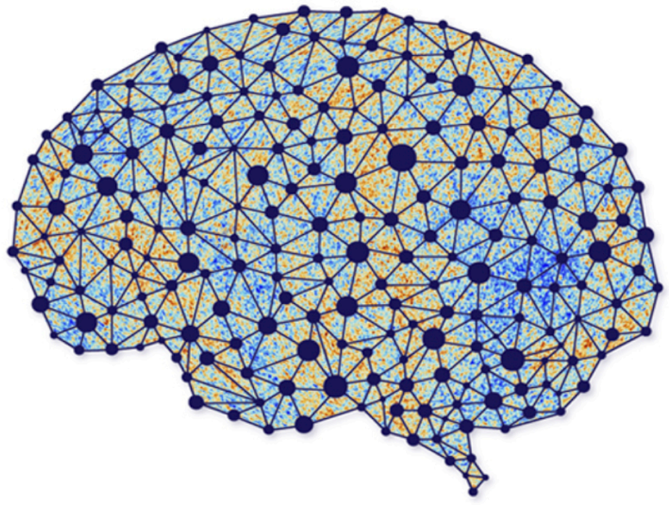
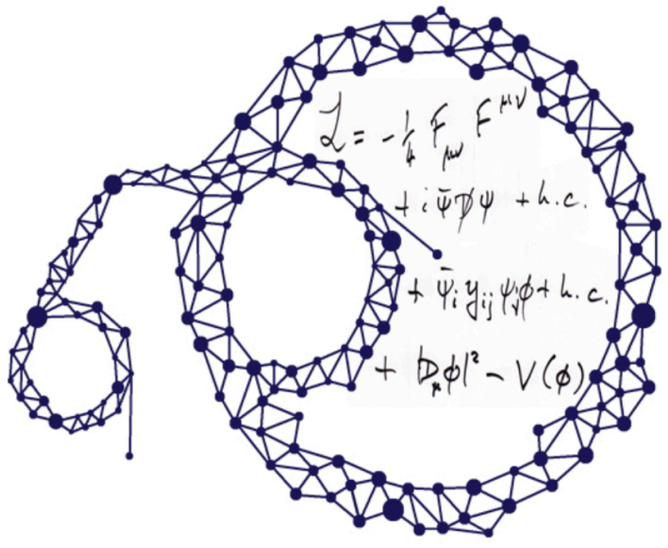
Physics 361 - Machine Learning in Physics

Lecture 26 – Diffusion Models

April 25th 2024



AI
∩
Universe



Moritz Münchmeyer

Final project

- We will have short presentations on May 2nd during class.
- We have about 15 projects in 75min, **so 5min per presentation**.
- I will share **a google drive or box folder** where you can all upload your slides and edit them right until the lecture.
- Grading will be primarily based on your submitted paper, so no need to squeeze a lot of information in the talk.
- Your main goal is to show the other students what you are working on.
- **Papers are due on Sunday May 5th at midnight**.

Final project

- You will write **a paper on an application of machine learning to physics** of your choice. Your paper needs to contain a computational analysis, which generally will mean applying a machine learning method to some data set.
- You can **work alone or in groups of two.**
- The paper should be **5 to 10 pages** and contain the following:
    - A short review of at least one research paper related to your topic. This is to encourage you to learn how to browse the literature.
    - A description of the data set you will be working with and its properties.
    - A brief description of the machine learning method you will use. Don't re-explain basics such as how CNNs work, rather describe the detailed properties of your approach.
    - Train the model and put the results in your paper. Explore some variations such as different hyper parameters.
    - Describe successes and problems in your analysis.
    - Also submit the **Colab notebook** you used for training/evaluation.
        - I will not re-run the notebook. Notebook submission is **not required but encouraged.**
    - The project should take you ~three days of work, spread over the last weeks of the semester.

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/
  - 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!)**.

