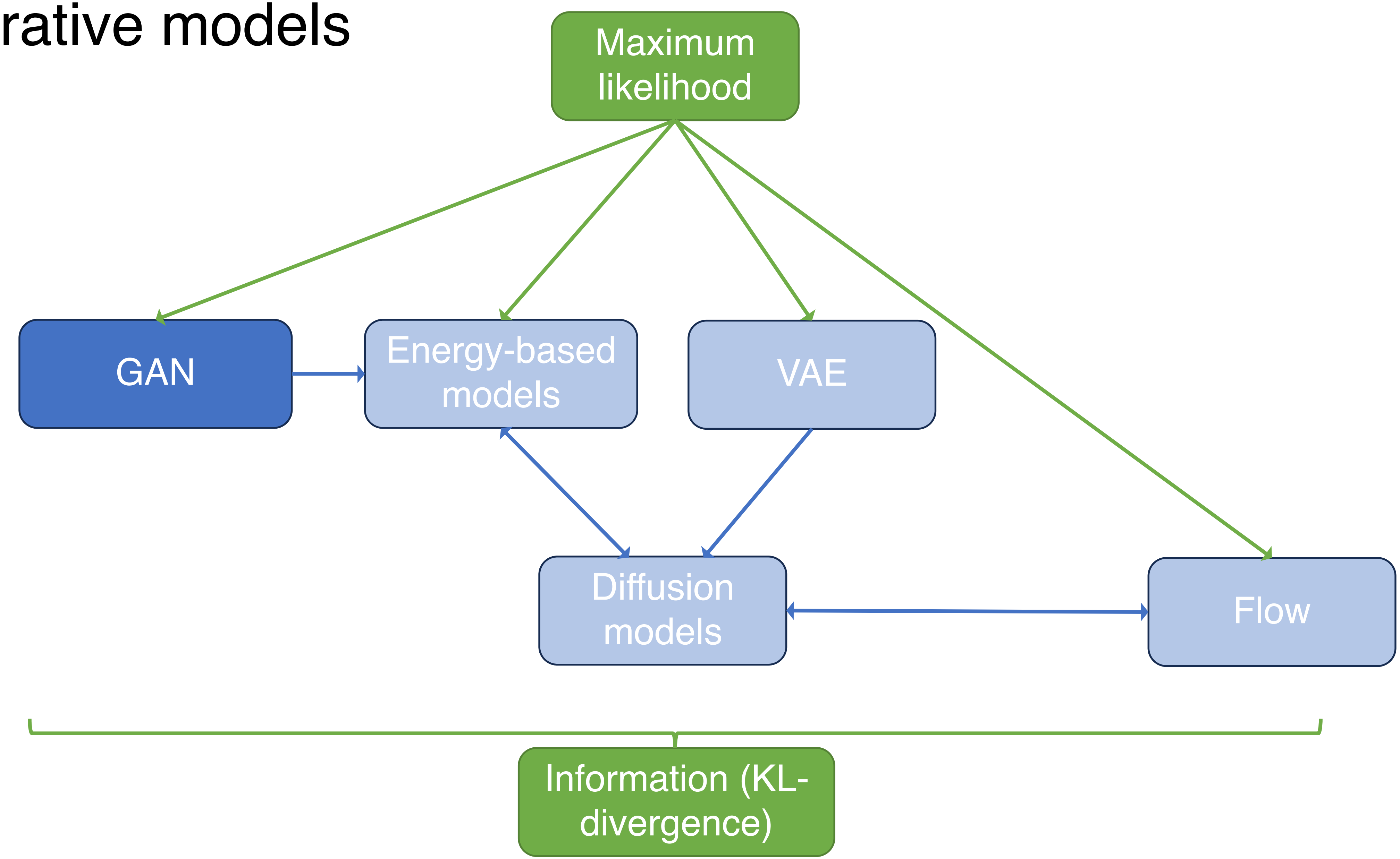
Variational autoencoder: Properties

- It can sample new examples of $\boldsymbol{x}$
- Can learn structured latent representation, given prior knowledge of $p_\theta(\boldsymbol{z})$
- Cannot compute the likelihood of data



$$\max_{\theta,\phi} L(\theta,\phi) = \mathbb{E}_{z\sim q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)}[\log p_\theta(\boldsymbol{x}_i|\boldsymbol{z})] - \mathbb{D}_{KL}\left[q_\phi(\boldsymbol{z}|\boldsymbol{x}_i)||p_\theta(\boldsymbol{z})\right]$$

Generative models



Generative model 2: Generative Adversarial Networks (GAN)

We want to learn a classifier $q_\theta\ (y|\boldsymbol{x}) = \begin{cases} q_\theta\ (y=1|\boldsymbol{x}) \coloneqq D_\theta(\boldsymbol{x}), & y=1 \\ q_\theta\ (y=0|\boldsymbol{x}) \coloneqq 1-D_\theta(\boldsymbol{x}), & y=0 \end{cases}$ for true and fake data.

We maximize the negative cross-entropy:

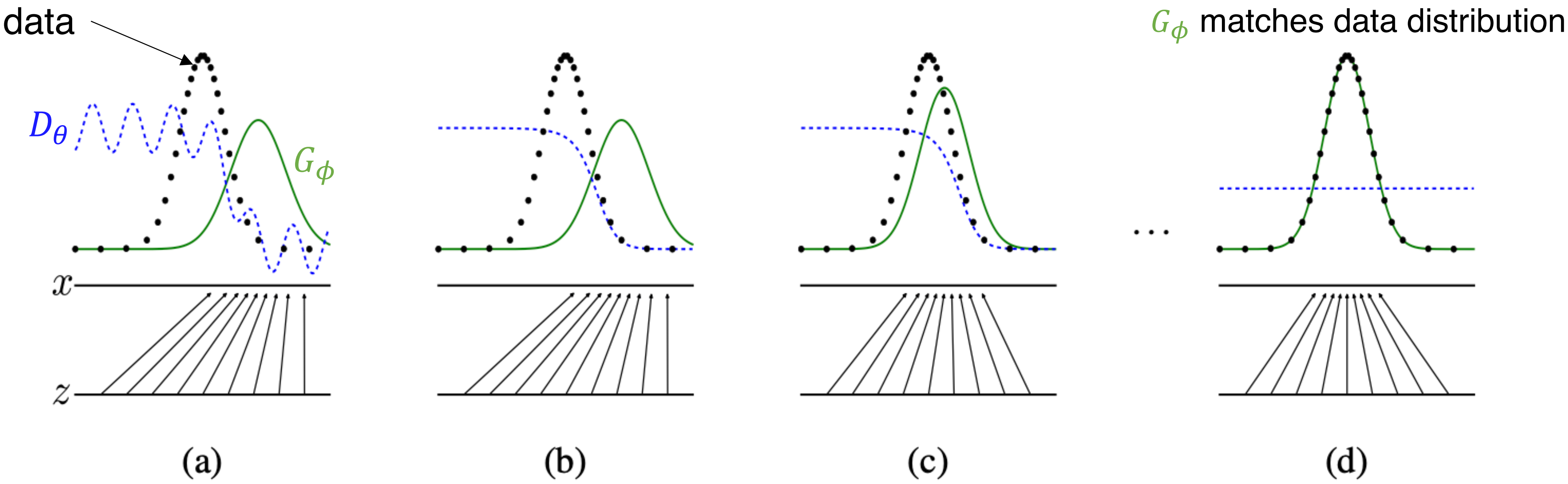
$$\max_{D_\theta} V(D_\theta) = \mathbb{E}_{x\sim p_{data}(\boldsymbol{x})}[\log q_\theta\ (y=1|\boldsymbol{x})] + \mathbb{E}_{x\sim p_{gen}(x)}[\log q_\theta\ (y=0|\boldsymbol{x})]$$

$$= \mathbb{E}_{x\sim p_{data}(\boldsymbol{x})}[D_\theta(\boldsymbol{x})] + \mathbb{E}_{x\sim p_{gen}(\boldsymbol{x})}[1 - D_\theta(\boldsymbol{x})]$$

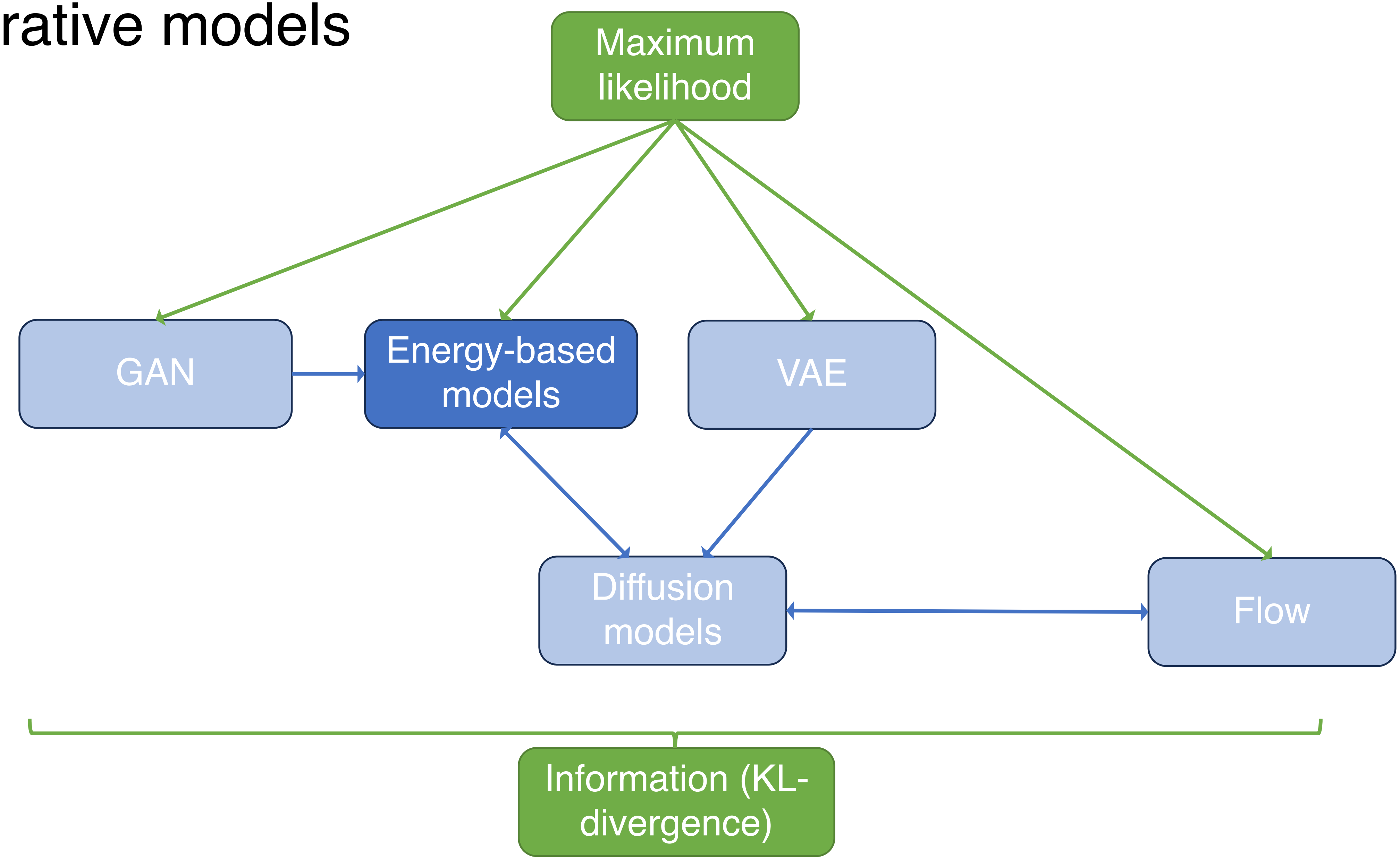
What if we also want to generate a distribution similar to $p_{data}(\boldsymbol{x})$?

Generative model 2: Generative Adversarial Networks (GAN)

$$\min_{G_\phi} \max_{D_\theta} V\left(D_\theta, G_\phi\right) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}(\boldsymbol{x})}[D_\theta(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_z(\boldsymbol{z})}\left[1 - D_\theta\left(G_\phi(\boldsymbol{z})\right)\right]$$



Generative models



Generative model 3: Energy-based models (EBM)

$$p_\theta(\boldsymbol{x}) = \frac{\exp\left(-E_\theta(\boldsymbol{x})\right)}{Z_\theta}$$

- $E_\theta(\boldsymbol{x})$ is an energy function that maps the input $\boldsymbol{x}$ to a scalar energy. The lower the $E_\theta(\boldsymbol{x})$, the higher the probability $p_\theta(\boldsymbol{x})$.
- $Z_\theta = \int \exp\left(-E_\theta(\boldsymbol{x})\right) d\boldsymbol{x}$ is a normalizing constant that is intractable.

Energy-based model: Inference

Stochastic Gradient Langevin Dynamics:

$$\boldsymbol{x}^{k+1} \leftarrow \boldsymbol{x}^{k} - \frac{\lambda}{2} \nabla_x E_\theta(\boldsymbol{x}) + \sqrt{\lambda}\boldsymbol{z}^{k}$$

where $\boldsymbol{z}^k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), k = 1,2, \ldots K$

Essentially: gradient descent with noise on the learned energy function $E_\theta(\boldsymbol{x})$.

When $\lambda \to 0, K \to +\infty$, the generated samples converge to $p_\theta(\boldsymbol{x}) = \frac{\exp(-E_\theta(\boldsymbol{x}))}{Z_\theta}$

Energy landscape $E_\theta(\boldsymbol{x})$

Energy-based model: Training with contrastive divergence

Contrastive divergence:

$$
\begin{aligned}
\nabla_\theta[-\log p_\theta(\boldsymbol{x})] &= \nabla_\theta E_\theta(\boldsymbol{x}) + \nabla_\theta \log Z_\theta \\
&= \nabla_\theta \underline{E_\theta(\boldsymbol{x})} - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\nabla_\theta \underline{E_\theta(\boldsymbol{x})}]
\end{aligned}
$$

energy for data samples

energy for negative samples (generated by $p_\theta(\boldsymbol{x})$)

Energy landscape $E_\theta(\boldsymbol{x})$

Push down energy for data samples

Energy-based model: Relation to GAN

EBM:

$$
\begin{aligned}
\nabla_\theta[-\log p_\theta(\boldsymbol{x})] &= \nabla_\theta E_\theta(\boldsymbol{x}) + \nabla_\theta \log Z_\theta \\
&= \nabla_\theta \underline{E_\theta(\boldsymbol{x})} - \mathbb{E}_{\boldsymbol{x}\sim p_\theta(\boldsymbol{x})}[\nabla_\theta \underline{E_\theta(\boldsymbol{x})}]
\end{aligned}
$$

energy for data samples

energy for negative samples (generated by $p_\theta(\boldsymbol{x})$)

GAN:

$$
\min_{G_\phi} \max_{D_\theta} V(D_\theta, G_\phi) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}[D_\theta(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\left[1 - D_\theta\left(G_\phi(\boldsymbol{z})\right)\right]
$$

GAN can be think of as a kind of EBM where the discriminator $E_\theta(\boldsymbol{x})$ is both used for discrimination as $D_\theta(\boldsymbol{x})$ and generation as $G_\phi(\boldsymbol{z})$.

