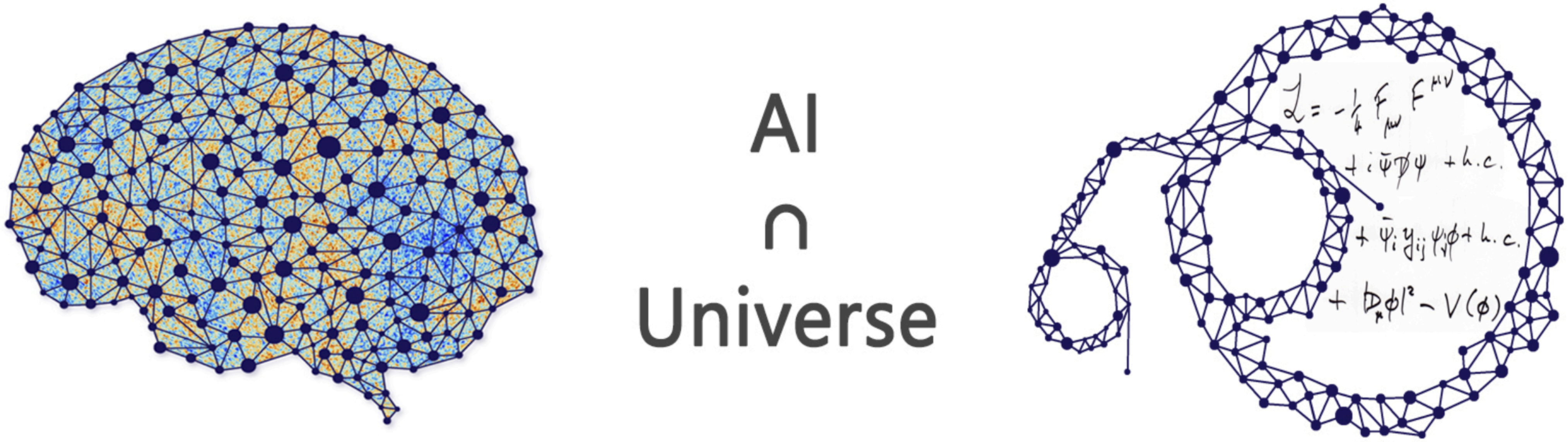
Physics 361 - Machine Learning in Physics

Lecture 22 – Diffusion Models

April 15th 2025


AI
∩
Universe
$\mathcal{L} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu}$
$+ i\bar{\psi} \not{D} \psi + h.c.$
$+ \bar{\psi}_i y_{ij} \psi_j \phi + h.c.$
$+ |D_\mu \phi|^2 - V(\phi)$

Moritz Münchmeyer

Recall Generative Models

Reference: Jared Kaplan’s lecture notes (see course website)

Unsupervised models

- In unsupervised learning we ask how can we **model the data distribution P(X)**?

- Note that a model for P(X) will allow us to sample from it, meaning that when we're doing UL, we are typically learning a generative model.

- P(X) could be the PDF of images of people for example. Not a prior a well-defined concept.

- Some generative models allow us "only" to sample (e.g. diffusion models), while others model the exact PDF (e.g. normalizing flows).

- There are also unsupervised algorithms that are not generative, e.g. k-means and t-SNE.

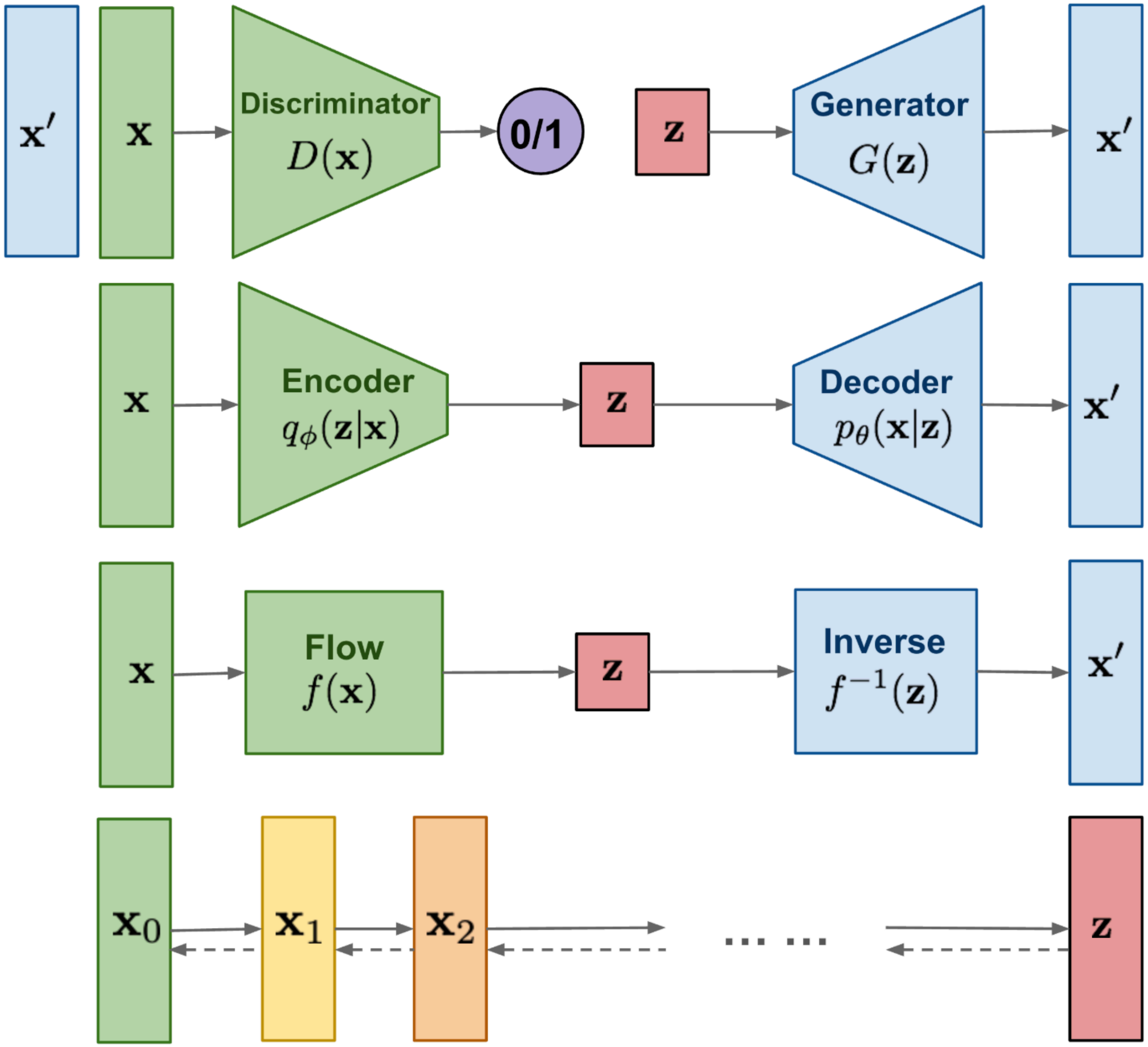
Common Generative Models

GAN: Adversarial training

VAE: maximize variational lower bound

Flow-based models: Invertible transform of distributions

Diffusion models: Gradually add Gaussian noise and then reverse



One can also add **autoregressive models such as transformer,** which we discussed before.

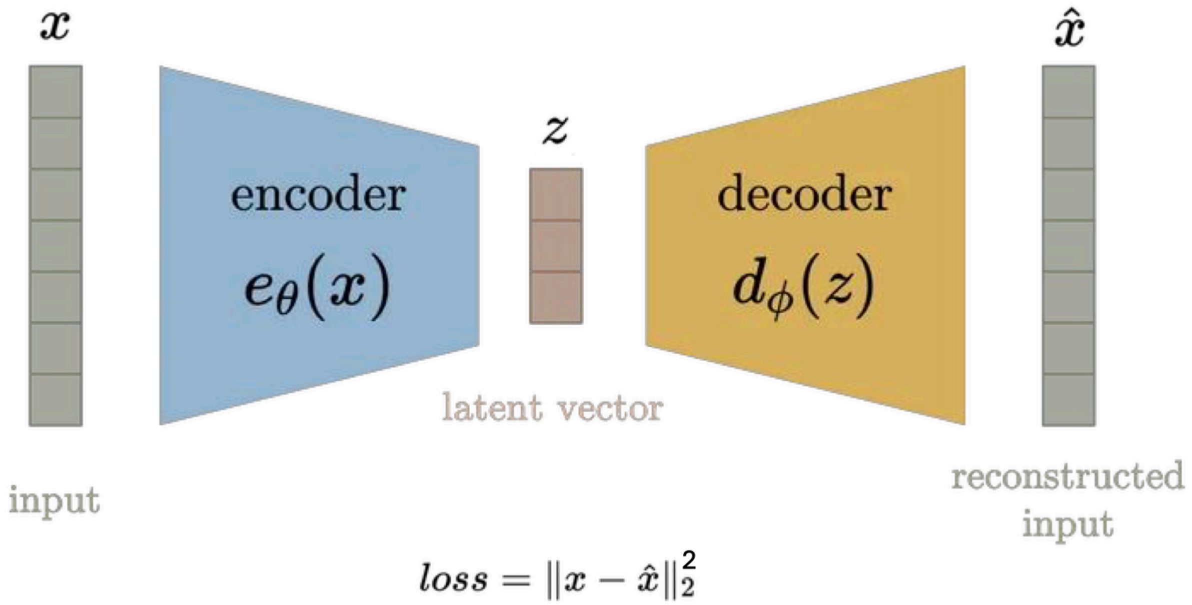
Figure credit: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Latent Space

- Underlying most approaches to UL is the idea of a 'latent space' where everything become simpler.

- Intuitively, 'understanding' something means knowing how to describe it efficiently. From this point of view, understanding has a lot to do with (mildly lossy) compression.