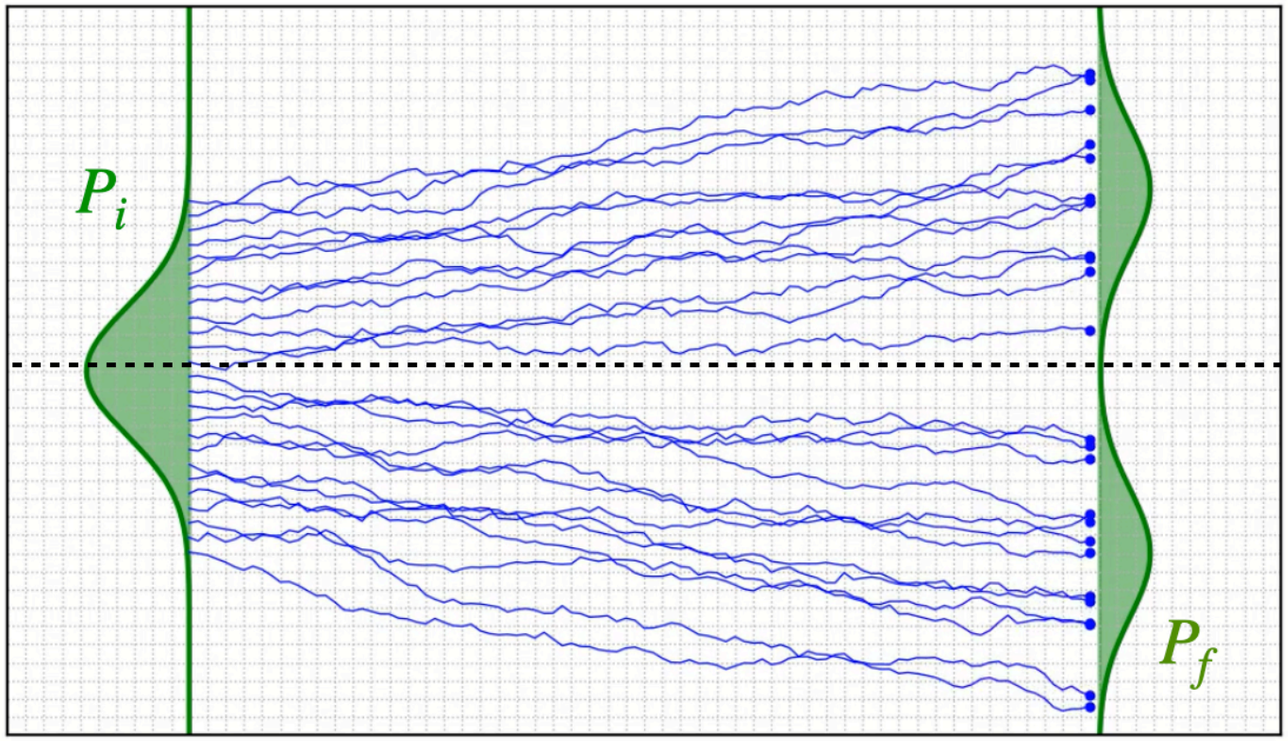


Figure credit: Akhil Premkumar, KICP

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

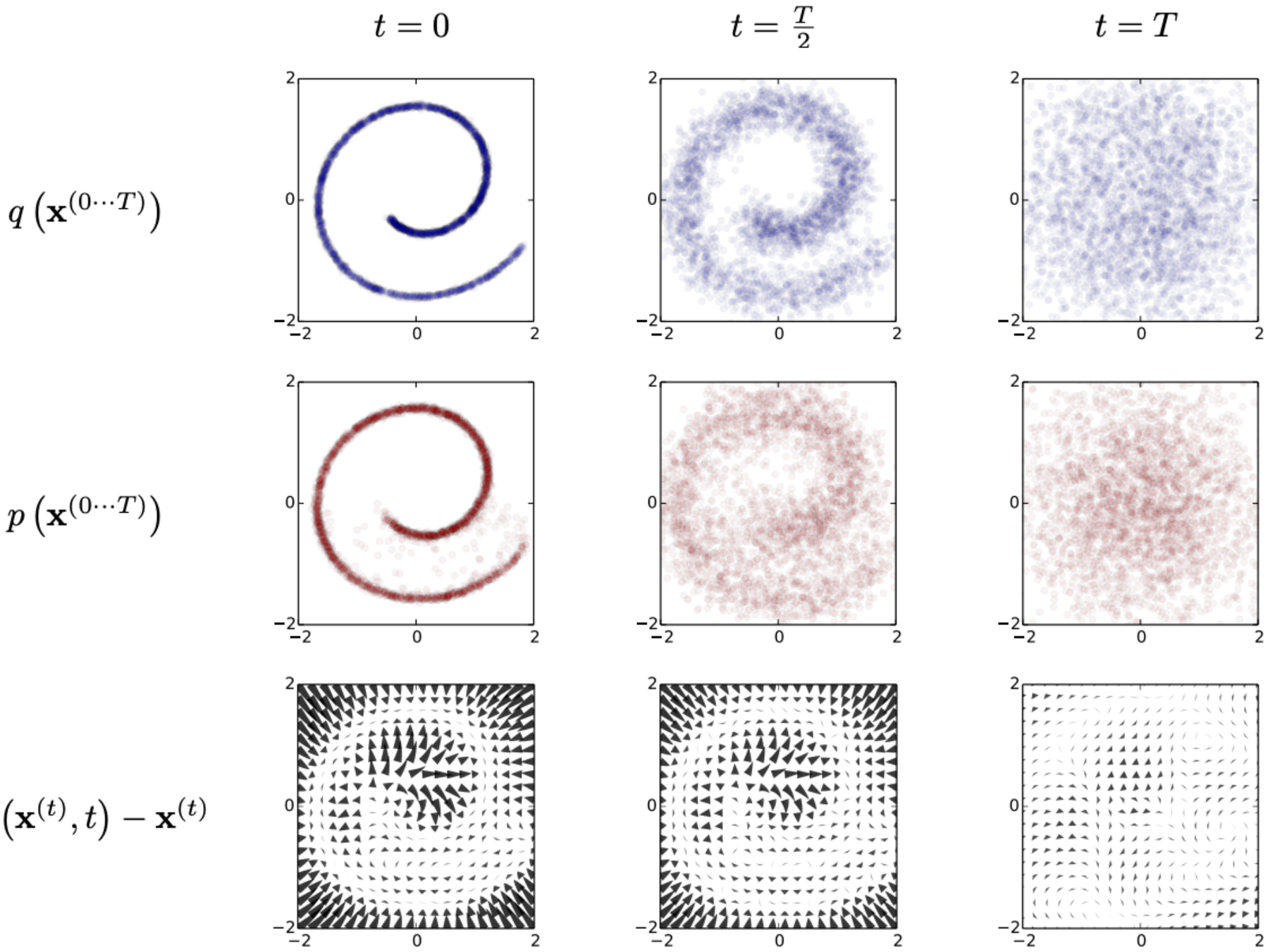
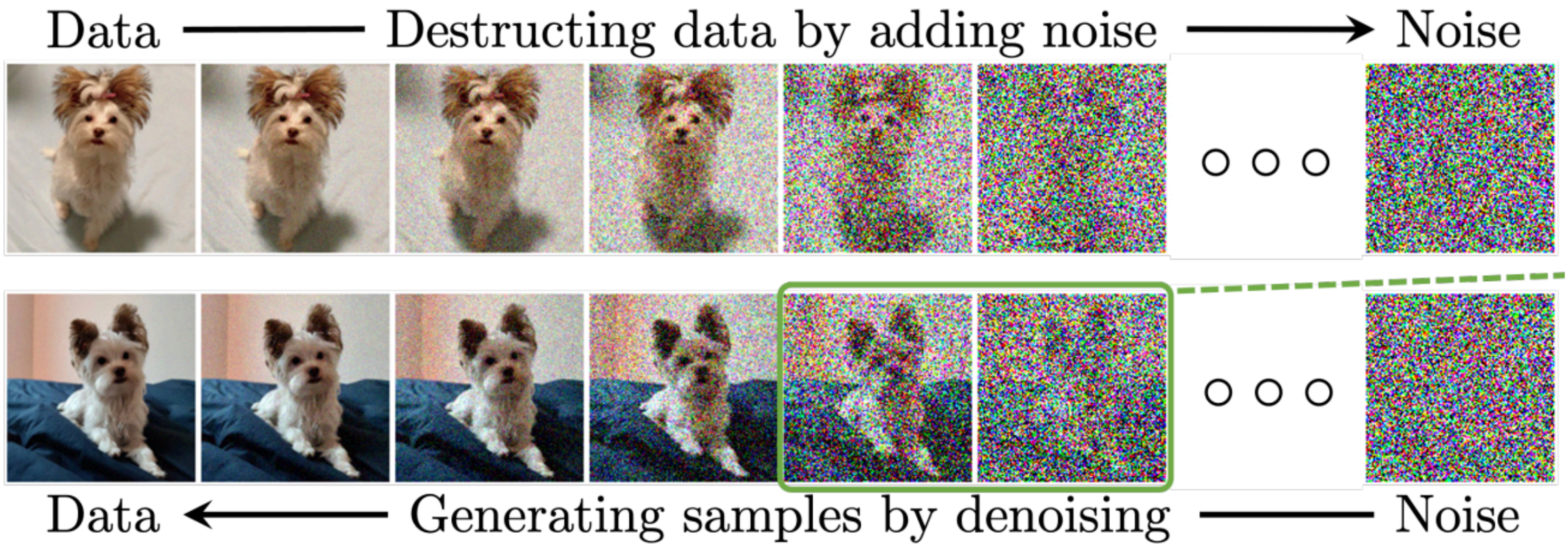


Figure 1. The proposed modeling framework trained on 2-d swiss roll data. The top row shows time slices from the forward trajectory $q\left(\mathbf{x}^{(0\cdots T)}\right)$. The data distribution (left) undergoes Gaussian diffusion, which gradually transforms it into an identity-covariance Gaussian (right). 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



- The noise to data (denoising) process is learned by a neural network, 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

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

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

- 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:

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

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

random Gaussian noise vector

Reverse Diffusion Process

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

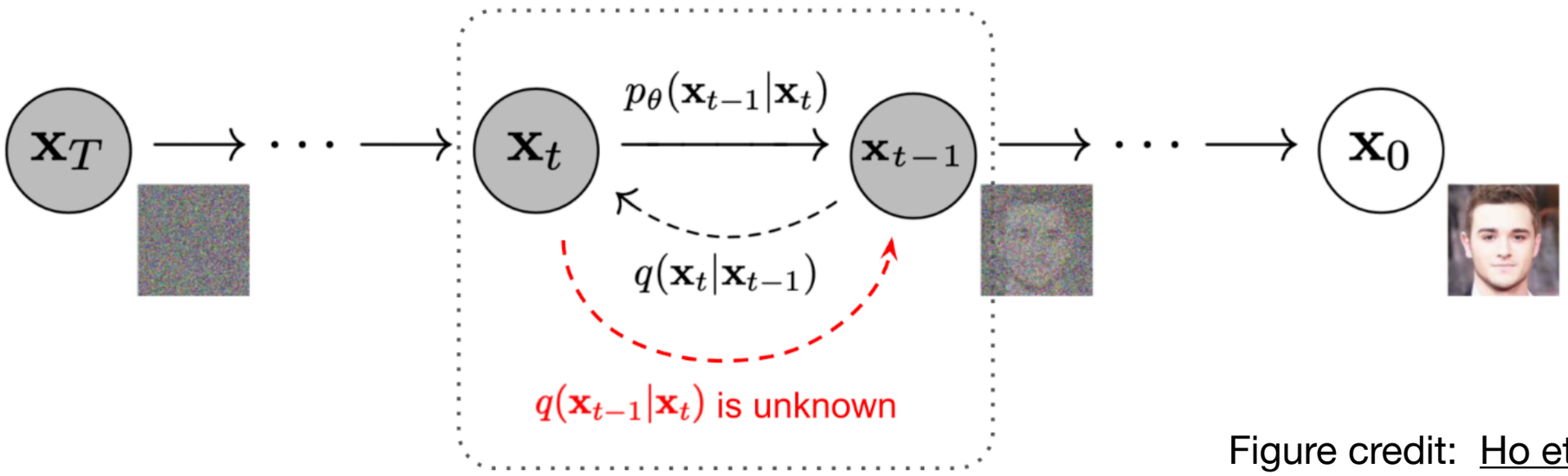


Figure credit: Ho et al. 2020

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

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

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

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

- The obvious training objective would be negative log-likelihood $-\log p_\theta(\mathbf{x}_0)$. 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.

KL-divergence of the two Markov chains

$$
\begin{aligned}
-\log p_\theta(\mathbf{x}_0) &\le -\log p_\theta(\mathbf{x}_0) + D_{\mathrm{KL}}(q(\mathbf{x}_{1:T}|\mathbf{x}_0) \| p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0)) \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_{\mathbf{x}_{1:T}\sim q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})/p_\theta(\mathbf{x}_0)}\right] \\
&= -\log p_\theta(\mathbf{x}_0) + \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})} + \log p_\theta(\mathbf{x}_0)\right] \\
&= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \\
\text{Let } L_{\mathrm{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \ge -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0)
\end{aligned}
$$

can be evaluated from training samples

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 actual loss function 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

$$L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[\left\| \epsilon_t - \epsilon_\theta(x_t, t) \right\|^2\right]$$

$$= \mathbb{E}_{t\sim[1,T],x_0,\epsilon_t}\left[\left\| \epsilon_t - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon_t, t\right) \right\|^2\right]$$

can calculate noisy image at time t in one step

- Next let us look into the final training and sampling algorithms to understand some more details.