Energy-based model: Compositional generation

Let $p_1(\boldsymbol{x}) \propto e^{-E_1(\boldsymbol{x})}, p_2(\boldsymbol{x}) \propto e^{-E_2(\boldsymbol{x})}$

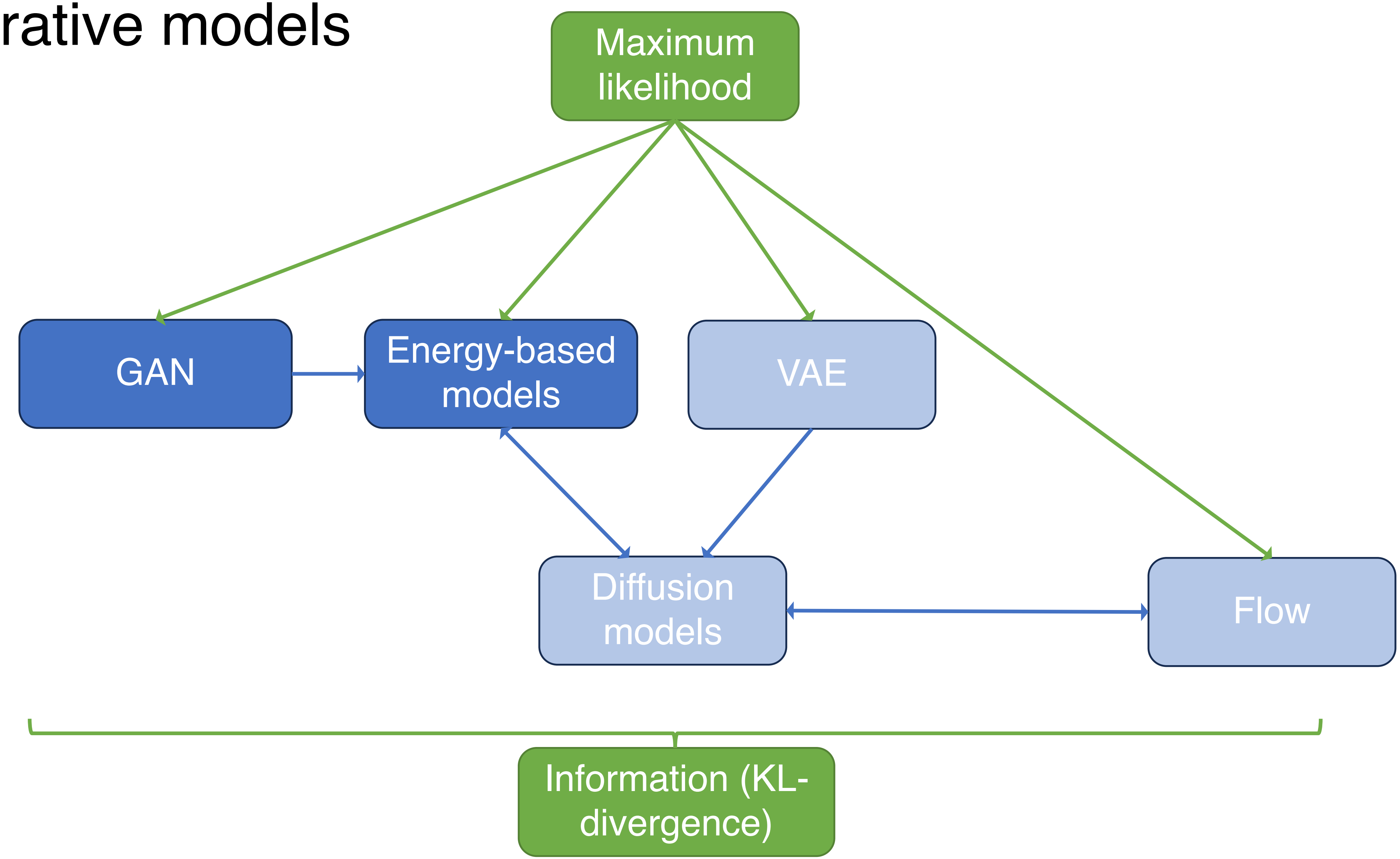
$\Rightarrow p_1(\boldsymbol{x})p_2(\boldsymbol{x}) \propto e^{-(E_1(\boldsymbol{x})+E_2(\boldsymbol{x}))}$

Product of probability corresponds to summation of the respective energy functions [1].

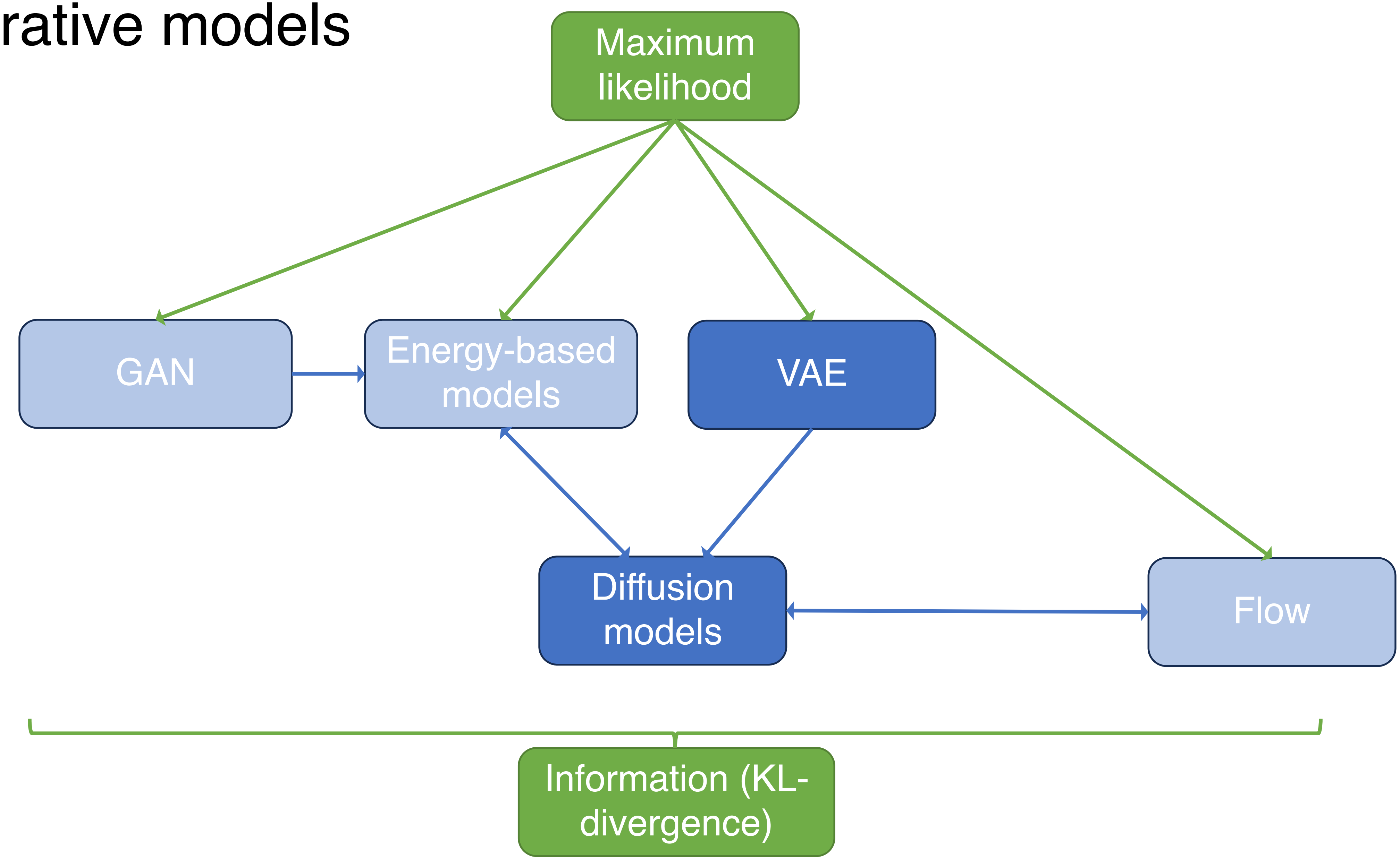
⇒ Allows inference-time generalization

[1] Du, Yilun, Shuang Li, and Igor Mordatch. "Compositional visual generation with energy based models." *NeurIPS* 2020: 6637-6647.

Generative models

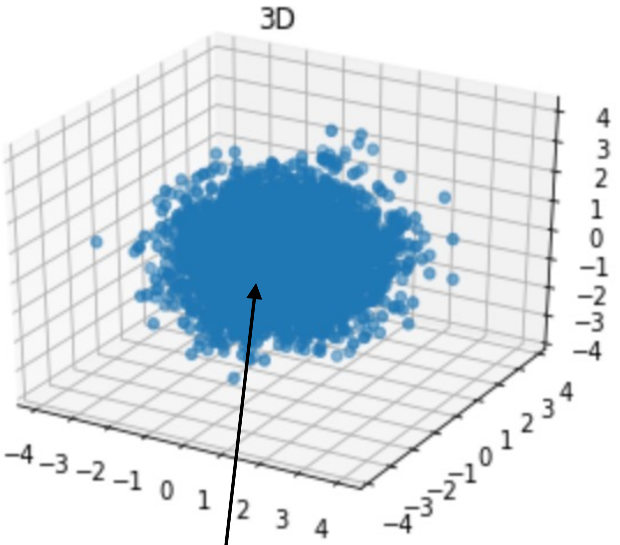


Generative models

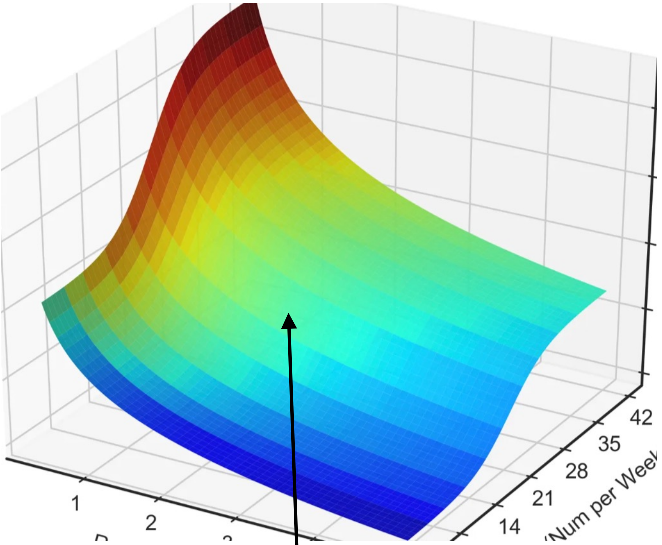


Generative model 4: Diffusion model

Insight: to construct a complex mapping from A to B, it is much easier to compose simple mappings



Diffusion model learns how to go from a random Gaussian sample to manifold

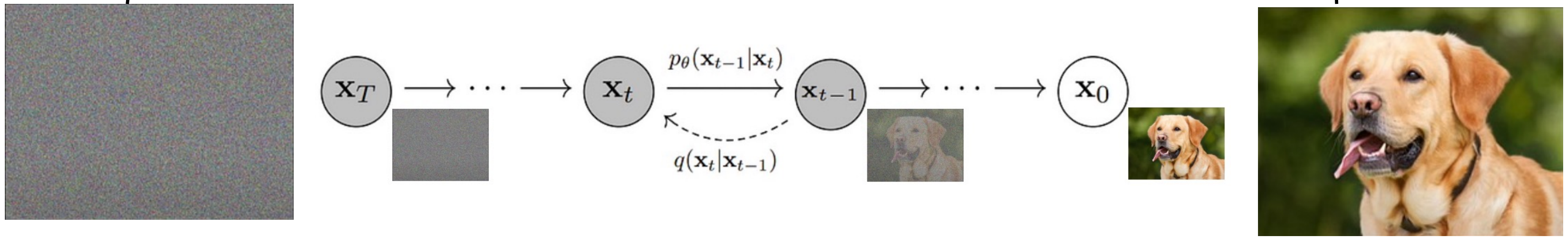


Each data sample is a point in a **manifold** in a high-dimensional space R^d

Gaussian distribution

diffusion model

data distribution



Diffusion model: Encoder

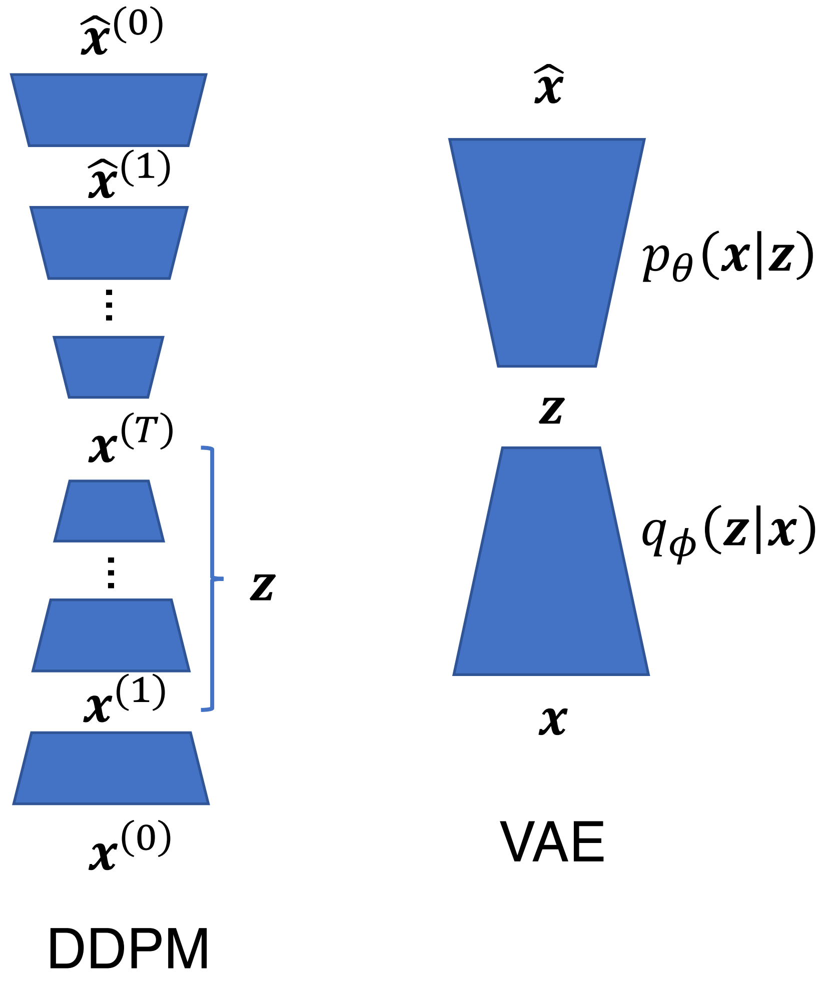
Recall VAE:

$$\text{ELBO} = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathbb{D}_{KL}[q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})]$$

Let $\boldsymbol{z} \coloneqq \boldsymbol{x}^{(1:T)}$, we define a unlearnable encoder:

$$q(\boldsymbol{z}|\boldsymbol{x}^{(0)}) = q(\boldsymbol{x}^{(1:T)}|\boldsymbol{x}^{(0)}) = \prod_{t=1}^{T} q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)})$$

where $q(\boldsymbol{x}^{(t)}|\boldsymbol{x}^{(t-1)}) = \mathcal{N}(\boldsymbol{x}^{(t)}; \sqrt{1-\beta_t}\boldsymbol{x}^{(t-1)}, \beta_t \boldsymbol{I})$

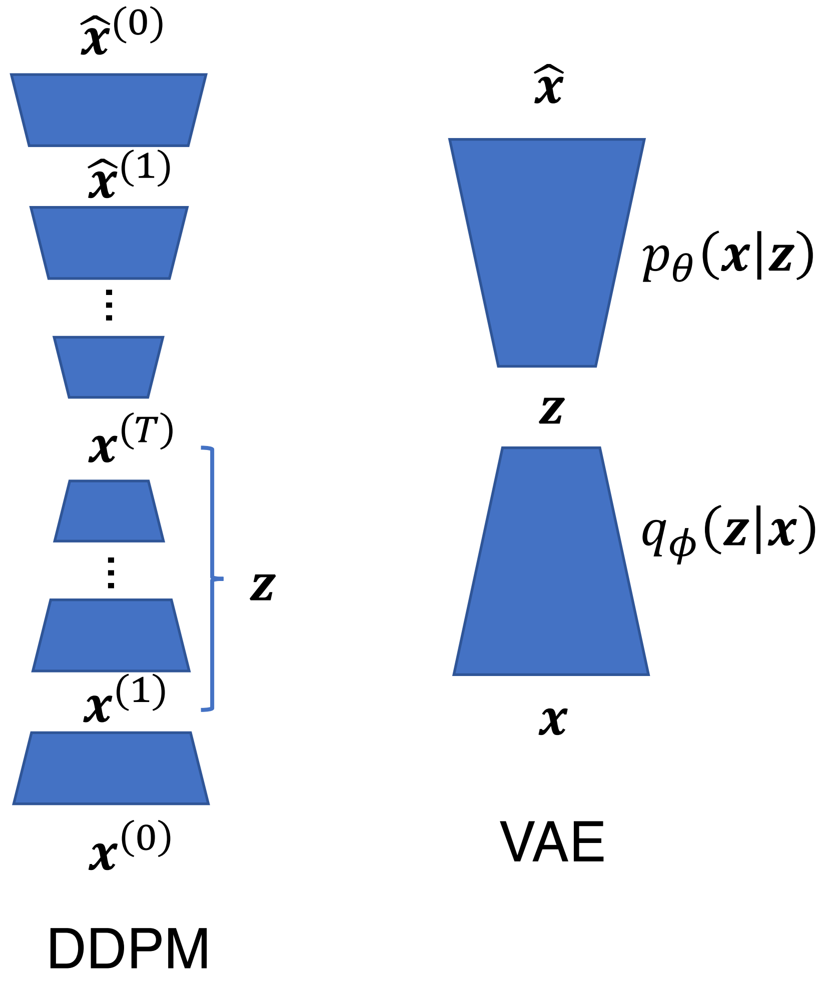


Diffusion model: Decoder

We define a learnable decoder:

$$p_\theta(\boldsymbol{x}^{(0)}, \boldsymbol{z}) = p(\boldsymbol{x}^{(T)}) \prod_{t=1}^{T} p_\theta(\boldsymbol{x}^{(t-1)} | \boldsymbol{x}^{(t)})$$

where $p_\theta(\boldsymbol{x}^{(t-1)} | \boldsymbol{x}^{(t)}) = \mathcal{N}(\boldsymbol{x}^{(t-1)}; \mu_\theta(\boldsymbol{x}^{(t)}, t), \tilde{\beta}_t \boldsymbol{I})$



Denoising Diffusion Probabilistic Models (DDPM) [1]

Maximizing the ELBO is equivalent to minimizing:

$$L = \mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_t}\boldsymbol{x}^{(0)} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|\right]$$

Training:

Algorithm 1 Training

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

Inference (sampling):

Algorithm 2 Sampling

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

[1] Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *NeurIPS* 2020: 6840-6851.

