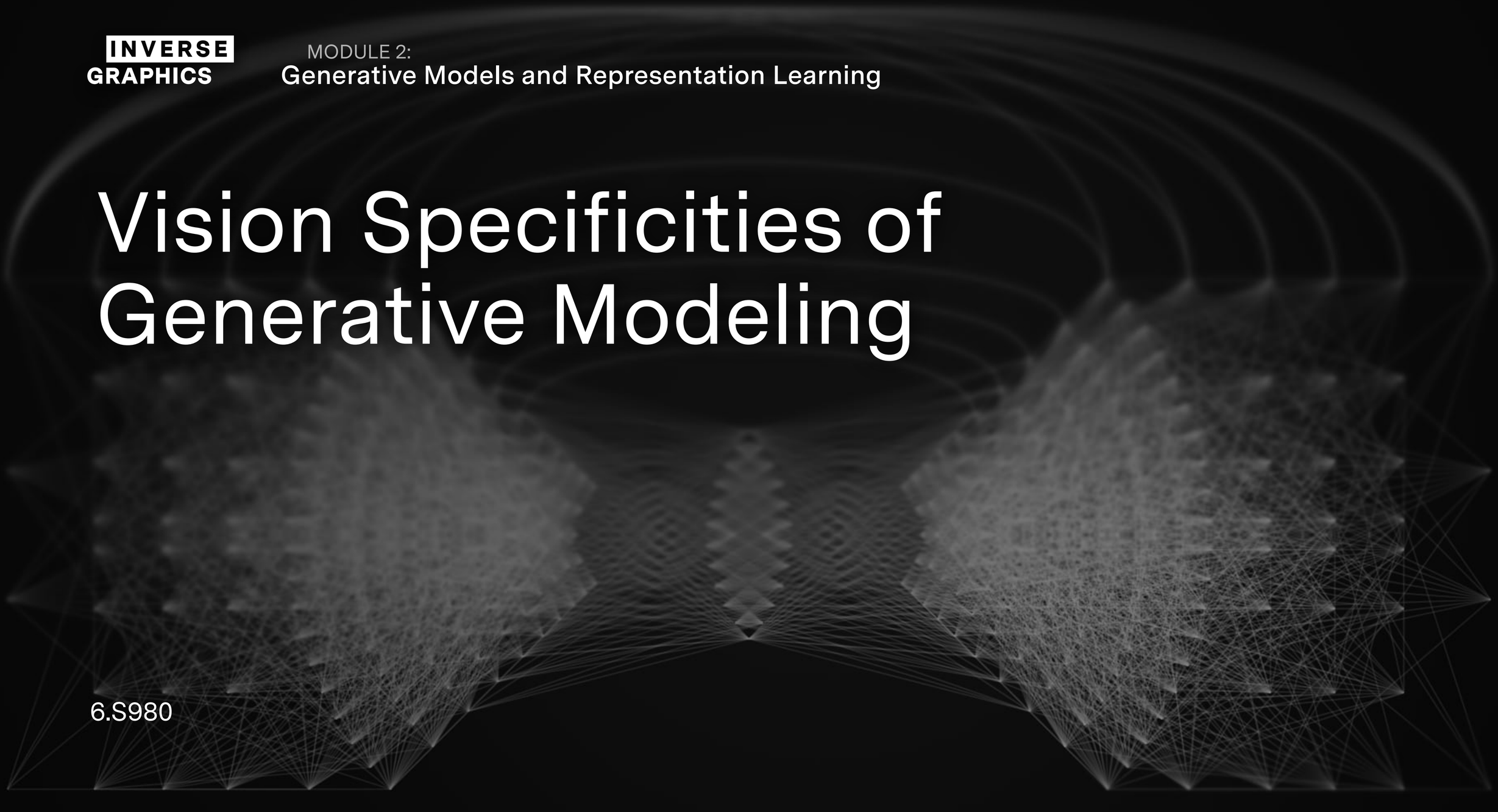
INVERSE GRAPHICS

MODULE 2:
Generative Models and Representation Learning

Vision Specificities of Generative Modeling



6.S980

High-Level Recap: Diffusion & Flow Matching

Empirical Data Distribution

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$

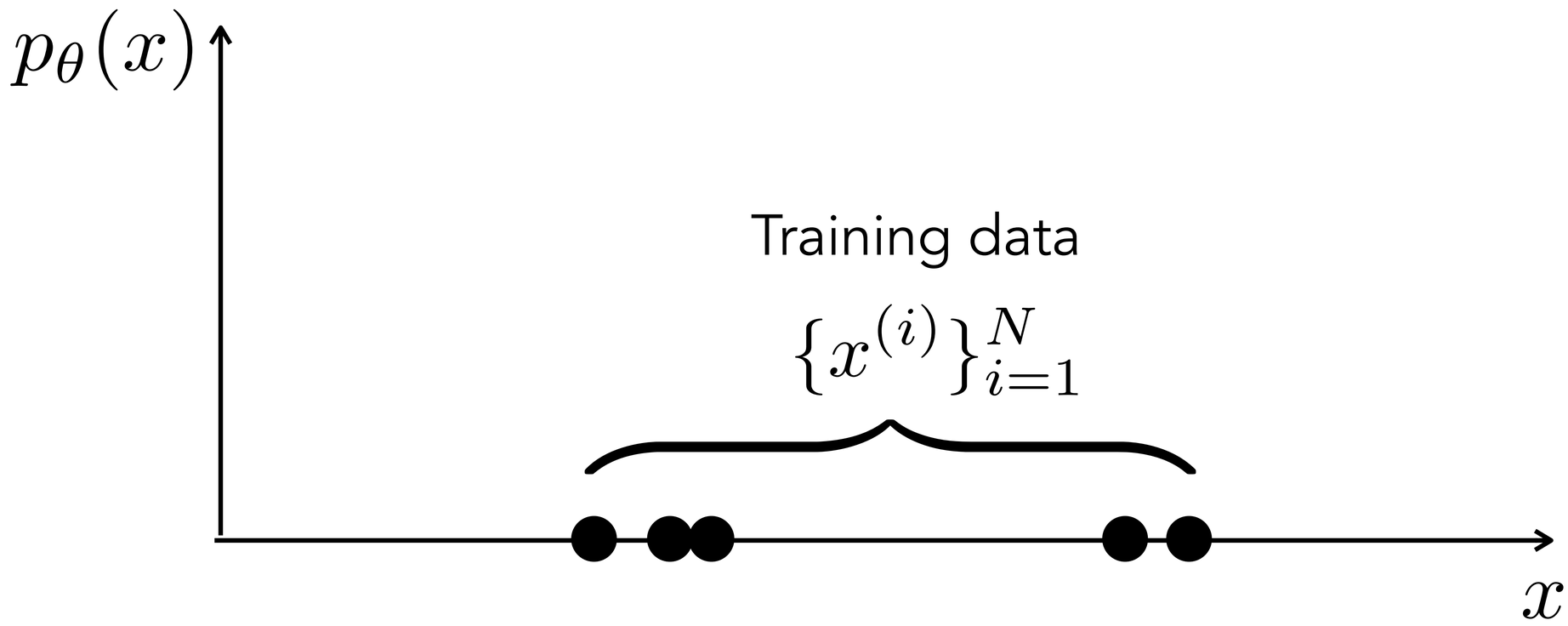
Empirical Data Distribution

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$



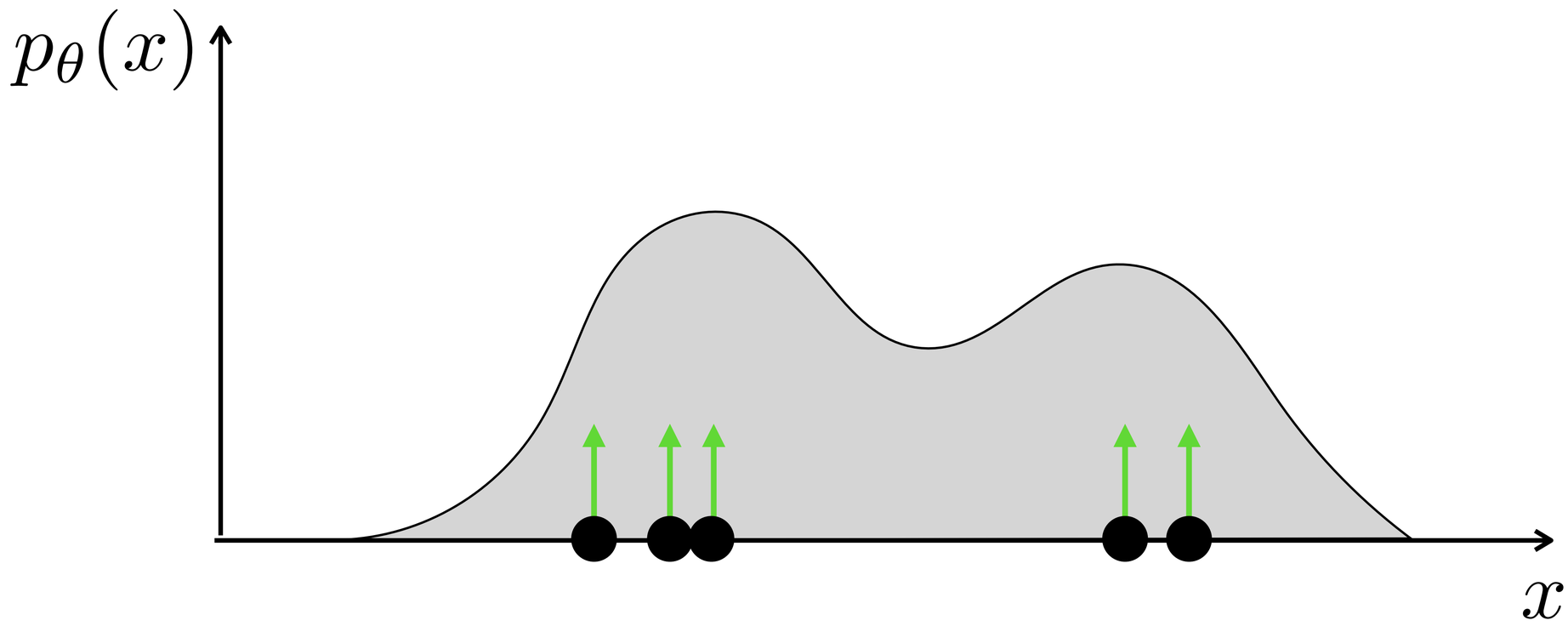
Empirical Data Distribution

$$p_\theta : \mathcal{X} \rightarrow [0, \infty)$$



Density Models

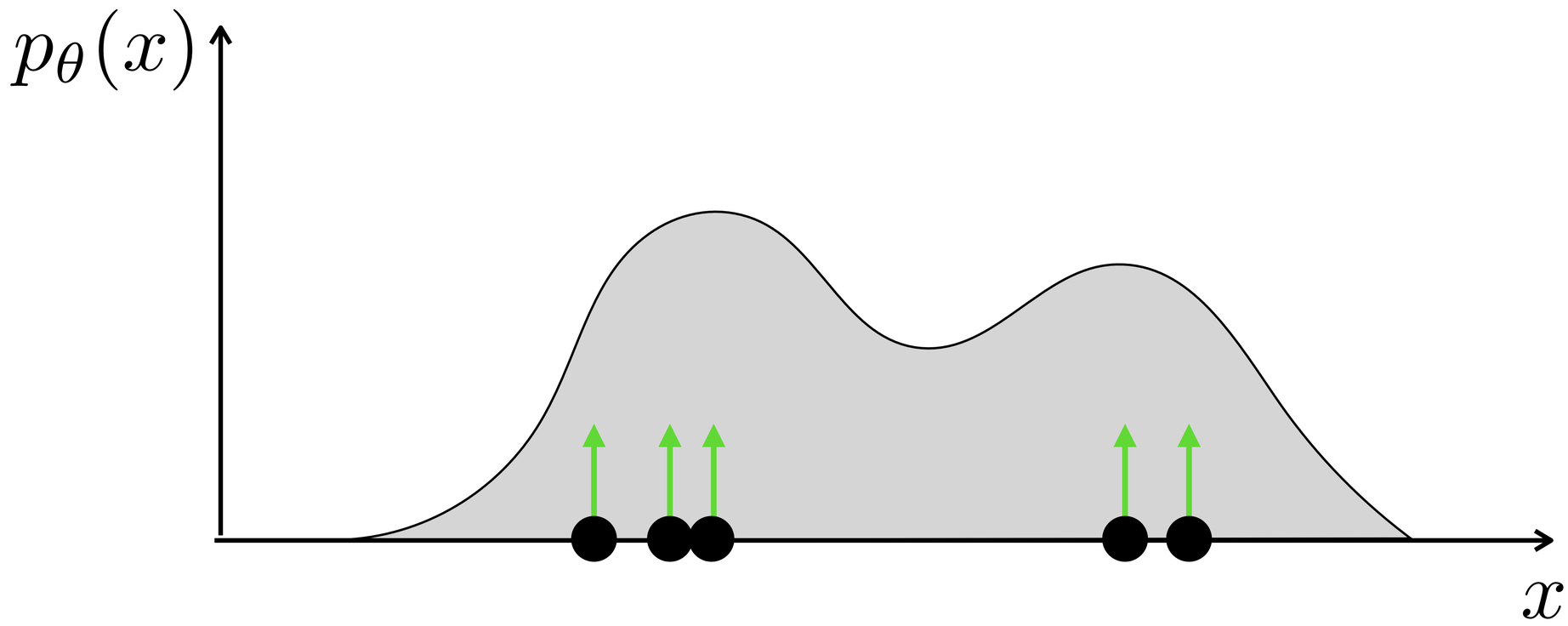
$$p(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}$$



Density Models

$$p(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}$$

Can use to define a continuous data distribution by defining $f_\theta(\mathbf{x})$, a real-valued function.

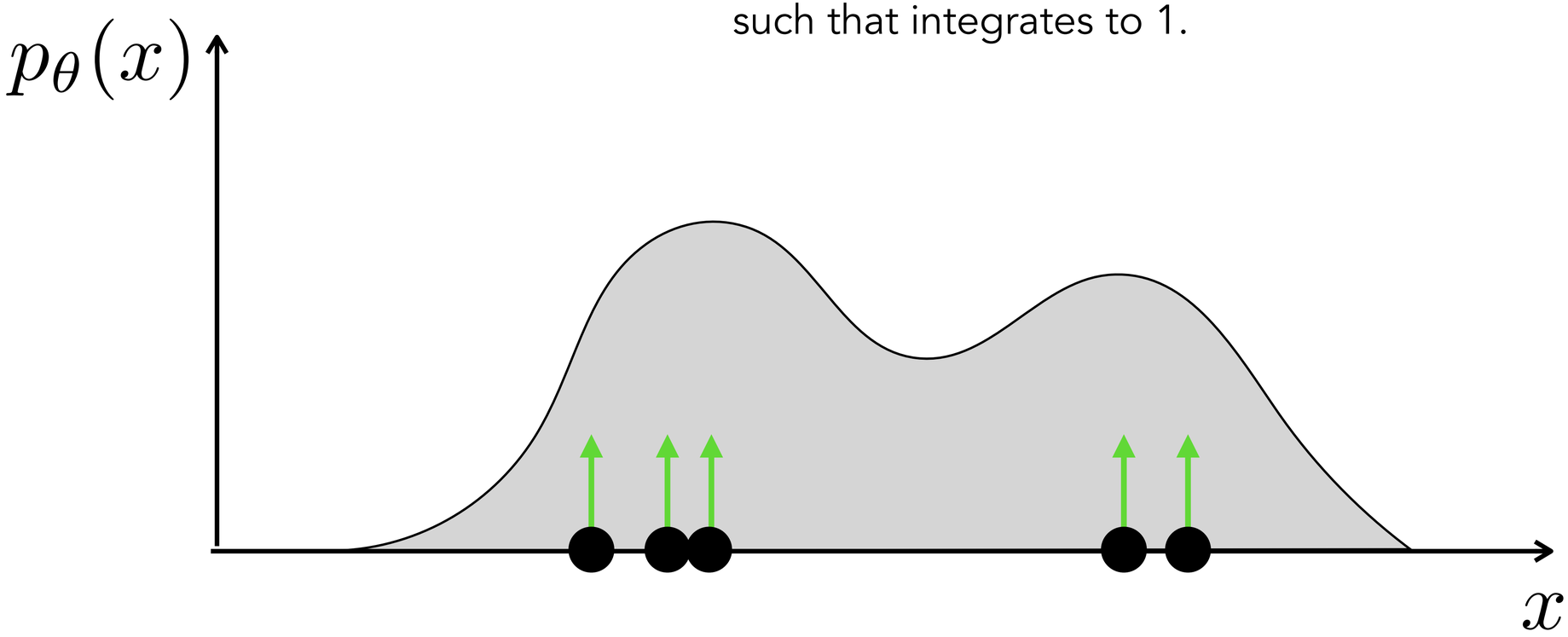


Density Models

$$p(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta}$$

Can use to define a continuous data distribution by defining $f_\theta(\mathbf{x})$, a real-valued function.

Z_θ is a normalizing constant such that integrates to 1.



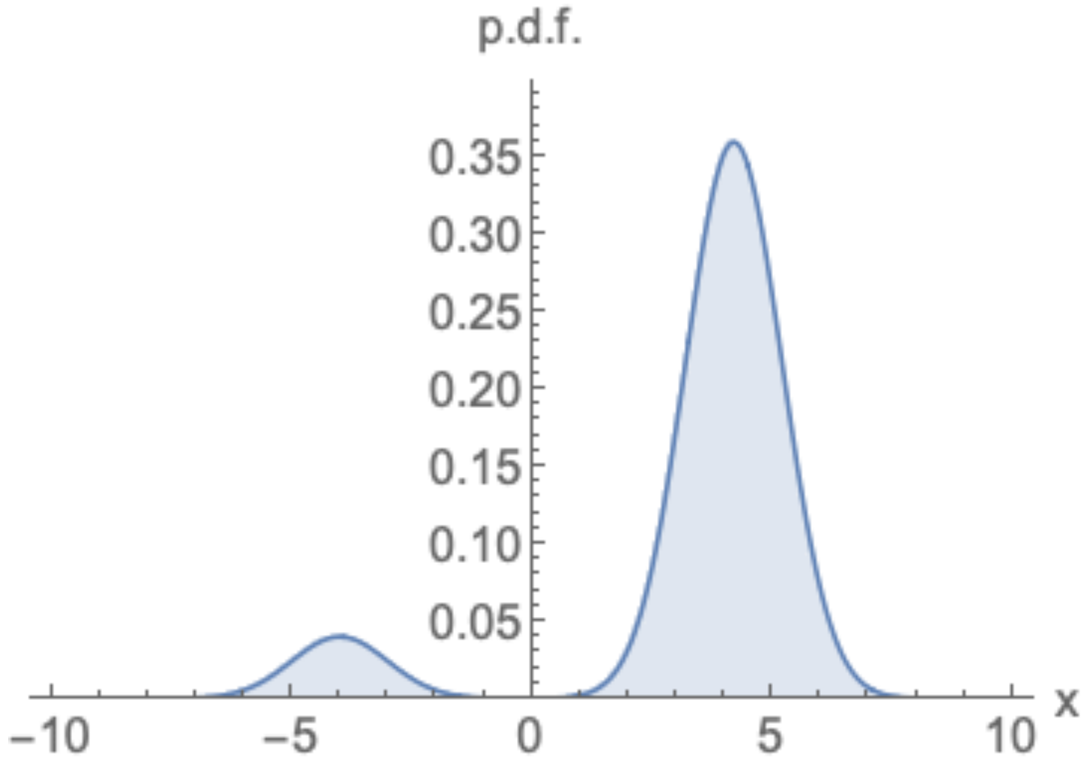
Score-Based Generative Modeling

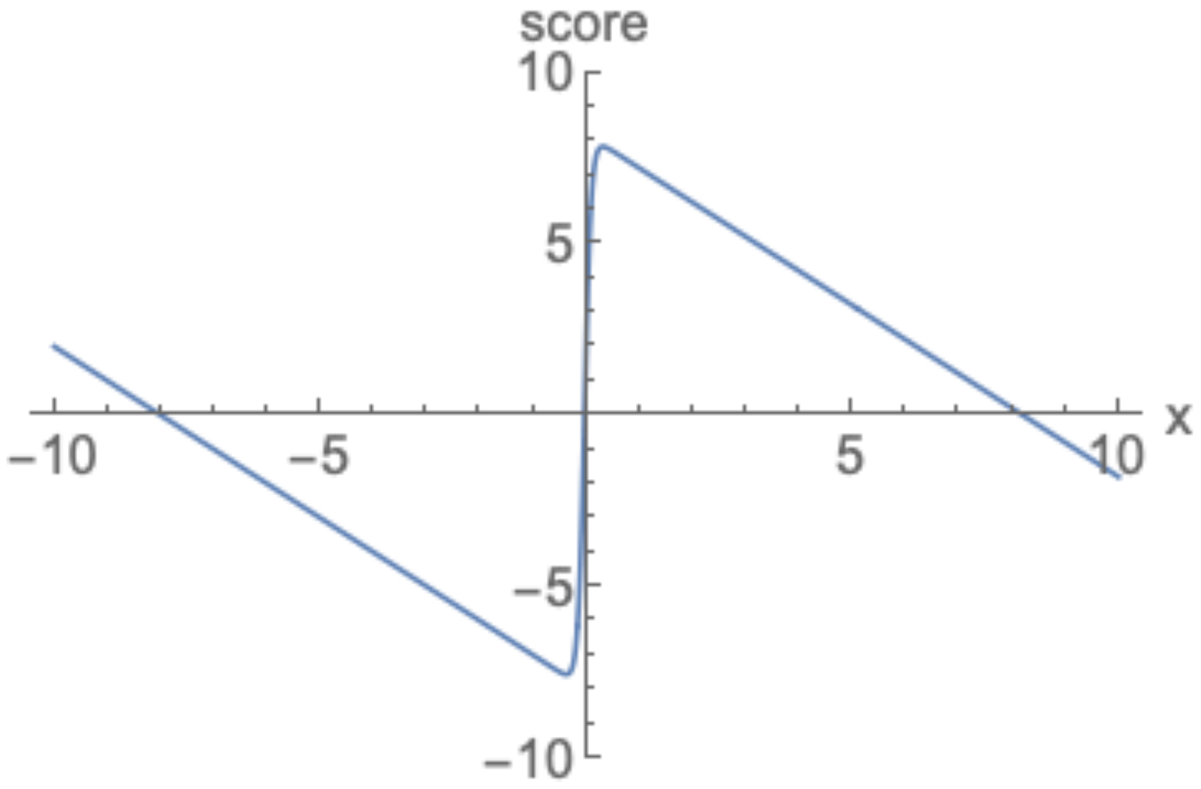
$$p(\mathbf{x}) = \frac{e^{-f(\mathbf{x})}}{Z_\theta}$$