Training algorithm

Algorithm 20.1: Training a denoising diffusion probabilistic model

Input: Training data $\mathcal{D} = \{\mathbf{x}_n\}$

Noise schedule $\{\beta_1, \ldots, \beta_T\}$

Output: Network parameters $\mathbf{w}$

---

for $t \in \{1, \ldots, T\}$ **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

  $t \sim \{1, \ldots, T\}$ // 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 $\mathbf{w}$

w: NN weights

g: NN output

Figure credit: Bishop Deep Learning

Sampling algorithm

initial random sampling

~1000 steps

random sampling at each time step

subtract estimated noise to find new mean

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 x

w: NN weights

g: NN output

Figure credit: Bishop Deep Learning

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.

Score-based Generative Models

- Given a probability density function $p(\mathbf{x})$, its score function is defined as the gradient of the log probability density $\nabla_{\mathbf{x}} \log p(\mathbf{x})$. The Stein score considered here is a function of 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.

- Clearly, learning the score is closely related to learning de-noising.

- 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 all noisy data distributions by training a deep neural network model conditioned on noise levels.

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

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

  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”, again a physics method.**

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

Relation to physics: Stochastic differential equations

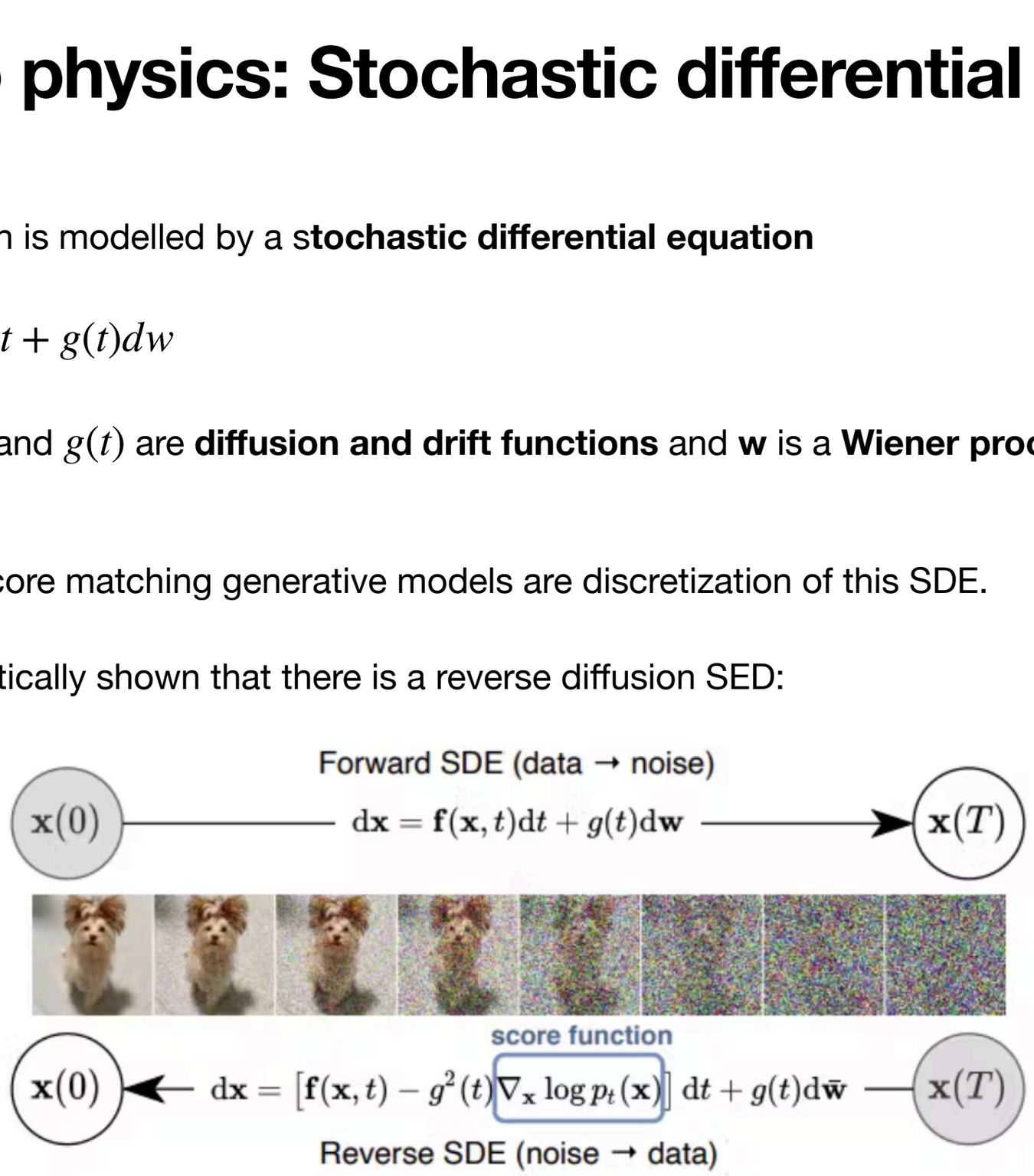
- In physics, diffusion is modelled by a s**tochastic differential equation**

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

where $f(x,t)$ and $g(t)$ are **diffusion and drift 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:



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**

Other physical processes as generative models?

- Diffusion models are a very nice example of how physics can be used to do machine learning (rather than the other way round).

- Since diffusion is a physical process, one may ask if there are other physical processes which can be use as generative models. This idea was developed here:

  - https://arxiv.org/abs/2209.11178 Poisson Flow Generative Models

  - https://arxiv.org/abs/2302.04265 PFGM++: Unlocking the Potential of Physics-Inspired Generative Models

  - https://arxiv.org/abs/2304.02637 GenPhys: From Physical Processes to Generative Models

- While so far these results have not been very important in practice, let's have a quick look in the papers because they are a great example of combining physics and ML.

Poisson flow from https://arxiv.org/pdf/2209.11178.pdf

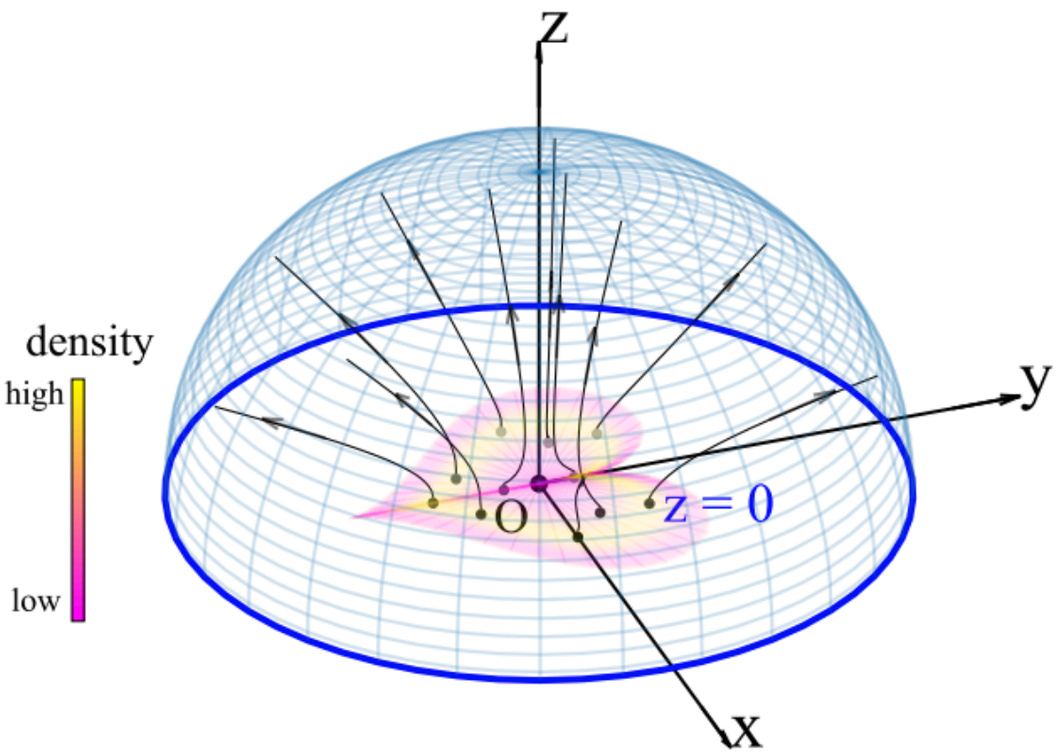


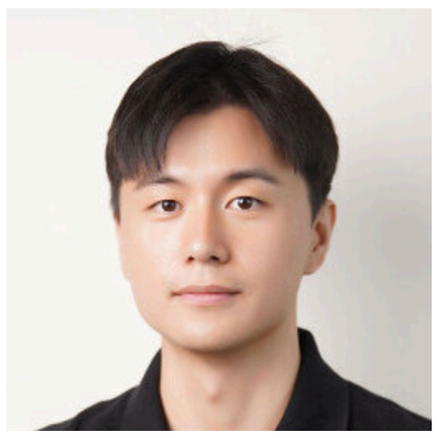
Figure 1: **(a)** 3D Poisson field trajectories for a heart-shaped distribution **(b)** The evolvements of a distribution (**top**) or an (augmented) sample (**bottom**) by the forward/backward ODEs pertained to the Poisson field.