DDPM as Energy-based model

Diffusion model essentially learns a "energy"-based model $E_\theta(\boldsymbol{x};t)$ to model the probability distribution

$$p_\theta(\boldsymbol{x};t) = \frac{1}{Z_\theta} e^{-E_\theta(\boldsymbol{x};t)}$$

The denoising function $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}^{(t)},t)$ is essentially the gradient of the energy-based model:

$$\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^{(t)},t\right) = \nabla_{\boldsymbol{x}} E_\theta(\boldsymbol{x};t) = -\nabla_{\boldsymbol{x}} \log p_\theta(\boldsymbol{x};t)$$

$$\hat{\boldsymbol{x}}^{(0)} = \mathrm{argmin}_{\boldsymbol{x}}\, E_\theta(\boldsymbol{x};t)$$

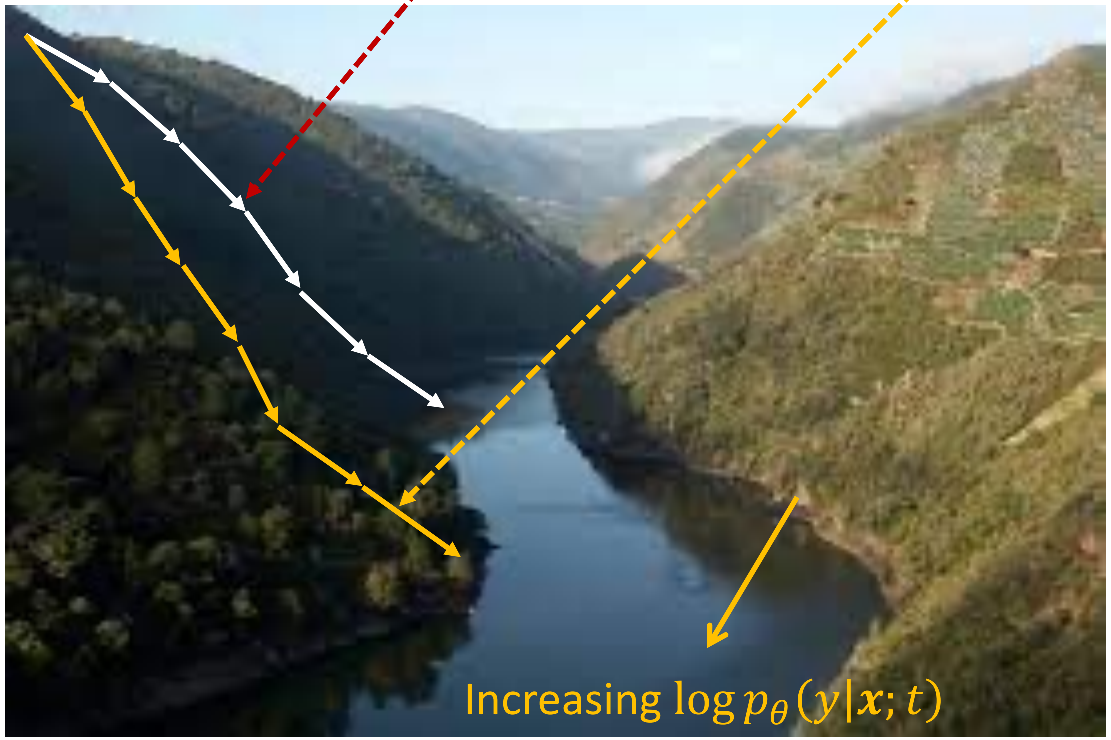
DDPM: classifier-based conditional generation

$$\hat{\boldsymbol{x}}^{(0)} = \mathrm{argmin}_{\boldsymbol{x}}(E_\theta(\boldsymbol{x}; t) - \log p_\theta(y|\boldsymbol{x}))$$

Suppose that we have trained $p(y|\boldsymbol{x})$, and want to generate $p(\boldsymbol{x}|y)$.

Only training $E_\theta(\boldsymbol{x}; t)$

Adding it during inference



Bayes rule:

$$p(\boldsymbol{x}|y; t) = \frac{p(\boldsymbol{x}; t)p(y|\boldsymbol{x})}{p(y; t)}$$

We have

$$\begin{aligned} & \nabla_{\boldsymbol{x}} \log p_\theta(\boldsymbol{x}|y; t) \\ = & \nabla_{\boldsymbol{x}} \log p_\theta(\boldsymbol{x}; t) + \nabla_{\boldsymbol{x}} \log p_\theta(y|\boldsymbol{x}) \\ = & -\nabla_{\boldsymbol{x}} E_\theta(\boldsymbol{x}; t) + \nabla_{\boldsymbol{x}} \log p_\theta(y|\boldsymbol{x}) \end{aligned}$$

DDPM: classifier-based conditional generation

$$\hat{\boldsymbol{x}}^{(0)} = \text{argmin}_{\boldsymbol{x}}\left(E_\theta(\boldsymbol{x};t) - \log p_\theta(y|\boldsymbol{x};t)\right)$$

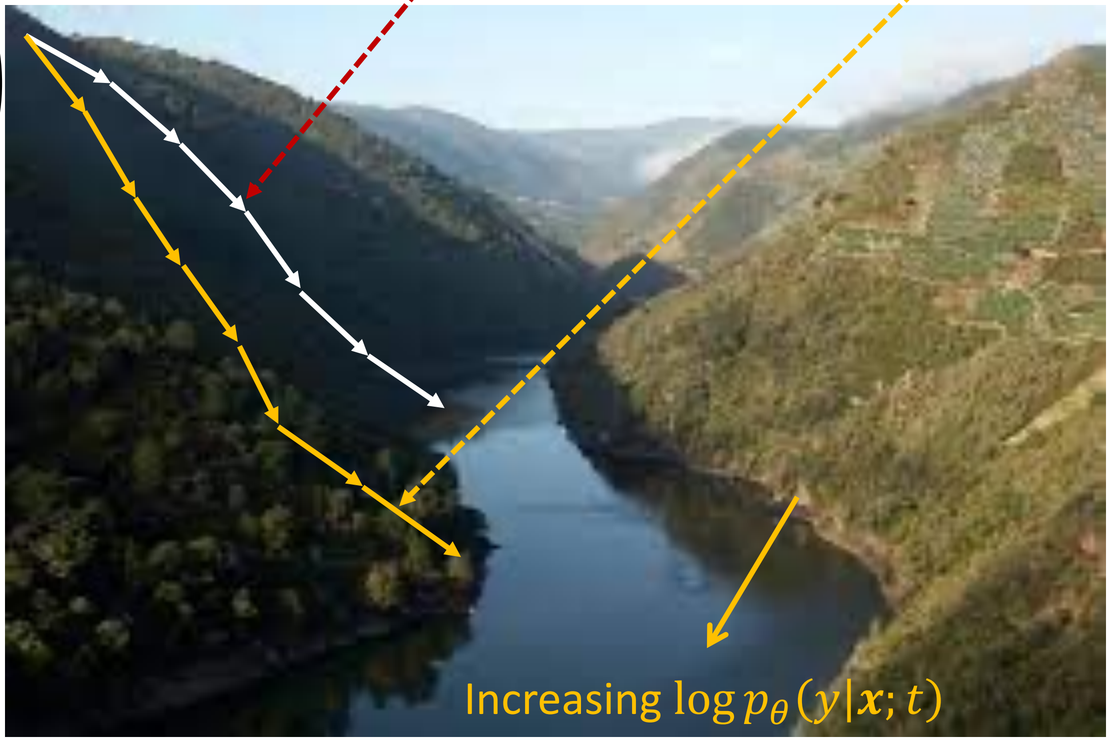
In inference:

$$\boldsymbol{x}^{(t-1)} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}^{(t)} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\left(\epsilon_\theta\left(\boldsymbol{x}^{(t)},t\right) \underline{- \nabla_{\boldsymbol{x}} \log p_\theta\left(y|\boldsymbol{x}^{(t)}\right)}\right)\right) + \sigma_t \boldsymbol{z}$$

Adding classifier-based guidance

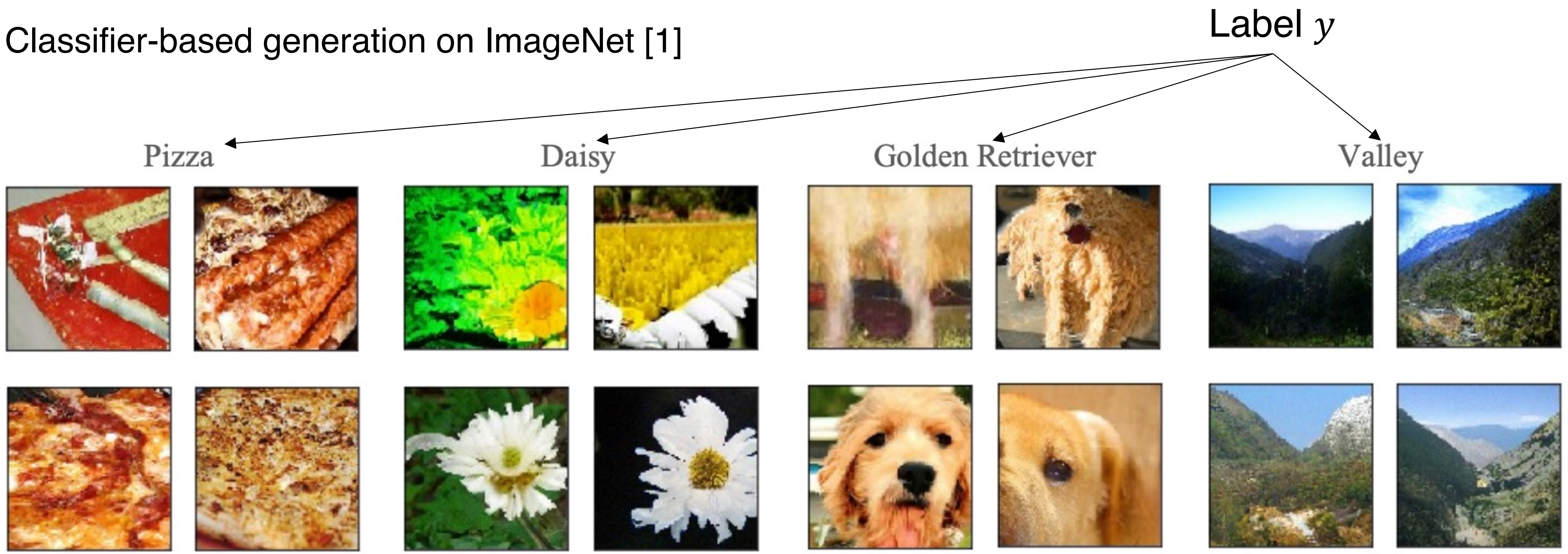
Only training $E_\theta(\boldsymbol{x};t)$

Adding it during inference



DDPM: classifier-based conditional generation

Classifier-based generation on ImageNet [1]



[1] Du, Yilun, et al. "Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc." *ICML* 2023

DDPM: classifier-based inverse design

$$\hat{\boldsymbol{x}}^{(0)} = \mathrm{argmin}_{\boldsymbol{x}}\big(E_\theta(\boldsymbol{x};t) + \mathcal{J}(\boldsymbol{x})\big)$$

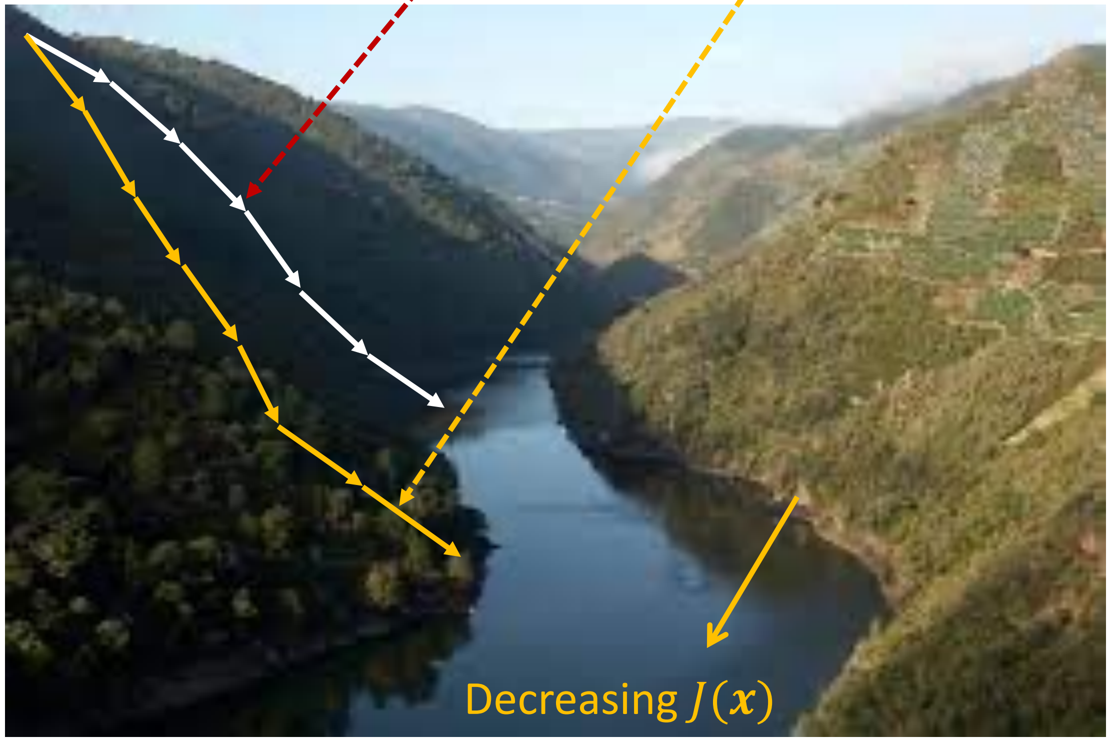
Suppose we have pre-specified inverse-design objective $\mathcal{J}(\boldsymbol{x})$ we want to minimize:

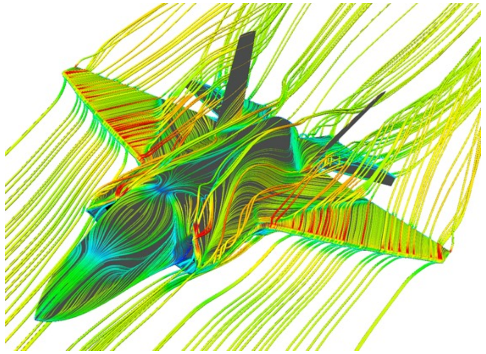
Only training $E_\theta(\boldsymbol{x};t)$

Adding it during inference

$$\boldsymbol{x}^{(t-1)} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}^{(t)} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\left(\epsilon_\theta\big(\boldsymbol{x}^{(t)},t\big) + \underline{\nabla_{\boldsymbol{x}}\mathcal{J}(\boldsymbol{x})}\right)\right) + \sigma_t \boldsymbol{z}$$

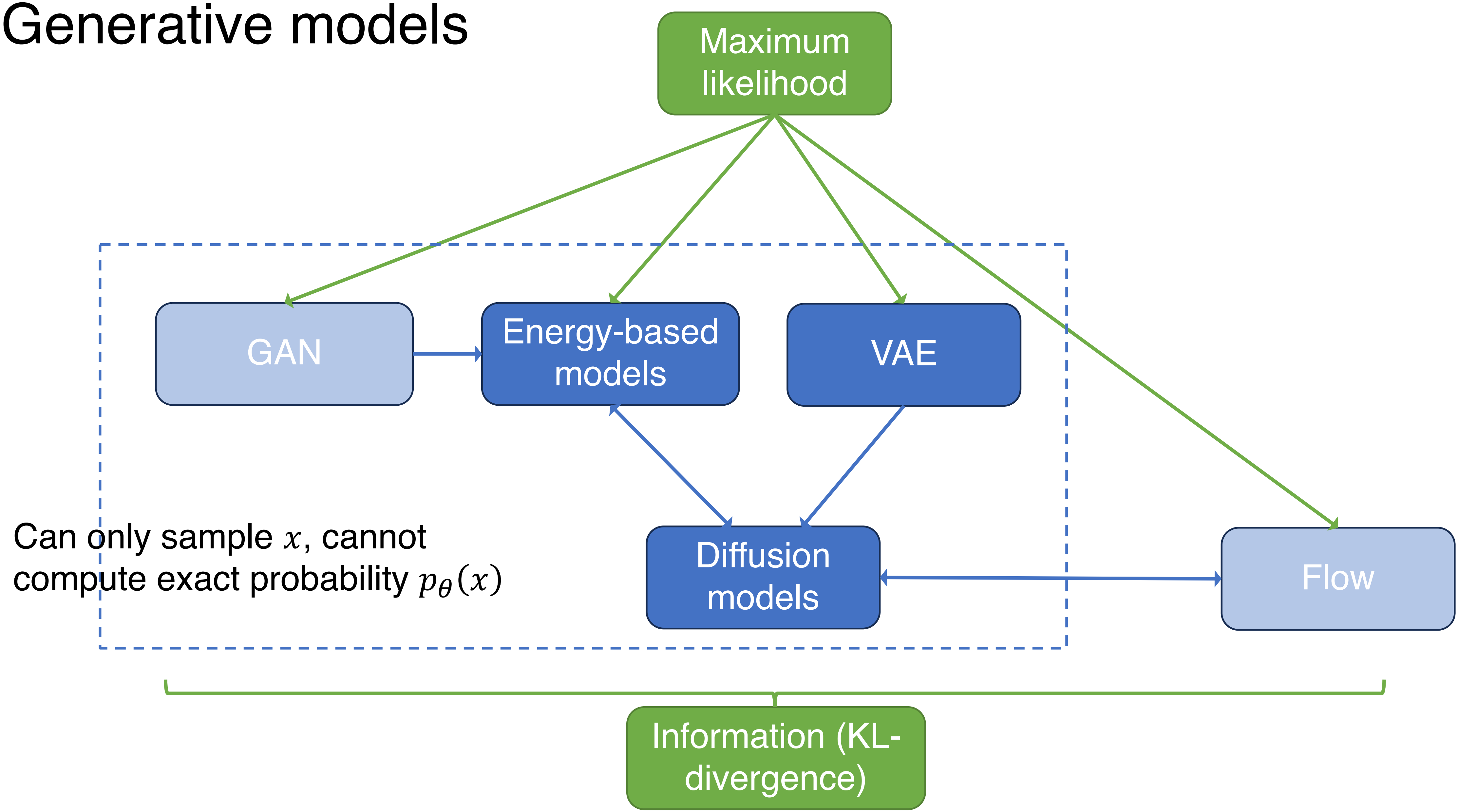
Adding guidance



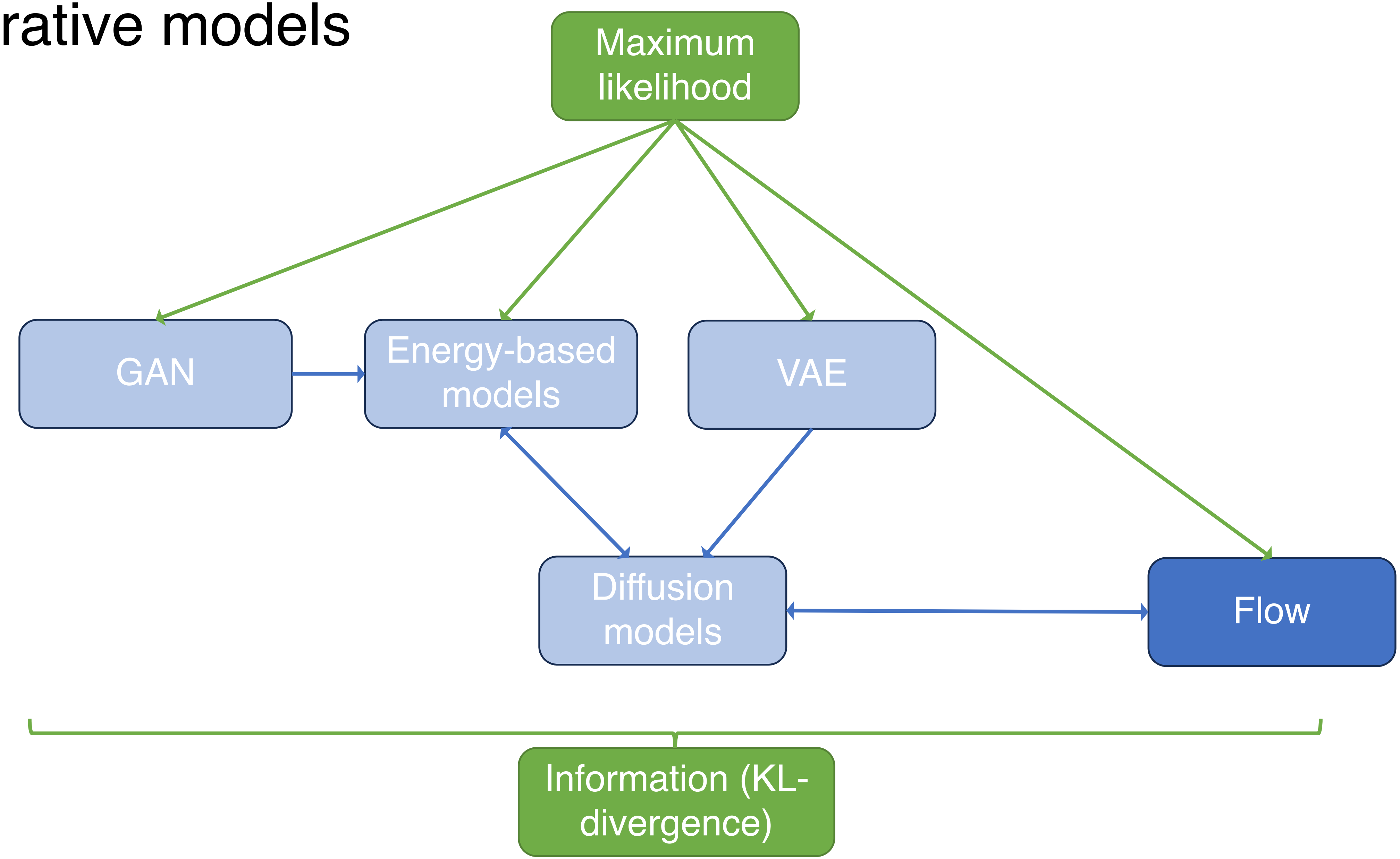


E.g. shape design of the airplane to minimize drag

Generative models



Generative models



Flow: Normalizing flow

Given prior density $p_z(z)$ and an invertible function ϕ,
$x = \phi(z)$, we have

$$\int p_z(z)dz = \int p_x(x)dx = 1$$

$$\Leftrightarrow p_z(z)dz = p_x(x)dx$$

The probability mass is conserved under change of variable

$$\Leftrightarrow p_x(x) = p_z(z)\left|\frac{dx}{dz}\right|^{-1} = p_z(z)\left|\frac{d\phi(z)}{dz}\right|^{-1}$$

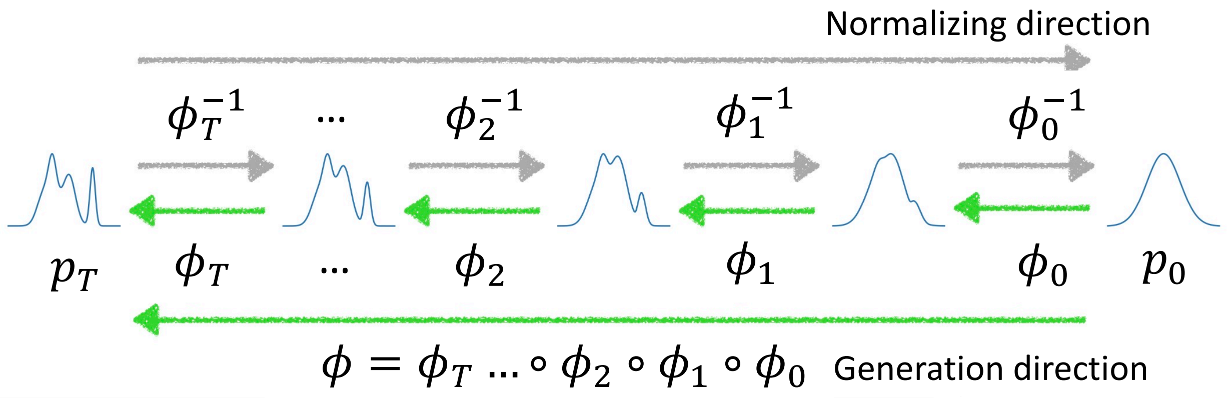
Generalizing to multivariate varables $\boldsymbol{x}$ and $\boldsymbol{z}$, we have

$$\log p_{\boldsymbol{x}}(\boldsymbol{x}) = \log p_{\boldsymbol{z}}(\boldsymbol{z}) - \log\det\frac{\partial\boldsymbol{\phi}(\boldsymbol{z})}{\partial\boldsymbol{z}}$$

Flow: Normalizing flow

Normalizing flow $\boldsymbol{\phi}_t(x)$

- $\boldsymbol{\phi}_t$: an invertible, differentiable function
- $p_T(\boldsymbol{x})$: real data
- $p_0(\boldsymbol{x})$: prior, e.g. Gaussian



$$\log p_{t+1}(\boldsymbol{x}) = \log p_t(\boldsymbol{x}) - \log \det \left[\frac{\partial \boldsymbol{\phi}_t(\boldsymbol{x})}{\partial \boldsymbol{x}}\right], t = 1,2 \ldots T$$

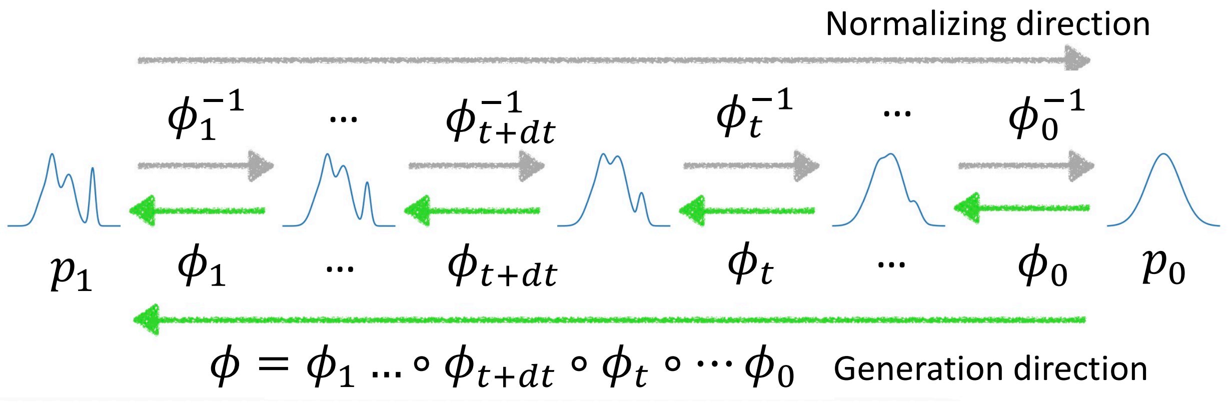
Design the layers ϕ_t in a way that is invertible and easy to compute $\det\left[\frac{\partial \boldsymbol{\phi}_t(\boldsymbol{x})}{\partial \boldsymbol{x}}\right]$, e.g., upper triangular

Neural ODE: Continuous normalizing flow [1]

[1] Chen, Ricky TQ, et al. "Neural ordinary differential equations." NeurIPS 2018

Making the layers ϕ_t continuous w.r.t. t

$$\begin{cases} \dfrac{d\boldsymbol{\phi}_t(\boldsymbol{x})}{dt} = \boldsymbol{v}_t(\phi_t(\boldsymbol{x}); \theta) \\ \boldsymbol{\phi}_0(\boldsymbol{x}) = \boldsymbol{x} \end{cases}$$



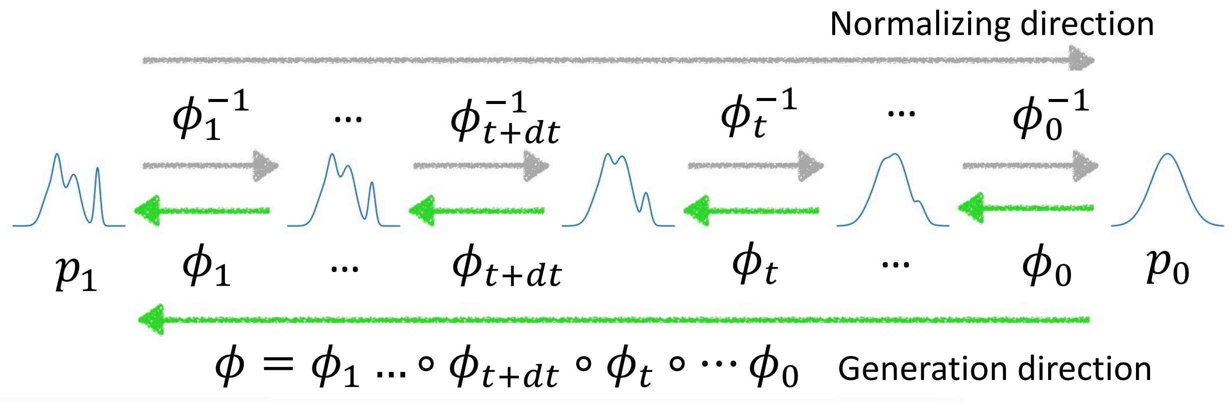
- The flow $\boldsymbol{\phi}_t$ is an transformation on the variable $\boldsymbol{x}$ at layer $t \in [0,1]$.
- $\boldsymbol{v}_t(\cdot, \theta)$ is a field that determines the dynamics of the flow $\boldsymbol{\phi}_t$ w.r.t. t.
- The flow $\phi_t(\boldsymbol{x})$ uniquely defines a probability density path $p_t(\boldsymbol{x}) = \phi_t \circ \phi_{t-dt} \circ \cdots \circ \phi_0 \, p_0(\boldsymbol{x}) = [\phi_t] p_0(\boldsymbol{x})$

Vector field $\boldsymbol{v}_t(\boldsymbol{x}; \theta)$ $\xrightarrow{\text{determines}}$ flow $\boldsymbol{\phi}_t$ $\xrightarrow{\text{generates}}$ Probability path $p_t = [\phi_t]_* p_0$

Neural ODE: Continuous normalizing flow, learning

Making the layers ϕ_t continuous w.r.t. t

$$\begin{cases} \dfrac{d\boldsymbol{\phi}_t(\boldsymbol{x})}{dt} = \boldsymbol{v}_t(\phi_t(\boldsymbol{x});\theta) \\ \boldsymbol{\phi}_0(\boldsymbol{x}) = \boldsymbol{x} \end{cases}$$



How to learn $\boldsymbol{v}_t(\cdot;\theta)$ such that it can transform a prior distribution $p_0(\boldsymbol{x})$ to a target distribution $p_{data}(\boldsymbol{x})$ specified by samples $\{\boldsymbol{x}_i\}_{i=1}^N$?