An important tool in generative modeling is the *score* of $p(\mathbf{x})$. For a particular datapoint $\mathbf{x}$, the score tells you the direction towards higher-probability regions:

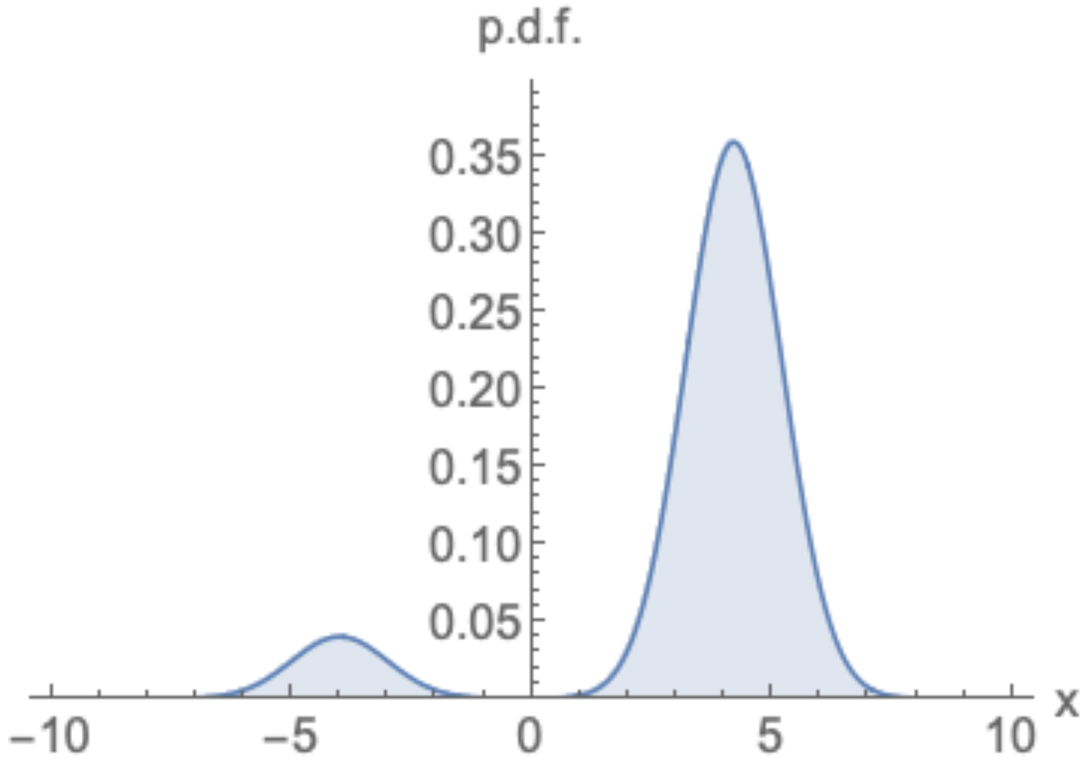
$$\nabla \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_\theta}_{=0} = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x})$$

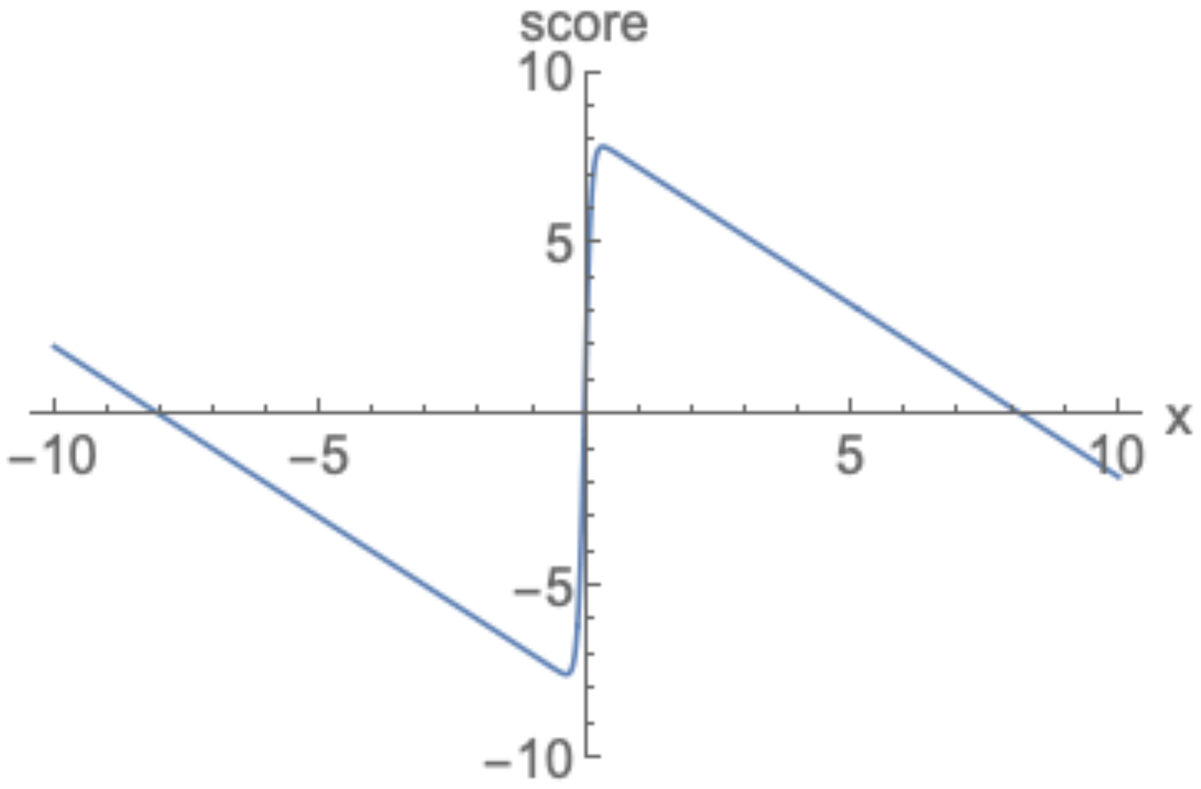
Score Functions = Gradients of the log-likelihood





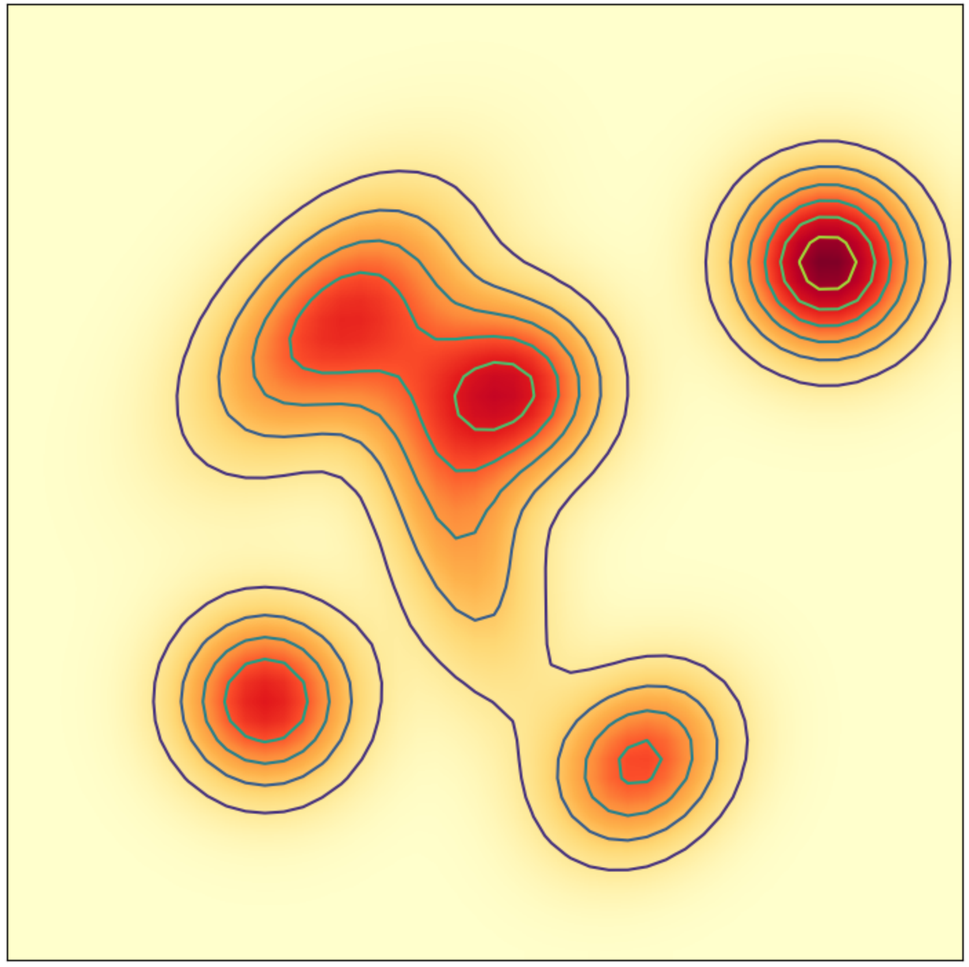
Score Functions = Gradients of the log-likelihood



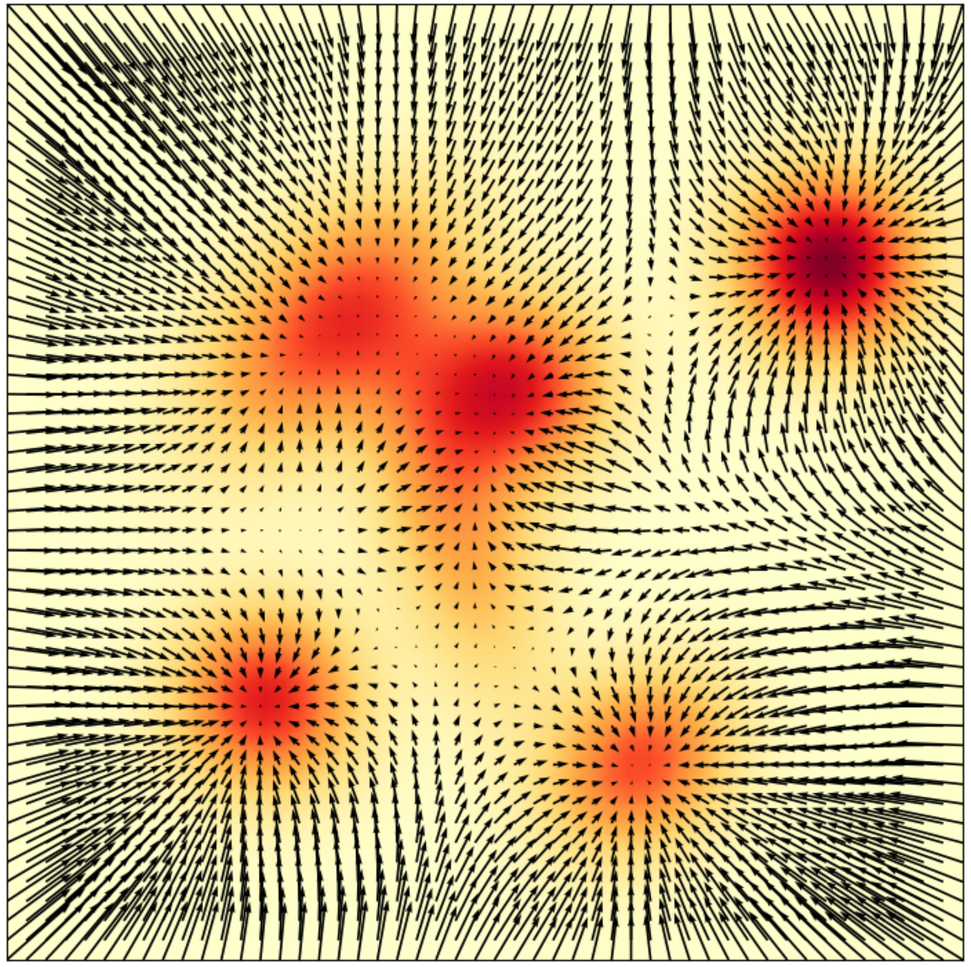


Score Functions = Gradients of the log-likelihood

Log-likelihood: $\log q(x)$



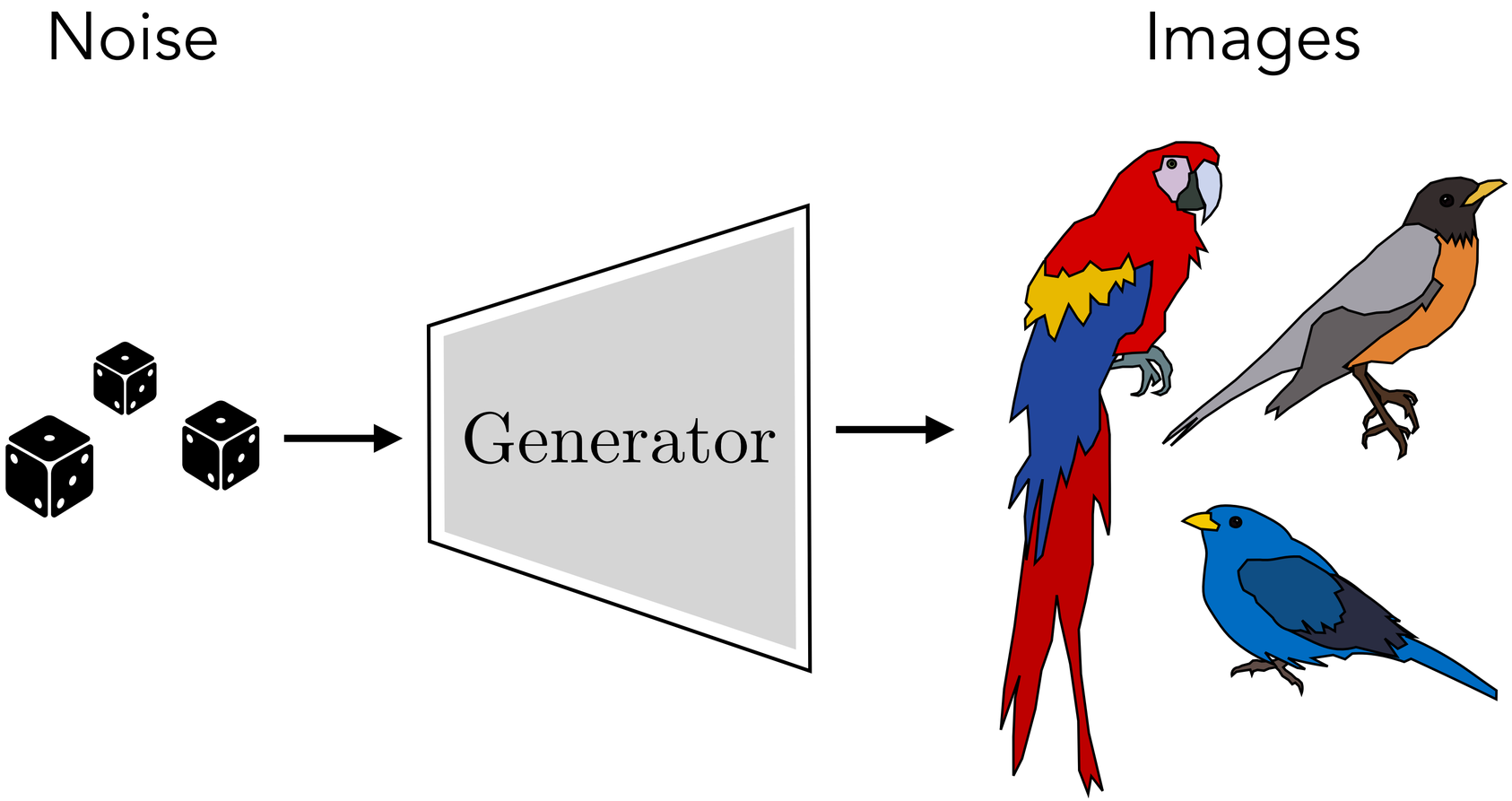
Score function: $\nabla \log q(x)$



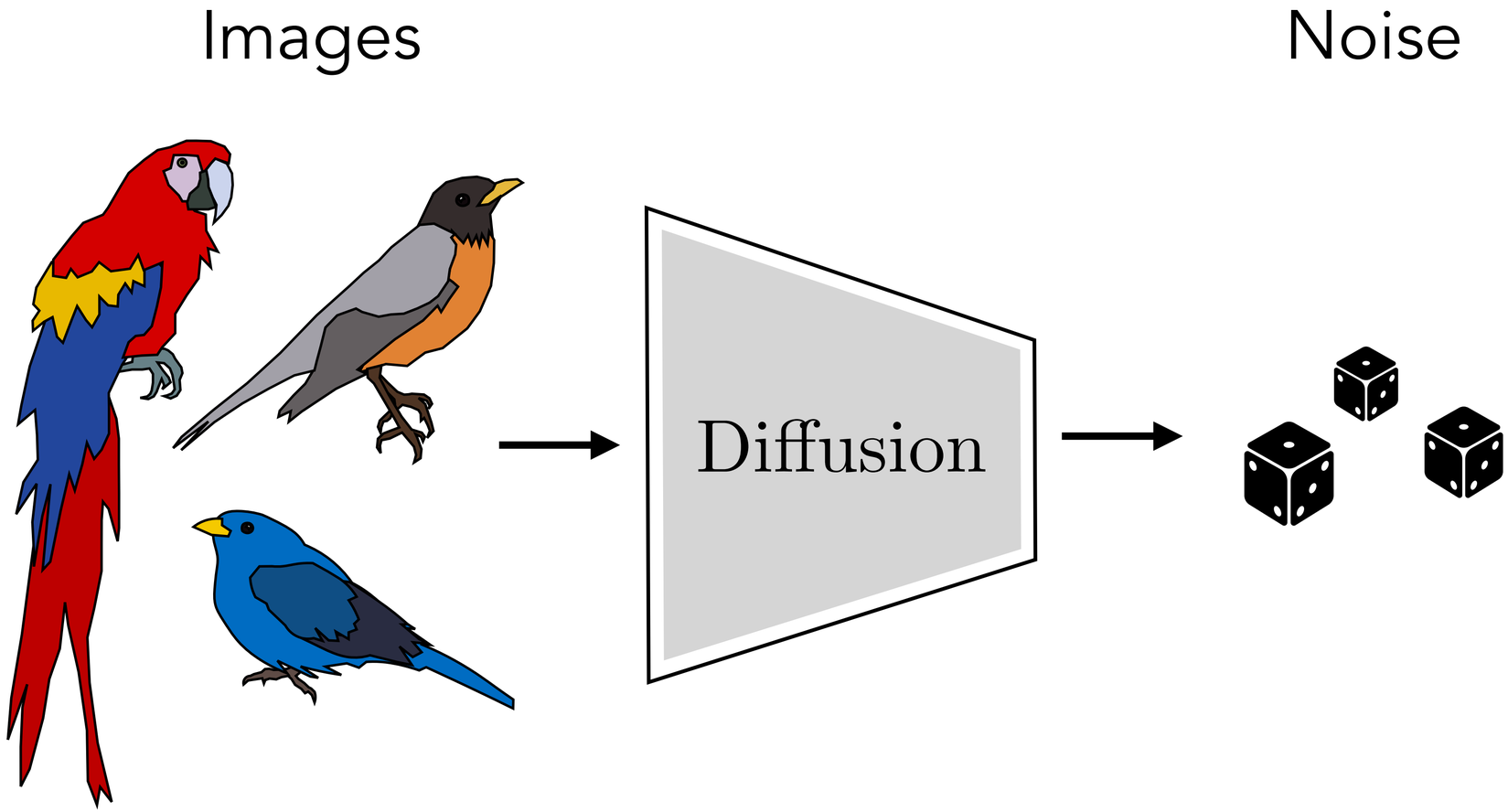
Diffusion Models as Score-Based Generative Modelers

One of many useful perspectives on diffusion models.