Example from my own research: Super-resolution Emulator

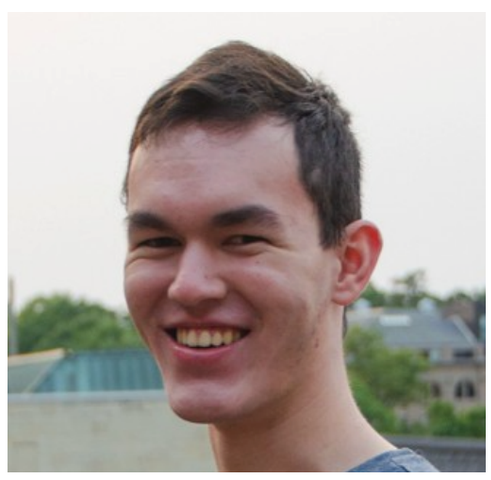
Super-Resolution Emulation of Large Cosmological Fields with a 3D Conditional Diffusion Model https://arxiv.org/abs/2311.05217

Collaborators:

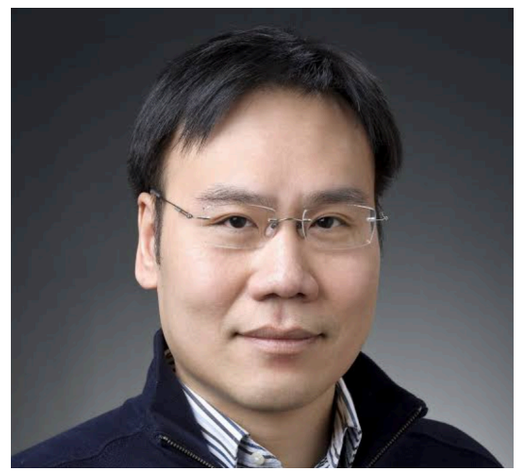
Adam Rouhiainen,
UW Madison Physics



Prof. Kangwook Lee,
UW Madison ECE & CS



Michael Gira,
UW Madison ECE & CS,
Microsoft



Prof. Gary Shiu,
UW Madison Physics

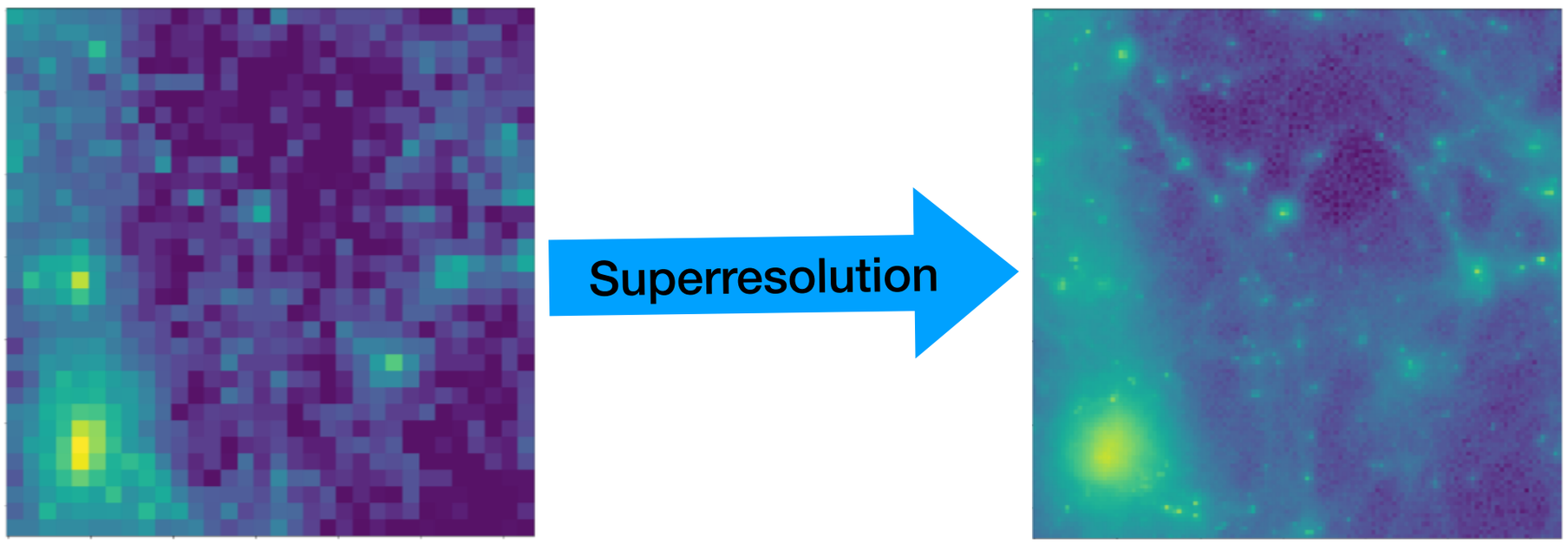
Super-resolution Emulators

- 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

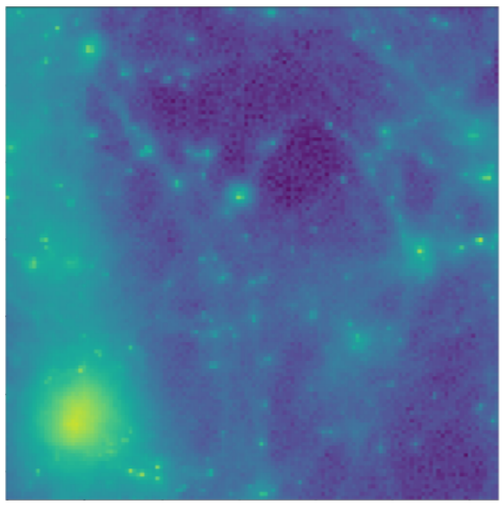
- Idea of **super-resolution (SR) emulators**: Run low-res (LR) dark matter sim and upgrade to high-res (HR) hydro simulation with a generative neural network.



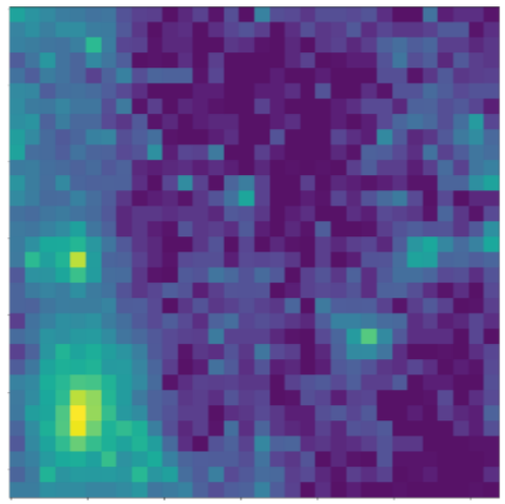
Conditional diffusion for Super-resolution