Adjoint method (simulation-based): generalizing gradient descent, where the gradient w.r.t. θ is another ODE from $t = 1$ to $t = 0$.

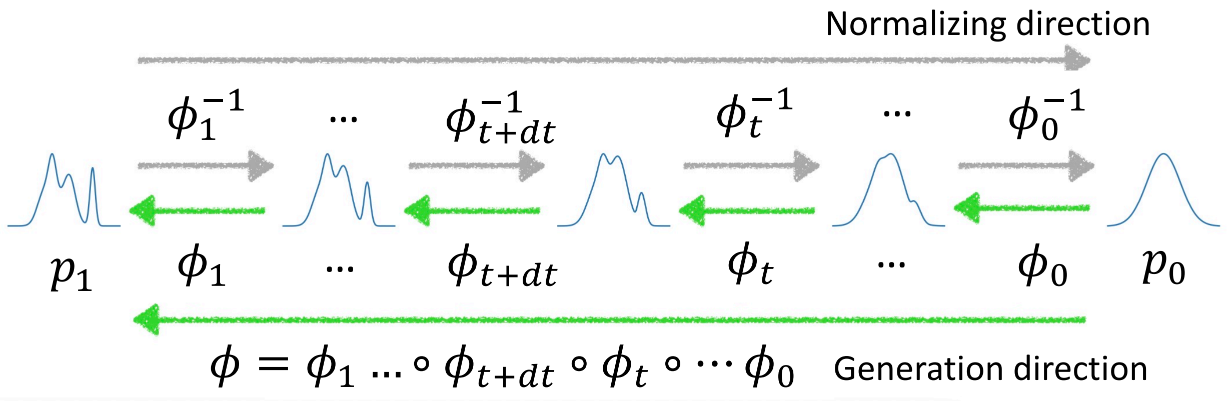
It is computationally expensive!

[1] Chen, Ricky TQ, et al. "Neural ordinary differential equations." *Advances in neural information processing systems* 31 (2018).

Flow matching [1]

Making the layers ϕ_t continuous w.r.t. t

$$\begin{cases} \dfrac{d\boldsymbol{\phi}_t(\boldsymbol{x})}{dt} = \boldsymbol{v}_t(\phi_t(\boldsymbol{x}); \theta) \\ \boldsymbol{\phi}_0(\boldsymbol{x}) = \boldsymbol{x} \end{cases}$$



Given a target path $p_t(\boldsymbol{x})$ and a corresponding vector field $u_t(\boldsymbol{x})$, we can directly regressing:

$$L_{FM}(\theta) = \mathbb{E}_{t,p_t(\boldsymbol{x})}[\|\boldsymbol{v}_t(\phi_t(\boldsymbol{x}); \theta) - u_t(\boldsymbol{x})\|^2]$$

The problem is that we only have data $\{\boldsymbol{x}_i\}_{i=1}^N$ and do not have $p_t(\boldsymbol{x})$ nor $u_t(\boldsymbol{x})$.

[1] Lipman, Yaron, et al. "Flow matching for generative modeling." ICLR 2022

Flow matching

We consider $p_t(\boldsymbol{x}|\boldsymbol{x}_1)$, where $\boldsymbol{x}_1 \sim p_{data}(\boldsymbol{x})$

Vector field $\boldsymbol{u}_t(\boldsymbol{x}|\boldsymbol{x}_1)$ $\xrightarrow{\text{determines}}$ flow $\boldsymbol{\psi}_t$ $\xrightarrow{\text{generates}}$ probability path $p_t(\boldsymbol{x}|\boldsymbol{x}_1) = [\boldsymbol{\psi}_t]_* p_0(\boldsymbol{x})$

$$\boldsymbol{u}_t(\boldsymbol{x}) = \int \boldsymbol{u}_t(\boldsymbol{x}|\boldsymbol{x}_1) \frac{p_t(\boldsymbol{x}|\boldsymbol{x}_1) p_{data}(\boldsymbol{x}_1)}{p_t(\boldsymbol{x})} d\boldsymbol{x}_1$$

The target vector field $\boldsymbol{u}_t(\boldsymbol{x})$ is a weighted sum of the conditional vector field $\boldsymbol{u}_t(\boldsymbol{x}|\boldsymbol{x}_1)$

$$p_t(\boldsymbol{x}) = \int p_t(\boldsymbol{x}|\boldsymbol{x}_1) p_{data}(\boldsymbol{x}_1)\, d\boldsymbol{x}_1$$

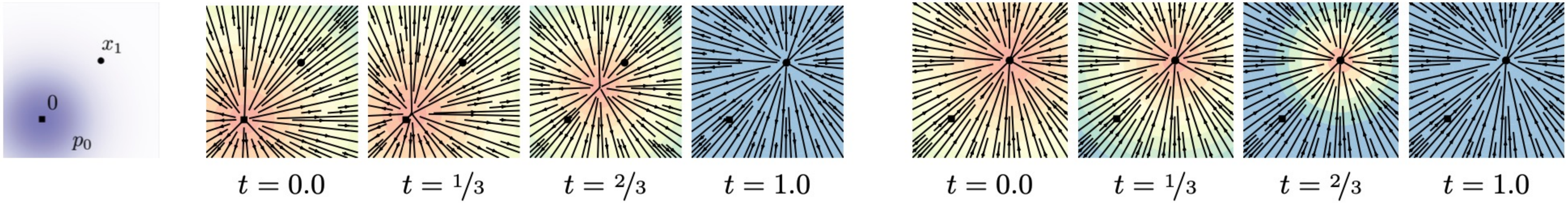
The target distribution $p_t(\boldsymbol{x})$ is a weighted sum of the conditional distribution $p_t(\boldsymbol{x}|\boldsymbol{x}_1)$

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t\sim U[0,1], \boldsymbol{x}_1 \sim p_{data}(\boldsymbol{x}_1), x \sim p_t(\boldsymbol{x}|\boldsymbol{x}_1)} \| v_t(\boldsymbol{x};\theta) - u_t(\boldsymbol{x}|\boldsymbol{x}_1) \|^2$$

Proposition: $\nabla_\theta \mathcal{L}_{FM}(\theta) = \nabla_\theta \mathcal{L}_{CFM}(\theta)$

Flow matching

We can then explicitly design the vector field $\boldsymbol{u}_t(\boldsymbol{x}|\boldsymbol{x}_1)$, e.g., using optimal transport (OT):



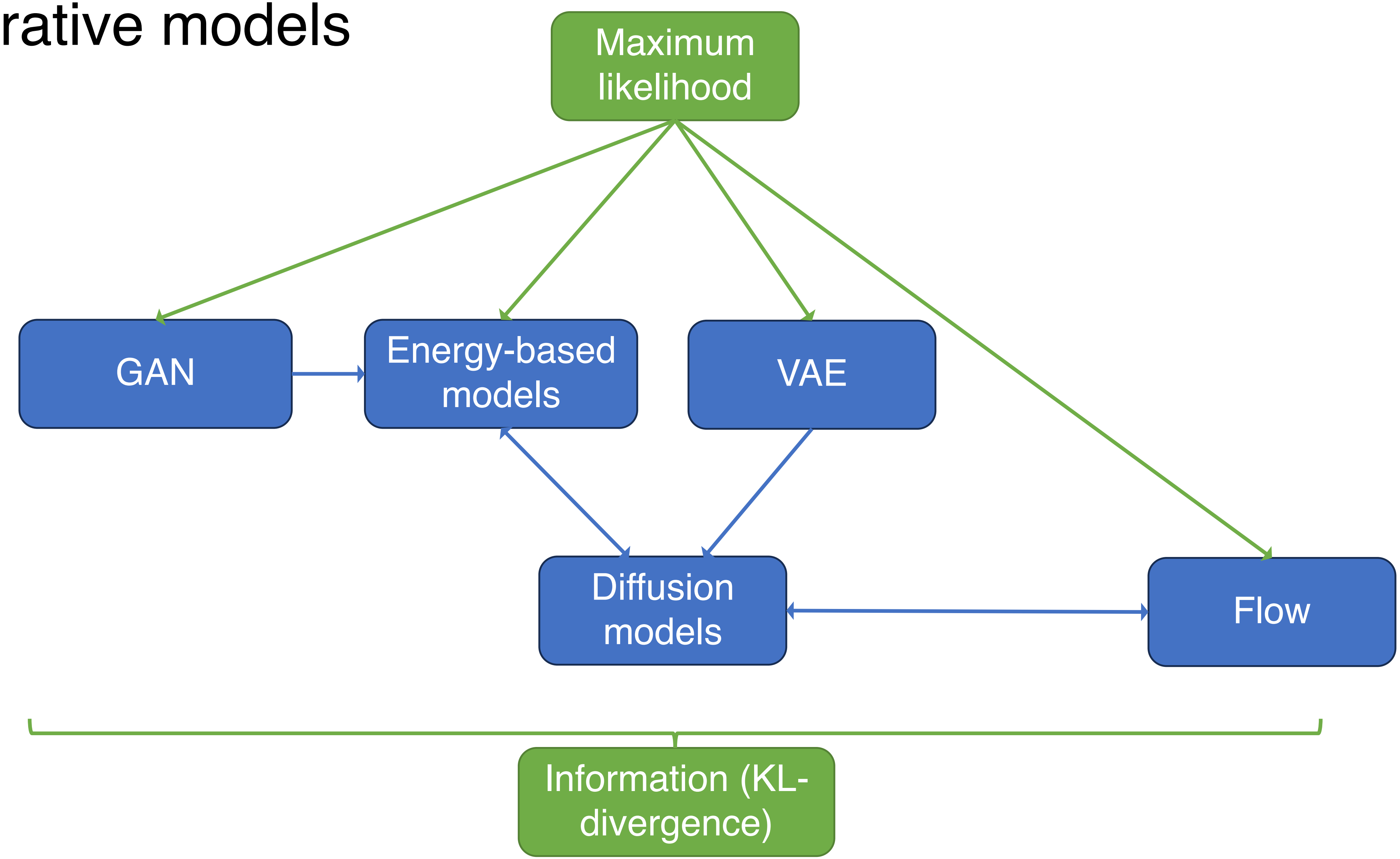
Diffusion path – conditional score function

OT path – conditional vector field

$\boldsymbol{\epsilon}_\theta\left(\boldsymbol{x}^{(t)}; t\right)$

$\boldsymbol{u}_t(\boldsymbol{x}|\boldsymbol{x}_1)$

Generative models



Outline

- Generative models
  - VAE
  - GAN
  - Energy-based models
  - Diffusion models
  - Flows
- Application of diffusion models
  - Image, video, and shape generation
  - Simulation
  - Inverse design/inverse problem
  - Control/planning

Application 1.1: image and video generation

Images and videos generated by diffusion models:

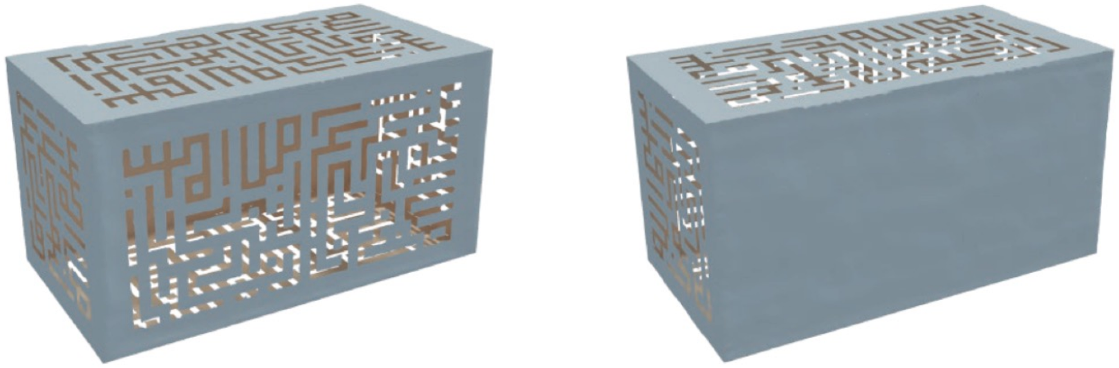
By DallE 2 [1]

Text to video generation by Sora [2]

[1] Ramesh, Aditya, et al. "Hierarchical text-conditional image generation with clip latents." *arXiv preprint arXiv:2204.06125* 1.2 (2022): 3.
[2] OpenAI team. “Video generation models as world simulators”, 2024

Application 1.2: 3D shape generation

By MeshDiffusion [1]

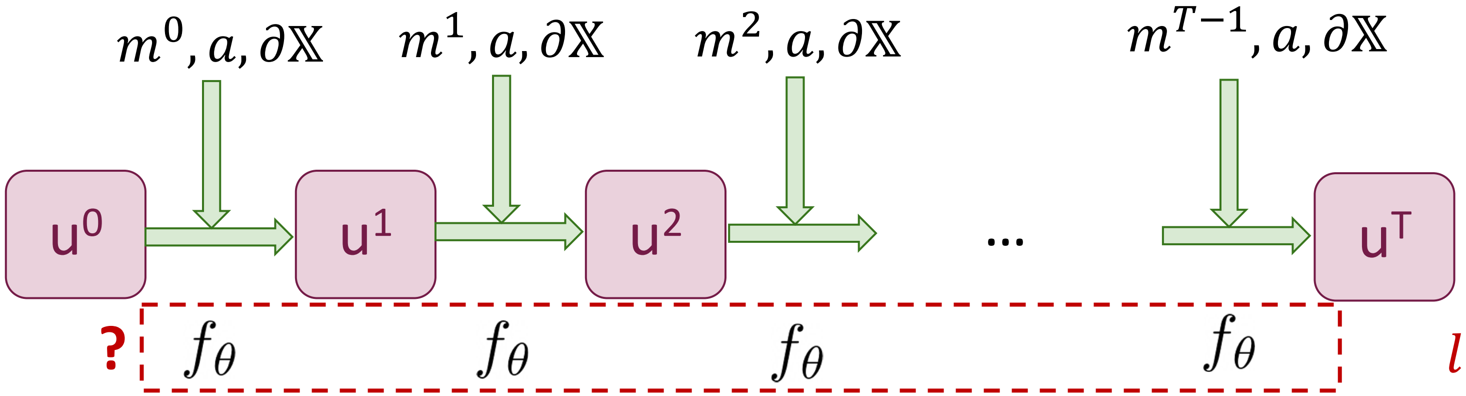


By G-Shell [1]

[1] Liu, Zhen, et al. "Meshdiffusion: Score-based generative 3d mesh modeling." *ICLR 2023*
[2] Liu, Zhen, et al. "Ghost on the Shell: An Expressive Representation of General 3D Shapes." *ICLR* 2024