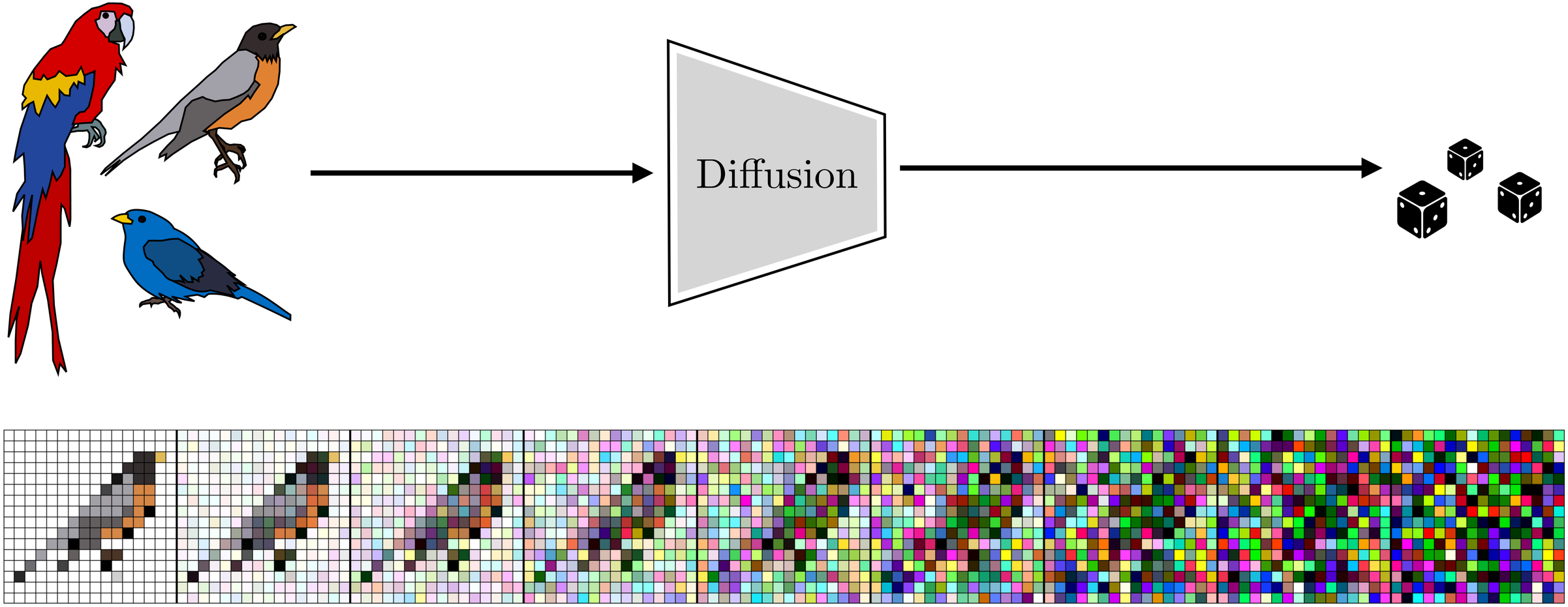
Diffusion Models



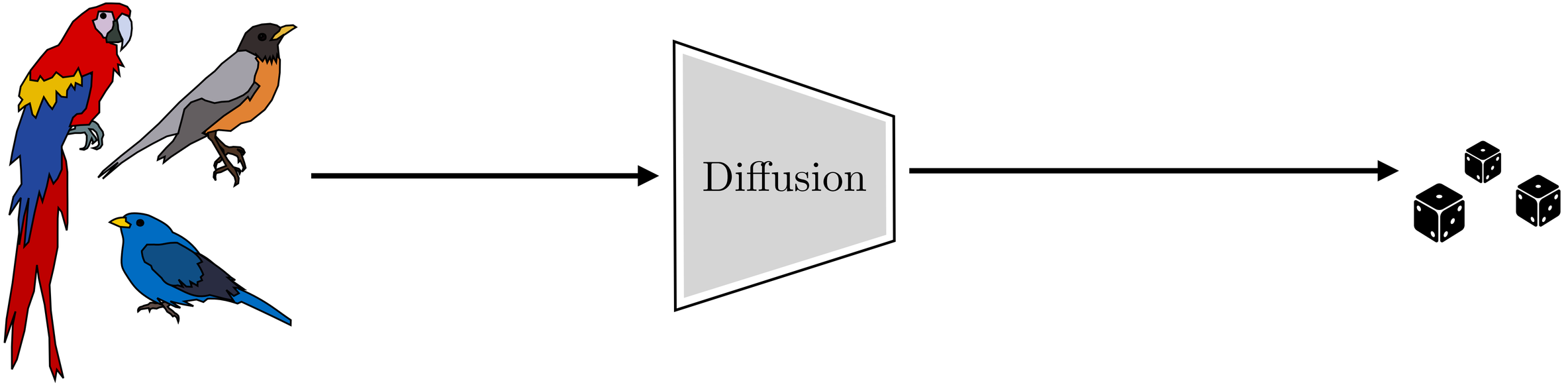
Diffusion Models

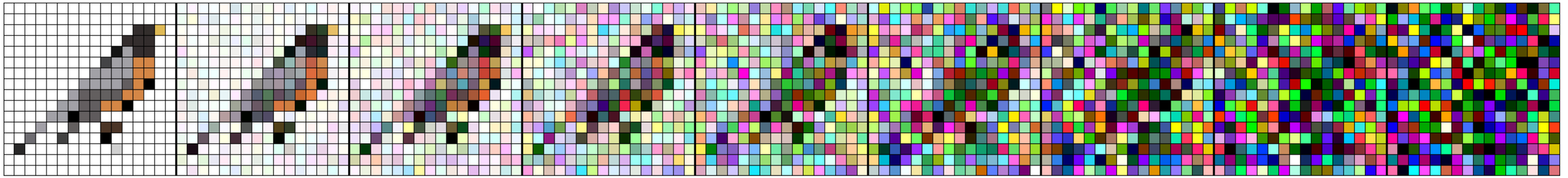


Diffusion Models



Diffusion Models

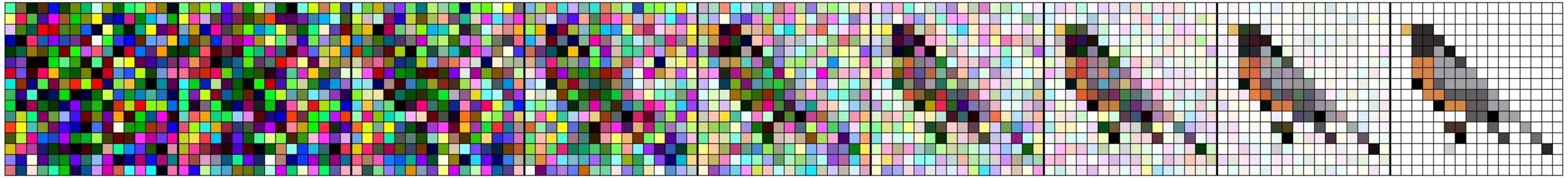




Diffusion: Just add noise

Diffusion Models

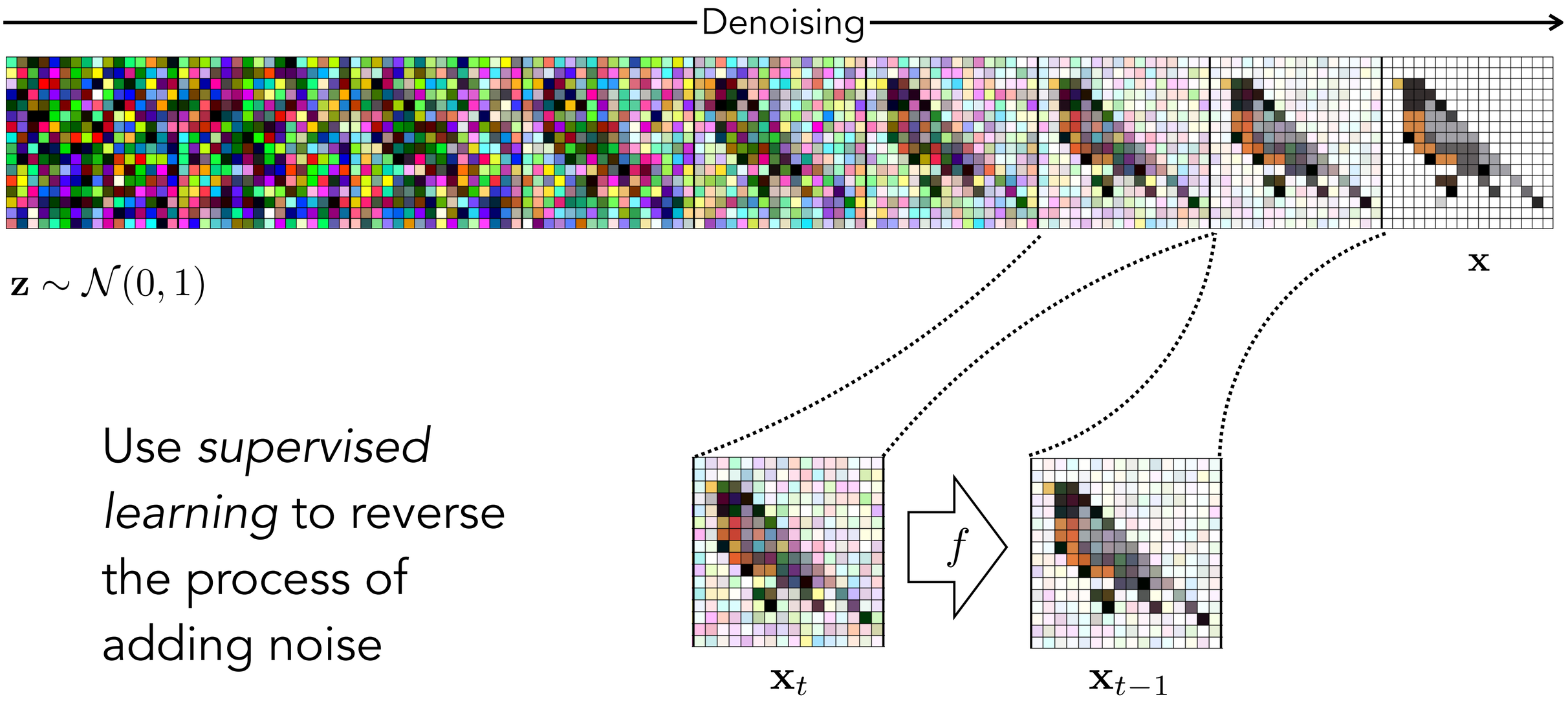
Denoising



$\mathbf{x}$

$\mathbf{z} \sim \mathcal{N}(0, 1)$

Diffusion Models

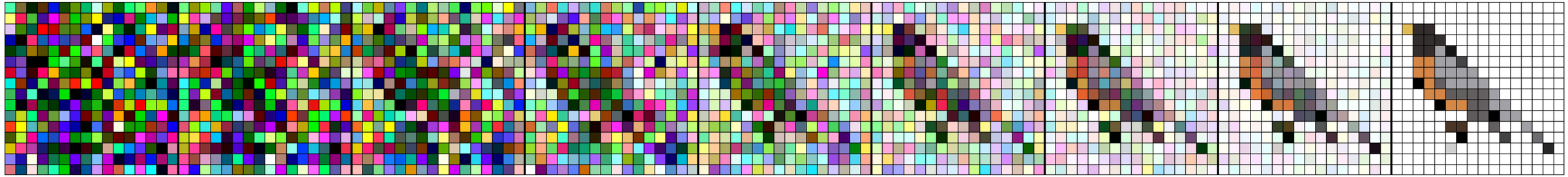


Use *supervised learning* to reverse the process of adding noise

Diffusion Models

$\mathbf{z} \sim \mathcal{N}(0, 1)$

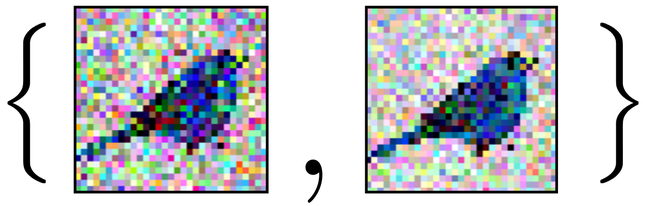
$\mathbf{x}$



Denoising

Training data

$\mathbf{x}_t$ $\mathbf{x}_{t-1}$



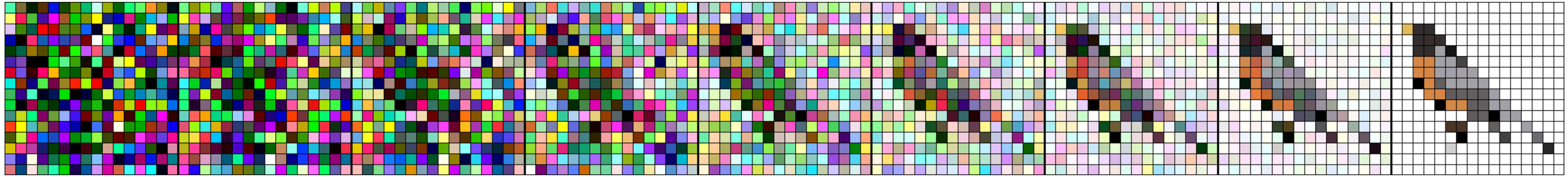




Diffusion Models

$\mathbf{z} \sim \mathcal{N}(0,1)$

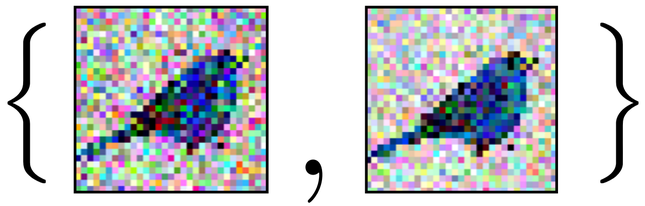
$\mathbf{x}$



Denoising

Training data

$\mathbf{x}_t$ $\mathbf{x}_{t-1}$





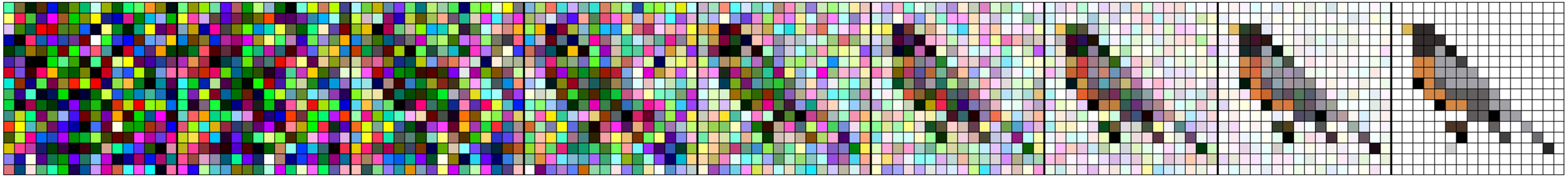


$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_t), \mathbf{x}_{t-1})$$

Diffusion Models

$\mathbf{z} \sim \mathcal{N}(0, 1)$

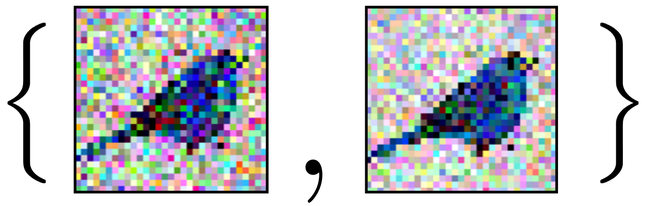
$\mathbf{x}$



Denoising

Training data

$\mathbf{x}_t$ $\mathbf{x}_{t-1}$







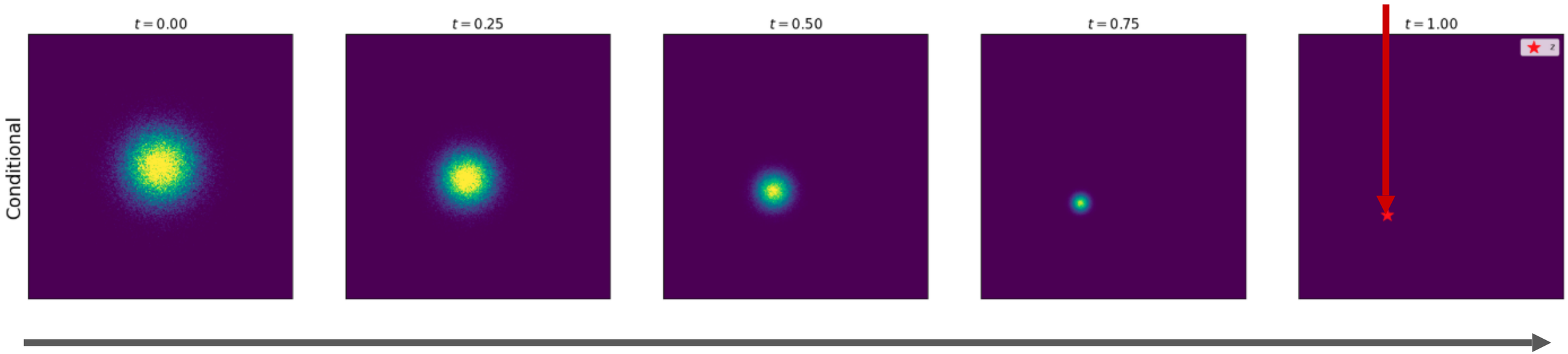
$$\arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \mathcal{L}(f(\mathbf{x}_t), \mathbf{x}_{t-1})$$

Converts generative modeling into a series of supervised prediction problems

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

p_{init}

z



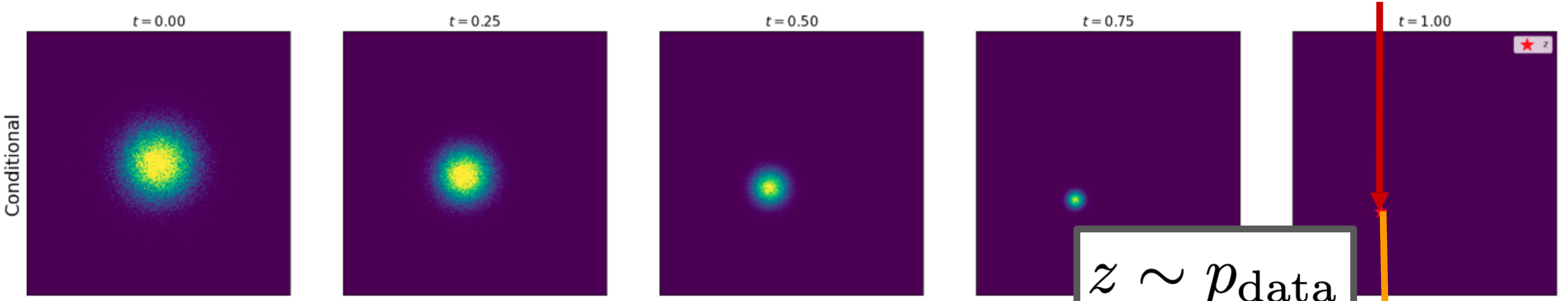
t=0

t=1

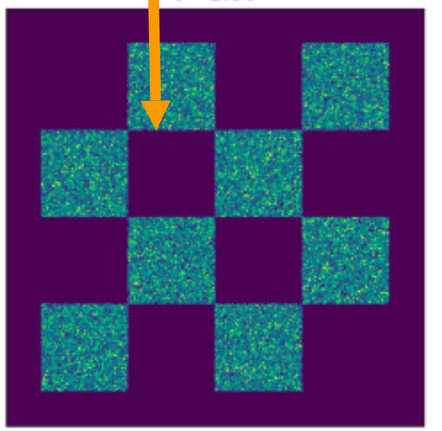
p_{init}

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

z



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

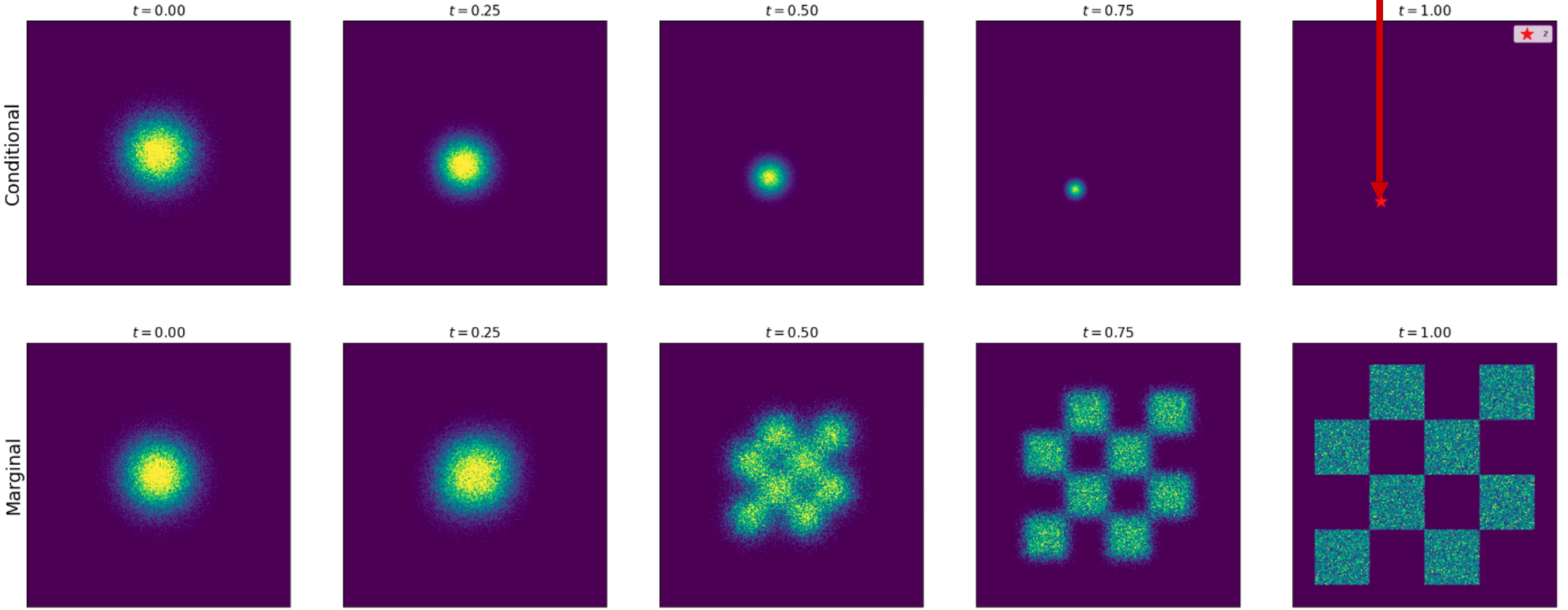


p_{data}

p_{init}

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

z

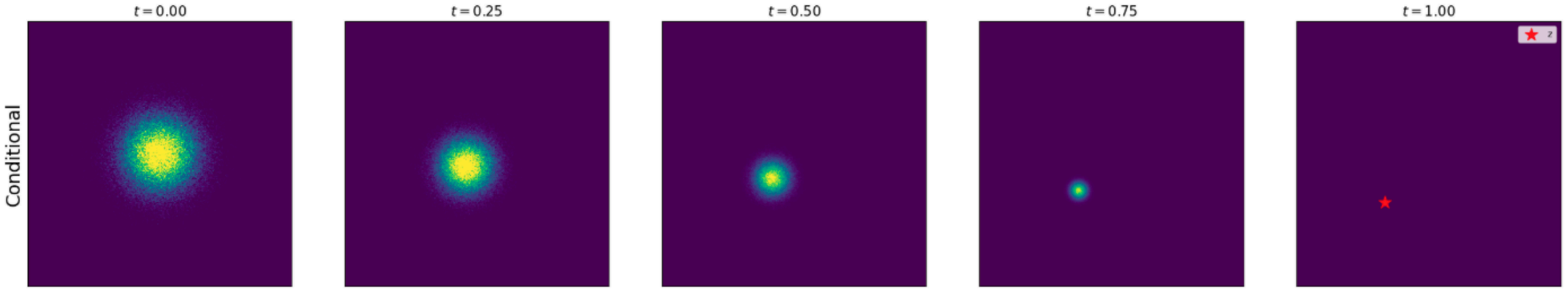


p_{init}

Marginal Probability Path p_t

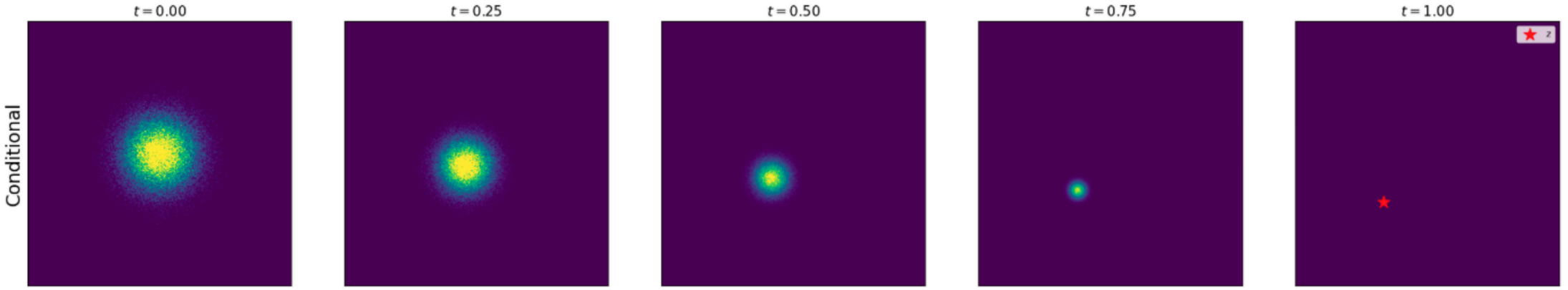
p_{data}

What is different about diffusion of, say, 2D points and images?