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

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



Here: High resolution IllustrisTNG gas density



Low resolution IllustrisTNG DM

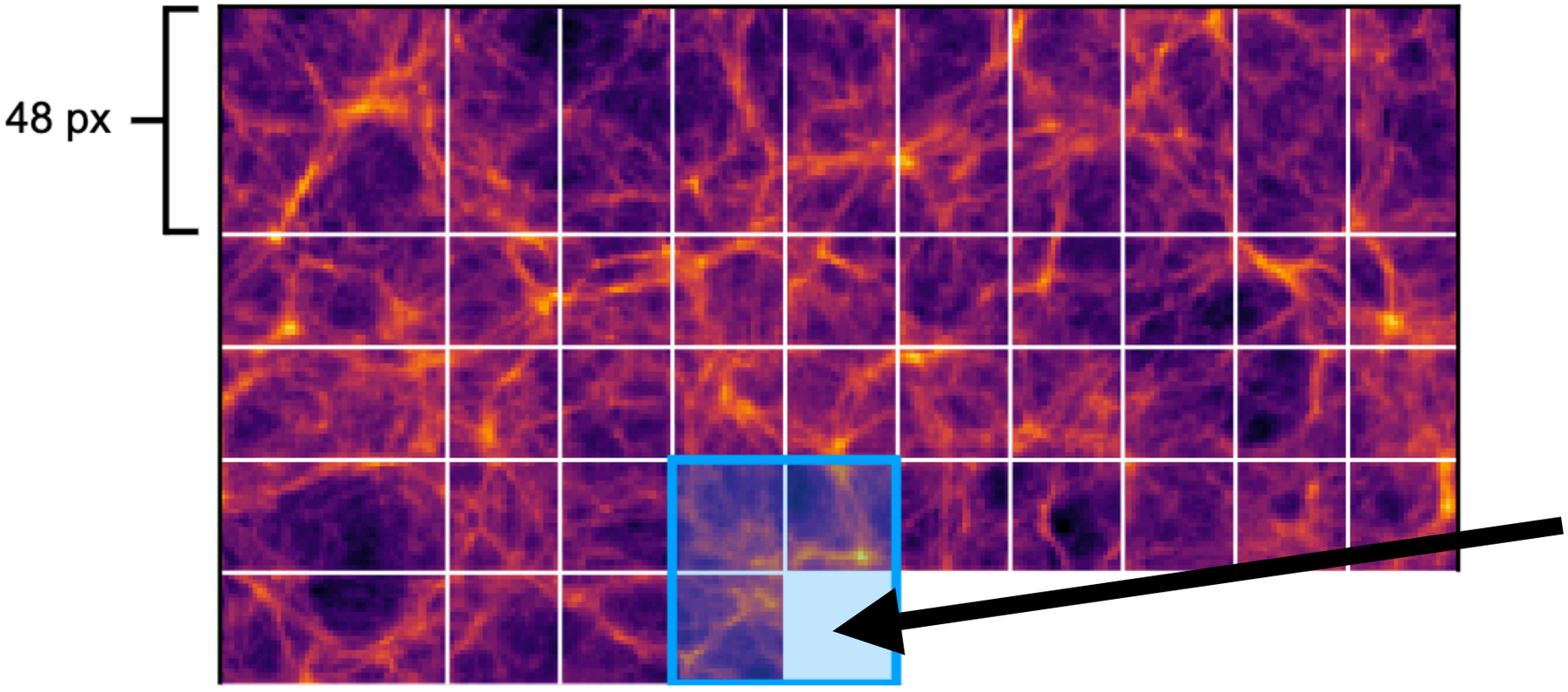
- We started with the flows from the last section. However we found that existing **NFs in 3d are not expressive enough, so we switched to diffusion models**.

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 **“outpointing” procedure to make in principle arbitrarily large SR simulations with smooth patching.**

$$P(\text{high resolution} \mid \text{low resolution, neighboring high resolution})$$



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

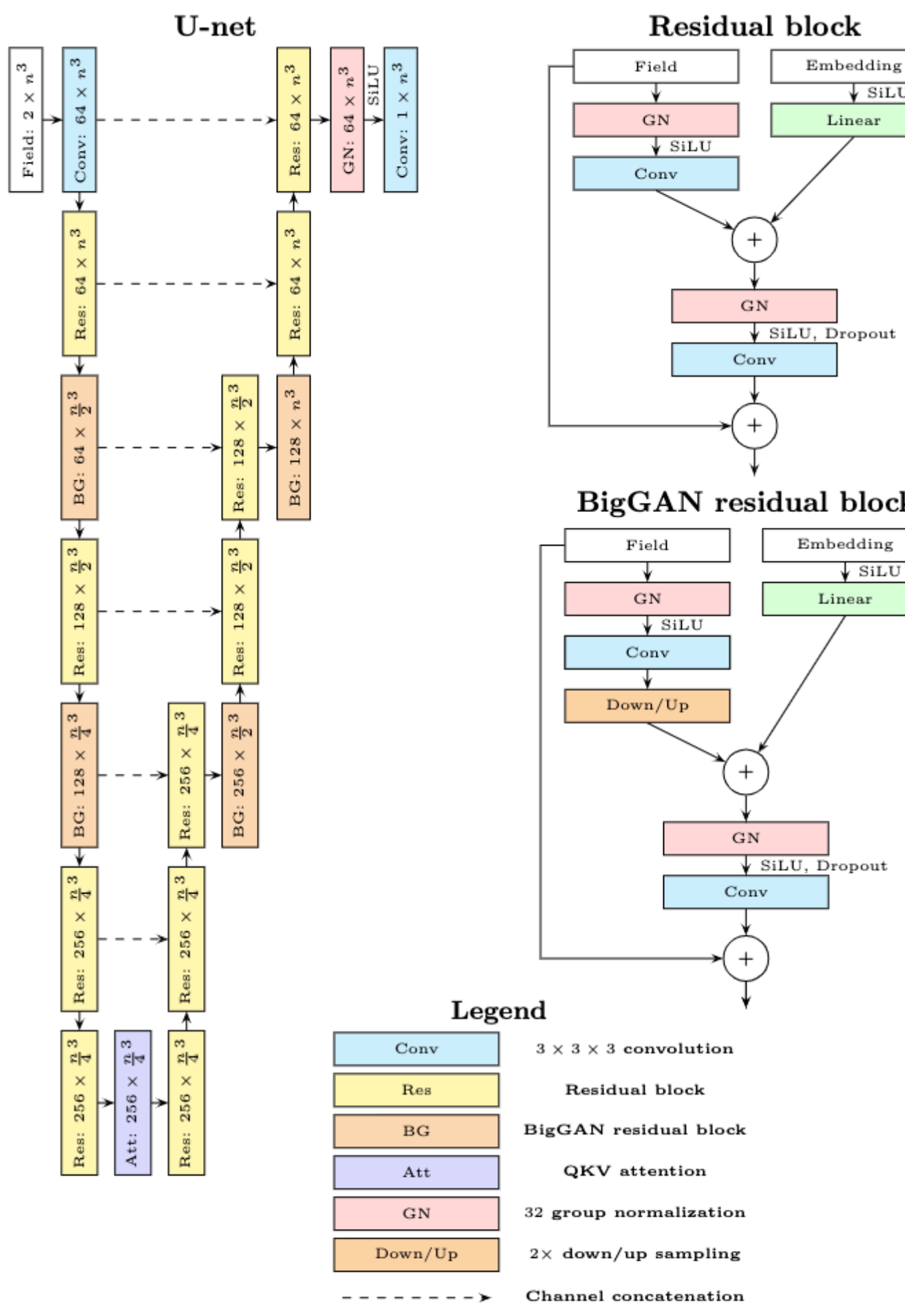
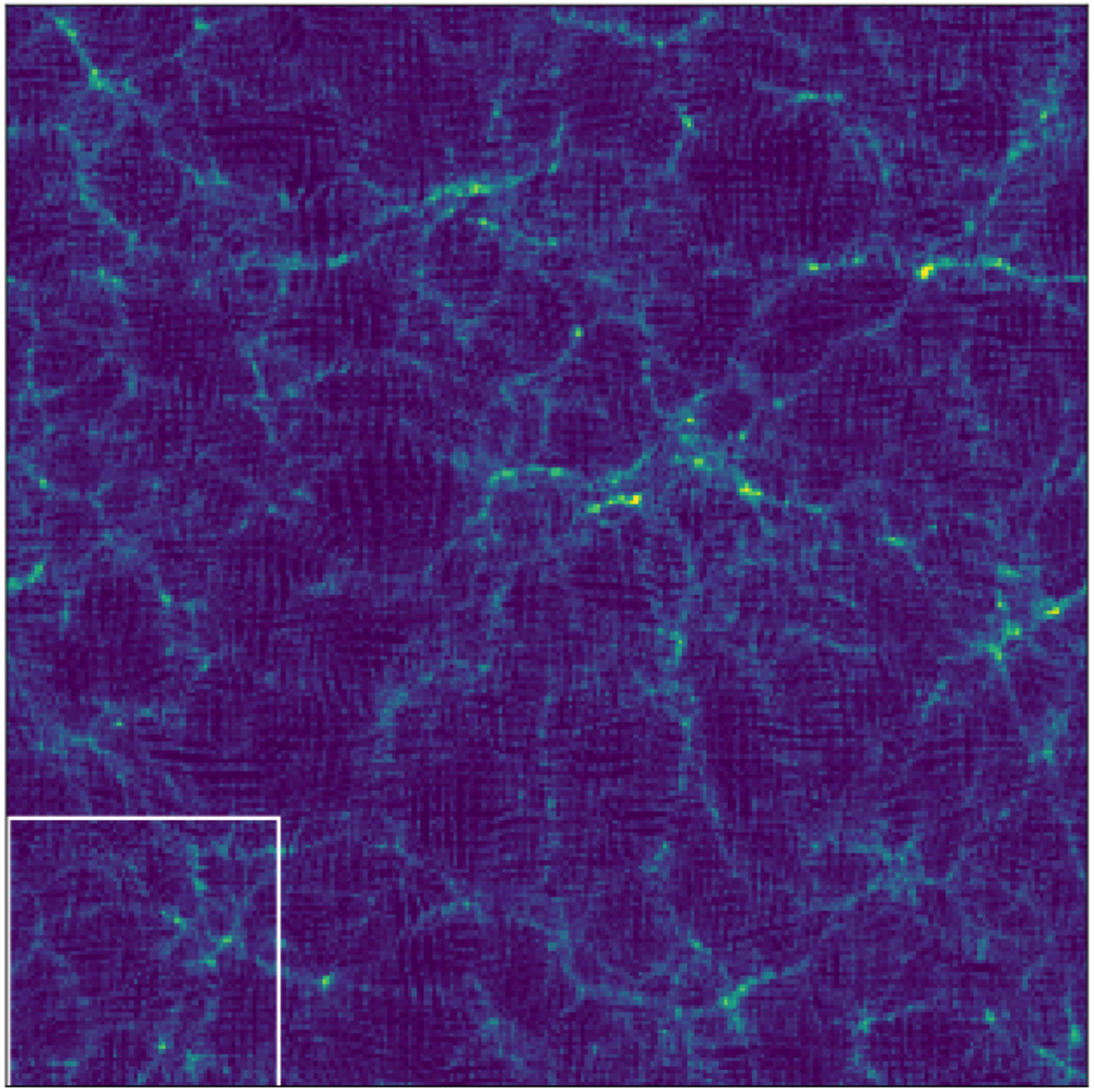