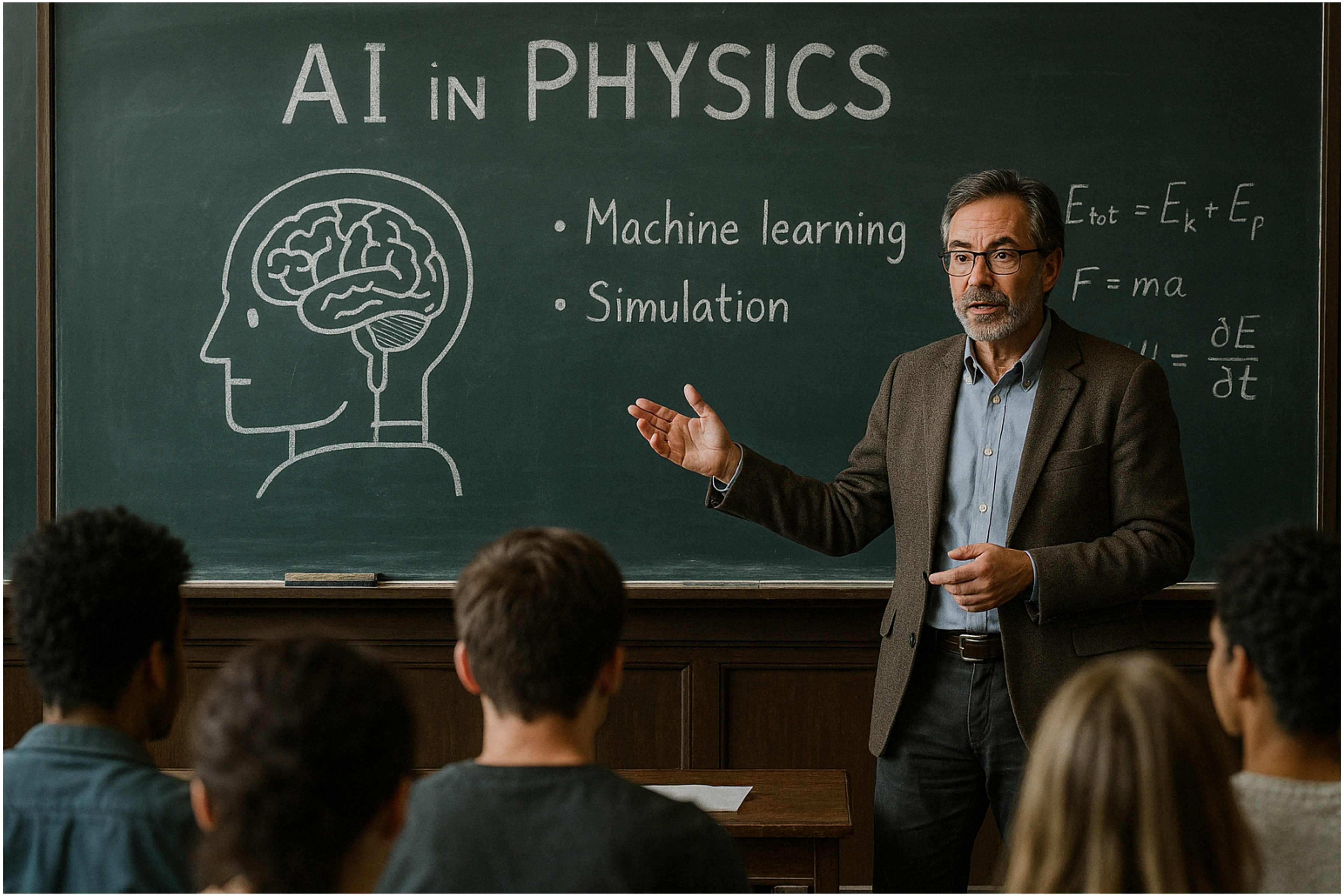
- Recall how this works for an auto-encoder:



- Many of the techniques we discuss are ways to learn better latent spaces.

Diffusion Models: DDPM

GPT4o: Make an image of a physics professor teaching a class about AI in physics. Make it photorealistic.



Importance of Diffusion Models

- The **landscape of generative models** is currently dominated by **transformers (for text)** and **diffusion models (for images, video)**. We already studied transformers, now let's talk about the latter.
- **Diffusion is a training process rather than an architecture**.
  - For example, OpenAI's Sora uses a Diffusion Transformer, i.e. the transformer model is used as a component of the diffusion model.
  - Diffusion **can be used to train many architectures as generative models: CNNs, transformers, graphs neural networks** etc.
- Diffusion models **have taken over GANs as the best performing models** / training process for generative models.
- They **scale well, are easy to parallelize, and are easier to train than GANs** (no mode collapse). But they are **computationally expensive to run**.

References

- This is a large topic that could easily cover a few weeks of classes. Much more details can be found here:
  - Bishop DL book https://www.bishopbook.com/ (main reference for this lecture)
  - https://arxiv.org/abs/2209.00796 Diffusion Models: A Comprehensive Survey of Methods and Applications
  - https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
  - Overview of diffusion model architectures: https://encord.com/blog/diffusion-models/

Figure credit: Akhil Premkumar, KICP

Latent space to target space

- Diffusion models are similar to normalizing flows and other generative models in that they **start with a simple (usually Gaussian) latent space distribution p(z)** and then **progressively deform it into a highly flexible distribution p(x)** of the data.

- This **deformation is done using a diffusion process (physics!)**.

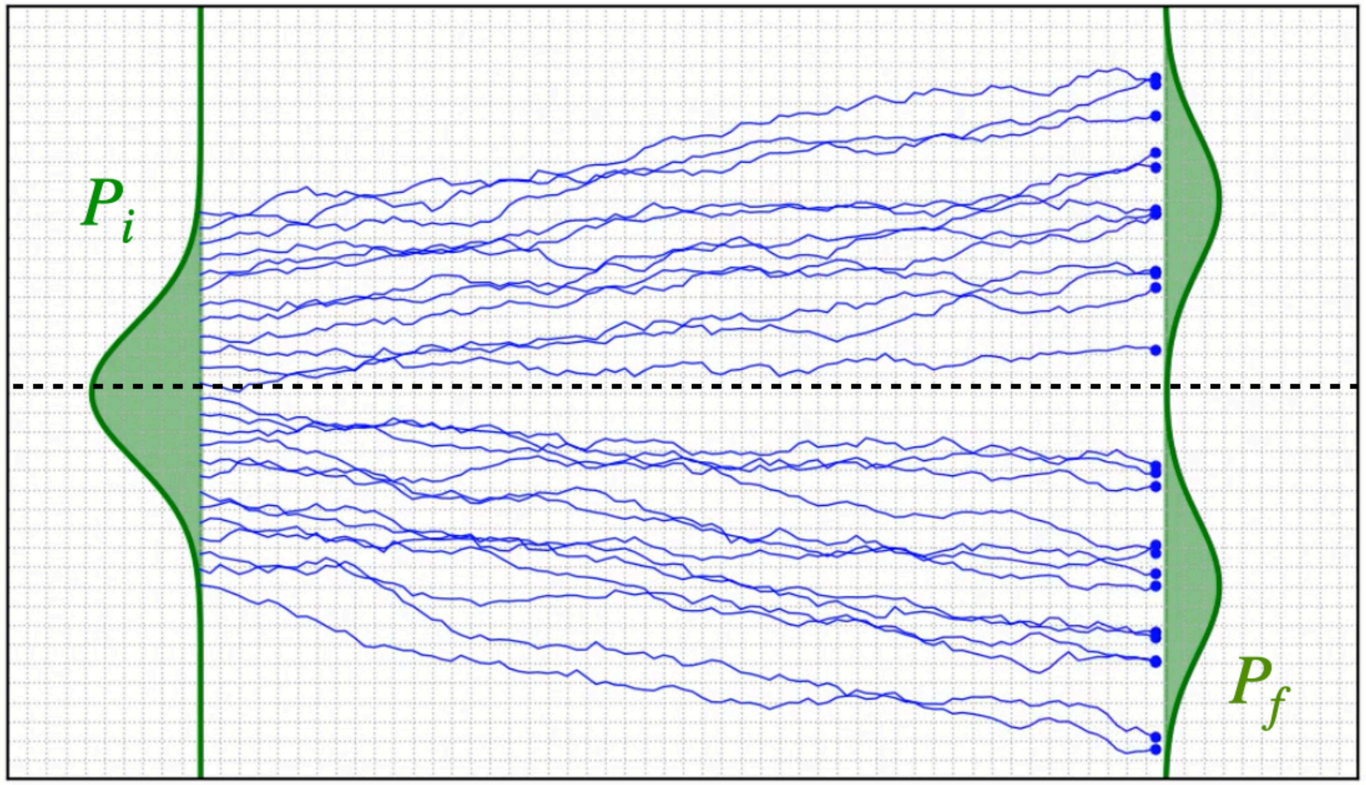


Figure credit: Akhil Premkumar, KICP

Original idea: https://arxiv.org/abs/1503.03585 **Deep Unsupervised Learning using Nonequilibrium Thermodynamics**

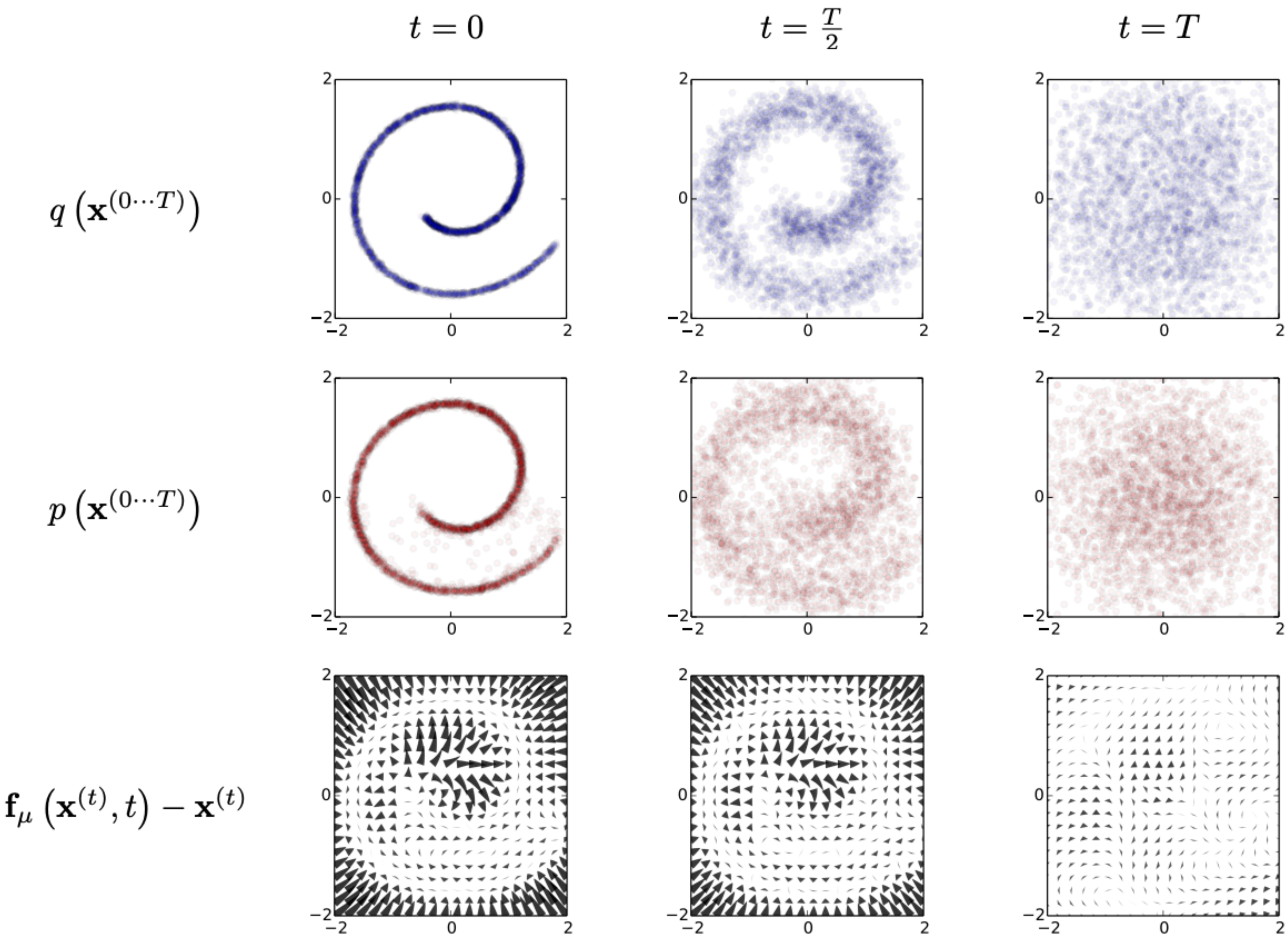
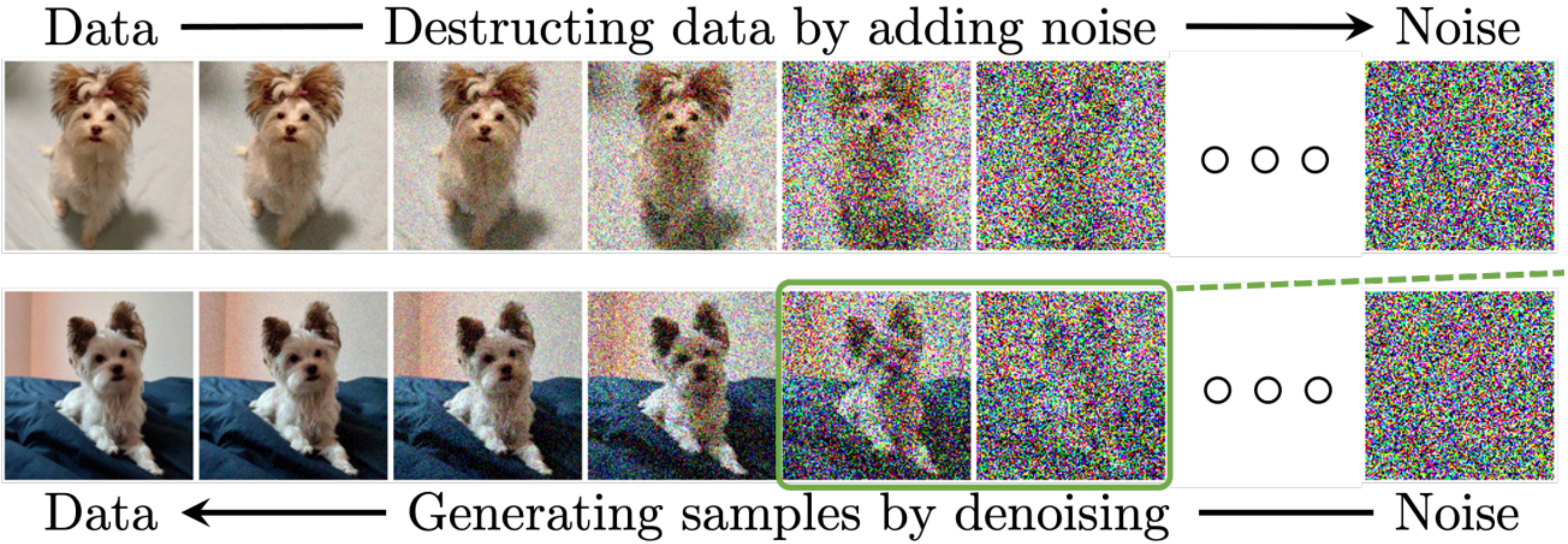