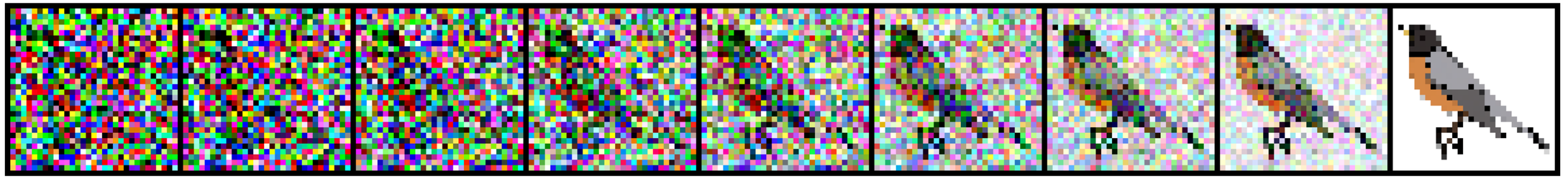


Different geometry!

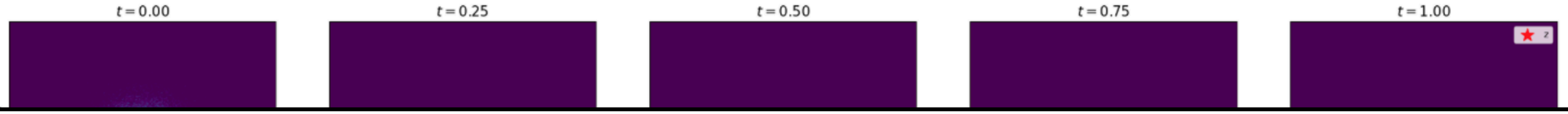
What is different about diffusion of, say, 2D points and images?



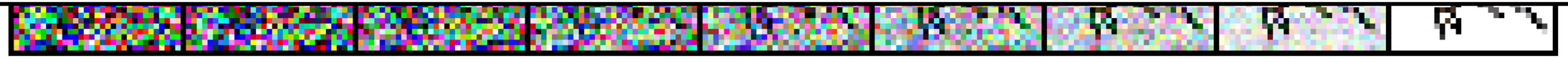
Different geometry!



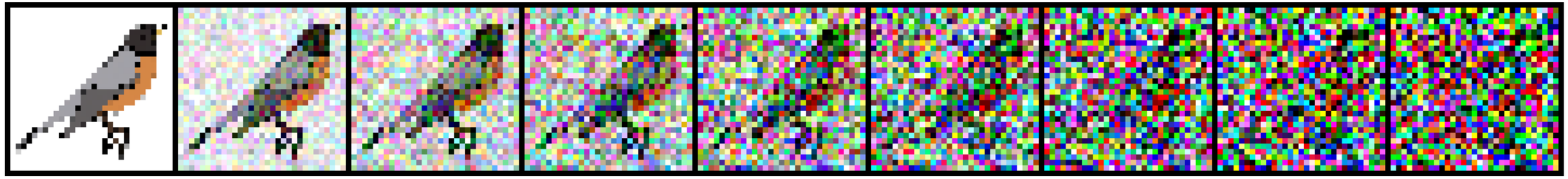
What is different about diffusion of, say, 2D points and images?



Let’s bring some of the tools for image processing to bear that we have learned about so far!

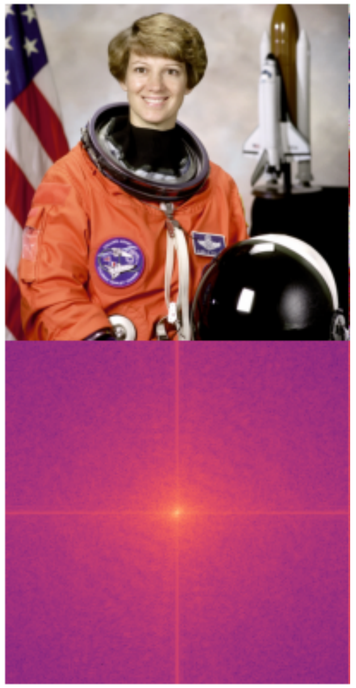


What is different about diffusion of, say,
2D points and images?



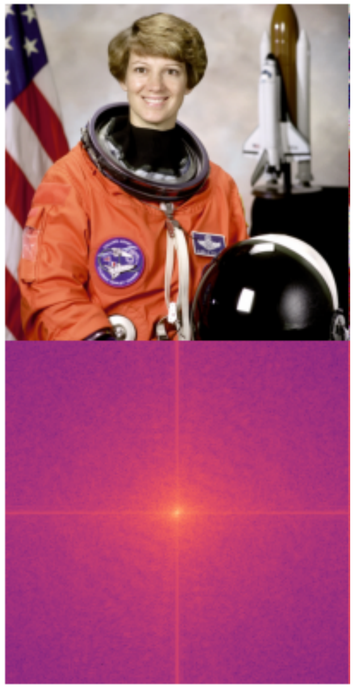
Let’s look at what happens to these images in frequency space!

Forward Diffusion in Frequency Space

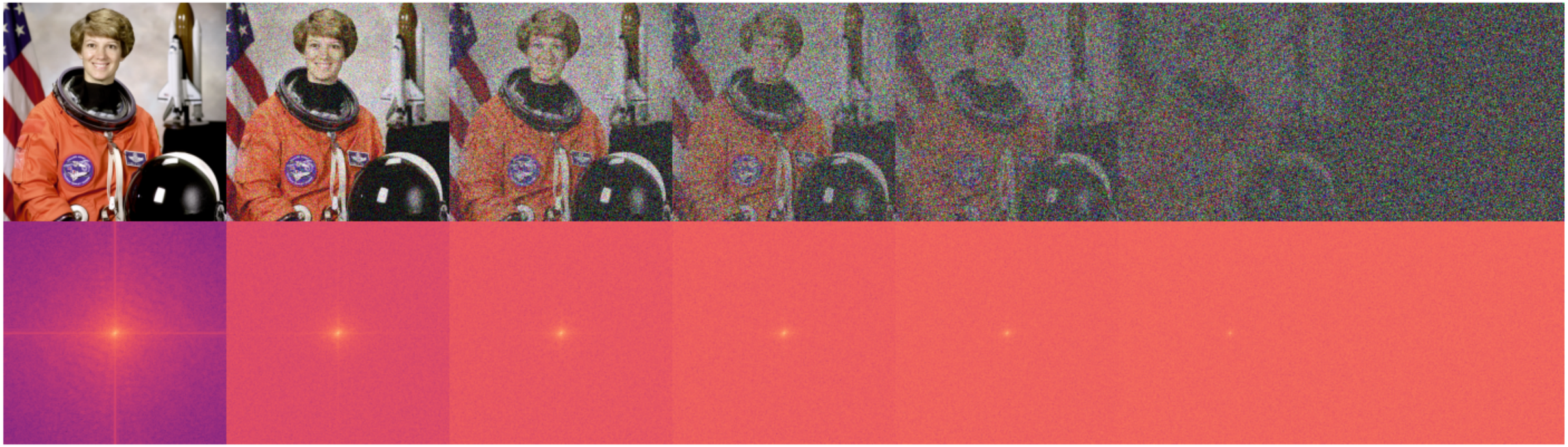


Forward Diffusion in Frequency Space

Diffusion

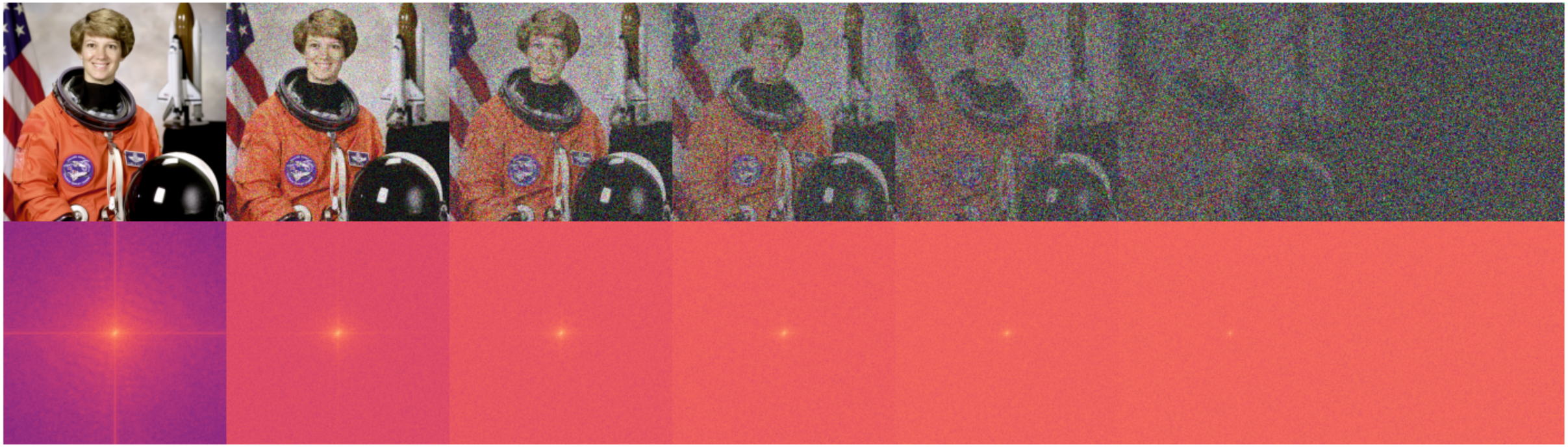


Forward Diffusion in Frequency Space

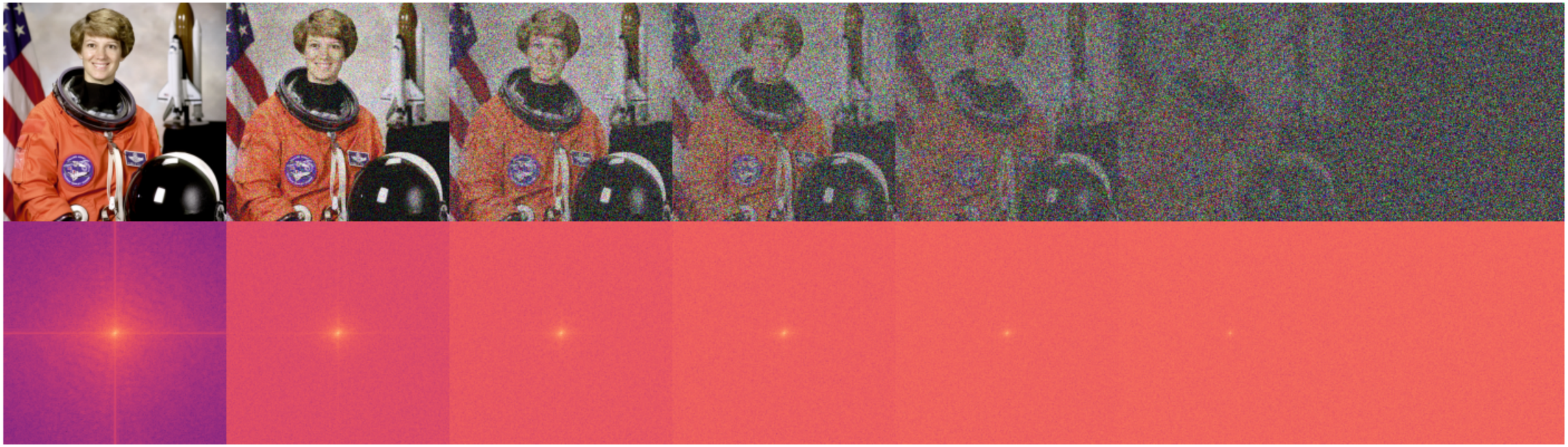


Forward Diffusion in Frequency Space

Diffusion



Forward Diffusion in Frequency Space

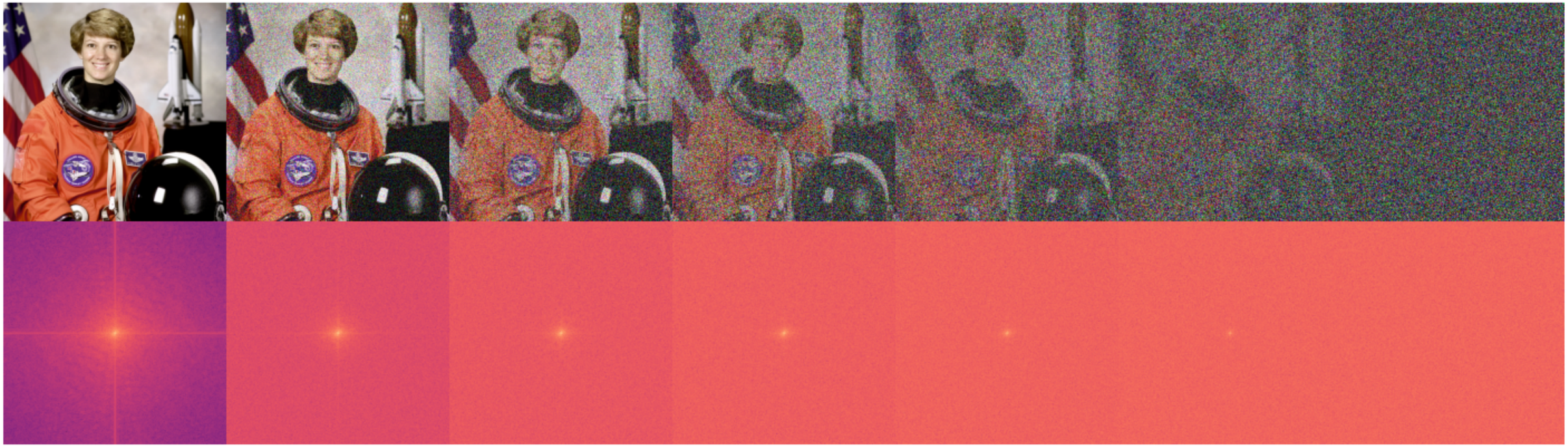


What do we see?

Low frequencies get “drowned out” last!

Forward Diffusion in Frequency Space

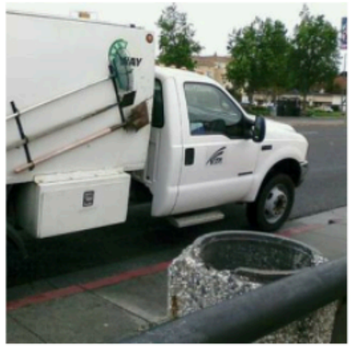
Diffusion →



What do we see?

Low frequencies get “drowned out” last!

The spectral density of natural images



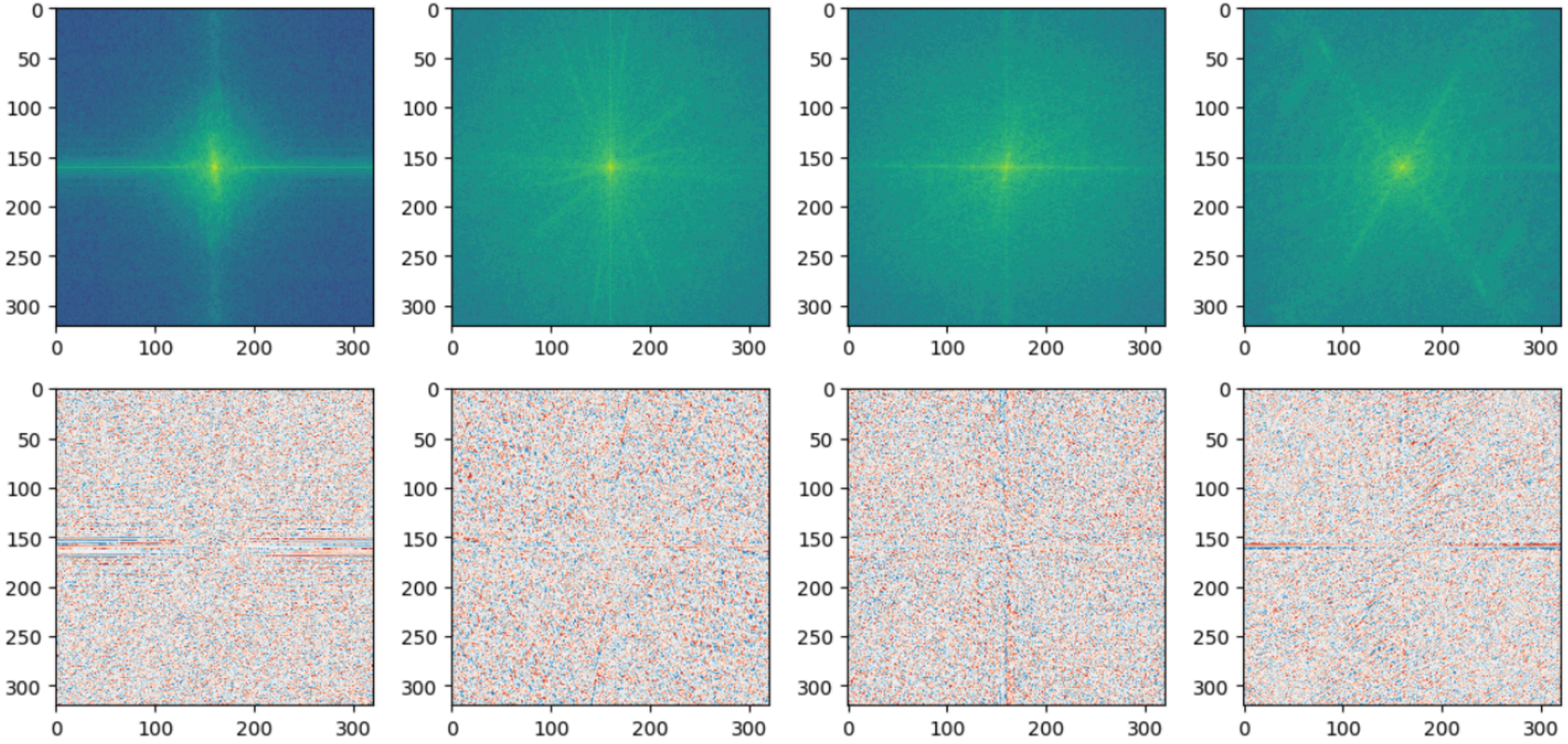


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

```
spec = np.fft.fft2(image)
spec = np.fft.fftshift(spec)
spdj = np.log(np.abs(spec))
```