Task 2: (Learning) simulation

Goal: learn the mapping f_θ from u^t to u^{t+1}:



$$loss = \mathbb{E}[MSE(f_\theta(u^t), u^{t+1})]$$

u^t: original **state**（状态）of the system. Can be a graph (e.g., mesh, particle-based systems, molecules), a tensor, or an infinite-dimensional function $u(t, x)$ as solution to a PDE

f_θ: **neural surrogate models**（神经网络代理模型）

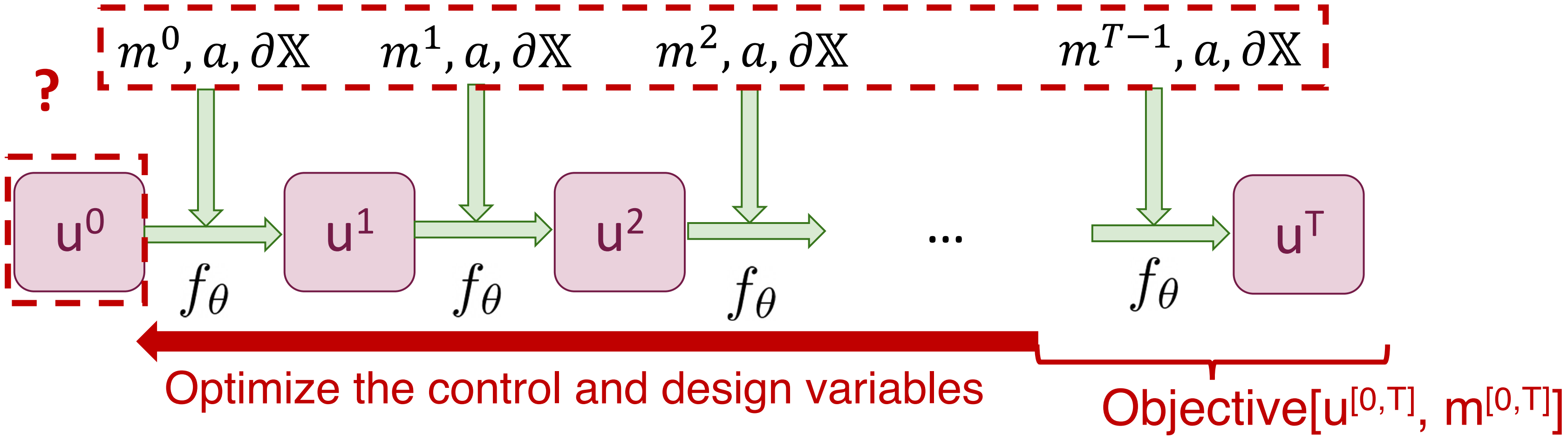
m^t: **external control**（外界控制）

a: **static parameters**（静态参数） of the system that does not change with time (e.g. parameters of PDE, spatially varying diffusion coefficient)

$\partial\mathbb{X}$: **boundary condition**（边界条件） of the system

PDE: partial differential equation
ODE: ordinary differential equation

Tasks 3 & 4: Inverse design, inverse problem, and control



u^t: original **state** of the system. Can be an infinite-dimensional function $u(t, x)$ as solution to a PDE, or a graph (e.g., mesh, particle-based systems, molecules)

f_θ: neural surrogate models

m^t: external **control（外界控制）** — control（控制）

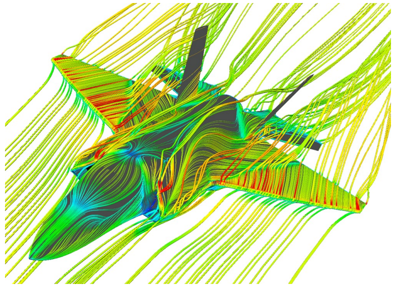
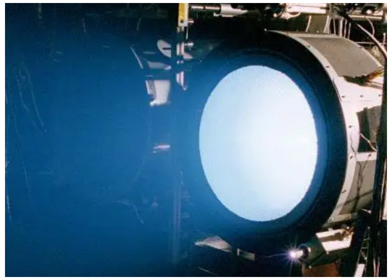
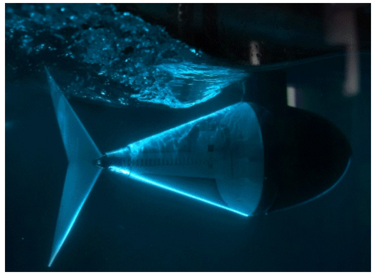
a: **static parameters（静态参数）** of the system that does not change with time (e.g. parameters of PDE, spatially varying diffusion coefficient)

$\partial\mathbb{X}$: **boundary condition（边界条件）** of the system

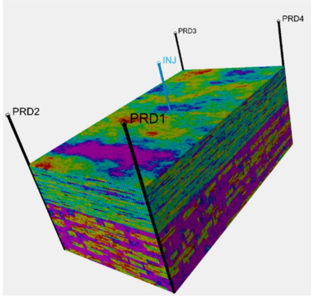
inverse design （反向设计）

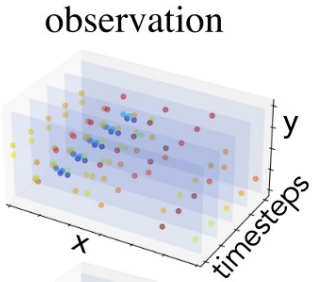
Tasks 3 & 4: Inverse design, inverse problem, and control

- **Inverse design:** boundary $\partial\mathbb{X}$, initial condition u^0, parameter a to **optimize design objective:** plane design, rocket shape, underwater robot shape

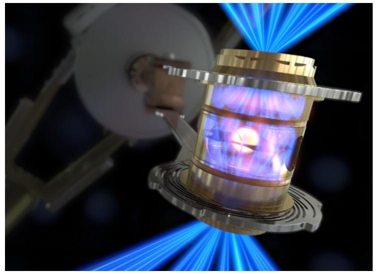




- **Inverse problem : infer initial condition** u^0 or parameter a to **match prediction with observation**





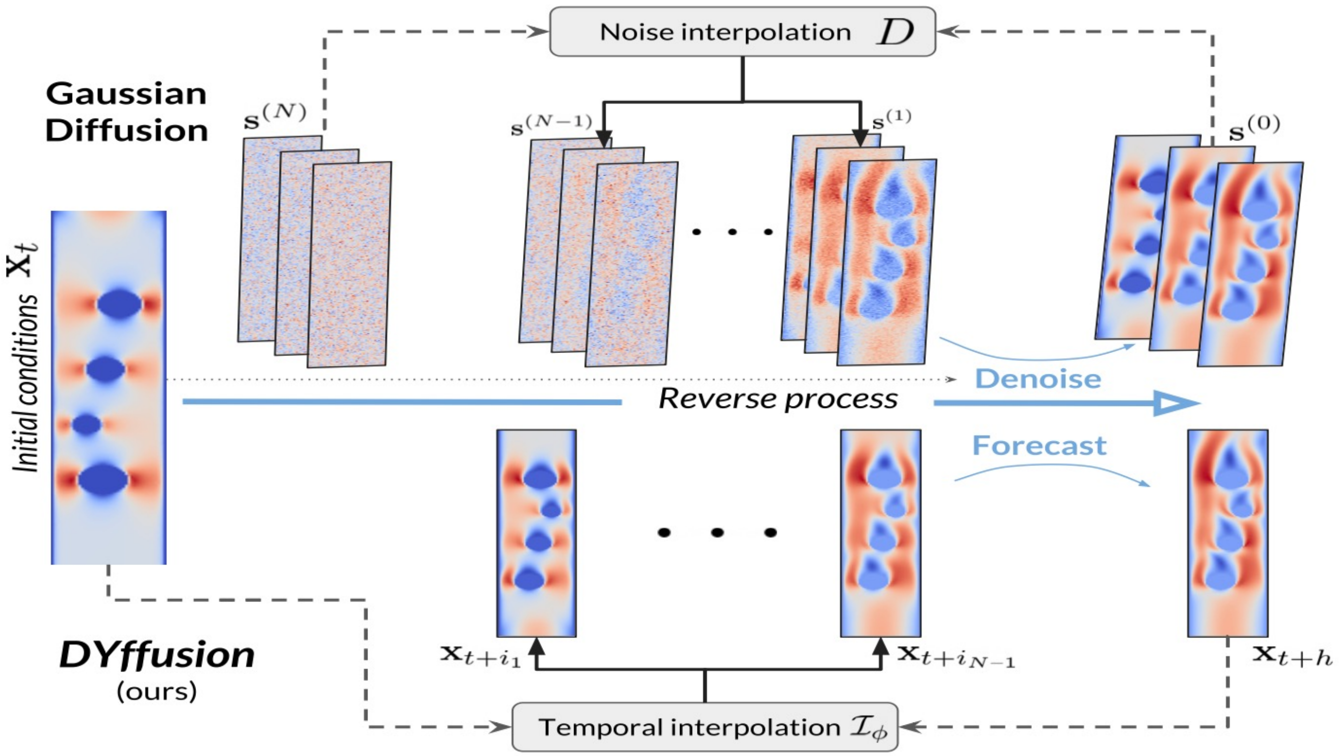
- **Control: optimize control** m^t to **optimize control objectives**: controlled nuclear fusion, robotics




Application 2.1: Simulation for PDE

For simulation（仿真）：

Learn $P\left(u^{[1,T]}|u^0\right)$:



[1] Cachay, Salva Rühling, et al. "DYffusion: A Dynamics-informed Diffusion Model for Spatiotemporal Forecasting." NeurIPS 2023, *arXiv preprint arXiv:2306.01984* (2023).

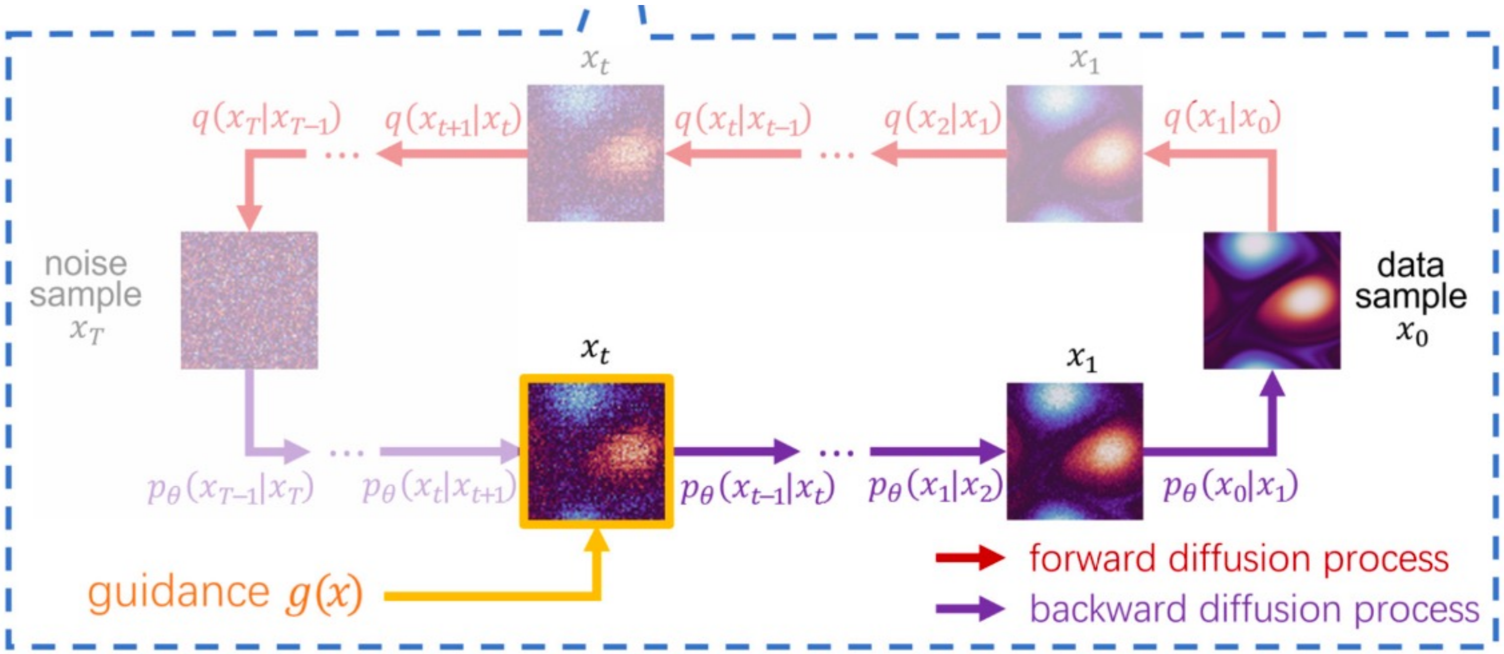
Application 2.2: Physics-informed simulation

Physics-informed diffusion models:

$$\hat{\boldsymbol{x}}^{(0)} = \mathrm{argmin}_{\boldsymbol{x}}\left(E_\theta(\boldsymbol{x};t) + \underline{\mathcal{J}(\boldsymbol{x})}\right)$$

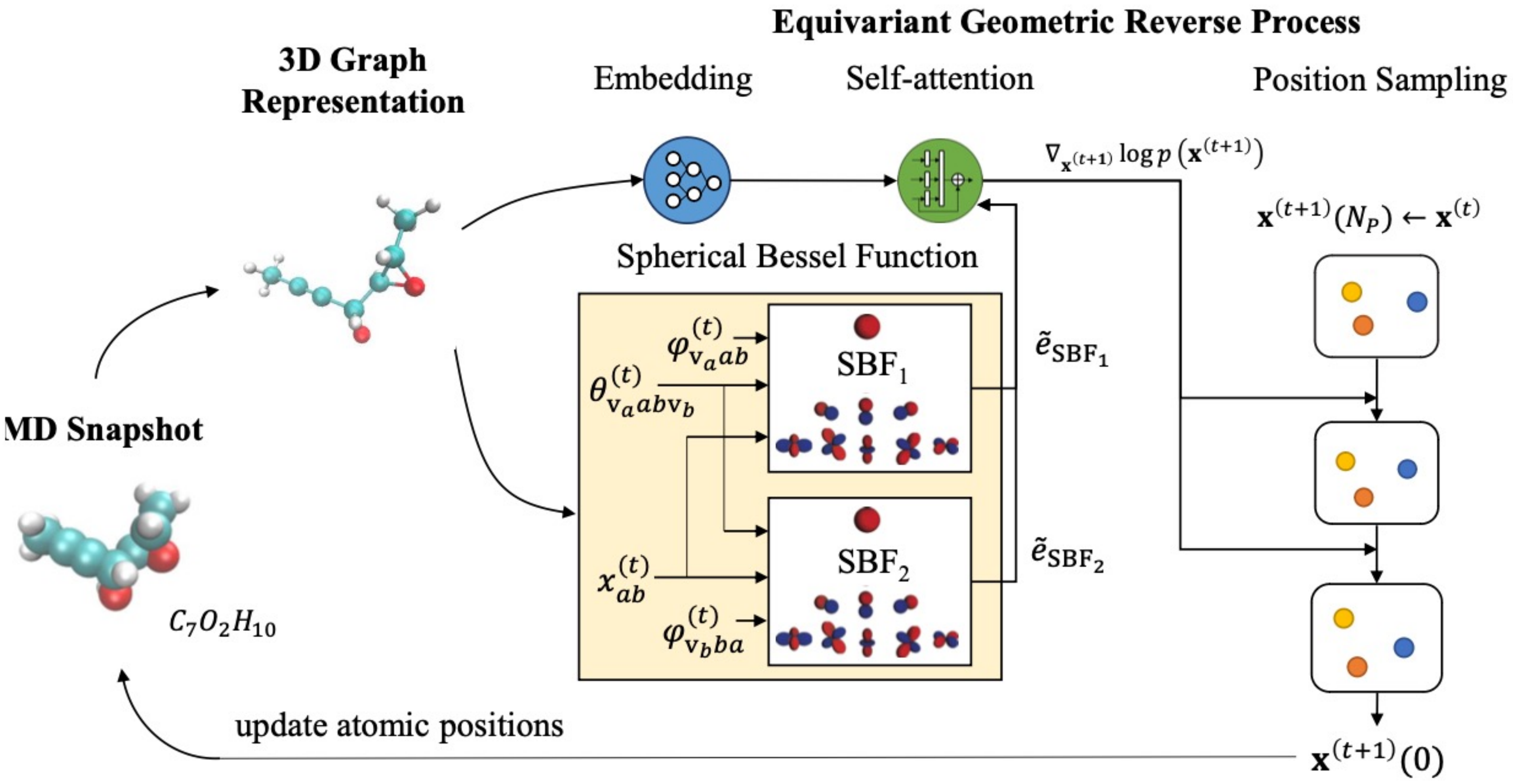
Equation loss

Key idea: using the PDE residual as additional term for the objective $J(x)$.