Figure 1. The proposed modeling framework trained on 2-d swiss roll data. The top row shows time slices from the forward trajectory $q\left(\mathbf{x}^{(0\cdots T)}\right)$. The data distribution (left) undergoes Gaussian diffusion, which gradually transforms it into an identity-covariance Gaussian (right). The middle row shows the corresponding time slices from the trained reverse trajectory $p\left(\mathbf{x}^{(0\cdots T)}\right)$. An identity-covariance Gaussian (right) undergoes a Gaussian diffusion process with learned mean and covariance functions, and is gradually transformed back into the data distribution (left). The bottom row shows the drift term, $\mathbf{f}_\mu\left(\mathbf{x}^{(t)}, t\right) - \mathbf{x}^{(t)}$, for the same reverse diffusion process.

DDPM

- We will focus on the most popular version of diffusion models, “denoising diffusion probabilistic models” (DDPM).

- Pictorially the process works as follows


Data
Destructing data by adding noise
Noise
Data
Generating samples by denoising
Noise

- The noise to data (denoising) process is **learned by a neural network (e.g. a U-Net)**, which is applied many times (roughly 100 to 1000 times).

https://arxiv.org/pdf/2209.00796.pdf

Forward Diffusion Process

- Given a data point sampled from a real data distribution $x_0 \sim q(x)$, we define a **forward diffusion process** in which we add small amounts of Gaussian noise to the sample in T steps, producing a sequence of noisy samples $x_1, \ldots, x_T$.

- The transition probability is

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t I)$$

$$q(x_{1:T} | x_0) = \prod_{t=1}^{T} q(x_t | x_{t-1})$$

- The β_t parameter, $\beta_t \ll 1$, controls the amount of noise and there are different possible values (noise schedules).

- It is possible sample x_t at any arbitrary time step t in a closed form due to the properties of Gaussians:

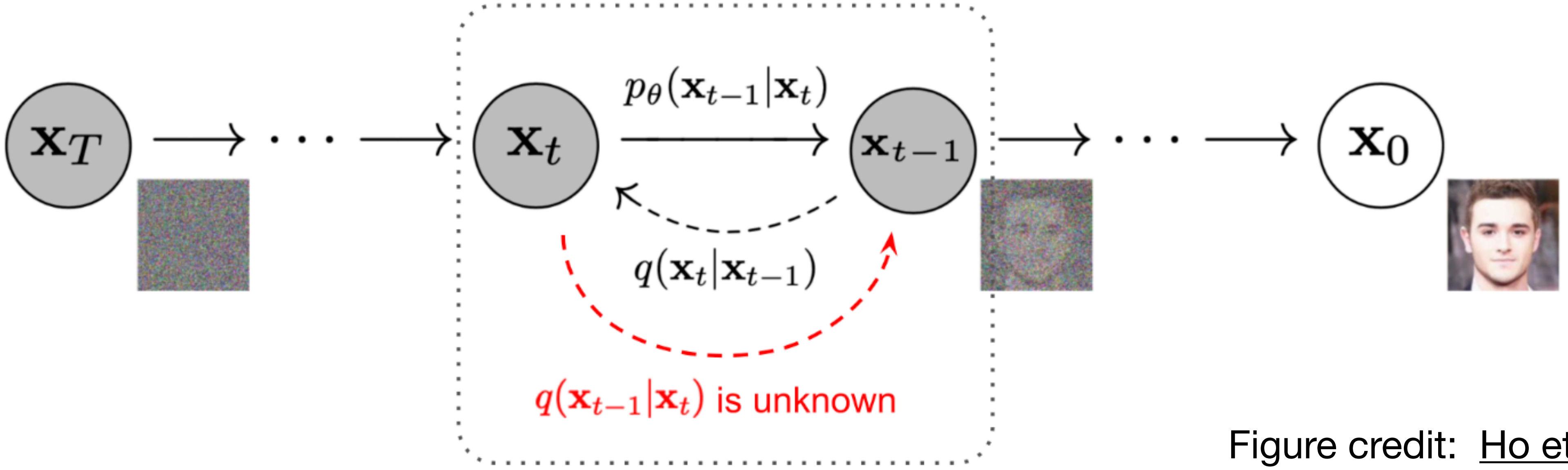
$$q(x_t | x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} x_0, (1-\bar{\alpha}_t) I) \qquad \text{and thus} \qquad x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon$$

random Gaussian noise vector

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

Reverse Diffusion Process

- To reverse the process we would need to know the transition probabilities $q(x_{t-1} \,|\, x_t)$. They cannot be directly calculated. Instead we learn a model p_θ to approximate these conditional probabilities.



$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown

Figure credit: Ho et al. 2020

- The reverse transition probabilities for $\beta_t \ll 1$ are also Gaussian (proof left out)

$$p_\theta(x_{t-1} \,|\, x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Learn denoised mean with a neural network

but now we need to parametrize the mean and covariance with a deep neural network.

Variational Lower Bound (VLB)

- Training the neural network that parametrizes the reverse diffusion process is somewhat involved.

- The obvious training objective would be negative log-likelihood $-\log p_\theta(\mathbf{x}_0)$ where x_0 is the data. Unfortunately this involves an intractable integral over all diffusion trajectories.

- So instead one uses a related quantity called the **evidence lower bound (ELBO) or variational lower bound (VLB)**. The VLB can be evaluated by sampling over the training set.

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\le -\log p_\theta(\mathbf{x}_0) + \overbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0)\|p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0))}^{\text{KL-divergence of the two Markov chains}\ \geq 0} \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\Big] \\
&= -\log \cancel{p_\theta(\mathbf{x}_0)} + \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log \cancel{p_\theta(\mathbf{x}_0)}\Big] \\
&= \mathbb{E}_q\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big]
\end{aligned}
$$

can be evaluated from training samples

$$\text{Let } L_{\mathrm{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\Big[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\Big] \geq -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)$$

$$D_{\mathrm{KL}}(q\|p) = \mathbb{E}_q\Big[\log \frac{q}{p}\Big]$$

$$p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0) = \frac{p_\theta(\mathbf{x}_{0:T})}{p_\theta(\mathbf{x}_0)}$$

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$$

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$$

Evaluating the VLB

$$\mathcal{L}_{\text{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right] \quad \Longrightarrow \quad \mathcal{L}_{\text{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log\left(\frac{p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{\prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\right)\right]$$

$$\Longrightarrow \quad \mathcal{L}_{\text{VLB}} = \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log p(\mathbf{x}_T) + \sum_{t=1}^{T}\left(\log p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) - \log q(\mathbf{x}_t \mid \mathbf{x}_{t-1})\right)\right]$$

This expectation is estimated using Monte Carlo sampling from the **forward process**, which is tractable since $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ is known.

Algorithm

1. **Sample** $\mathbf{x}_0 \sim \text{data}$.
2. **Sample** $\mathbf{x}_{1:T} \sim q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ (i.e. run the forward process).
3. For each $t \in \{1, \ldots, T\}$:
    - Compute $\log p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$
    - Compute $\log q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$
4. Add $\log p(\mathbf{x}_T)$, typically:

$$\log p(\mathbf{x}_T) = \log \mathcal{N}(\mathbf{x}_T; 0, \mathbf{I})$$

5. Sum the result and average over a batch to approximate the expectation.

Recall the **forward process**:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\,\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$$

We parameterize the **reverse diffusion** as a Gaussian with learned means and variances:

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(t))$$

Loss function

- Starting from the general equation for the VLB on the last slide there are a number of analytic tricks and simplifications that people use to arrive at **the loss function usually used in practice**. The steps are written out for example here: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

- It turns out that it is somewhat easier to **predict the noise** ϵ in an image rather than to predict the de-noised mean μ.