The spectral density of natural images

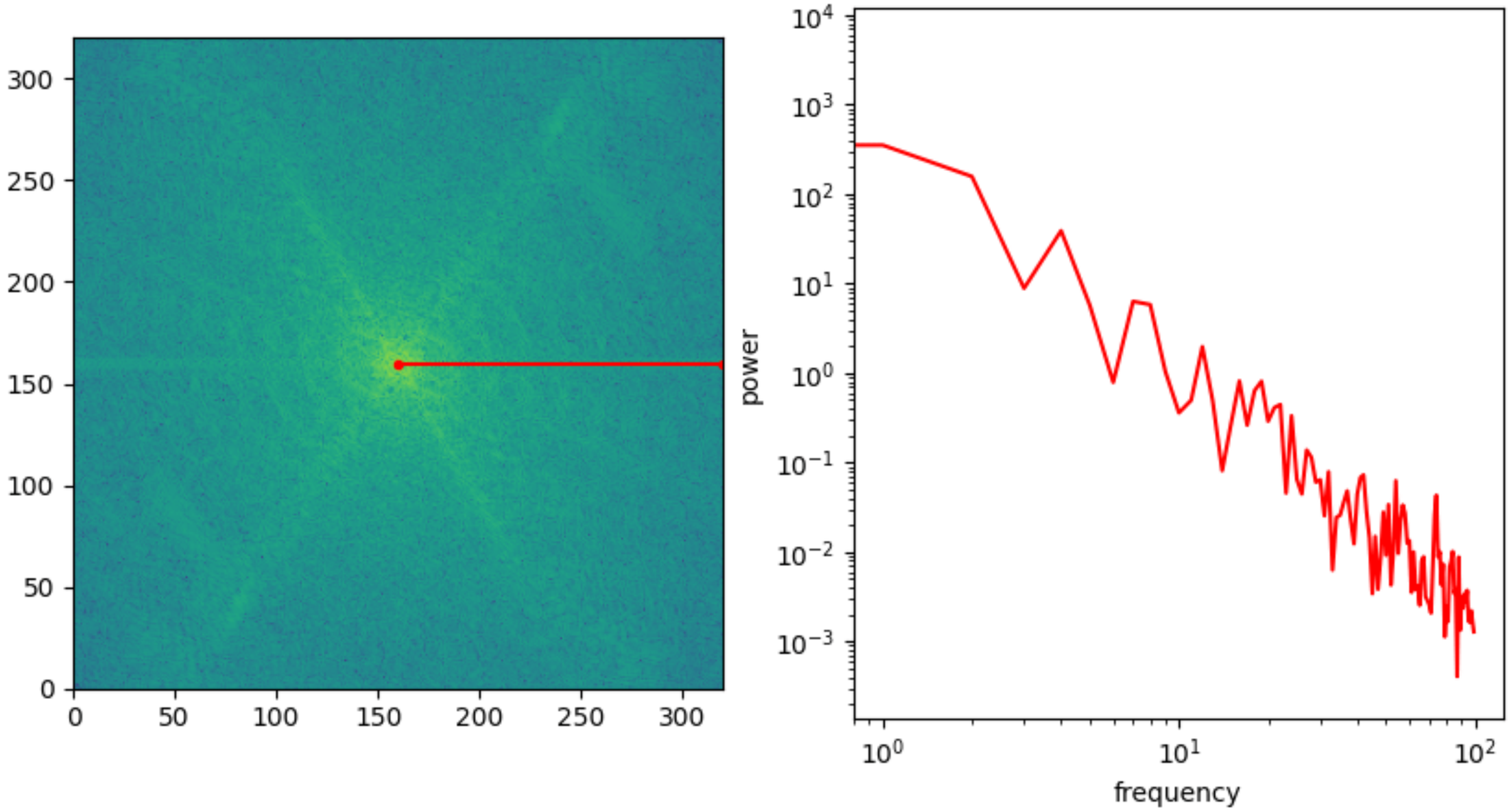


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

The spectral density of natural images

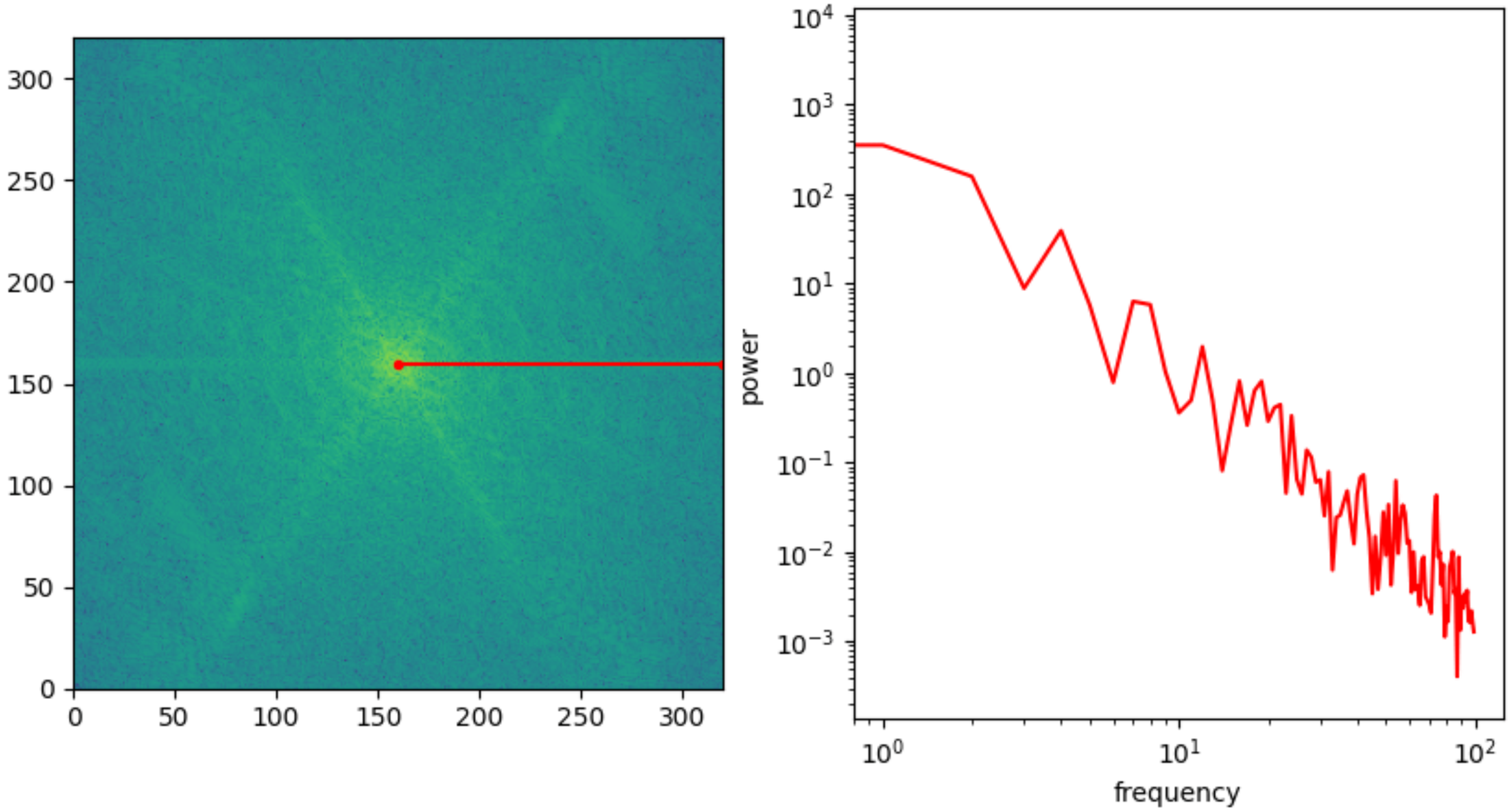


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

The spectral density of natural images

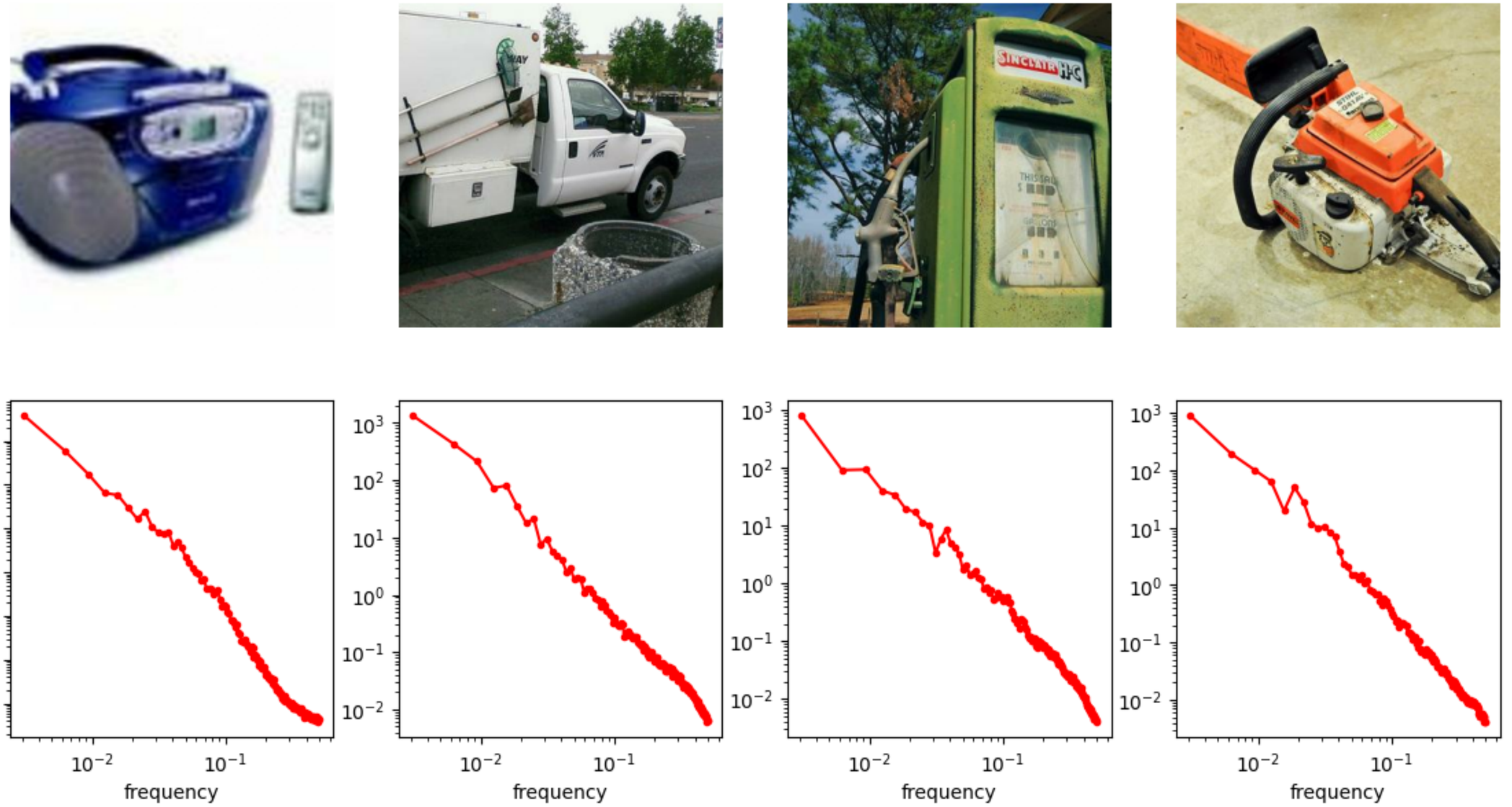


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

The spectral density of natural images



Insight: Spectral Power of images follows a “power law”, i.e., it’s linear in the log domain.

Note: this is true *on aggregate*, it is easy to find images that behave differently.

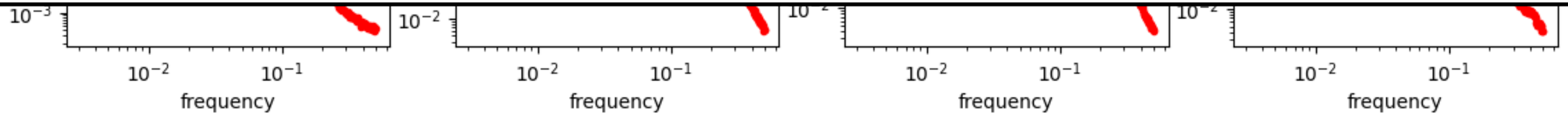


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

What happens when we add noise?

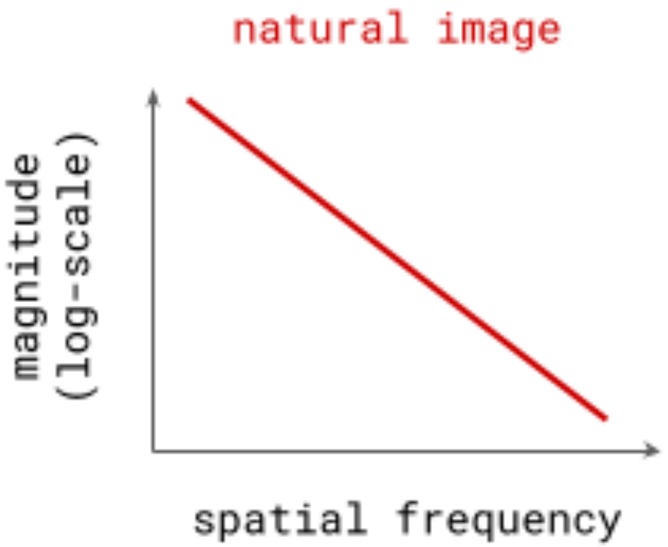


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

What happens when we add noise?

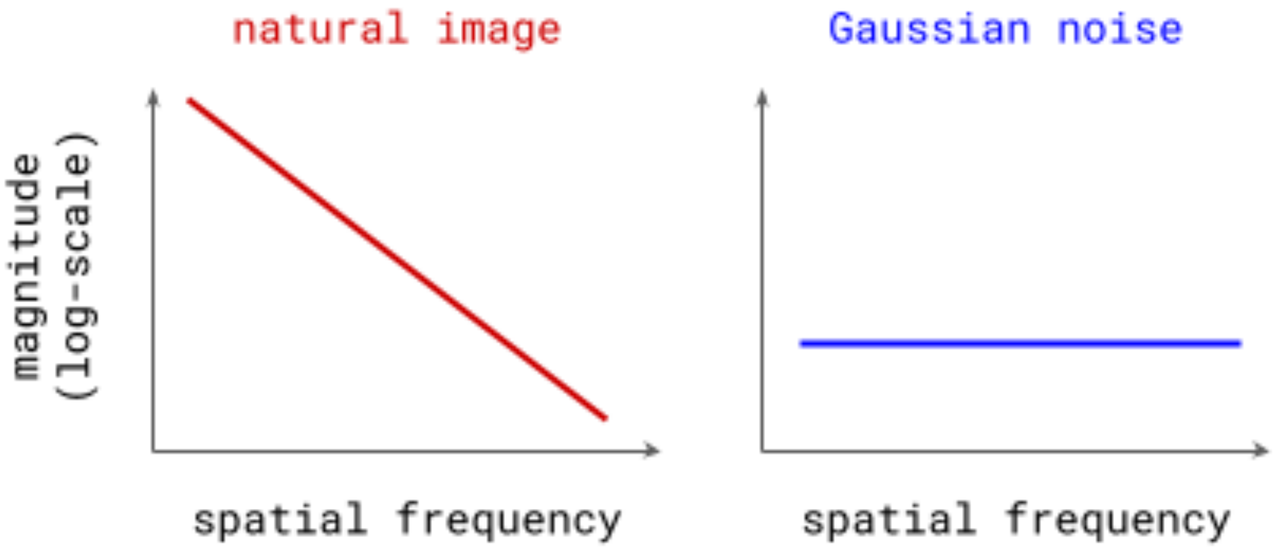


Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

What happens when we add noise?

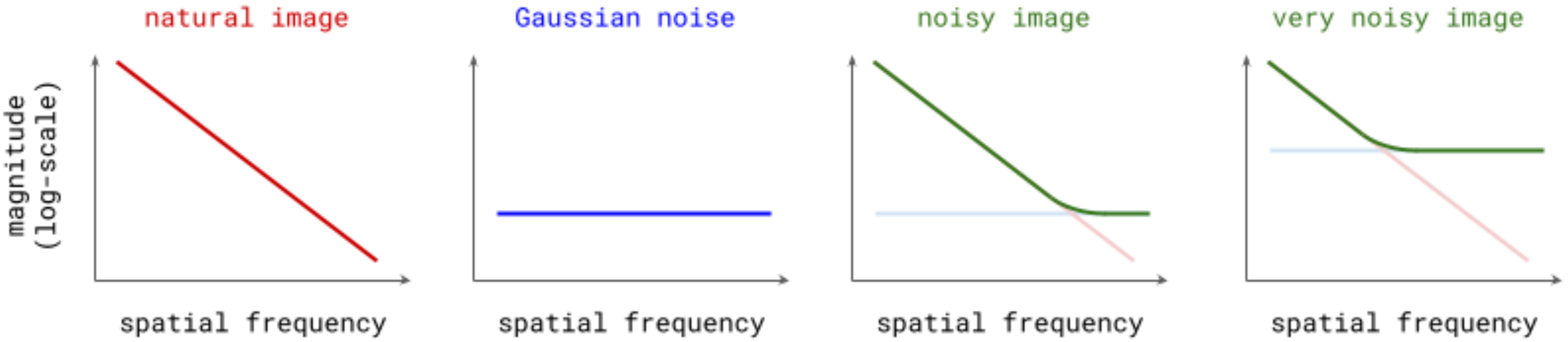
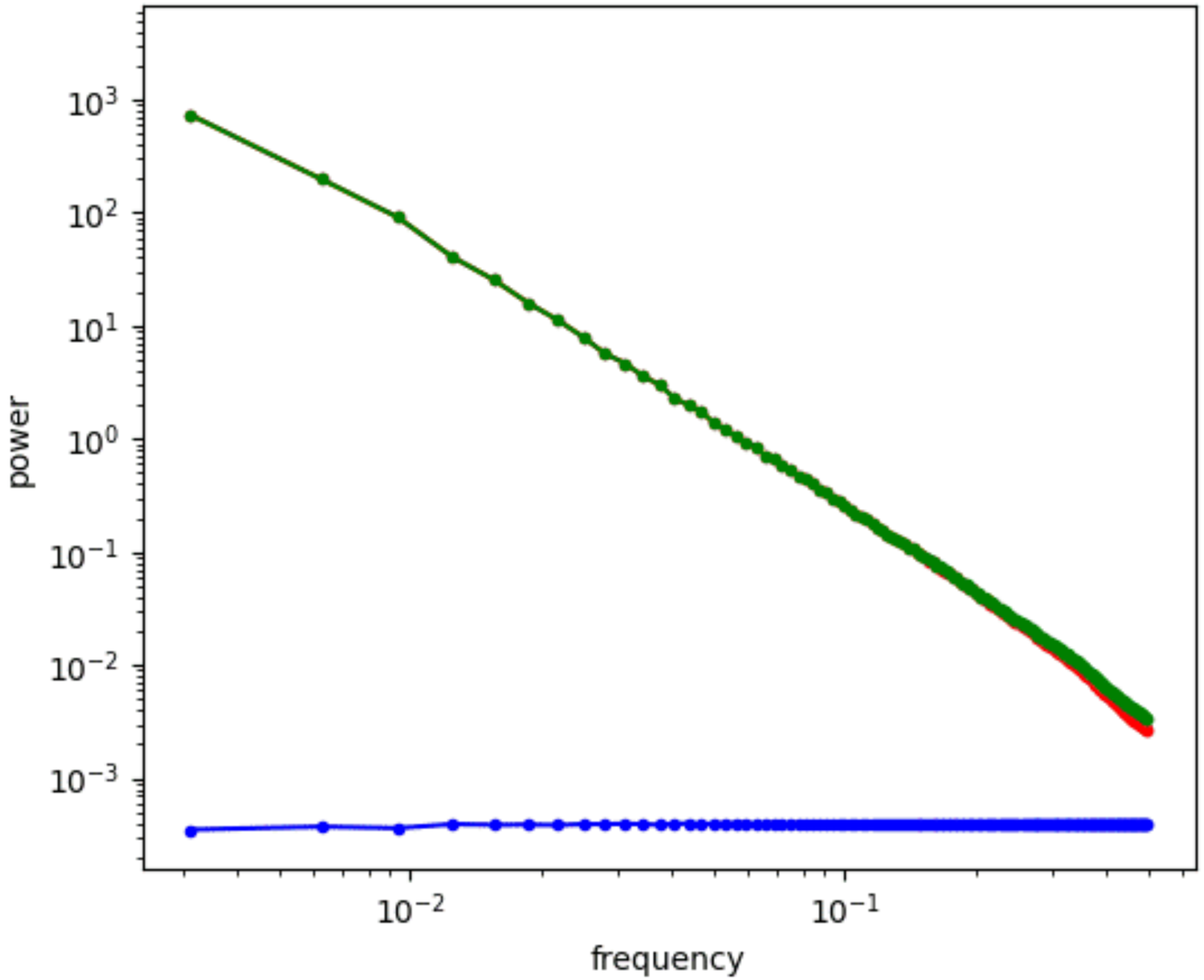
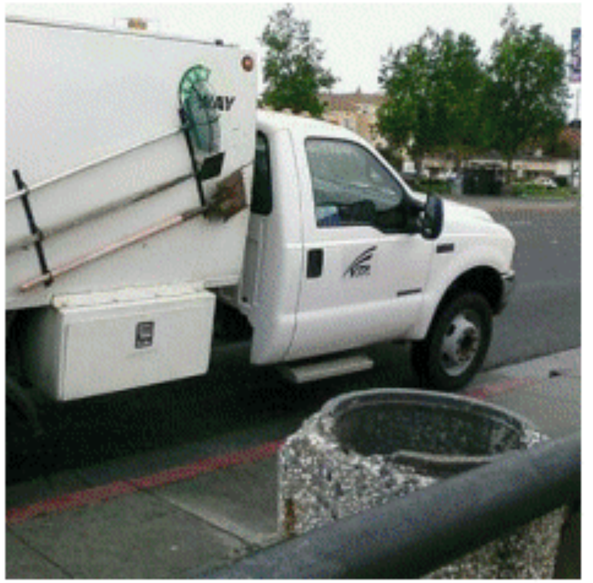


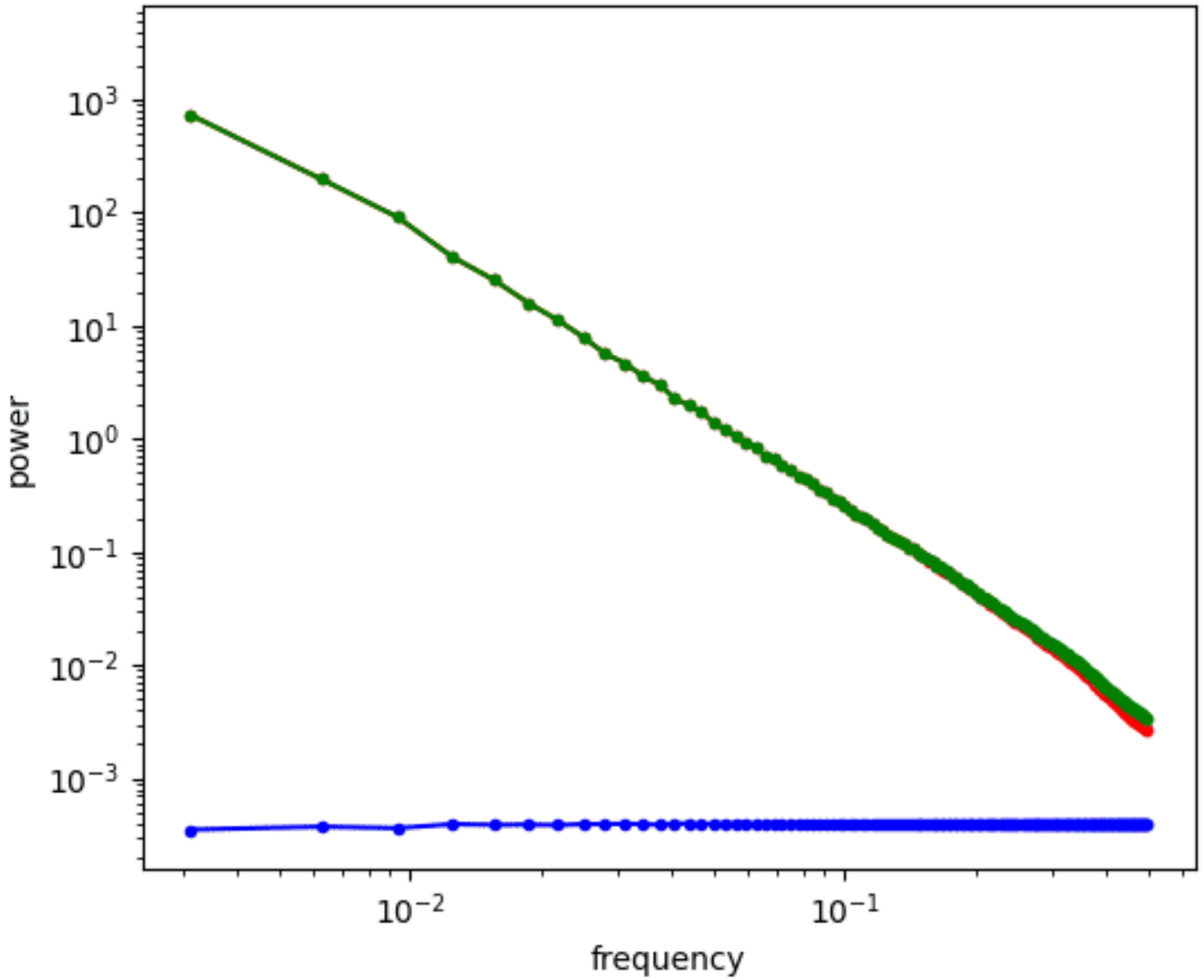
Illustration from Sander Dieleman, “Diffusion is spectral autoregression”

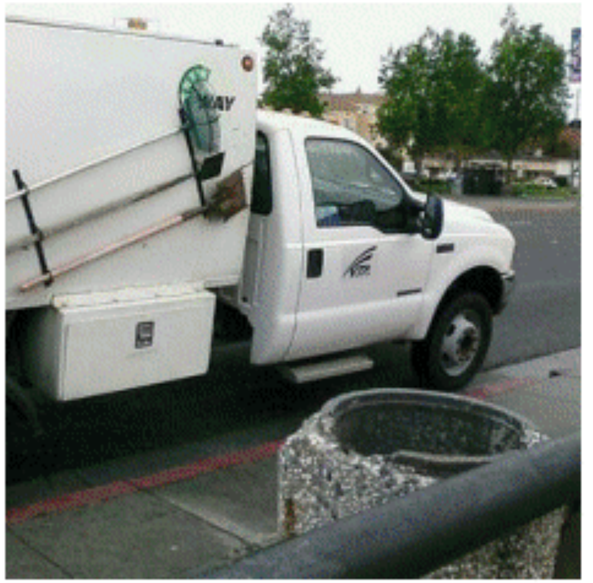
What happens when we add noise?