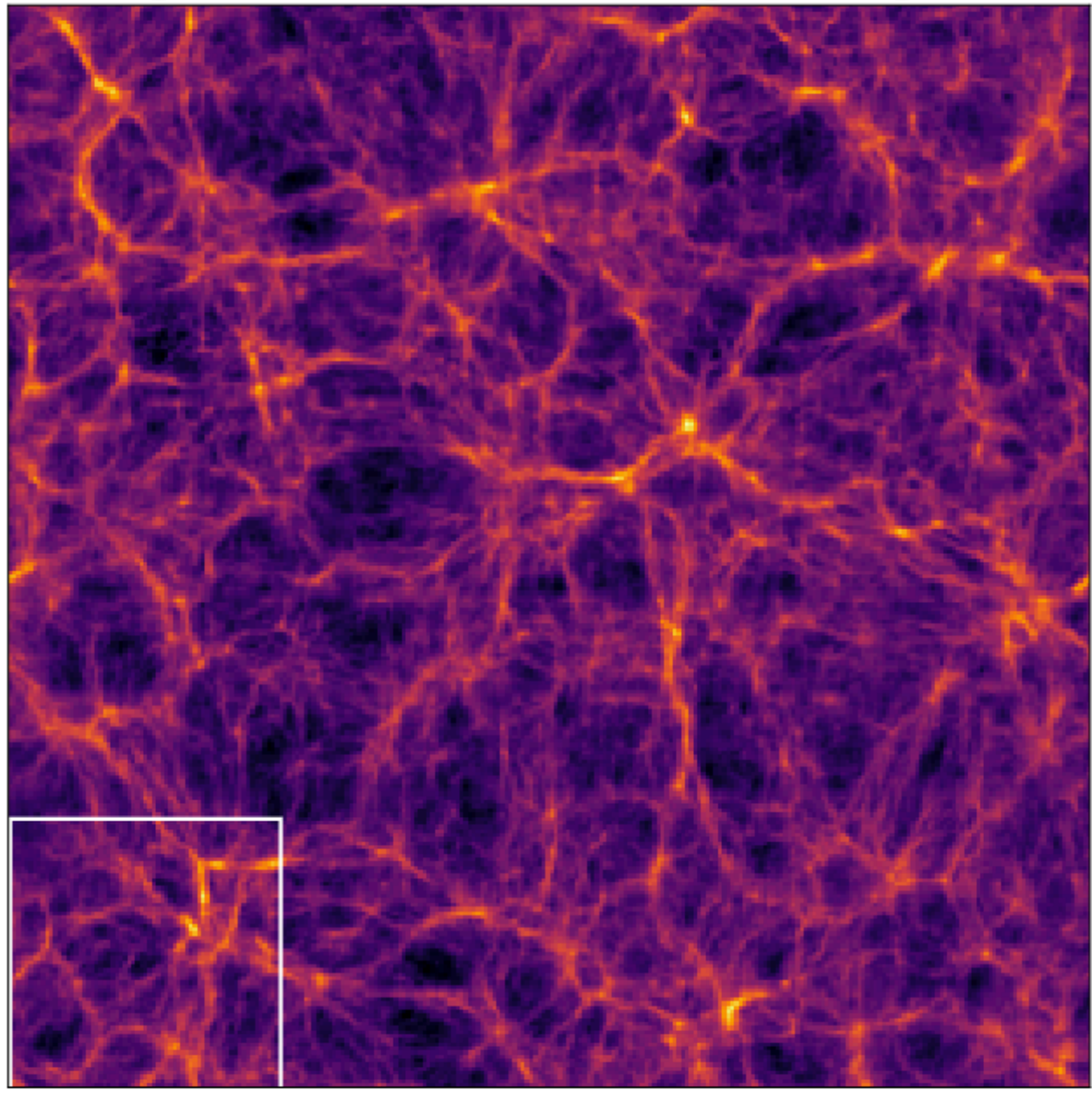
FIG. 11. The U-net used in this work (left) is made of residual blocks (top right), with BigGAN residual blocks (center right) used to downsample and upsample the fields. Shown after every layer name in the U-net is the layer's output channels × volume.