- While the calculations are cumbersome, the final simplified result is intuitive:

sample time steps, training examples, noise

train NN to estimate that noise

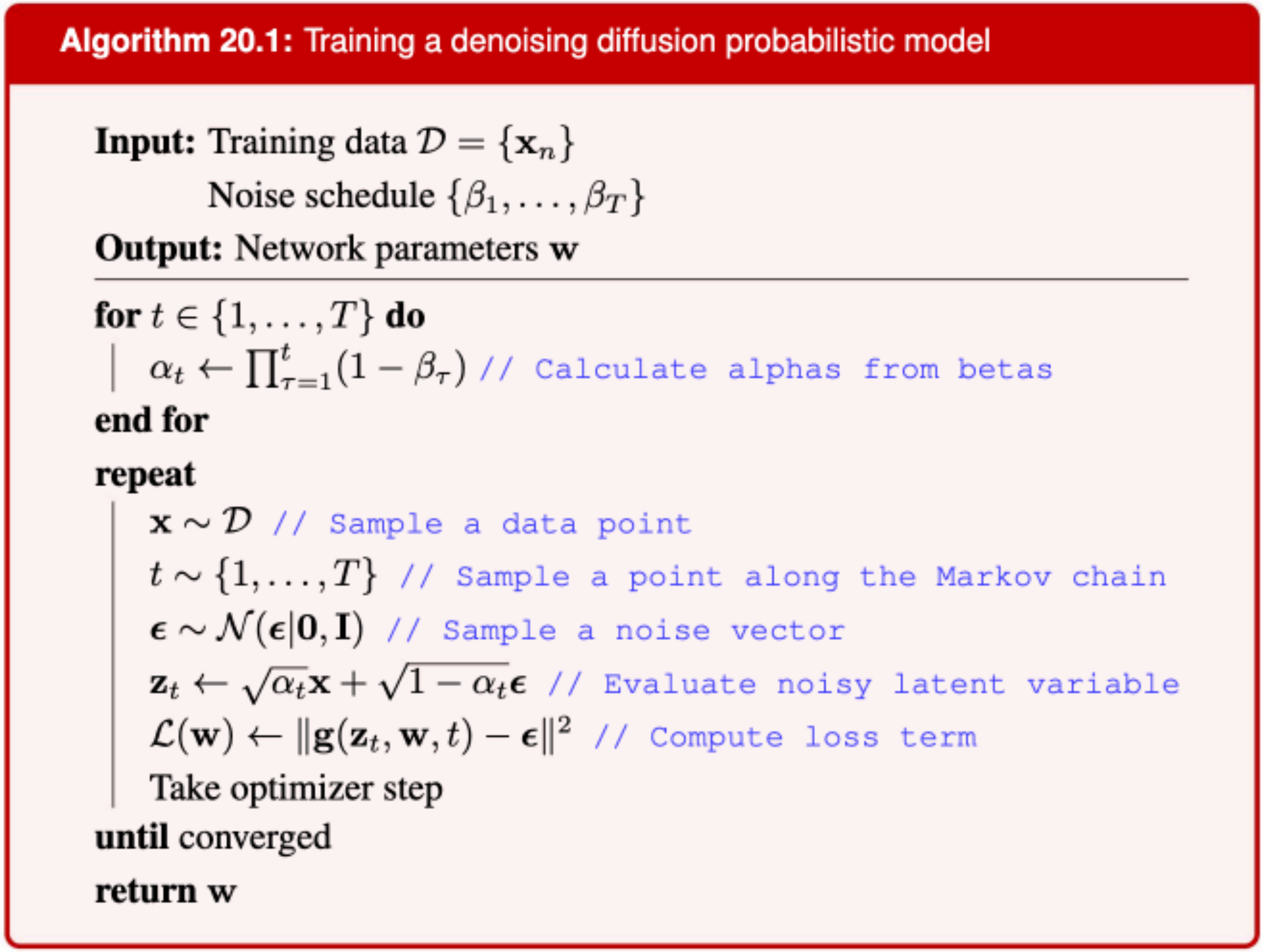
$$
\begin{aligned}
L_t^{\text{simple}} &= \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[\,\left\|\,\epsilon_t - \epsilon_\theta(x_t, t)\,\right\|^2\right] \\
&= \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[\,\left\|\,\epsilon_t - \epsilon_\theta\left(\underbrace{\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t, t}_{\text{can calculate noisy image at time t in one step}}\right)\right\|^2\right]
\end{aligned}
$$

The model is trained to **match the true noise ϵ_t** with the predicted noise, using MSE:

Training algorithm

w: NN weights

g: NN output



Algorithm 20.1: Training a denoising diffusion probabilistic model

```
Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$
       Noise schedule $\{\beta_1, \dots, \beta_T\}$
Output: Network parameters $\mathbf{w}$
for $t \in \{1, \dots, T\}$ do
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ // Calculate alphas from betas
end for
repeat
    $\mathbf{x} \sim \mathcal{D}$ // Sample a data point
    $t \sim \{1, \dots, T\}$ // Sample a point along the Markov chain
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
    $\mathbf{z}_t \leftarrow \sqrt{\alpha_t}\mathbf{x} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}$ // Evaluate noisy latent variable
    $\mathcal{L}(\mathbf{w}) \leftarrow \|\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t) - \boldsymbol{\epsilon}\|^2$ // Compute loss term
    Take optimizer step
until converged
return $\mathbf{w}$
```

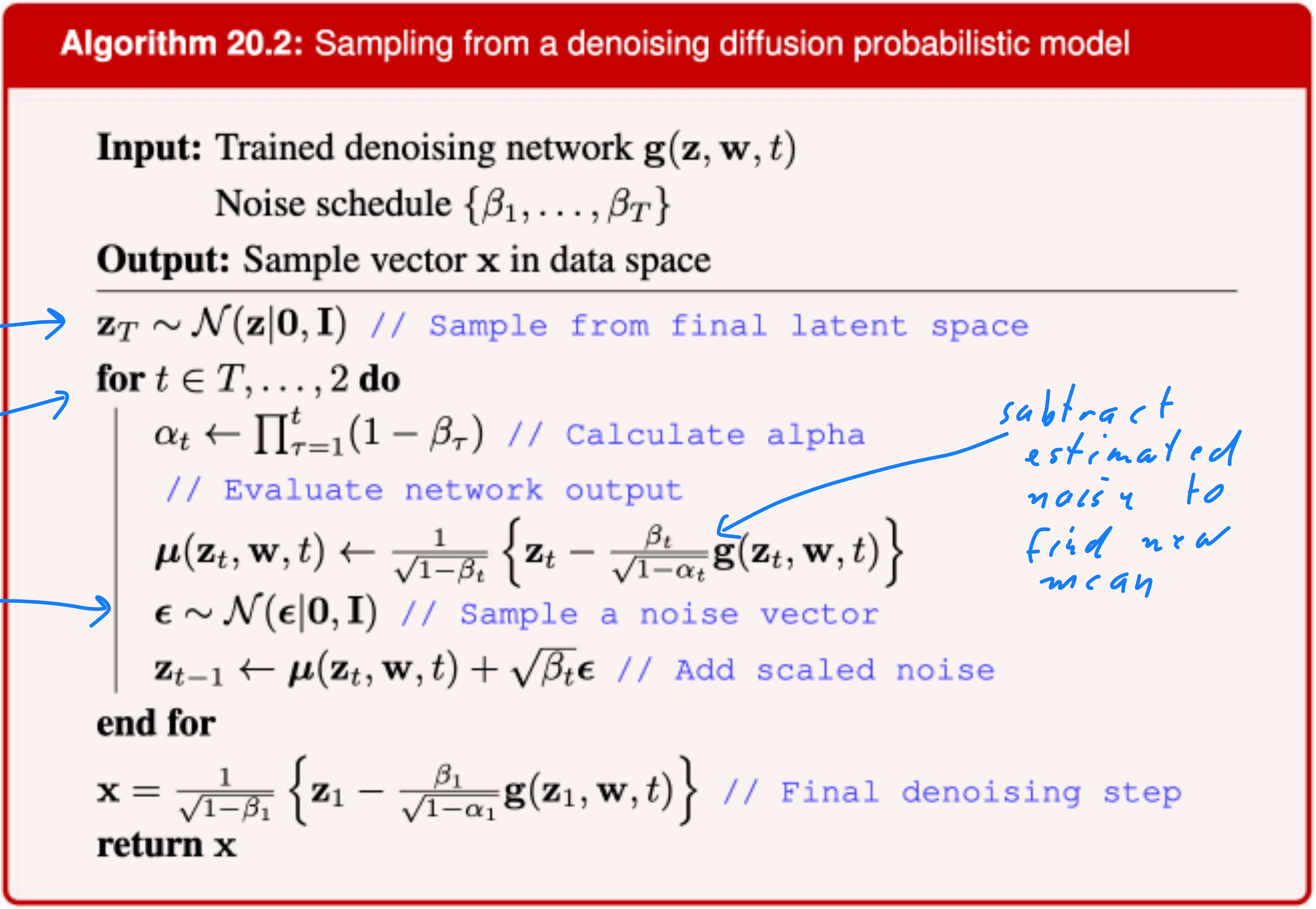
Figure credit: Bishop Deep Learning

Sampling algorithm

initial random sampling

~1000 steps

random sampling at each time step



Algorithm 20.2: Sampling from a denoising diffusion probabilistic model

Input: Trained denoising network $\mathbf{g}(\mathbf{z}, \mathbf{w}, t)$
Noise schedule $\{\beta_1, \ldots, \beta_T\}$

Output: Sample vector $\mathbf{x}$ in data space

```
$\mathbf{z}_T \sim \mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{I})$ // Sample from final latent space
for $t \in T, \ldots, 2$ do
    $\alpha_t \leftarrow \prod_{\tau=1}^{t}(1 - \beta_\tau)$ // Calculate alpha
    // Evaluate network output
    $\boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{z}_t, \mathbf{w}, t)\right\}$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{\epsilon}|\mathbf{0}, \mathbf{I})$ // Sample a noise vector
    $\mathbf{z}_{t-1} \leftarrow \boldsymbol{\mu}(\mathbf{z}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$ // Add scaled noise
end for
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\left\{\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}}\mathbf{g}(\mathbf{z}_1, \mathbf{w}, t)\right\}$ // Final denoising step
return $\mathbf{x}$
```

subtract estimated noise to find new mean

Figure credit: Bishop Deep Learning

w: NN weights

g: NN output

Why does diffusion work?

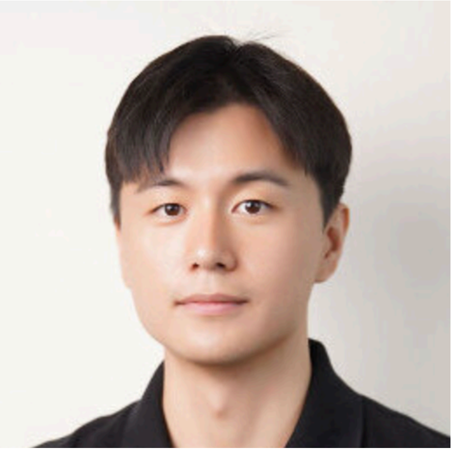
- Why do we need to run diffusion in many small steps rather than one large one?
    - From the original paper: **“The essential idea, inspired by non-equilibrium statistical physics, is to systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process.”** https://arxiv.org/pdf/1503.03585.pdf
    - If we were to add all the noise in one step, it would be akin to destroying all the data's structure immediately, which is very difficult to reverse. By adding noise slowly, the model learns a **series of simpler denoising steps**, which together can effectively reconstruct the original data from noise.
    - Adding noise gradually helps maintain the stability of the training process. **Abrupt changes can lead to training instabilities, while gradual changes allow the model to adapt slowly** and steadily.