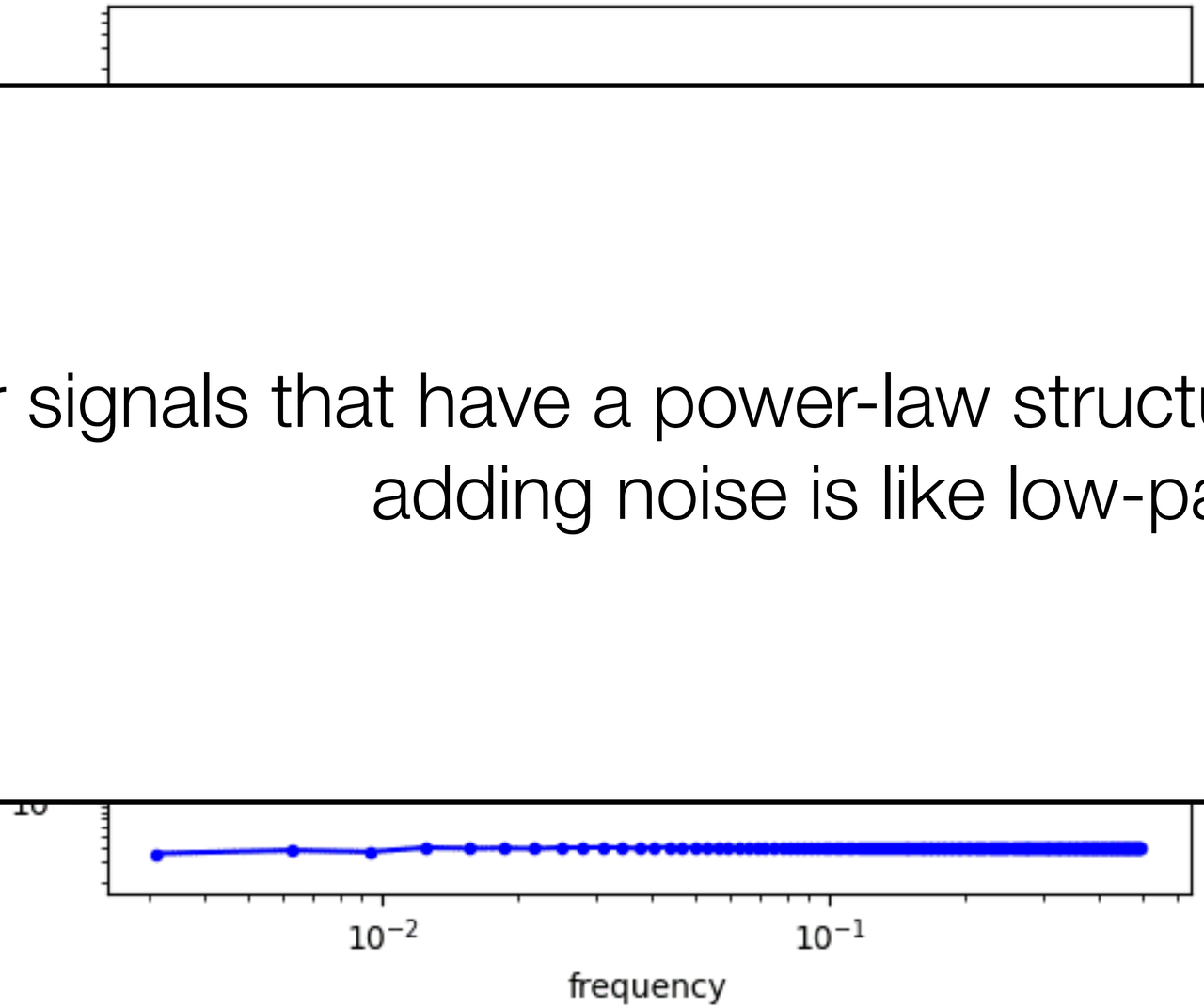


What happens when we add noise?



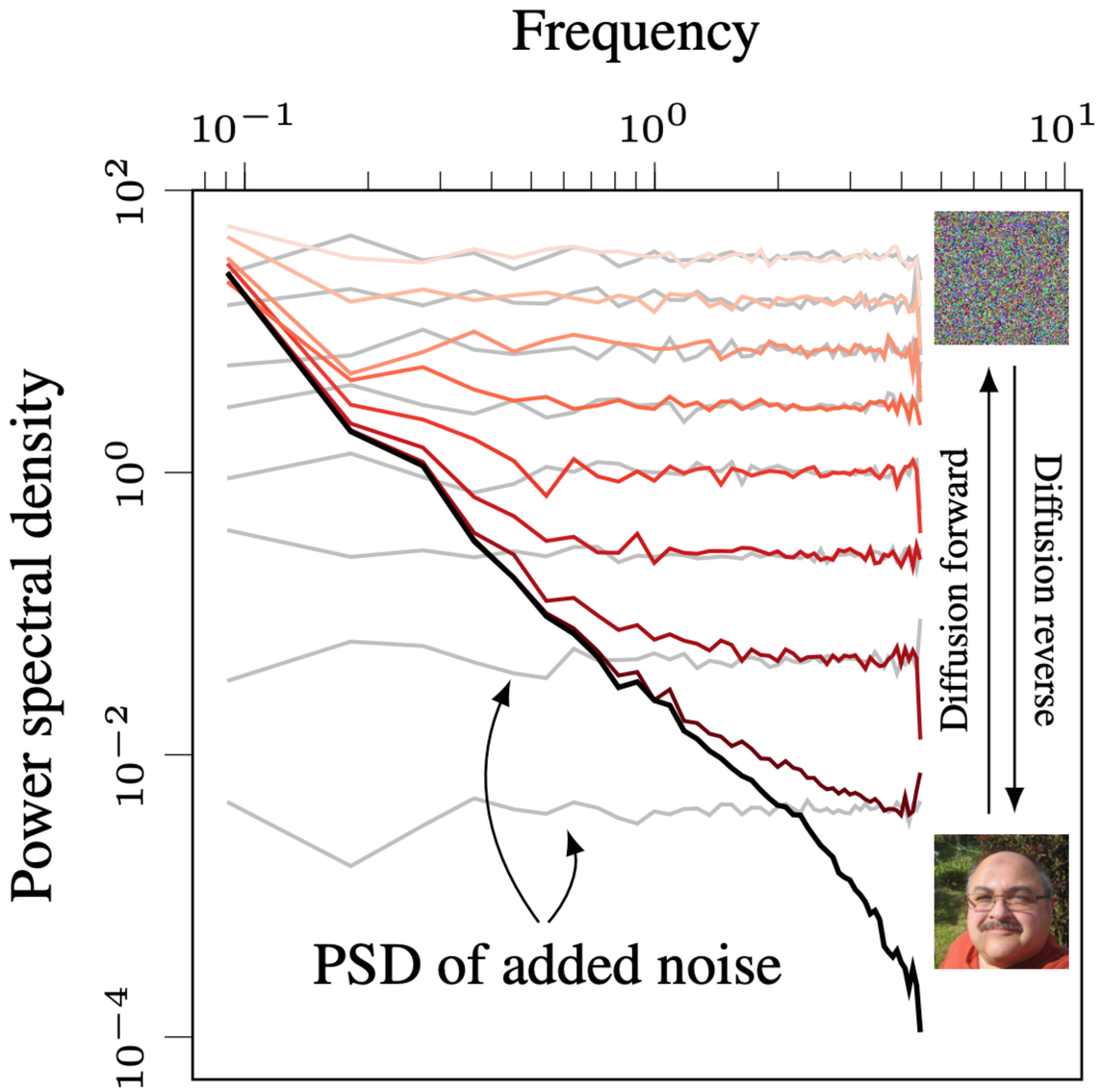


What happens when we add noise?



For signals that have a power-law structure in the spectral density, adding noise is like low-pass filtering!

What is Diffusion Doing?



- Diffusion models can be seen as doing “spectral auto-regression”
- Given lower frequency of image, incrementally predict higher frequencies!

Generative Modeling with Inverse Heat Dissipation, Rissanen et al. 2020

Edify Image: High-Quality Image Generation with Pixel Space Laplacian Diffusion Models

NVIDIA[1]

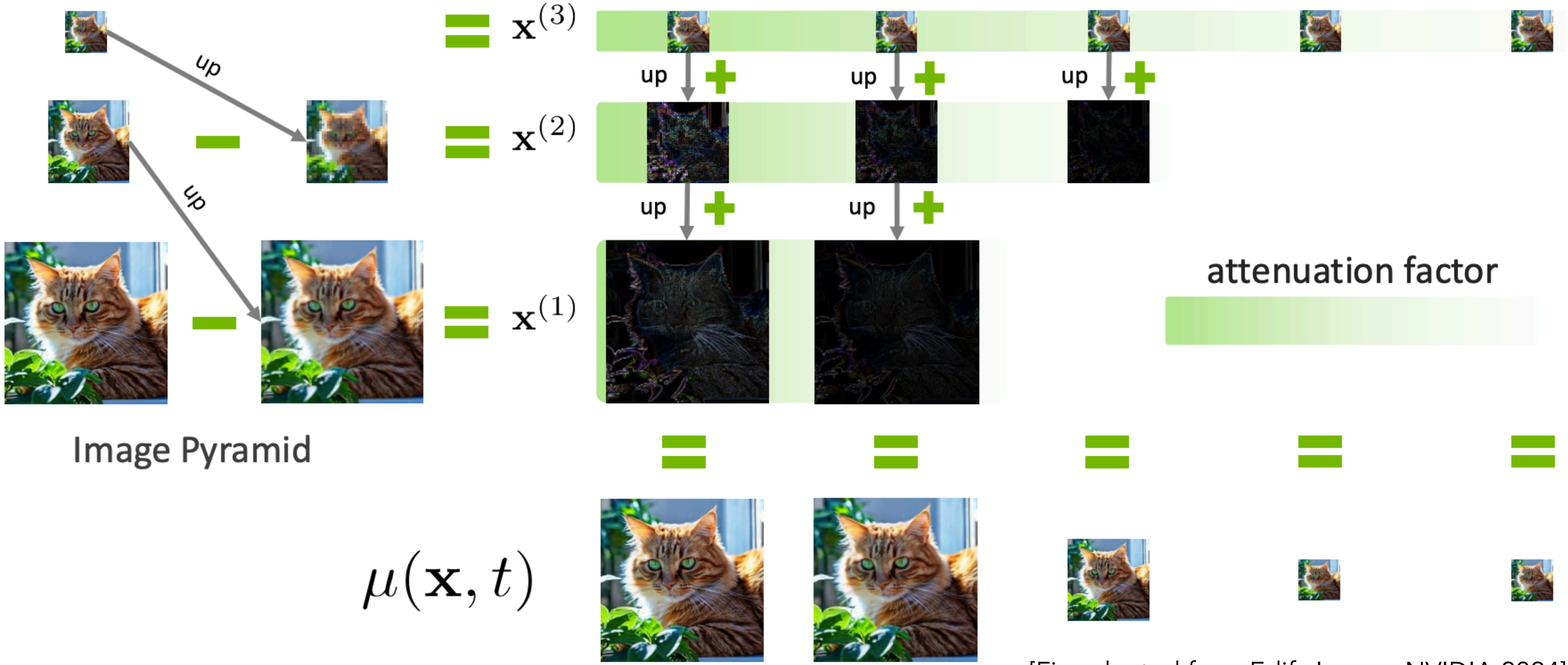


A photo of a couple doing pottery together in a well-lit room



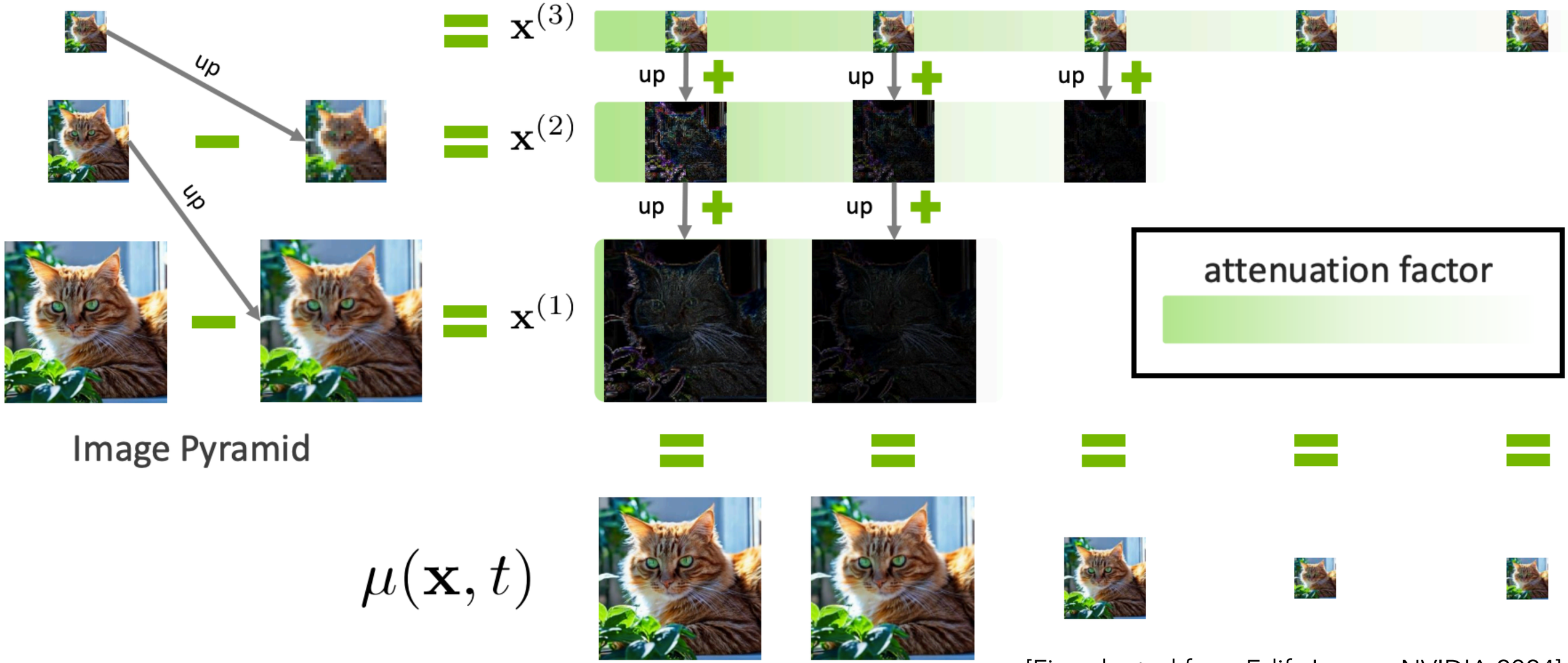
A chameleon showing colorful scales

Foundation: Laplacian Pyramid



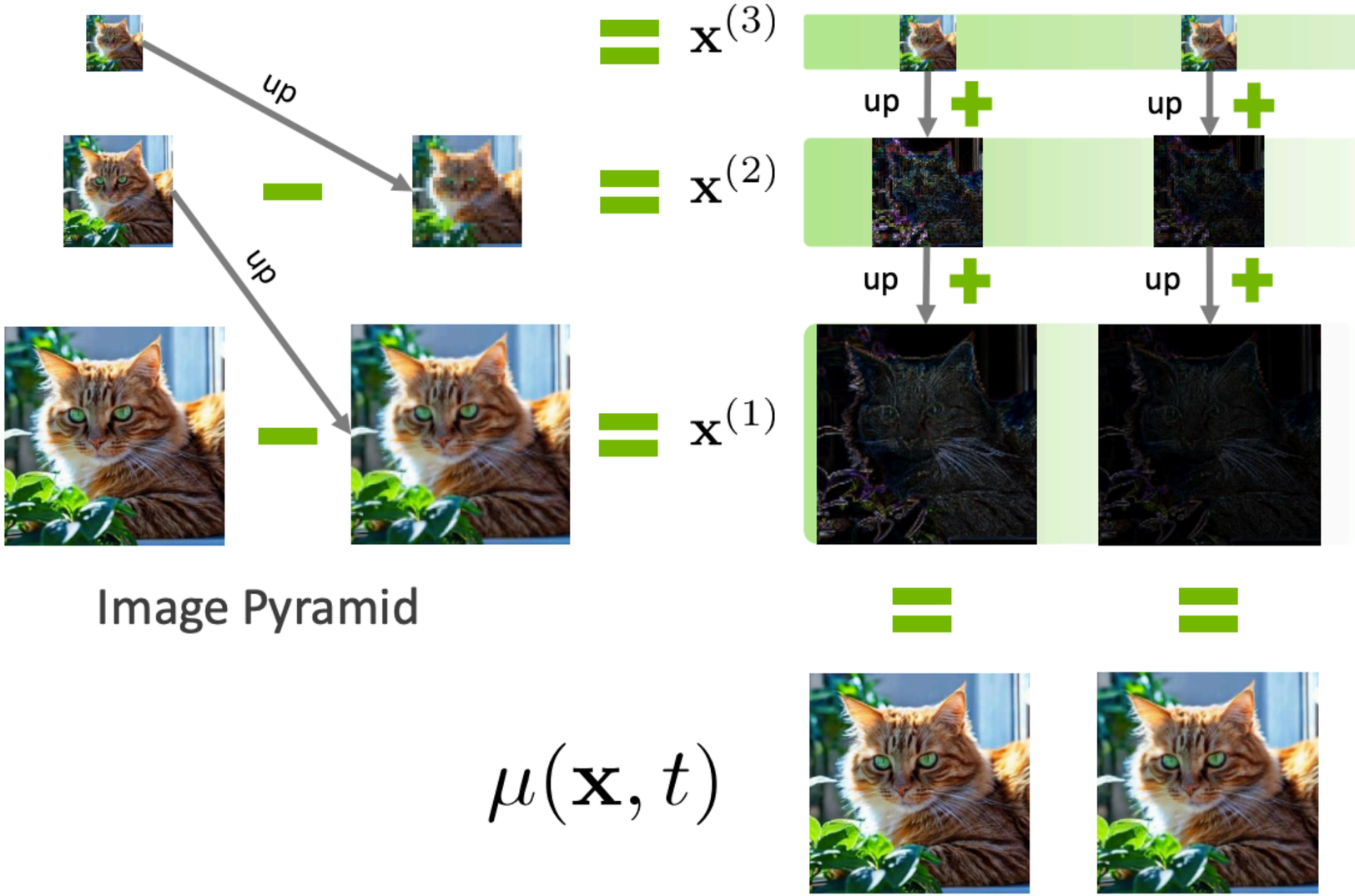
[Fig adapted from Edify Image, NVIDIA 2024]

Foundation: Laplacian Pyramid



[Fig adapted from Edify Image, NVIDIA 2024]

Foundation: Laplacian Pyramid



Core idea: Noise higher frequencies faster

[Fig adapted from Edify Image, NVIDIA 2024]

[Fig adapted from Edify Image, NVIDIA 2024]