[1] Shu, Dule, Zijie Li, and Amir Barati Farimani. "A physics-informed diffusion model for high-fidelity flow field reconstruction." *Journal of Computational Physics* 478 (2023): 111972.

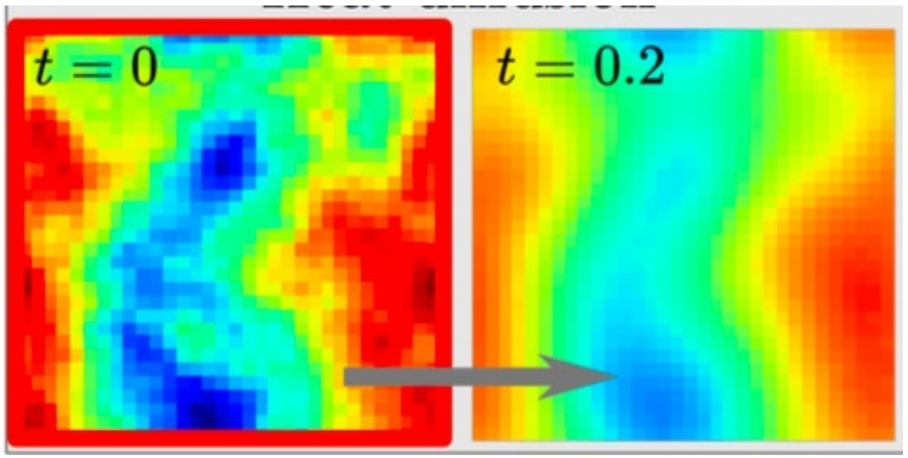
Application 2.3: Molecular dynamics simulation [1]



[[1] Wu, Fang, and Stan Z. Li. "DIFFMD: a geometric diffusion model for molecular dynamics simulations." AAAI 2023

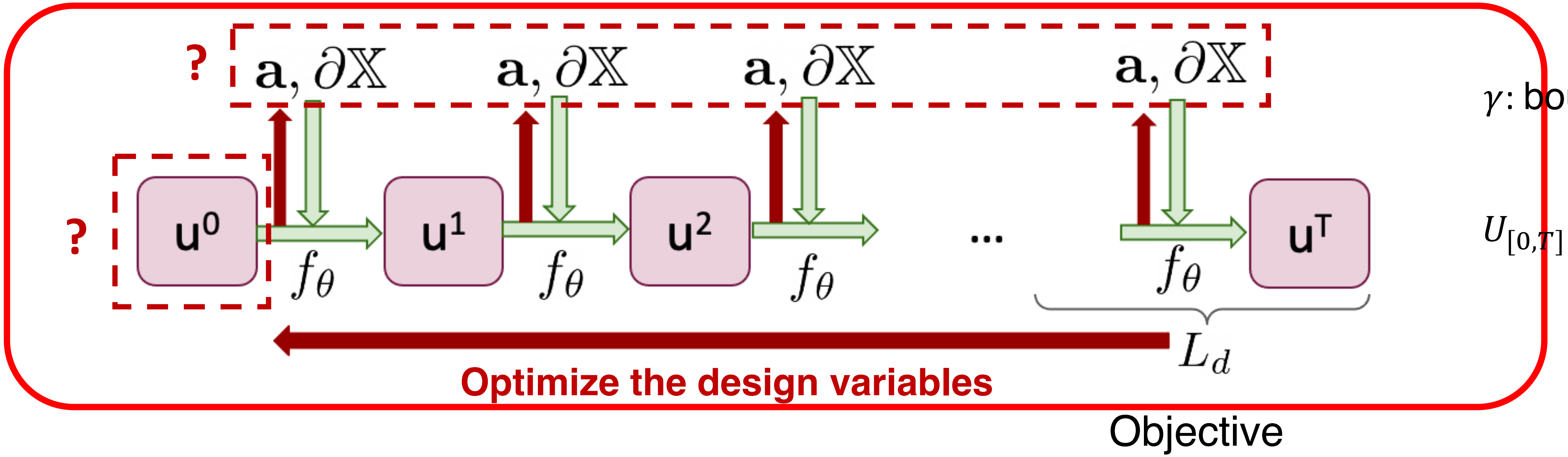
Application 3.1: Inverse problem

Given sparse observations, infer the full state, or parameters a



Holzschuh, Benjamin, Simona Vegetti, and Nils Thuerey. "Solving Inverse Physics Problems with Score Matching." *NeurIPS* 2023

Application 3.2: inverse design



γ: boundary condition

$U_{[0,T]}$: state sequence

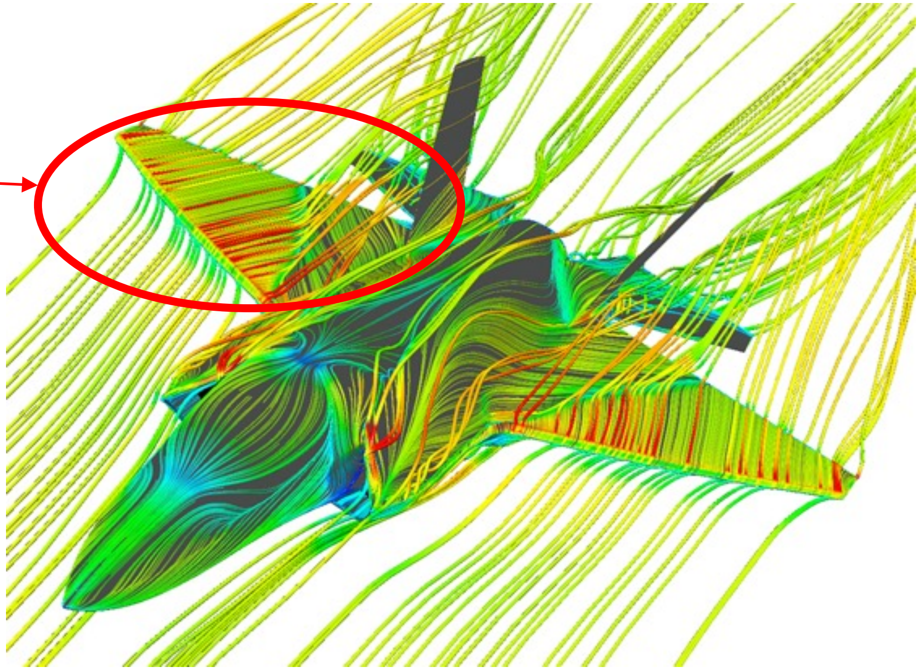
Treat all the variables as a single variable $(U_{[0,T]}, \gamma)$ and learn to generate simultaneously

Application 3.2: Compositional inverse design: definition

Given objective $J(U(\gamma),\gamma)$, find design parameters γ that minimize J, where the parameters γ and/or the state U are more complex than in training.

For example:
Training: we only see how the fluid interacts with each part of the airplane

Test: design the whole airplane shape



Application 3.2: CinDM method

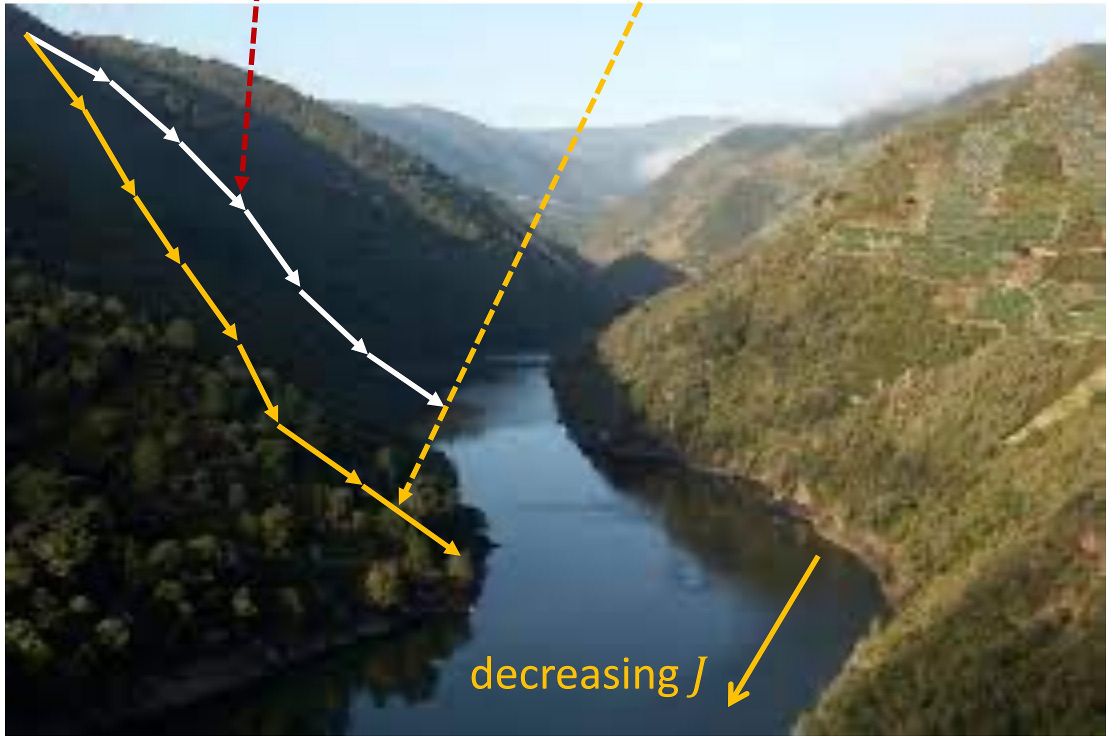
[1] **Wu, Tailin**, et al. "Compositional Generative Inverse Design." ICLR 2024 spotlight

$$\hat{\gamma} = \arg\min_{\gamma, U_{[0,T]}} \left[E_\theta(U_{[0,T]}, \gamma) + \lambda \cdot \mathcal{J}(U_{[0,T]}, \gamma) \right]$$

In inference, can also compose multiple E_θ on subsets of variables

Training: only learn E_θ

Inference: have an additional objective J

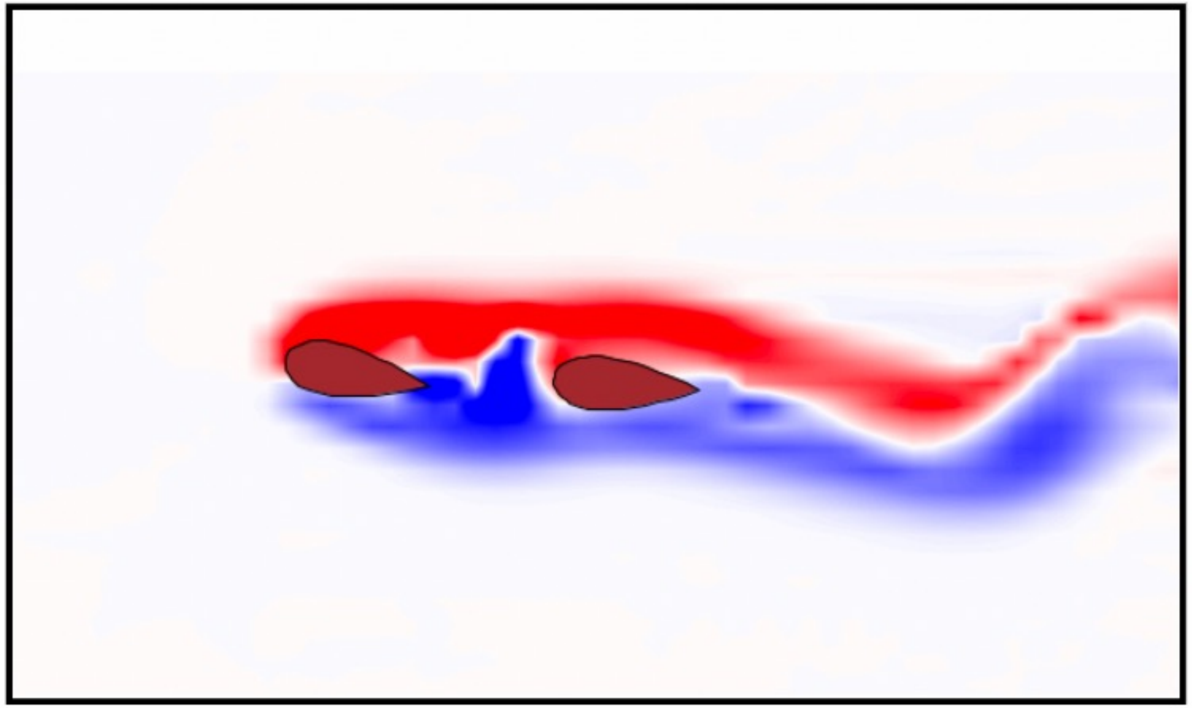


$U_{[0,T]}$: state sequence
γ: boundary condition

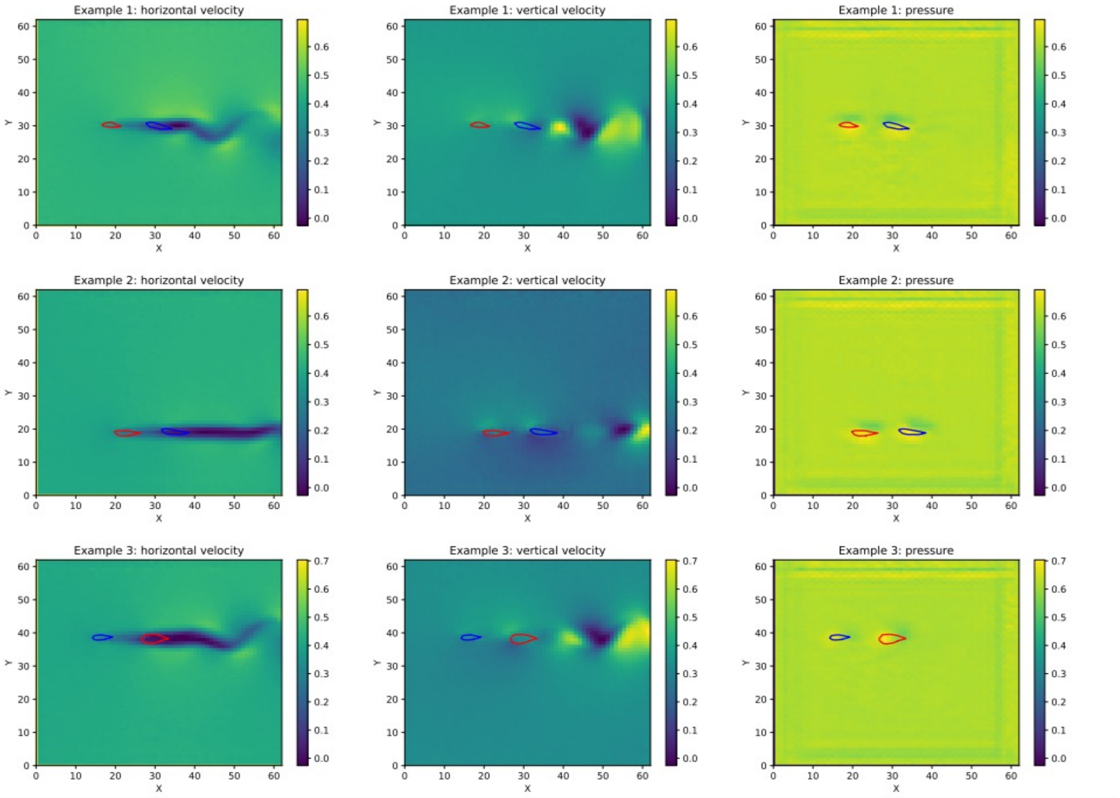
Application 3.2: CinDM method, part-to-whole generalization

Training: consider a **single airfoil** interacting with the air flow

Inference: consider **multiple airfoils**, maximize life-to-drag ratio: $(= \frac{lift}{drag})$

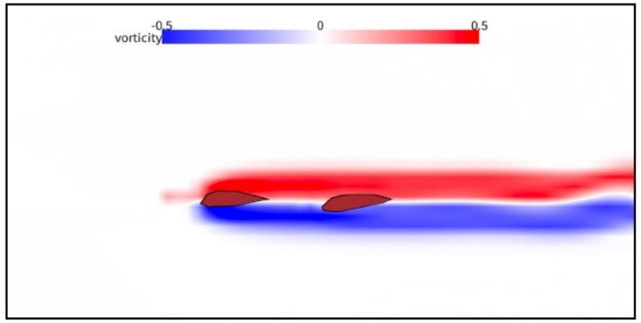


Example of Lily-Pad simulation

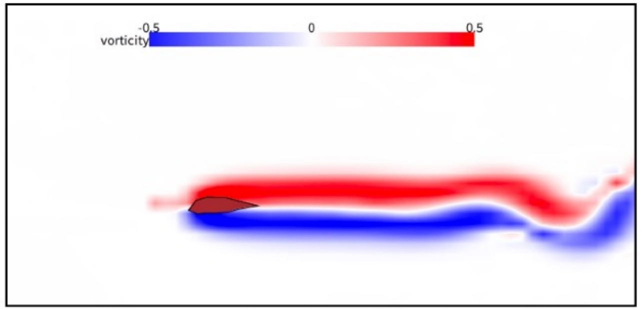


Compositional design results of our method in 2D airfoil generation. Each row represents an example. We show the heatmap of velocity in horizontal and vertical direction and pressure in the initial time step, inside which we plot the generated airfoil boundaries.