Example from my own research: Super-resolution Emulator

Super-Resolution Emulation of Large Cosmological Fields with a 3D Conditional Diffusion Model https://arxiv.org/abs/2311.05217

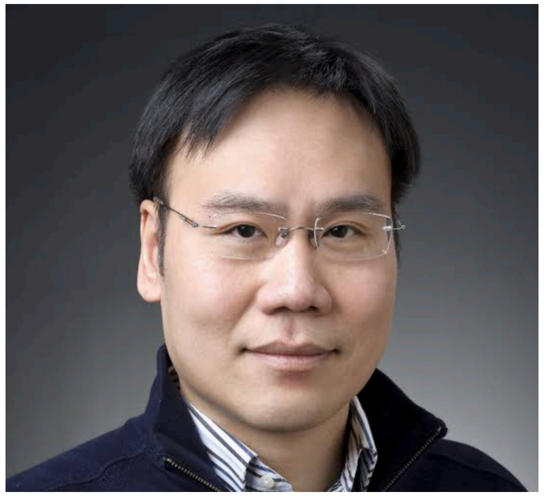
Collaborators:

Adam Rouhiainen,
UW Madison Physics



Prof. Kangwook Lee,
UW Madison ECE & CS

Michael Gira,
UW Madison ECE & CS,
Microsoft



Prof. Gary Shiu,
UW Madison Physics

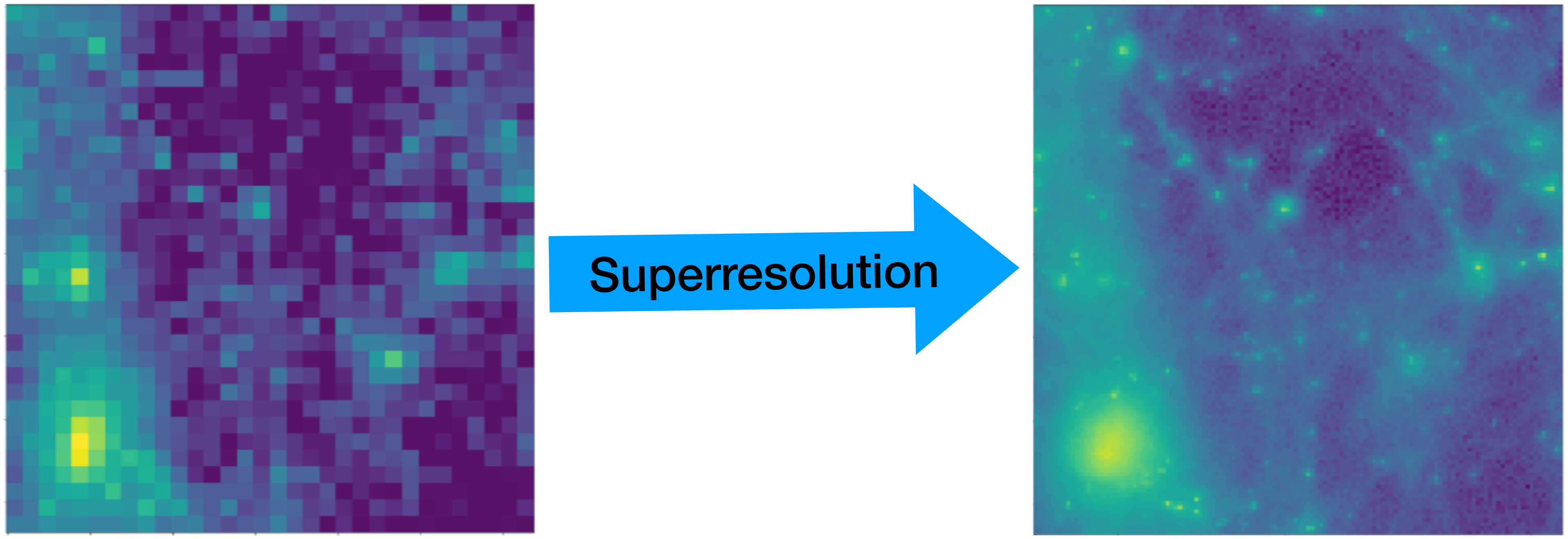
Super-resolution Emulators

- Cosmology is increasingly simulation-driven. Simulations are required for theoretical studies, statistical method development and parameter inference.

- **High-resolution baryonic hydro-simulations** are computationally extremely expensive. Not possible on a realistic survey volume.

- **Low-resolution dark matter simulations** on the other hand are cheap to make on large volumes

- Idea of **super-resolution (SR) emulators**: Run low-res (LR) dark matter sim and upgrade to high-res (HR) hydro simulation with a generative neural network.

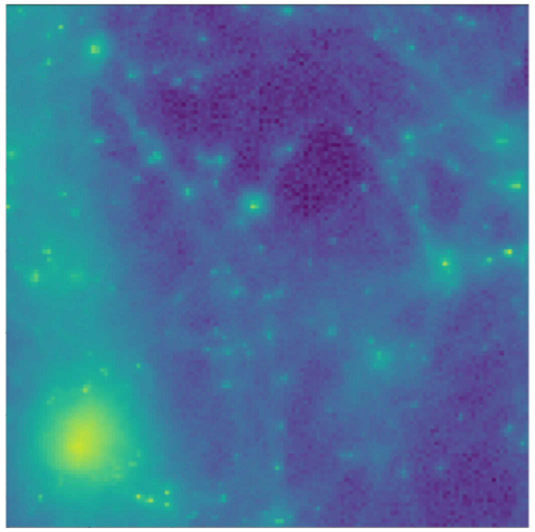


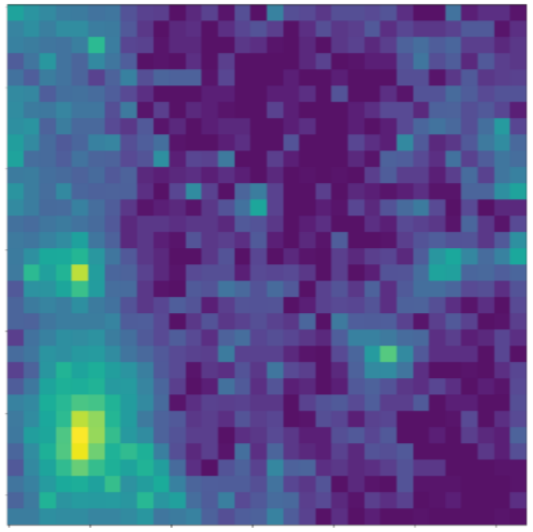
Superresolution

Conditional diffusion for Super-resolution

- A **conditional flow or conditional diffusion model** can learn how small-scale structure reacts to large-scale structure, probabilistically, at field level.

$$P(\text{small-scale structure} \,|\, \text{large-scale structure})$$





Here: High resolution IllustrisTNG gas density

Low resolution IllustrisTNG DM

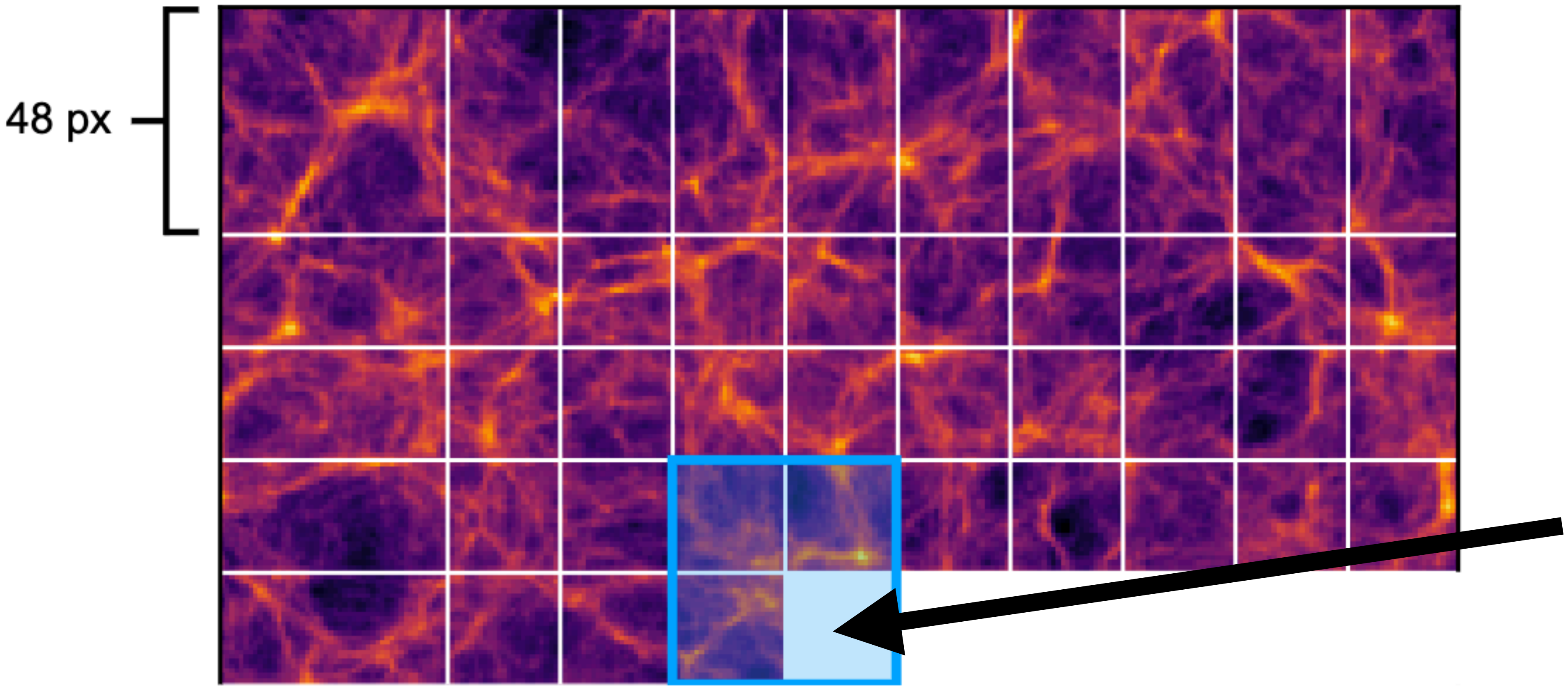
- We started with the normalizing flows. However we found that existing **NFs in 3d are not expressive enough, so we switched to diffusion models**.

Outpainting (Conditional Patching)

- Important ingredient: Locality of structure formation. **Need only small volume HR training simulations to learn the complicated hydro physics**.

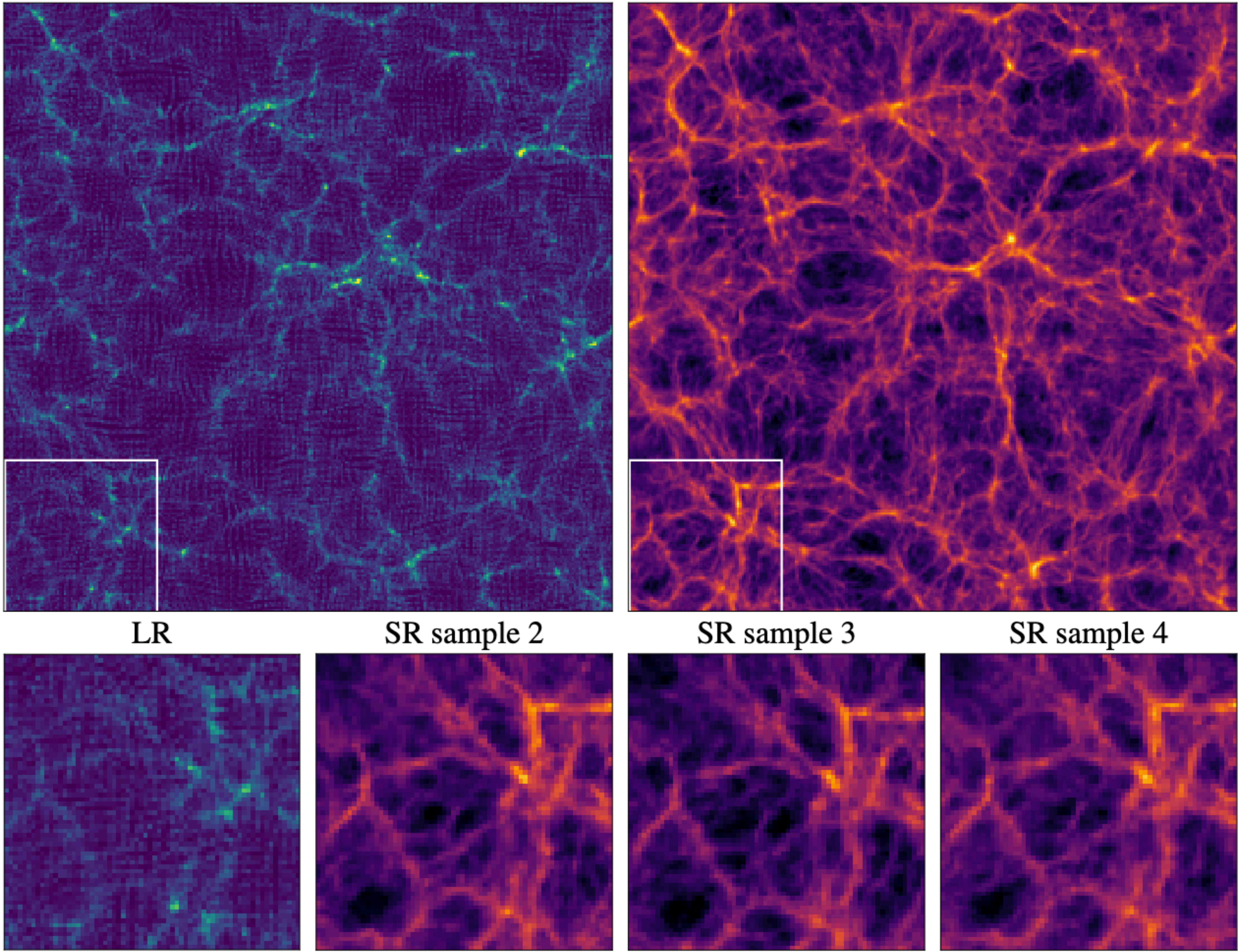
- We developed a 3d **"outpainting" procedure to make in principle arbitrarily large SR simulations with smooth patching.**

$P($high resolution | low resolution, neighboring high resolution$)$



New super-resolution region gets generated conditioned on low resolution map and neighbour regions.