time

Forward Noising

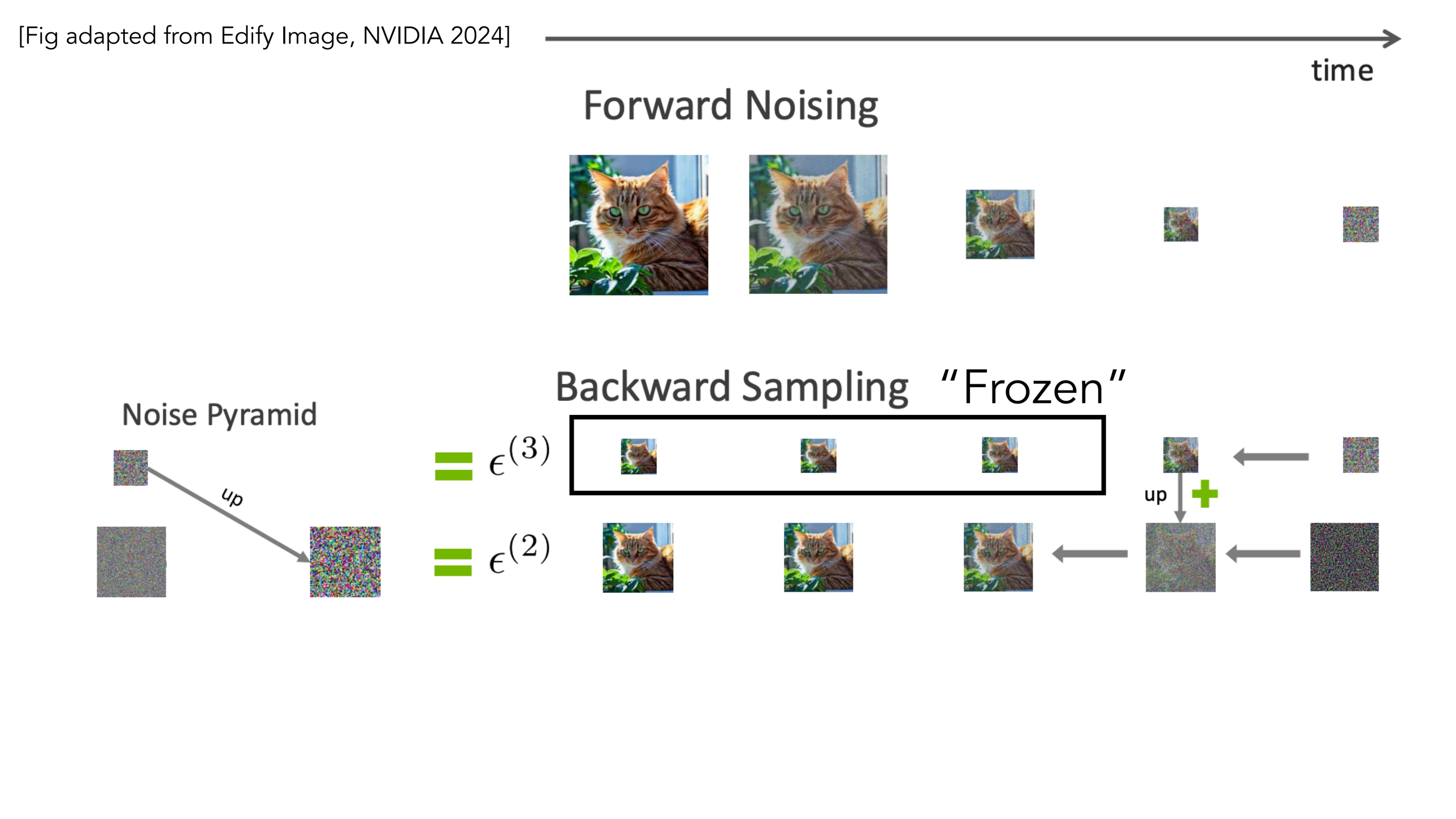


Backward Sampling

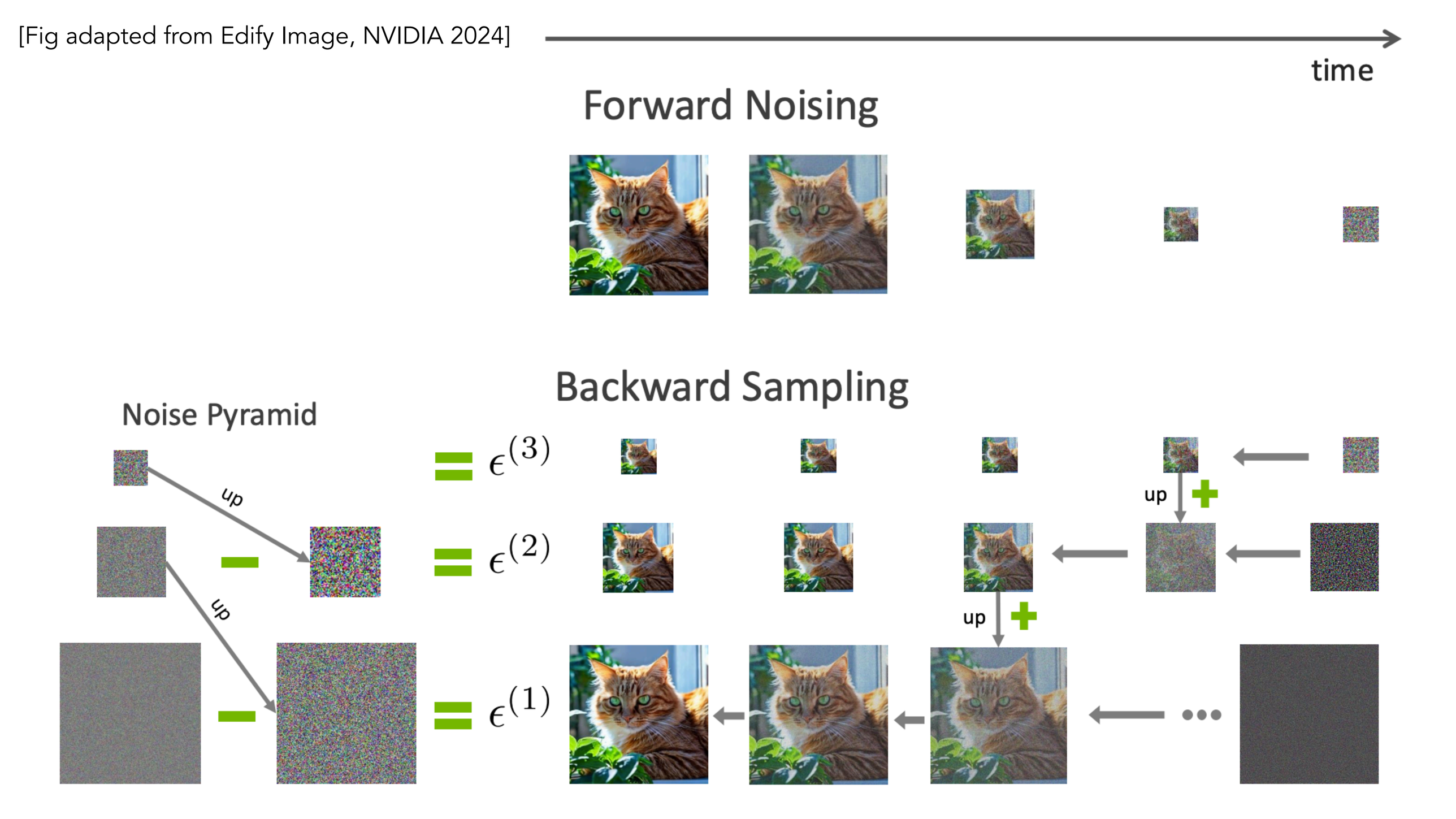
Noise Pyramid



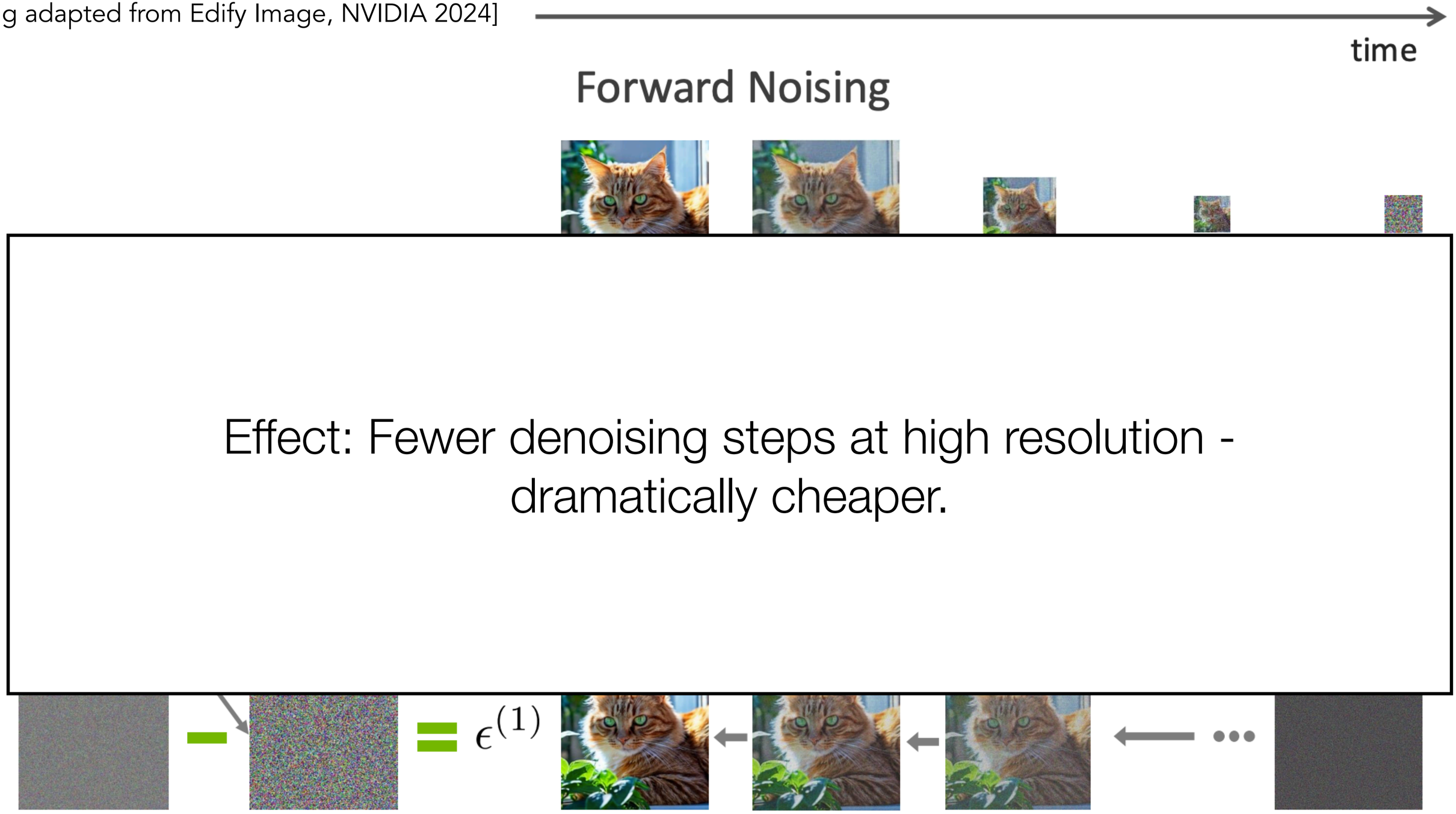
[Fig adapted from Edify Image, NVIDIA 2024]



[Fig adapted from Edify Image, NVIDIA 2024]



[Fig adapted from Edify Image, NVIDIA 2024]

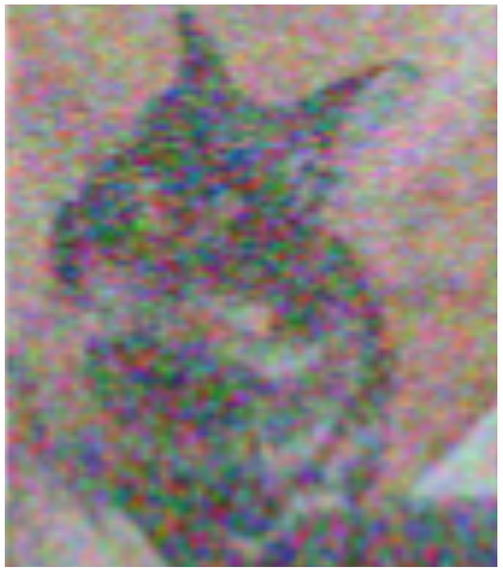


Effect: Fewer denoising steps at high resolution - dramatically cheaper.

Classifier-Free Guidance

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”

Noisy Image x_t



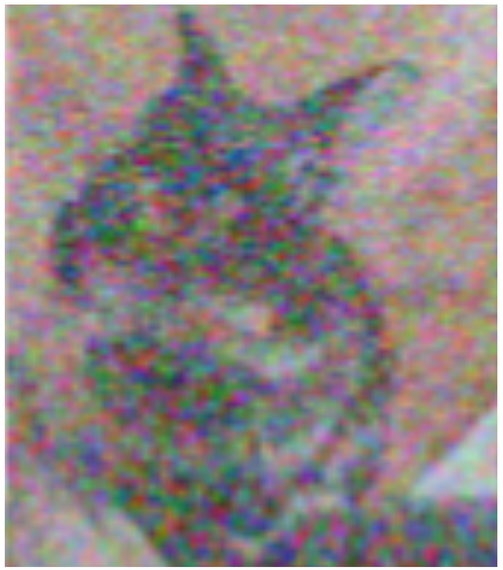
Text Prompt

“A Chinchilla”

Classifier-Free Guidance

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”

Noisy Image x_t

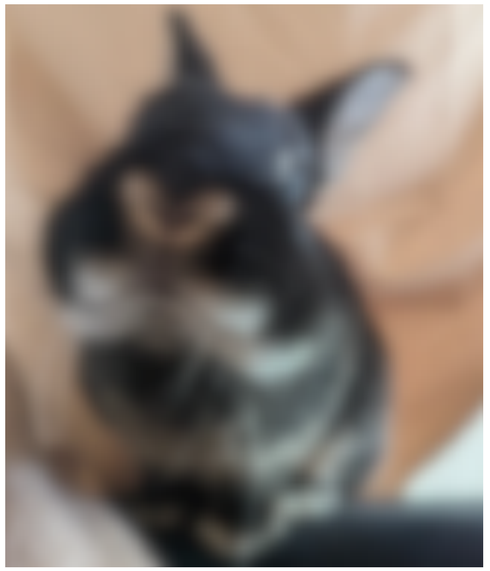


Text Prompt

“A Chinchilla”

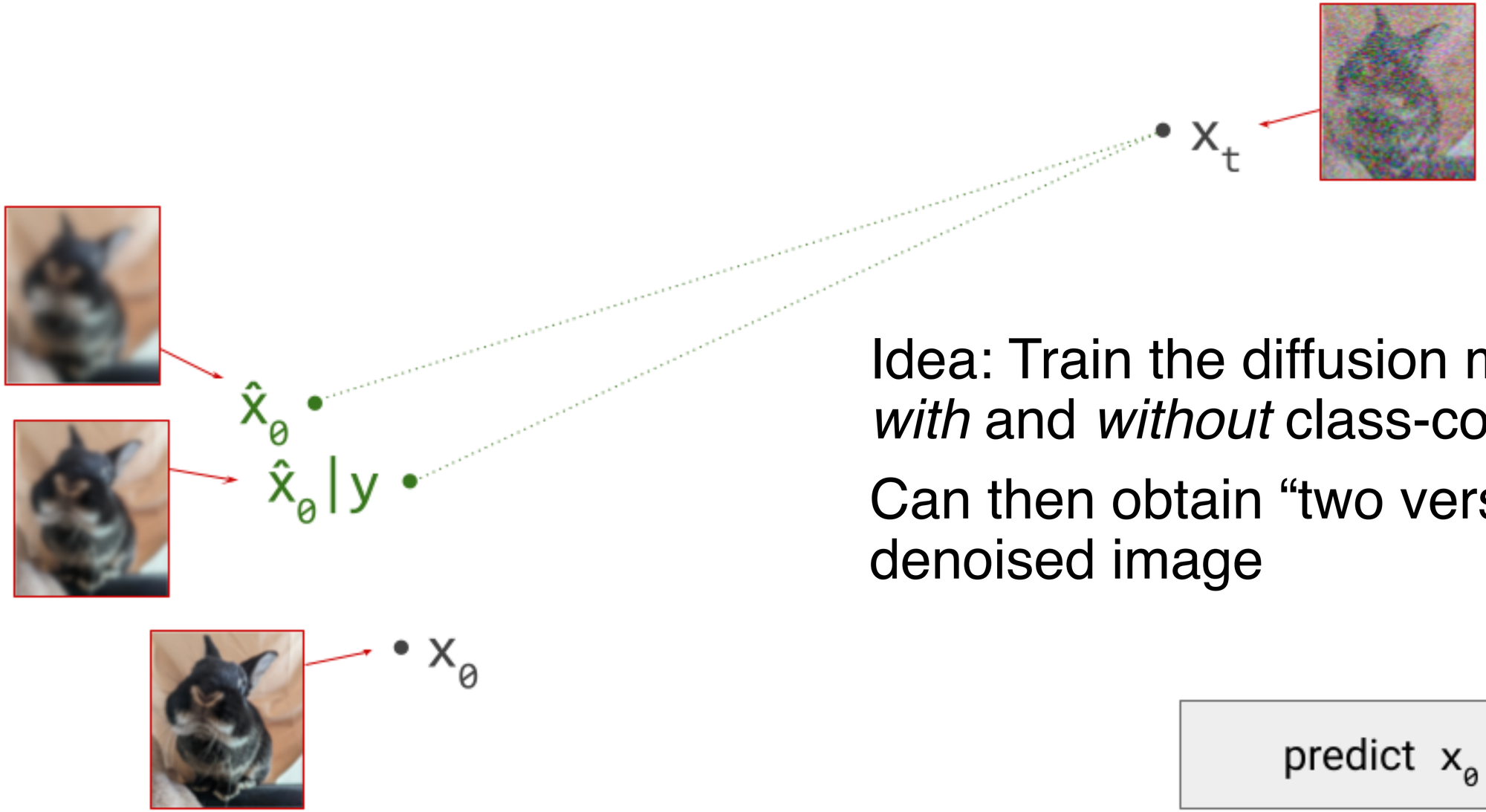
Image Diffusion Model

predict $x_0|y$



$\hat{x}_0|y$

Classifier-Free Guidance



Idea: Train the diffusion model both *with* and *without* class-conditioning

Can then obtain “two versions” of a denoised image

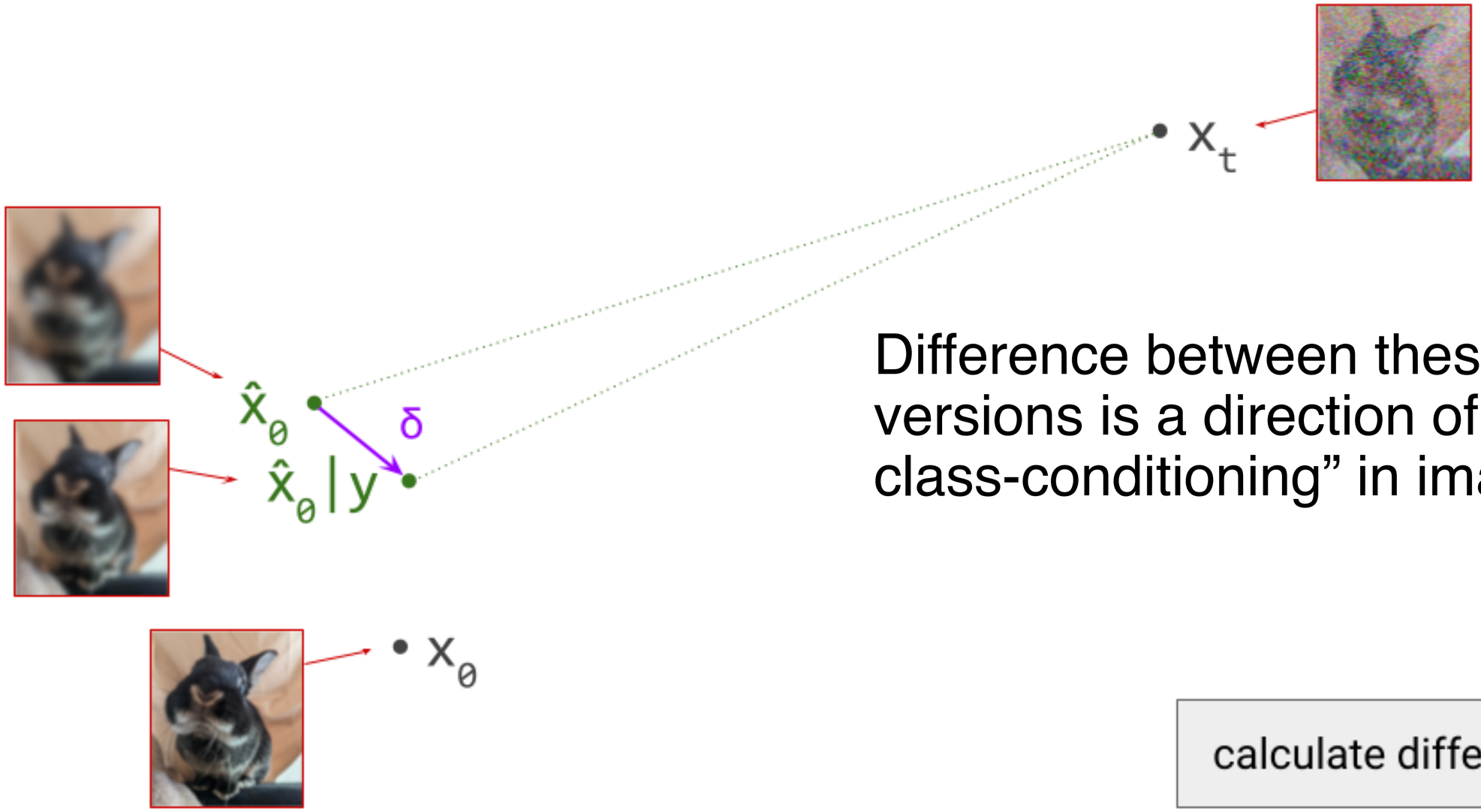
predict $x_0 | y$

Classifier-Free Diffusion Guidance, Ho et al. 2022

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”

Classifier-Free Guidance

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”



Difference between these two versions is a direction of “increased class-conditioning” in image space!

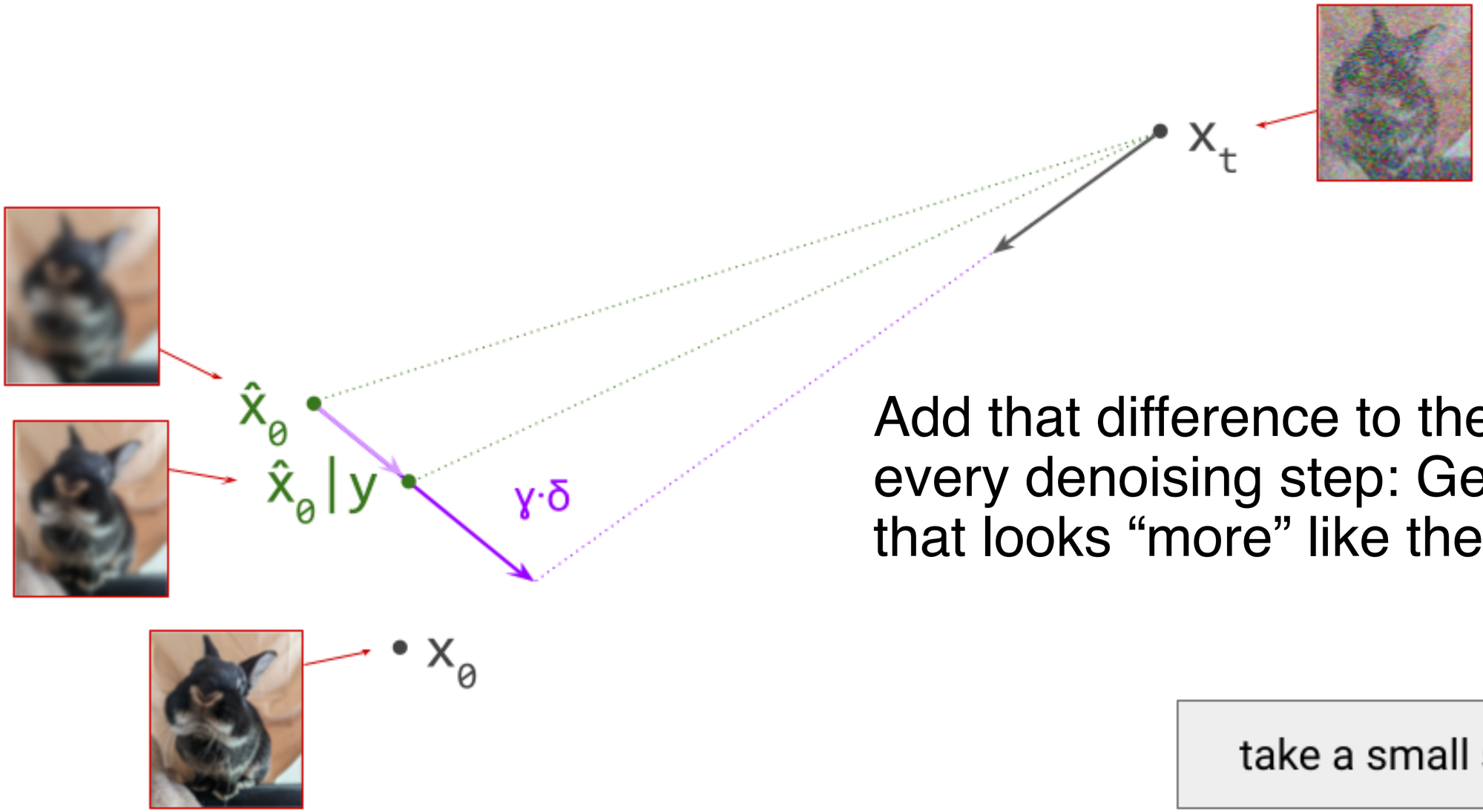
calculate difference

Classifier-Free Diffusion Guidance, Ho et al. 2022

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”

Classifier-Free Guidance

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”



Add that difference to the image at every denoising step: Get a sample that looks “more” like the class!

take a small step

Classifier-Free Diffusion Guidance, Ho et al. 2022

Illustration from Sander Dieleman, “Guidance: A Cheat Code for Diffusion Models”

Classifier-Free Guidance



Figure 2: The effect of guidance on a mixture of three Gaussians, each mixture component representing data conditioned on a class. The leftmost plot is the non-guided marginal density. Left to right are densities of mixtures of normalized guided conditionals with increasing guidance strength.