Results

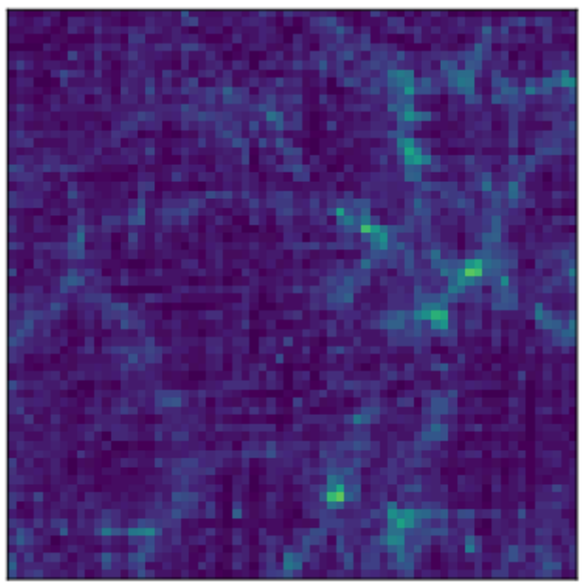
Low-resolution conditional



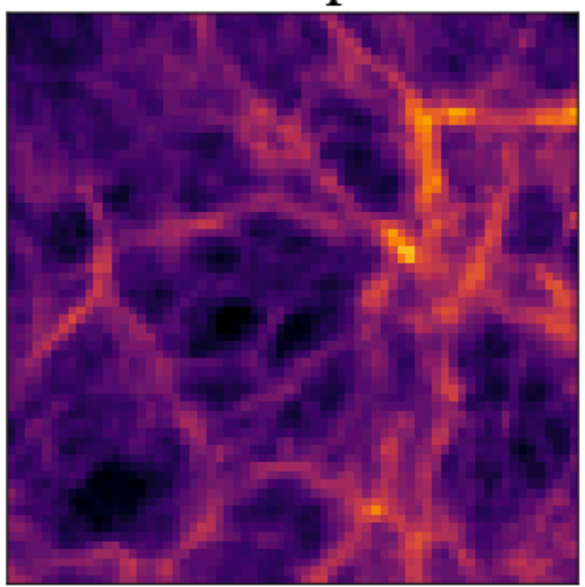
Super-resolution model output, sample 1



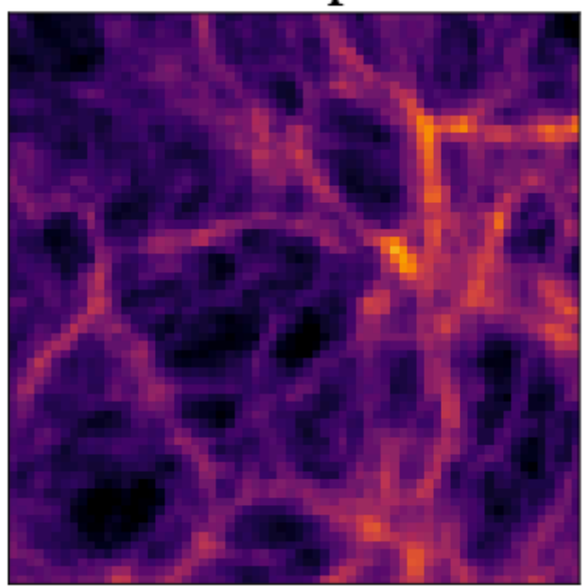
LR



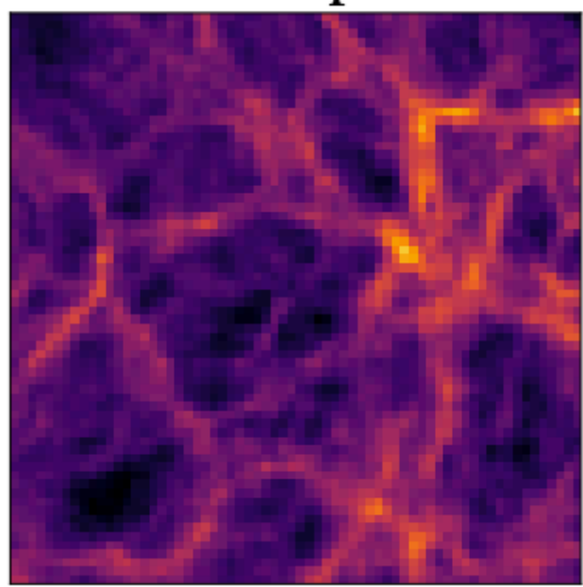
SR sample 2



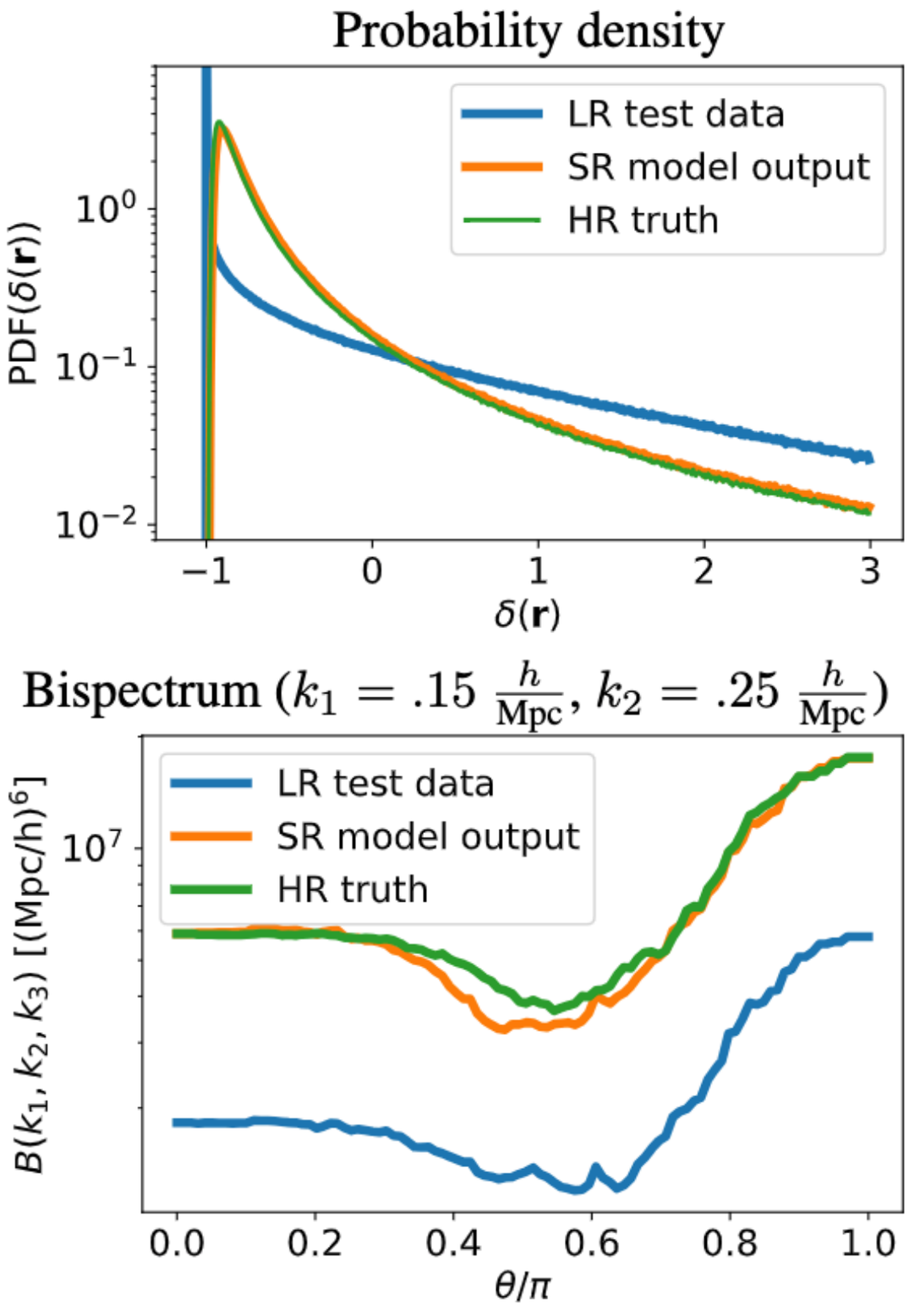
SR sample 3

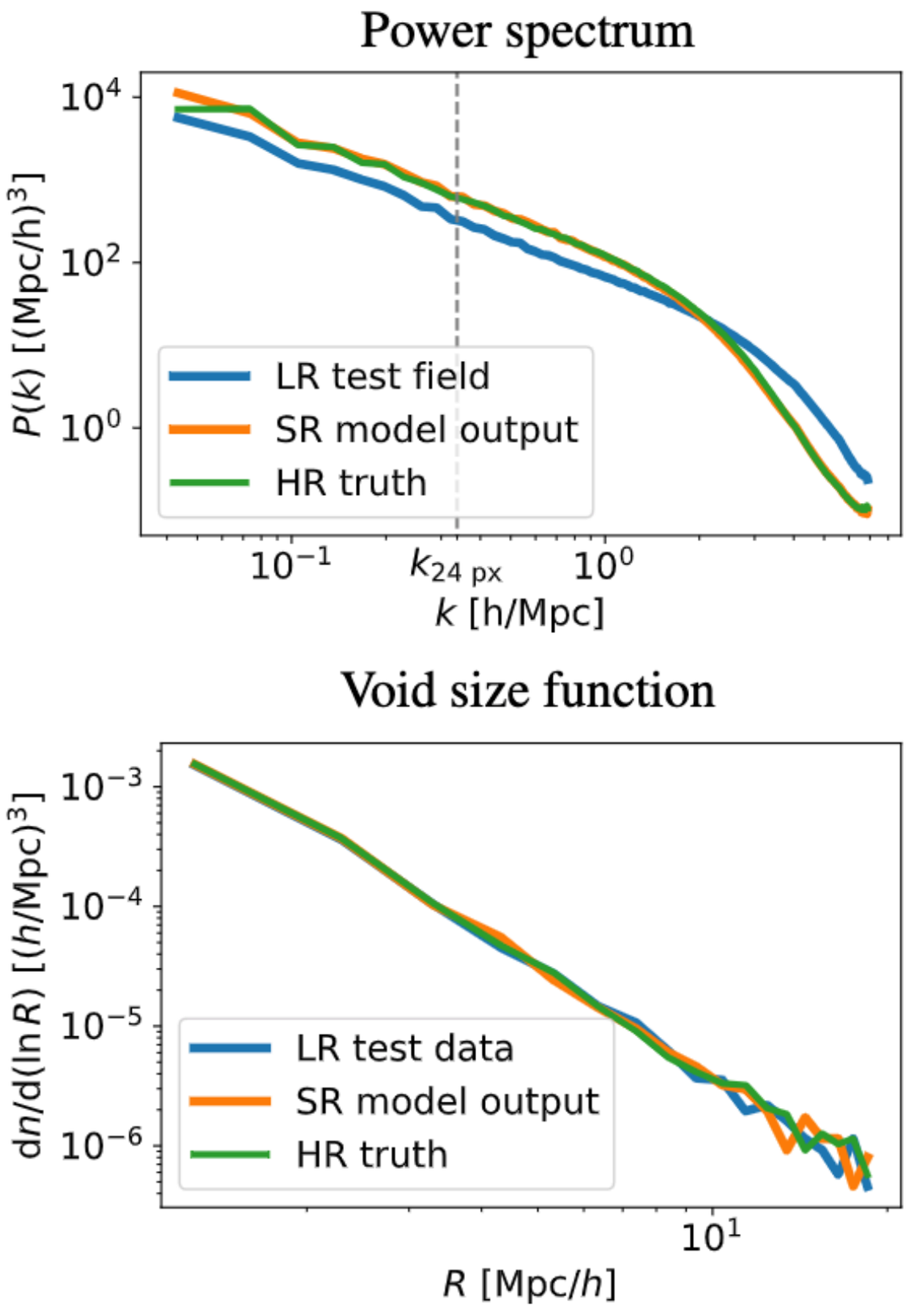


SR sample 4

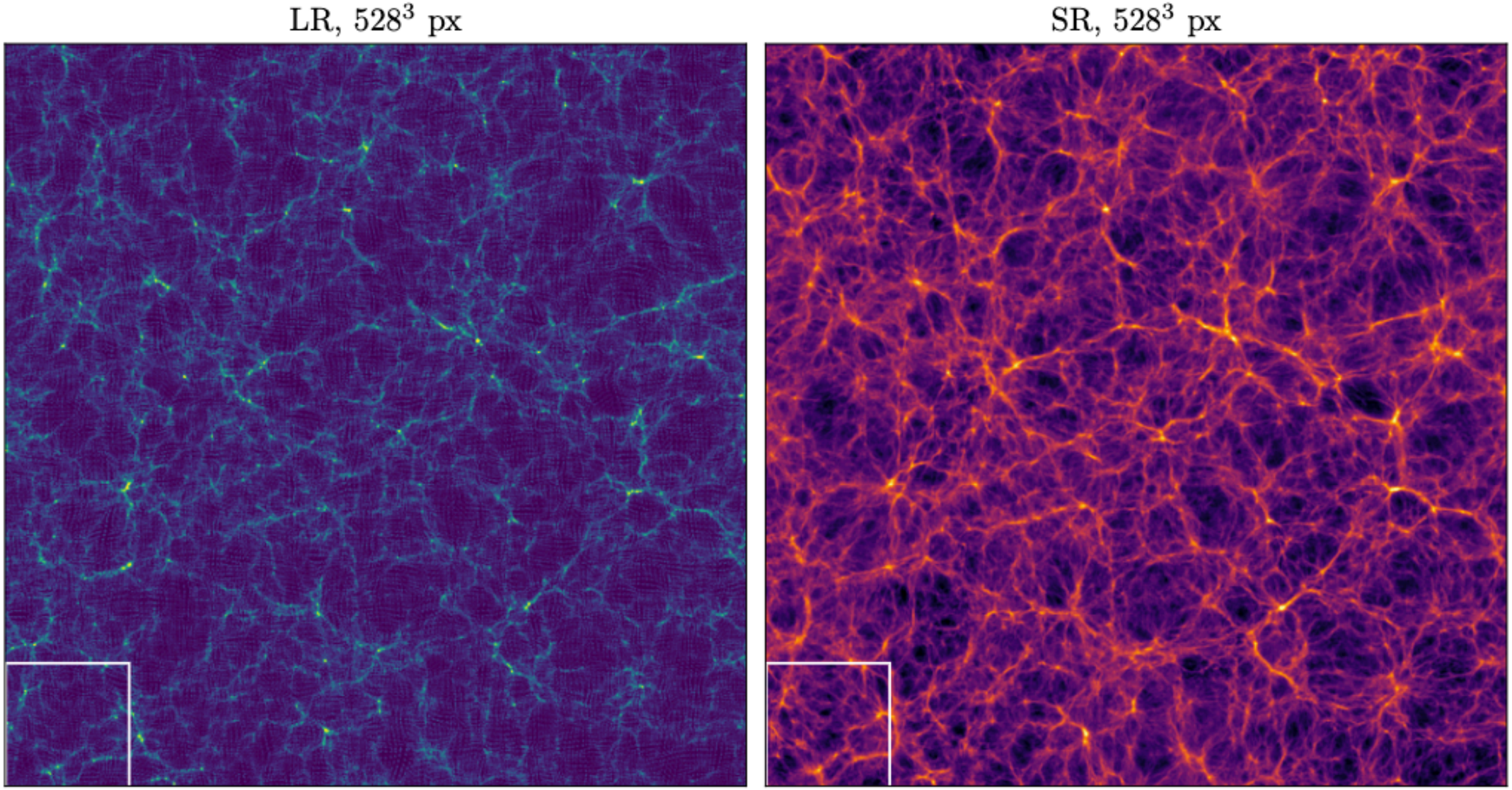


Results: SR matches HR on validation data