Results



Low-resolution conditional

Super-resolution model output, sample 1

LR

SR sample 2

SR sample 3

SR sample 4

Details of our Setup

- **Model:**
  - For the model we use a **DDPM** based on the **Palette image-to-image code** which we generalized from 2d to 3d.
  - The de-noising is learned by a U-Net with added self-attention layers, with about 30 million parameters in total. Training ~3 days.
  - We use the standard DDPM sampler with 2000 steps.
- **Training data:**
  - High res: **Illustris-TNG 300** baryon density, sampled on 264^3 px cube.
  - Low res: Custom **AREPO dark matter simulation** using the same initial conditions as Illustris-TNG.
- Test data: New **low-res AREPO dark matter** simulations with different initial conditions.

U-Net architecture for de-noising

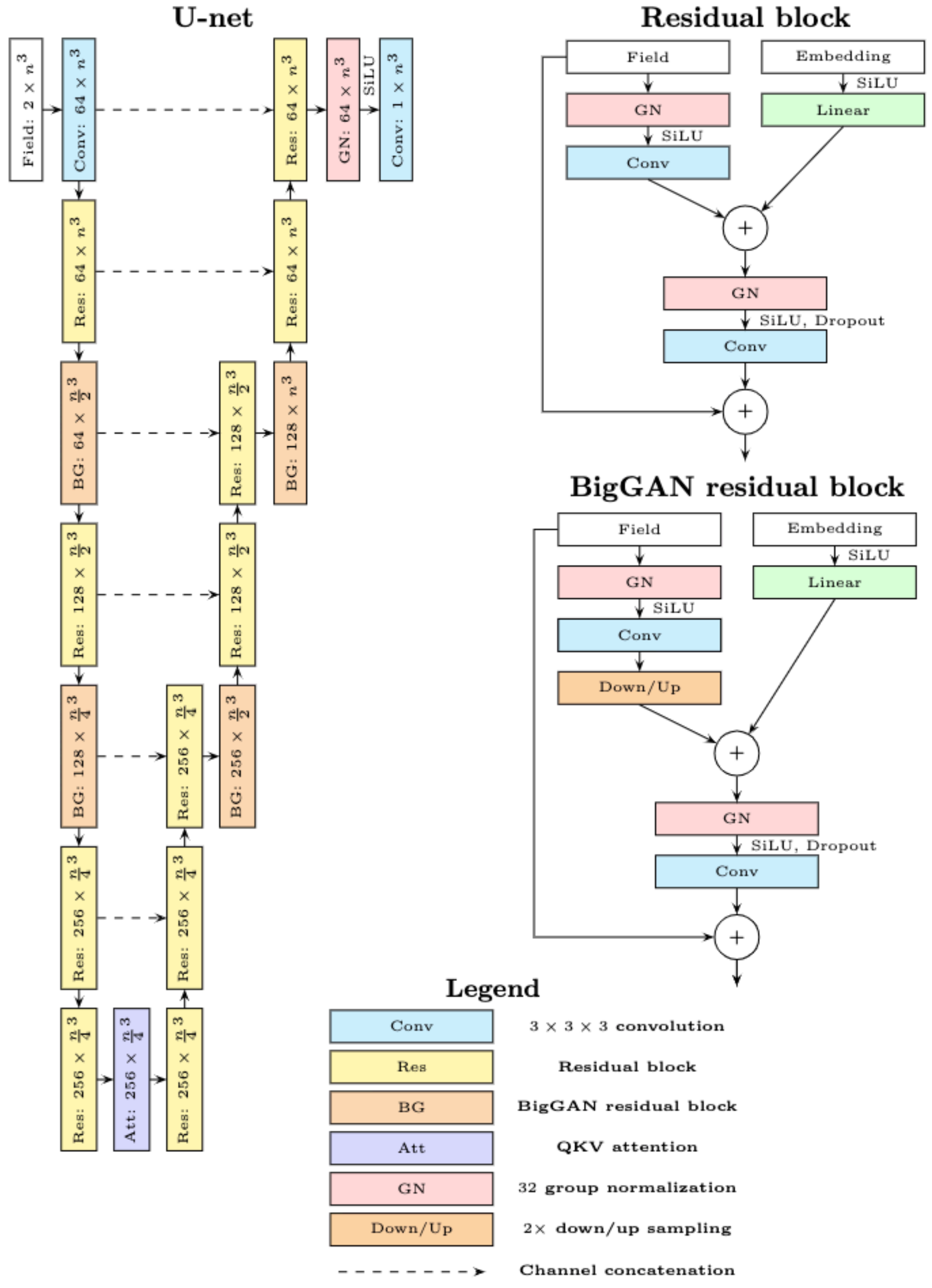
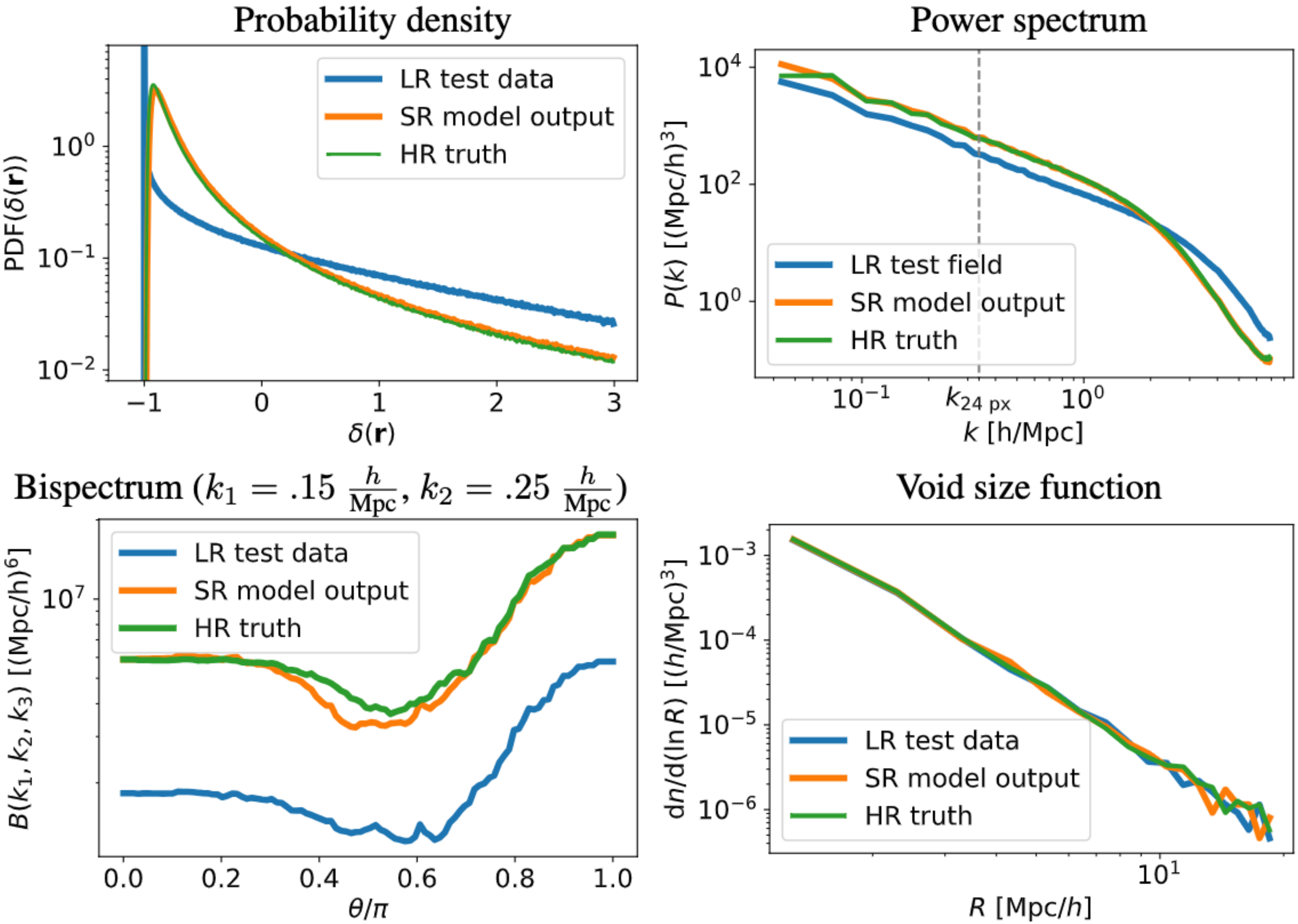


FIG. 11. The U-net used in this work (left) is made of residual blocks (top right), with BigGAN residual blocks (center right) used to downsample and upsample the fields. Shown after every layer name in the U-net is the layer's output channels × volume.