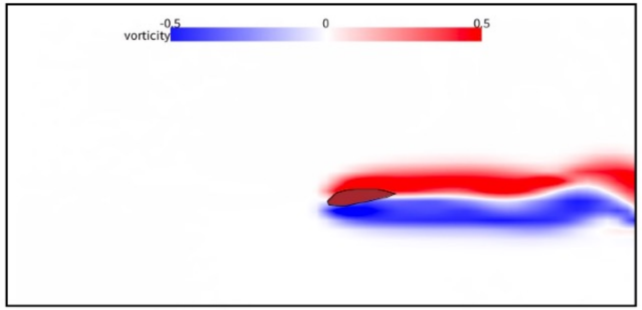
Application 3.2: CinDM method, part-to-whole generalization



(a) Formation flying of airfoils A and B



(b) Single flying of airfoil A

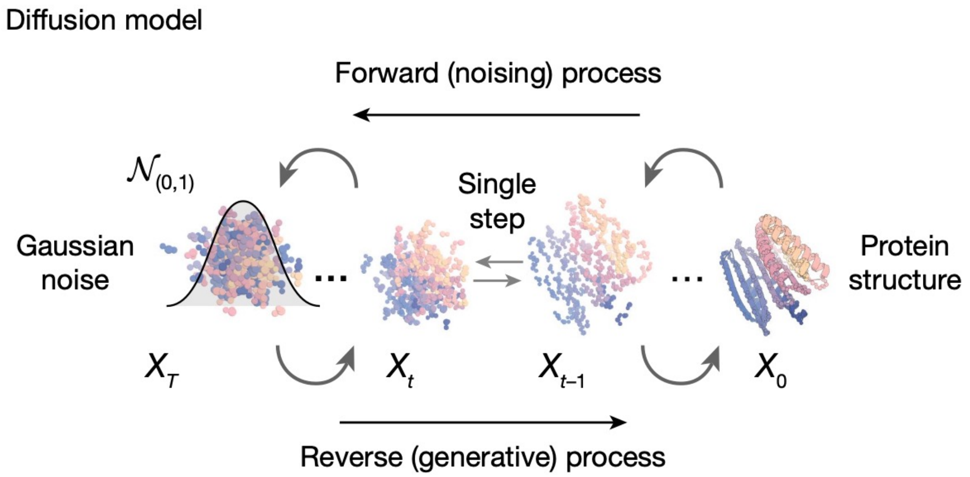


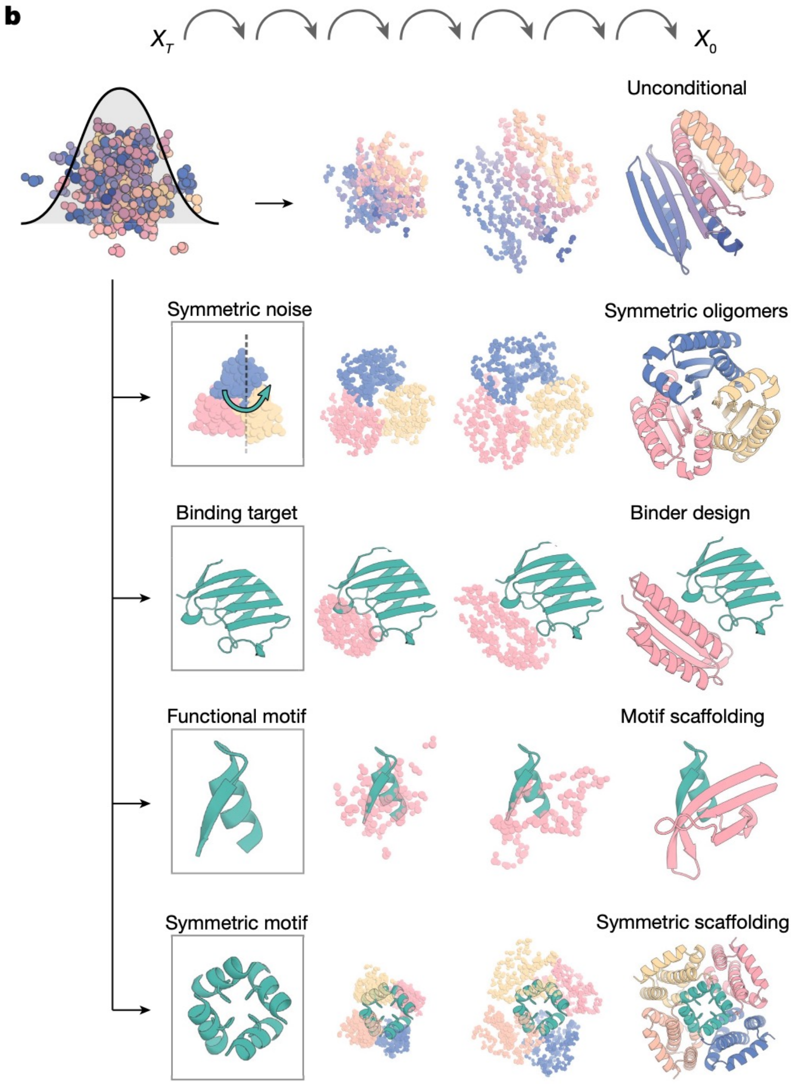
(c) Single flying of airfoil B

Our model discovers **formation flying（编队飞行）**
- Reducing the drag by 53.6%
- increasing the lift-to-drag ratio by 66.1%

Application 3.3: Protein design

De novo protein design [1]:

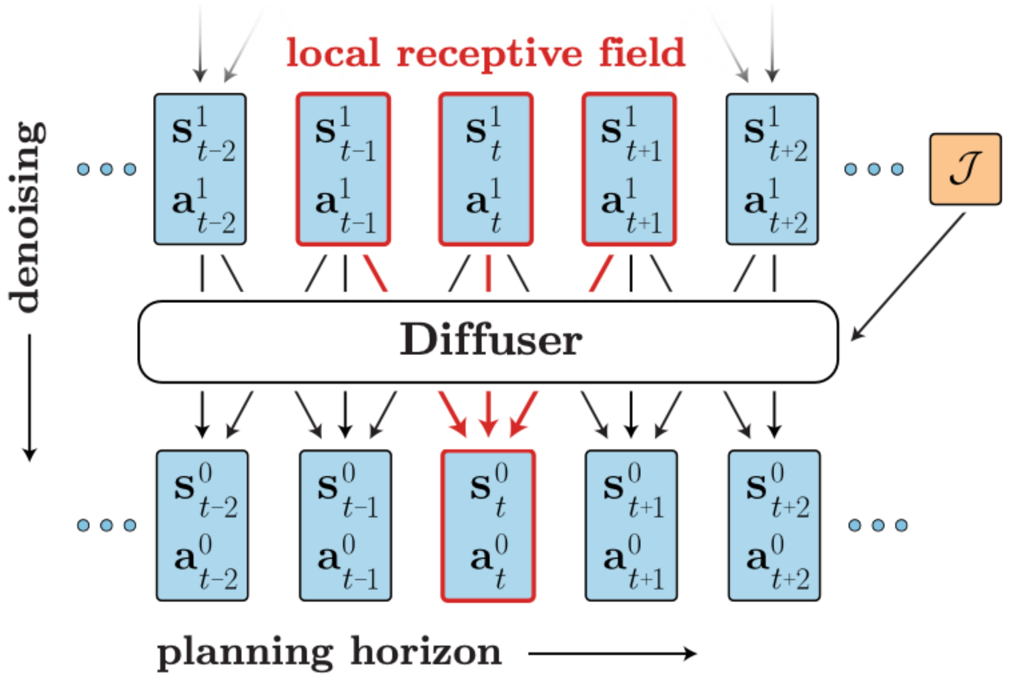


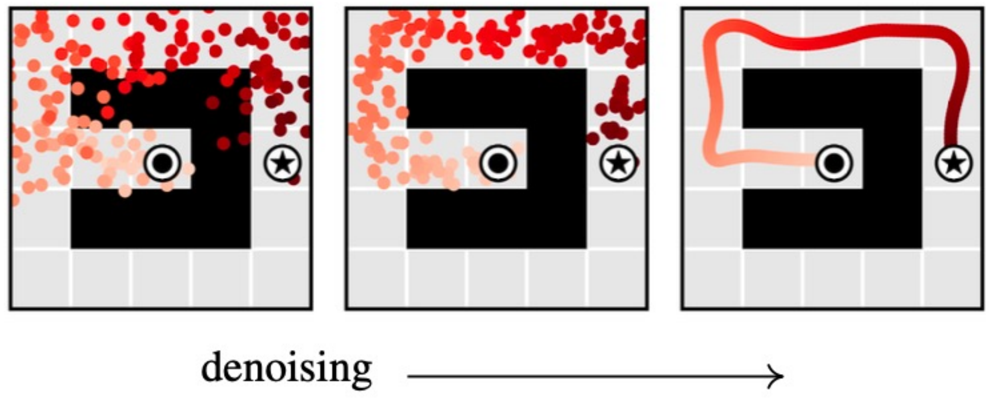


[1] Watson, Joseph L., et al. "De novo design of protein structure and function with RFdiffusion." *Nature* 620.7976 (2023): 1089-1100.

Application 4.1: Planning [1]

For agent interacting with an environment with action sequence $\{a_t\}$, envionrment state $\{s_t\}$:





[1] Janner, Michael, et al. "Planning with diffusion for flexible behavior synthesis." *ICML 2022*

Application 4.2: Diffusion Policy [1]

For agent interacting with an environment with action sequence $\{a_t\}$, envionrment state $\{s_t\}$:

[1] Chi, Cheng, et al. "Diffusion policy: Visuomotor policy learning via action diffusion." *arXiv preprint arXiv:2303.04137* (2023).

Other applications

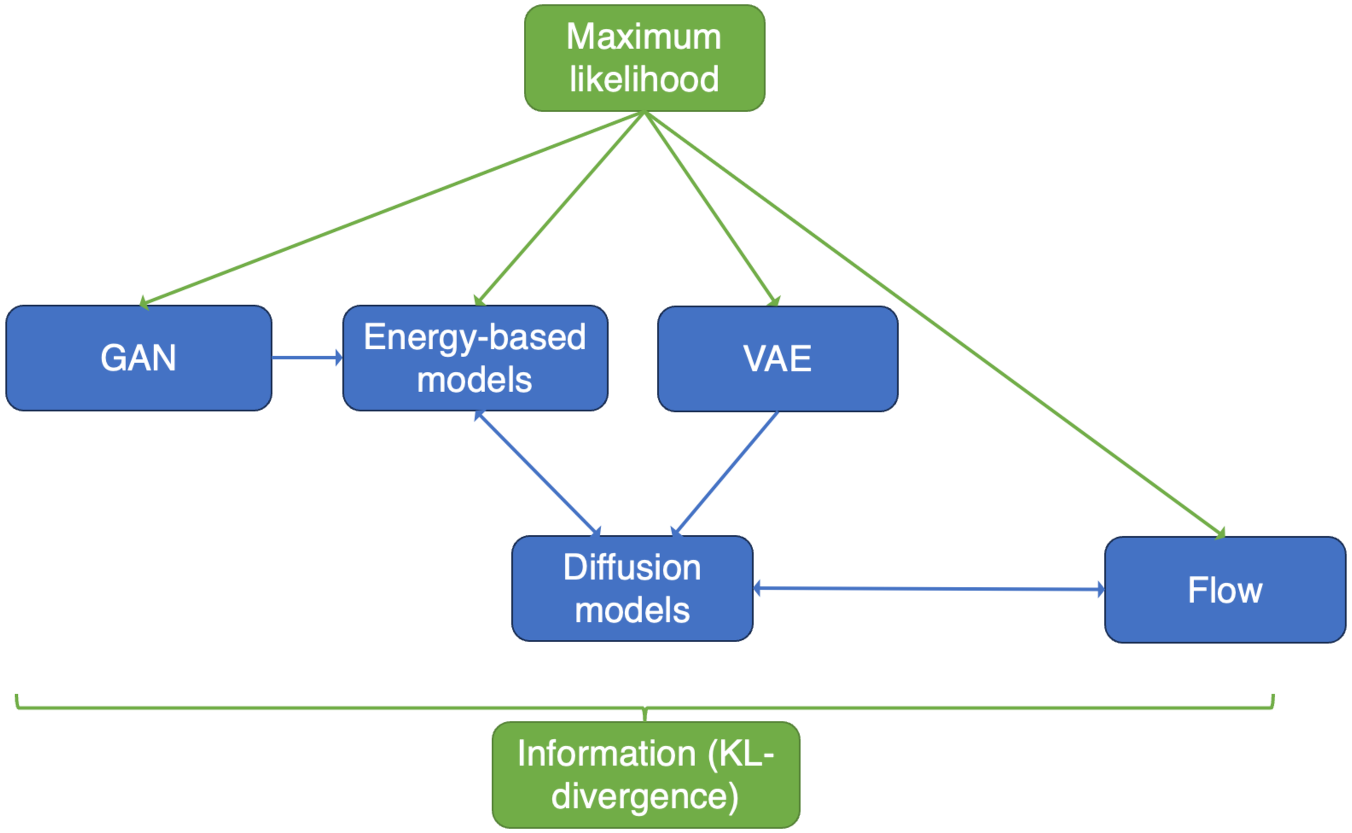
- **Algorithm:** Diffusion models as plug-and-play priors, NeurIPS 2022
- **Finetuning:** ControlNet: Adding Conditional Control to Text-to-Image Diffusion Models
- **Inpainting:** RePaint: Inpainting using Denoising Diffusion Probabilistic Models
- **Graph generation:** Autoregressive Diffusion Model for Graph Generation, ICML 2023
- **Antibody generation:** Antigen-Specific Antibody Design and Optimization with Diffusion-Based Generative Models for Protein Structures, NeurIPS 2022
- **Material generation:** Crystal Diffusion Variational Autoencoder for Periodic Material Generation, ICLR 2022
- **Composing video sequences:** Synthesizing Long-Term Human Motions with Diffusion Models via Coherent Sampling
- **Point cloud generation:** Point-E: A System for Generating 3D Point Clouds from Complex Prompts

Faster diffusion (1-8 steps):

- Progressive distillation for fast sampling of diffusion models, ICLR 2022
- On distillation of guided diffusion models, CVPR 2023
- SnapFusion: Text-to-Image Diffusion Model on Mobile Devices within Two Seconds, NeurIPS 2023

Summary

- Generative models
  - VAE
  - GAN
  - Energy-based models
  - Diffusion models
  - Flows
- Application of diffusion models
  - Image, video, and shape generation
  - Simulation
  - Inverse design/inverse problem
  - Control/planning



Useful materials

- Diffusion models (DDPM): "Denoising diffusion probabilistic models." NeurIPS 2020
- DDPM tutorial: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
- DDPM code: https://github.com/lucidrains/denoising-diffusion-pytorch
- Flow matching: "Flow matching for generative modeling." ICLR 2022