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)

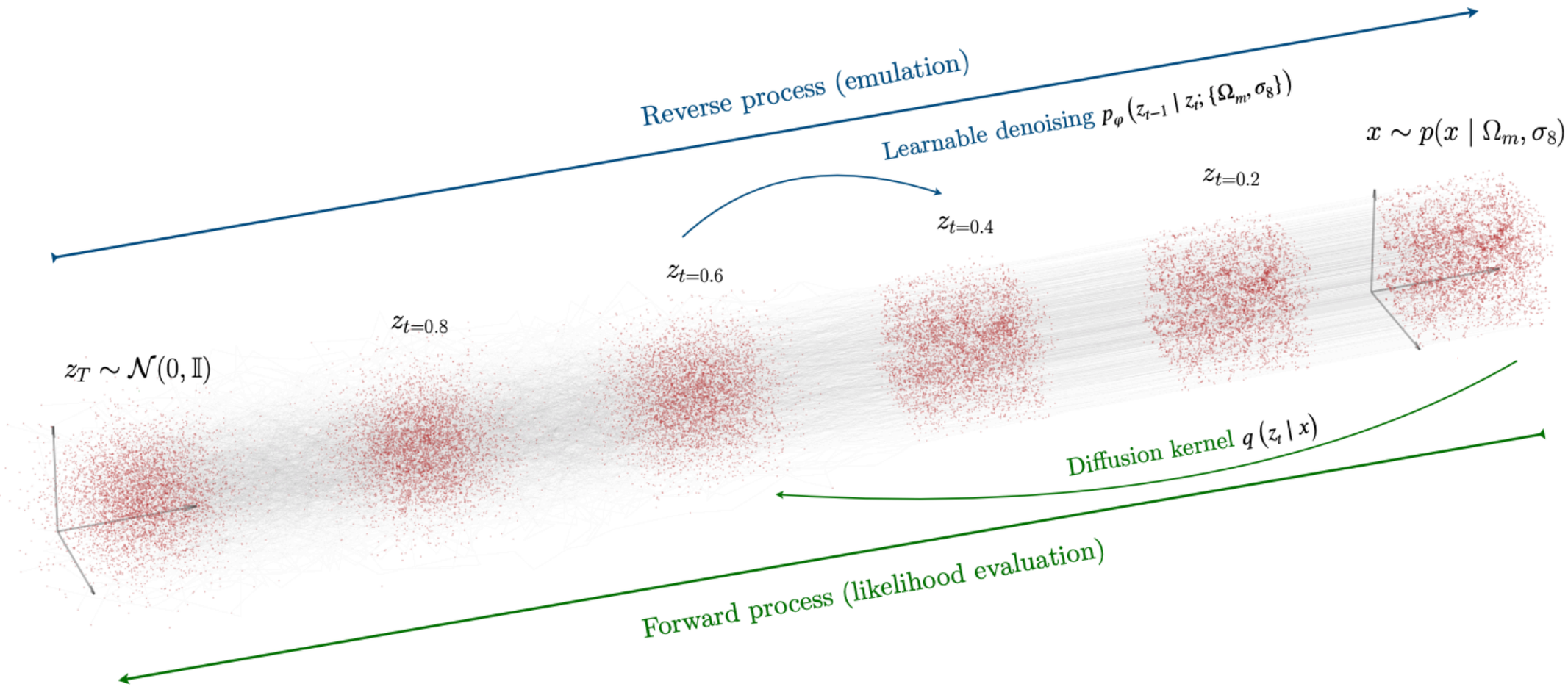


Figure 1. A schematic overview of the point cloud diffusion model, showing samples from the diffusion process at different diffusion times. During training, noise is added to a data sample x using the diffusion kernel $q(z_t \mid x)$ and a denoising distribution $p_\varphi(z_{t-1} \mid z_t)$ is learned.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