Results: SR matches HR on validation data



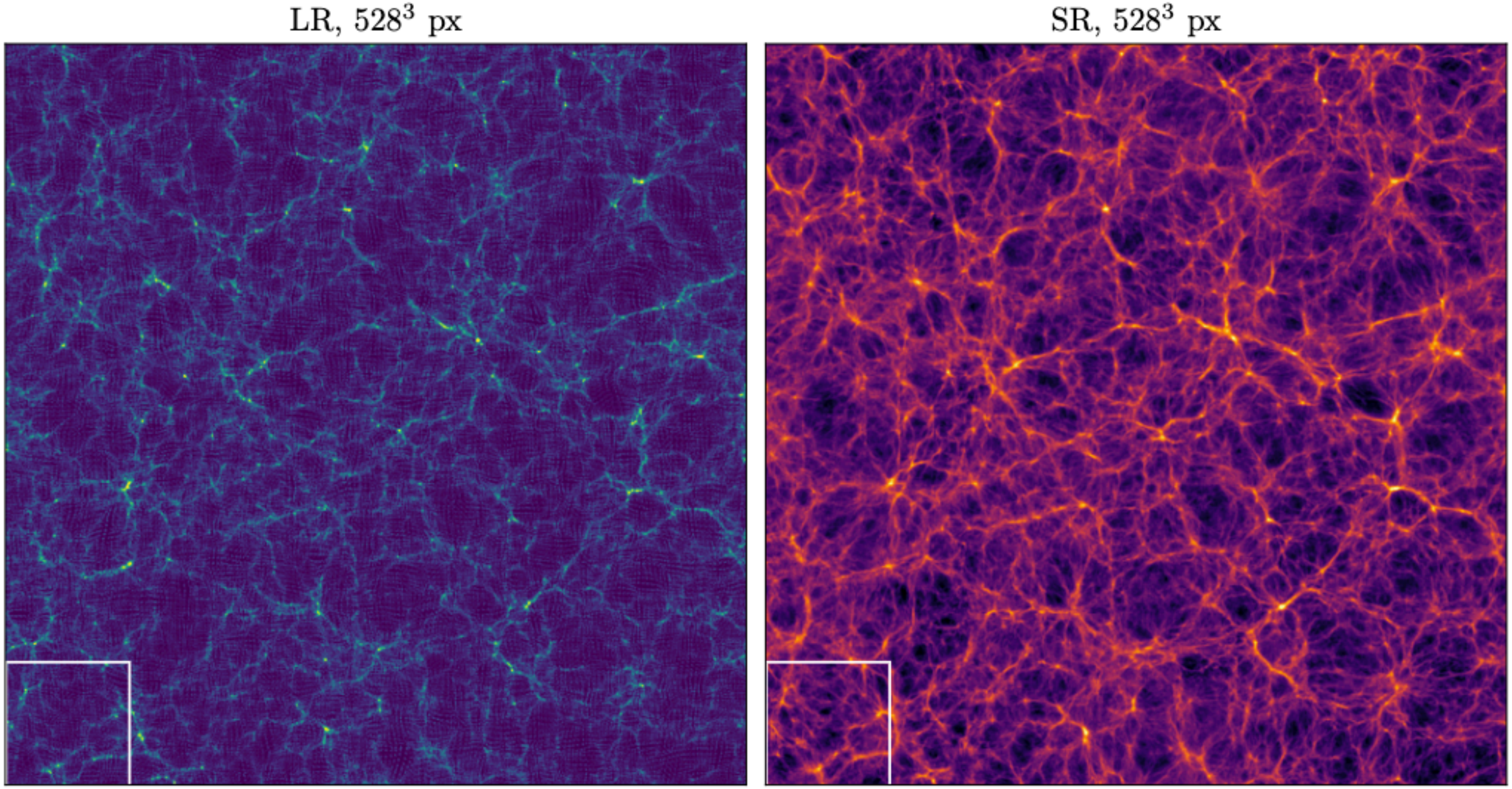
Probability density

Power spectrum

Bispectrum ($k_1 = .15\ \frac{h}{\mathrm{Mpc}}$, $k_2 = .25\ \frac{h}{\mathrm{Mpc}}$)

Void size function

Making larger volumes than the training data



This is a 3d “Illustris-TNG 600”, where we increased the volume by a factor of 8 compared to the training data. Current limitation: Sample generation time.

Another example in cosmology: Diffusion on Graphs NNs

- https://arxiv.org/abs/2311.17141 (Not from my group)

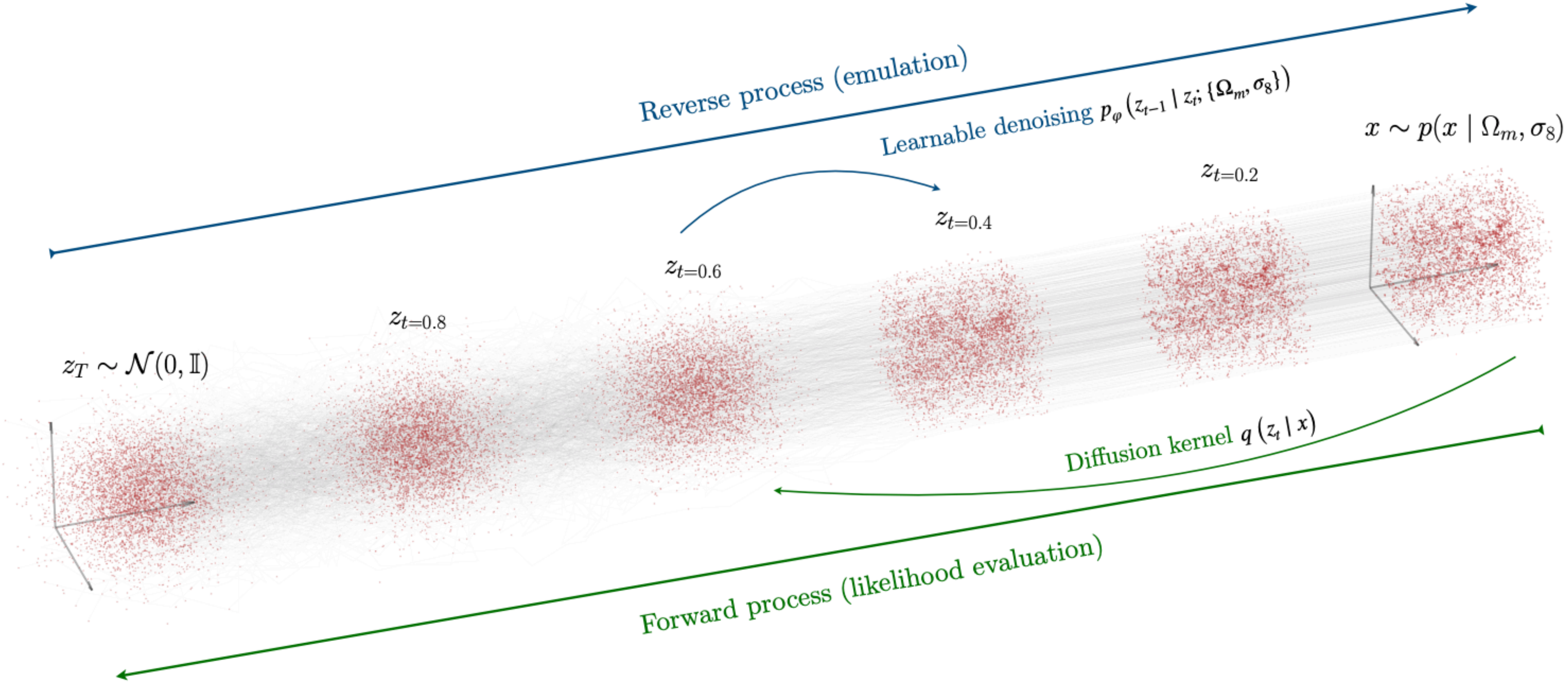

Reverse process (emulation)
Learnable denoising $p_\varphi(z_{t-1} \mid z_t; \{\Omega_m, \sigma_8\})$
$x \sim p(x \mid \Omega_m, \sigma_8)$
$z_{t=0.2}$
$z_{t=0.4}$
$z_{t=0.6}$
$z_{t=0.8}$
$z_T \sim \mathcal{N}(0, \mathbb{I})$
Diffusion kernel $q(z_t \mid x)$
Forward process (likelihood evaluation)

Figure 1. A schematic overview of the point cloud diffusion model, showing samples from the diffusion process at different diffusion times. During training, noise is added to a data sample x using the diffusion kernel $q(z_t \mid x)$ and a denoising distribution $p_\varphi(z_{t-1} \mid z_t)$ is learned. To generate samples, we simulate the reverse process – we sample noise from a standard Gaussian distribution and denoise it iteratively using the learned denoising distribution.

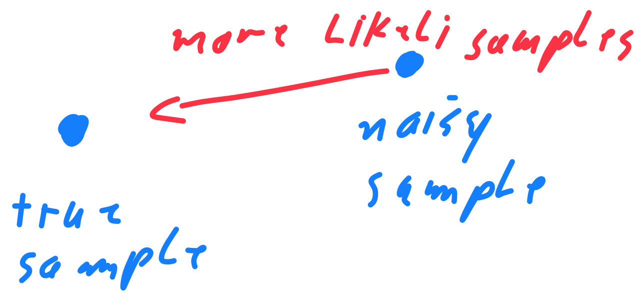
Score-based diffusion, Relation to physics

The score function

- Given a probability density function $p(\mathbf{x})$, its (Stein) score function is defined as the gradient of the log probability density $\nabla_{\mathbf{x}} \log p(\mathbf{x})$ with respect to the data $\mathbf{x}$ rather than the model parameter θ.

- It is a vector field that points to directions along which the probability density function has the largest growth rate.

$$\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \ln p(\mathbf{x})$$



- Learning the score function is closely related to learning how to denoise. In fact, **denoising a noisy image toward the clean data manifold gives an estimate of the score function** — the gradient of the log-density. Score-based generative models use this connection to train on noisy data and learn how to reverse the corruption process.

Score-based Generative Models (briefly)

- The key idea of score-based generative models (SGMs) is to **perturb data with a sequence of intensifying Gaussian noise and jointly estimate the score functions** for noisy data distributions by training a deep neural network model conditioned on noise levels.

- For **score matching**, the loss function is

$$\mathbb{E}_{t\sim[1,T],x_0\sim q(x_0),x_t\sim q(x_t|x_0)}\left[\lambda(t)^2 \parallel \nabla_{x_t}\log q(x_t) - s_\theta(x_t, t) \parallel ^2\right]$$

  which can be re-written in a computable way for Gaussian noise perturbations.

- After one has estimated the score function, there are several different methods how one can sample from it. The classic one is called **“Langevin Dynamics”, a physics related method.**

- The learned score can also be used as a **generative prior in a Bayesian data analysis**.

Relation to physics: Stochastic differential equations

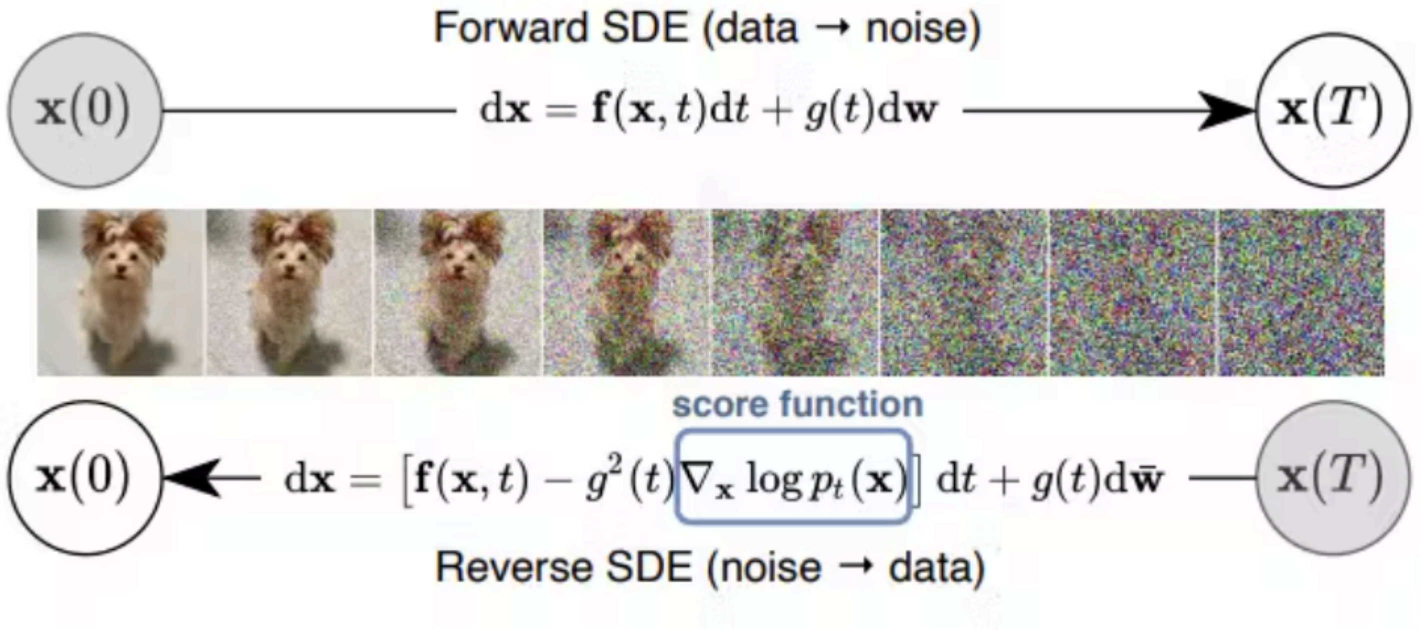
- In physics, diffusion is modelled by a s**tochastic differential equation**

  $$dx = f(x, t)dt + g(t)dw$$

  where $f(x, t)$ and $g(t)$ are the **drift and diffusion functions** and **w** is a **Wiener process (Brownian motion)**

- Both DDPM and score matching generative models are discretization of this SDE.

- It can be mathematically shown that there is a reverse diffusion SED:



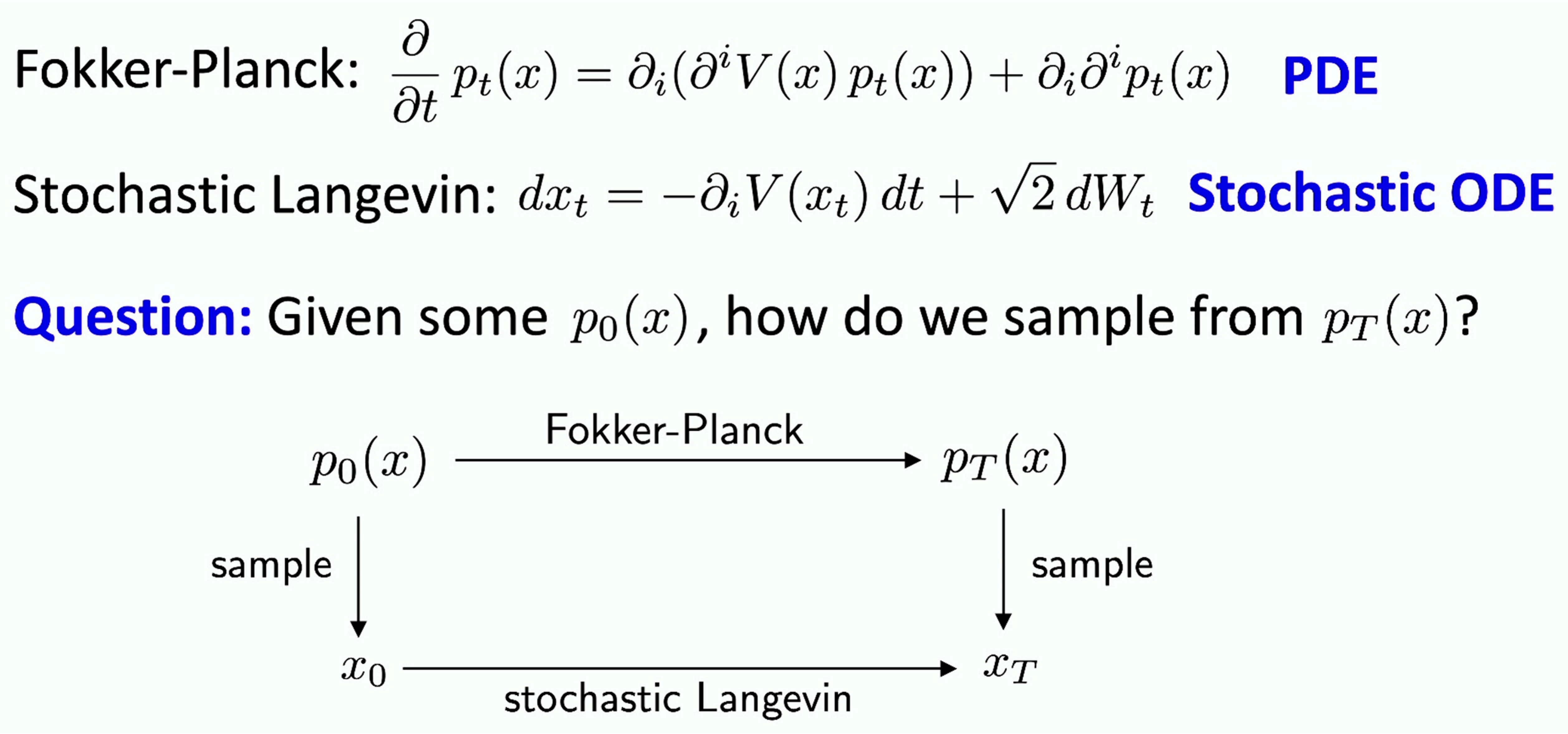
Forward SDE (data → noise)
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g^2(t)\boxed{\nabla_\mathbf{x} \log p_t(\mathbf{x})}]\,\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$
$\mathbf{x}(T)$
Reverse SDE (noise → data)

https://arxiv.org/abs/2011.13456

- For an analysis of diffusion models by a physicist see https://arxiv.org/abs/2310.04490 **Generative Diffusion From An Action Principle**

Relation to physics: PDE vs SDE (briefly)

There are two different ways to sample from a PDF by evolving a system in time.



Fokker-Planck: $\frac{\partial}{\partial t} p_t(x) = \partial_i(\partial^i V(x)\, p_t(x)) + \partial_i \partial^i p_t(x)$ **PDE**

Stochastic Langevin: $dx_t = -\partial_i V(x_t)\, dt + \sqrt{2}\, dW_t$ **Stochastic ODE**

Question: Given some $p_0(x)$, how do we sample from $p_T(x)$?

$$\begin{tikzcd}
p_0(x) \arrow[r, "\text{Fokker-Planck}"] \arrow[d, "\text{sample}"'] & p_T(x) \arrow[d, "\text{sample}"] \\
x_0 \arrow[r, "\text{stochastic Langevin}"'] & x_T
\end{tikzcd}$$

This slide is from Jordan Cotler’s talk here: https://pdf.pirsa.org/files/25040095.pdf

Fokker–Planck Equation

$$\frac{\partial}{\partial t} p_t(x) = \partial_i \left(\partial^i V(x)\, p_t(x) \right) + \partial_i \partial^i p_t(x)$$

- This is a **PDE** describing how the probability density $p_t(x)$ evolves over time.
- $V(x)$ is often a **potential function** (e.g. a log-density).
- The right-hand side describes **drift** (first term) and **diffusion** (second term).

Think of it as describing how a “cloud of particles” spreads and drifts over time under certain forces.

In statistical mechanics and information theory, the **Fokker–Planck equation** is a partial differential equation that describes the **time evolution of the probability density function of the velocity** of a particle under the influence of **drag forces and random forces, as in Brownian motion**.

We **do not sample using Fokker–Planck** — it’s just the theoretical backbone describing the evolution of the **density**, not individual samples.

Stochastic Langevin Equation

$$dx_t = -\partial_i V(x_t)\, dt + \sqrt{2}\, dW_t$$

- This is a **stochastic differential equation (SDE)**.
- It's the **stochastic process** whose marginal density $p_t(x)$ evolves according to the Fokker–Planck PDE.
- The term $-\partial_i V(x_t)\, dt$ is drift toward high-probability regions.
- The $\sqrt{2}\, dW_t$ term is **random noise** (Brownian motion).

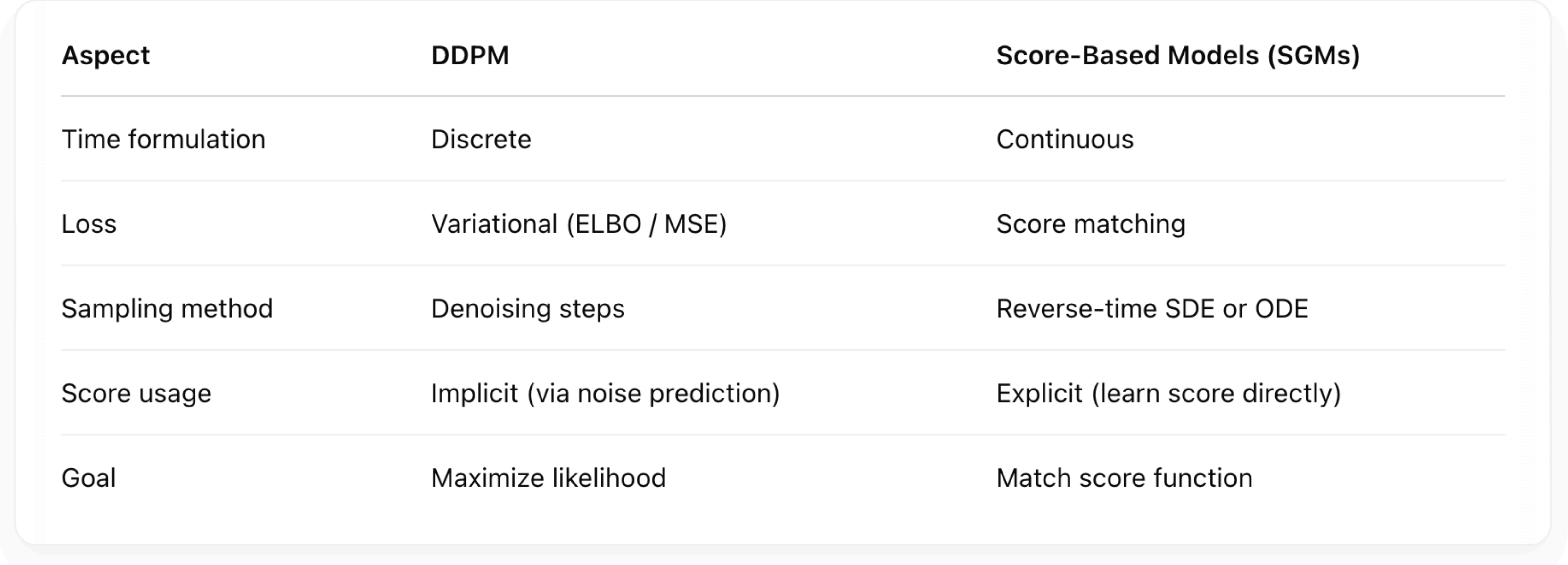
The stochastic Langevin equation is a type of stochastic differential equation that **describes the time evolution of a system subject to both deterministic and stochastic forces.** It is often used to model phenomena like **Brownian motion**, where a small particle is subject to random impacts from the surrounding fluid molecules.

This equation tells you how a particle x_t **moves in time**, and if you simulate many particles this way, their distribution follows the Fokker–Planck equation.

In diffusion models, we sample using Langevin-like or reverse-time SDEs

DDPM vs SGM

You can think of SGMs as a generalization or alternative formulation of DDPMs.



Aspect	DDPM	Score-Based Models (SGMs)
Time formulation	Discrete	Continuous
Loss	Variational (ELBO / MSE)	Score matching
Sampling method	Denoising steps	Reverse-time SDE or ODE
Score usage	Implicit (via noise prediction)	Explicit (learn score directly)
Goal	Maximize likelihood	Match score function

Course logistics

- **Reading for this lecture:**
  - This lecture was based in part on the book by Bishop linked on the website.