Classifier-Free Diffusion Guidance, Ho et al. 2022

Classifier-Free Guidance



Figure 1: Classifier-free guidance on the malamute class for a 64x64 ImageNet diffusion model. Left to right: increasing amounts of classifier-free guidance, starting from non-guided samples on the left.

Classifier-Free Diffusion Guidance, Ho et al. 2022

Back to Basics: Let Denoising Generative Models Denoise

Tianhong Li Kaiming He

MIT

Back to Basics: Let Denoising Generative Models Denoise

Tianhong Li Kaiming He

	(a) x-pred $\boldsymbol{x}_\theta := \texttt{net}_\theta(\boldsymbol{z}_t, t)$	(b) ϵ-pred $\boldsymbol{\epsilon}_\theta := \texttt{net}_\theta(\boldsymbol{z}_t, t)$	(c) v-pred $\boldsymbol{v}_\theta := \texttt{net}_\theta(\boldsymbol{z}_t, t)$
(1) x-loss: $\mathbb{E}\|\boldsymbol{x}_\theta - \boldsymbol{x}\|^2$	$\boldsymbol{x}_\theta$	$\boldsymbol{x}_\theta = (\boldsymbol{z}_t - (1-t)\boldsymbol{\epsilon}_\theta)/t$	$\boldsymbol{x}_\theta = (1-t)\boldsymbol{v}_\theta + \boldsymbol{z}_t$
(2) ϵ-loss: $\mathbb{E}\|\boldsymbol{\epsilon}_\theta - \boldsymbol{\epsilon}\|^2$	$\boldsymbol{\epsilon}_\theta = (\boldsymbol{z}_t - t\boldsymbol{x}_\theta)/(1-t)$	$\boldsymbol{\epsilon}_\theta$	$\boldsymbol{\epsilon}_\theta = \boldsymbol{z}_t - t\boldsymbol{v}_\theta$
(3) v-loss: $\mathbb{E}\|\boldsymbol{v}_\theta - \boldsymbol{v}\|^2$	$\boldsymbol{v}_\theta = (\boldsymbol{x}_\theta - \boldsymbol{z}_t)/(1-t)$	$\boldsymbol{v}_\theta = (\boldsymbol{z}_t - \boldsymbol{\epsilon}_\theta)/t$	$\boldsymbol{v}_\theta$

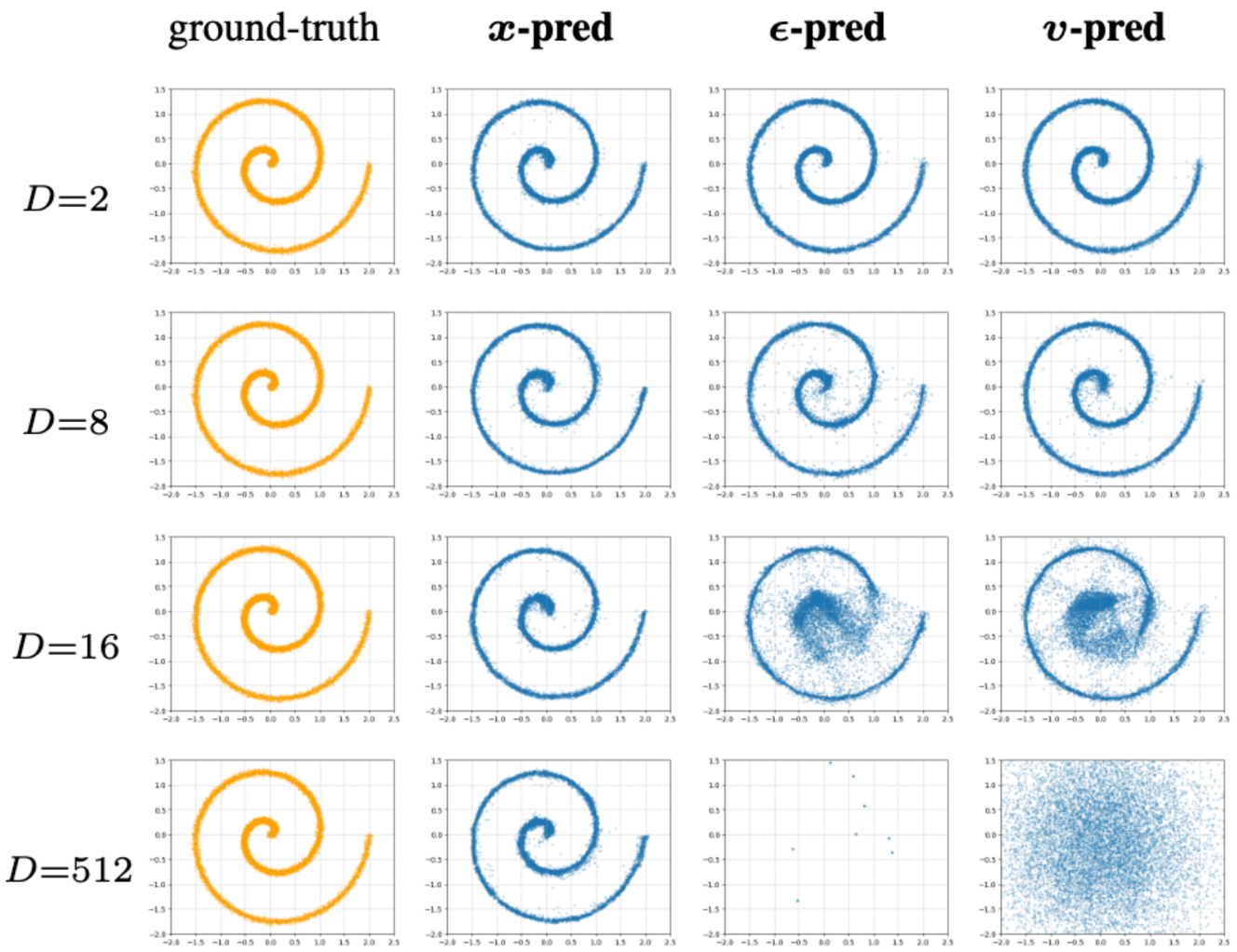
Back to Basics: Let Denoising Generative Models Denoise

Tianhong Li Kaiming He

	(a) x-pred $\boldsymbol{x}_\theta := \texttt{net}_\theta(\boldsymbol{z}_t, t)$	(b) ϵ-pred $\boldsymbol{\epsilon}_\theta := \texttt{net}_\theta(\boldsymbol{z}_t, t)$	(c) v-pred $\boldsymbol{v}_\theta := \texttt{net}_\theta(\boldsymbol{z}_t, t)$
(1) x-loss: $\mathbb{E}\|\boldsymbol{x}_\theta - \boldsymbol{x}\|^2$	$\boldsymbol{x}_\theta$	$\boldsymbol{x}_\theta = (\boldsymbol{z}_t - (1-t)\boldsymbol{\epsilon}_\theta)/t$	$\boldsymbol{x}_\theta = (1-t)\boldsymbol{v}_\theta + \boldsymbol{z}_t$
(2) ϵ-loss: $\mathbb{E}\|\boldsymbol{\epsilon}_\theta - \boldsymbol{\epsilon}\|^2$	$\boldsymbol{\epsilon}_\theta = (\boldsymbol{z}_t - t\boldsymbol{x}_\theta)/(1-t)$	$\boldsymbol{\epsilon}_\theta$	$\boldsymbol{\epsilon}_\theta = \boldsymbol{z}_t - t\boldsymbol{v}_\theta$
(3) v-loss: $\mathbb{E}\|\boldsymbol{v}_\theta - \boldsymbol{v}\|^2$	$\boldsymbol{v}_\theta = (\boldsymbol{x}_\theta - \boldsymbol{z}_t)/(1-t)$	$\boldsymbol{v}_\theta = (\boldsymbol{z}_t - \boldsymbol{\epsilon}_\theta)/t$	$\boldsymbol{v}_\theta$

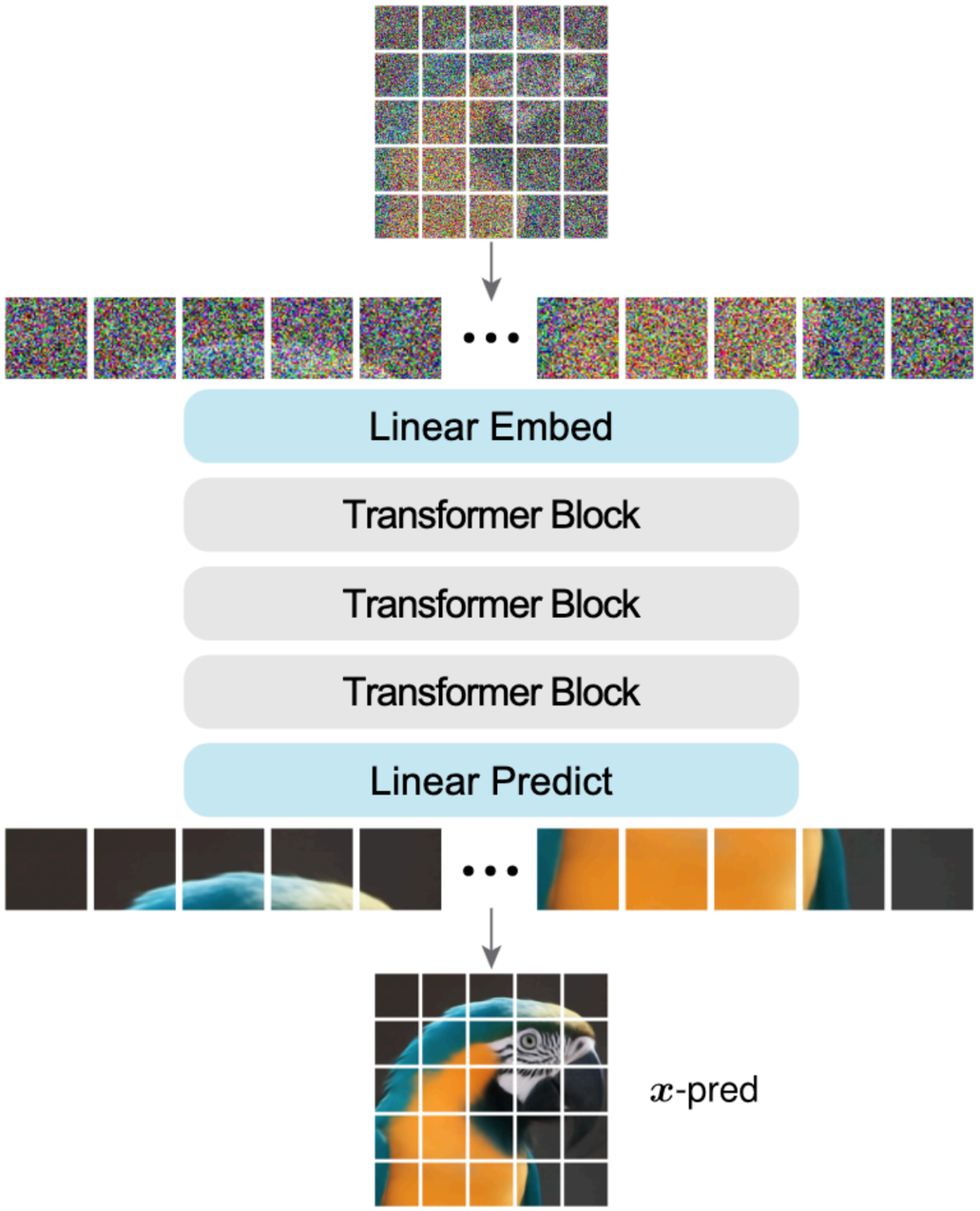
Mathematically, all of these ways of training a diffusion model are equivalent: you can write one of them as the other.

Practically, your neural network has to infer a different quantity!



d-dimensional (d = 2) underlying data is “buried” in a D-dimensional space, by a fixed, random, column-orthogonal projection matrix. In the D-dim space, we train a simple generative model (5-layer ReLU MLP with 256-dim hidden units). The projection matrix is unknown to the model, and we only use it for visualizing the output.

In this toy experiment, with the observed dimension D increasing, only x-prediction can produce reasonable results.



The “JiT” Just Image Transformer

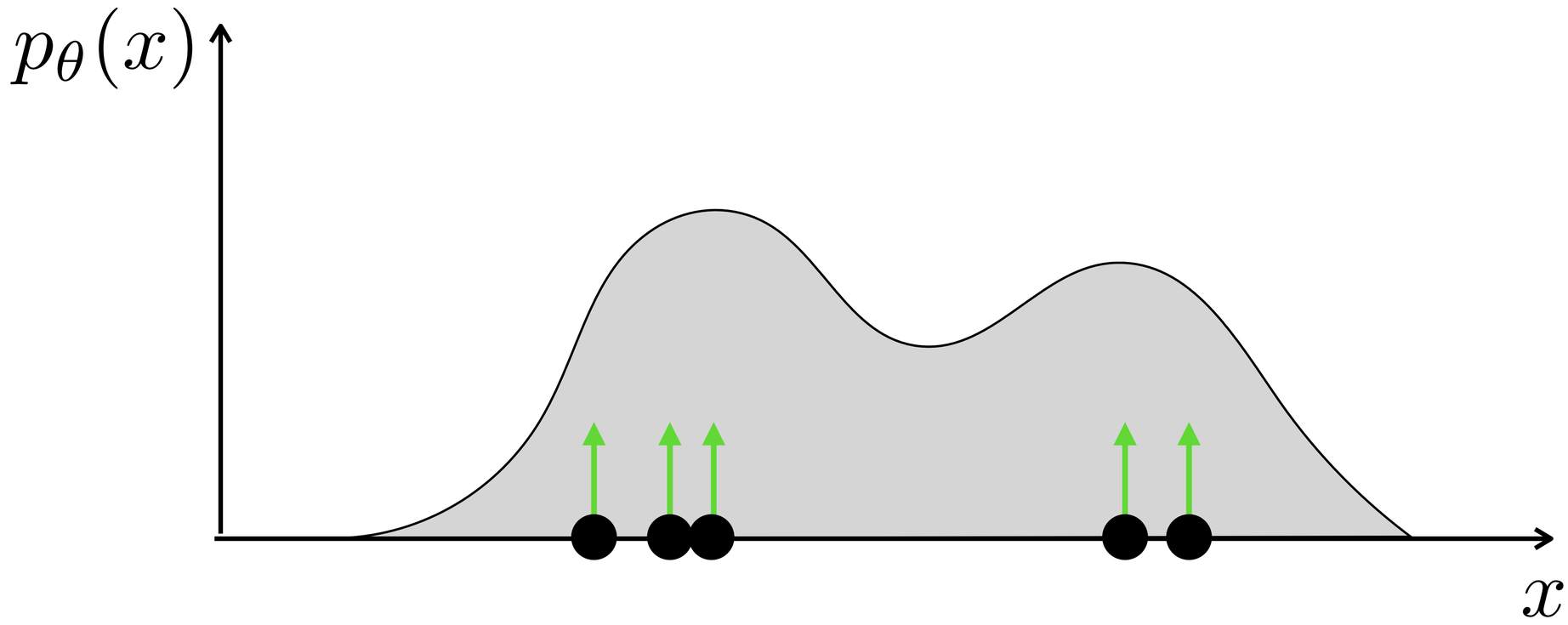
Keep this in mind: The neural net is the “star” of the show.

The neural net input and output are the most important thing :)

Can we *understand* what the diffusion model is doing *mechanistically?*

Diffusion's Analytical Probability Density

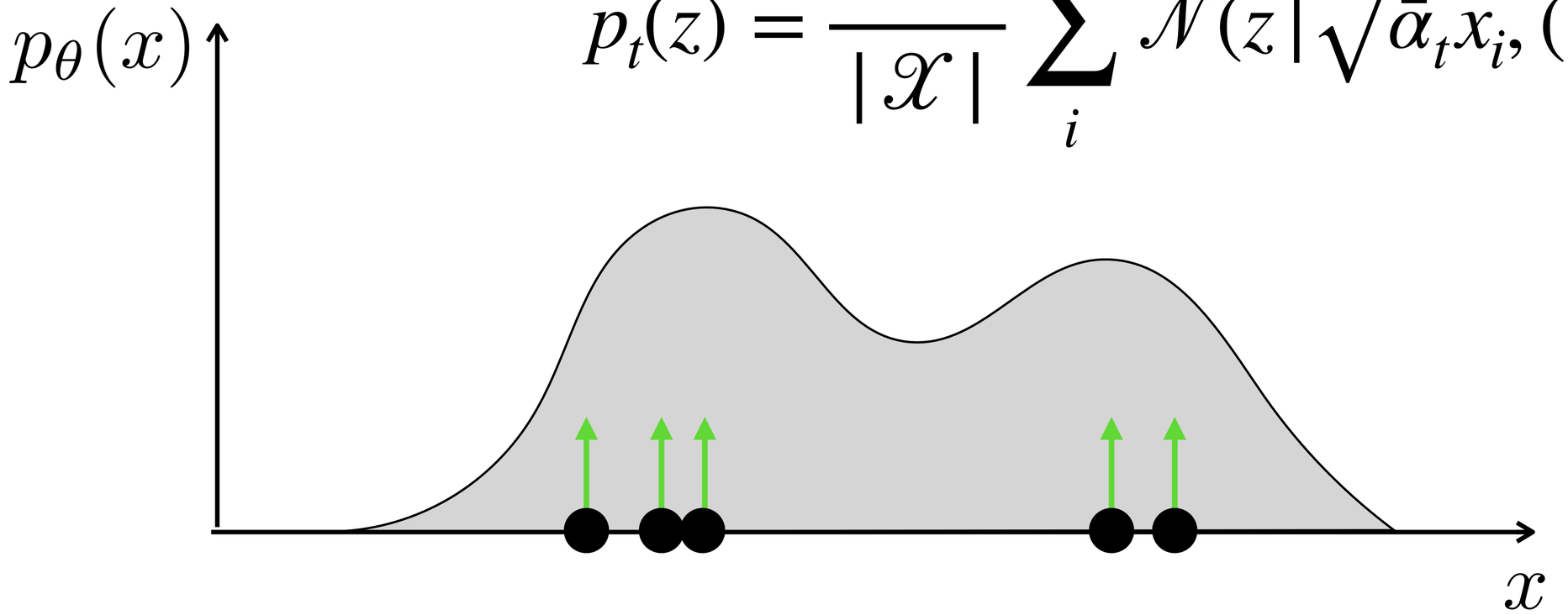
For diffusion models, $p_t(z)$ is a mixture of Gaussians centered at each data point $x_i \in \mathcal{X}$, where the variance depends on the timestep!



Diffusion's Analytical Probability Density

For diffusion models, $p_t(z)$ is a mixture of Gaussians centered at each data point $x_i \in \mathcal{X}$, where the variance depends on the timestep!

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z \,|\, \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

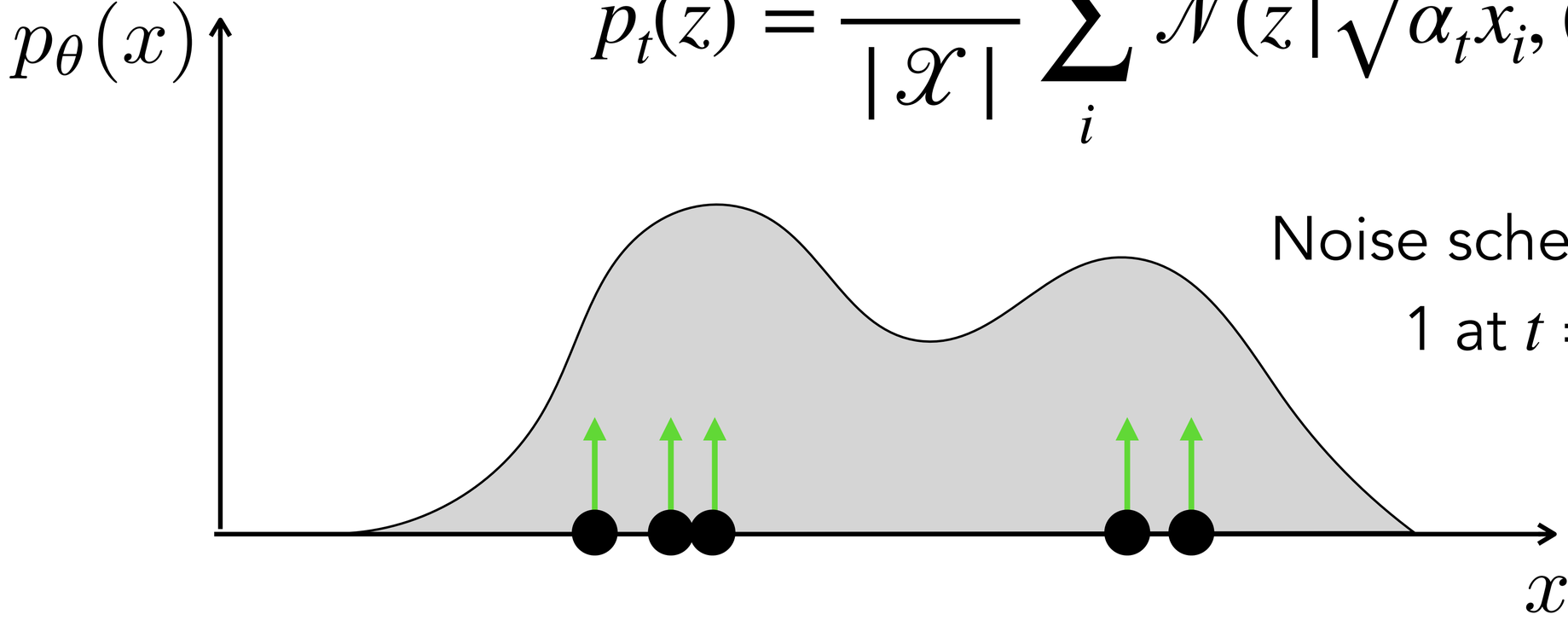


Diffusion's Analytical Probability Density

For diffusion models, $p_t(z)$ is a mixture of Gaussians centered at each data point $x_i \in \mathcal{X}$, where the variance depends on the timestep!

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z \,|\, \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

Noise schedule, decreases from 1 at $t = 0$ to 0 at $t = T$



Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2} \left(k_t(z) - z \right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax \left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

Diffusion's Analytical Probability **Score**

$$p(z) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{N}(z \mid \sqrt{\bar{\alpha}}\, x_i, (1-\bar{\alpha})I)$$

What happens if we generate samples using this score?

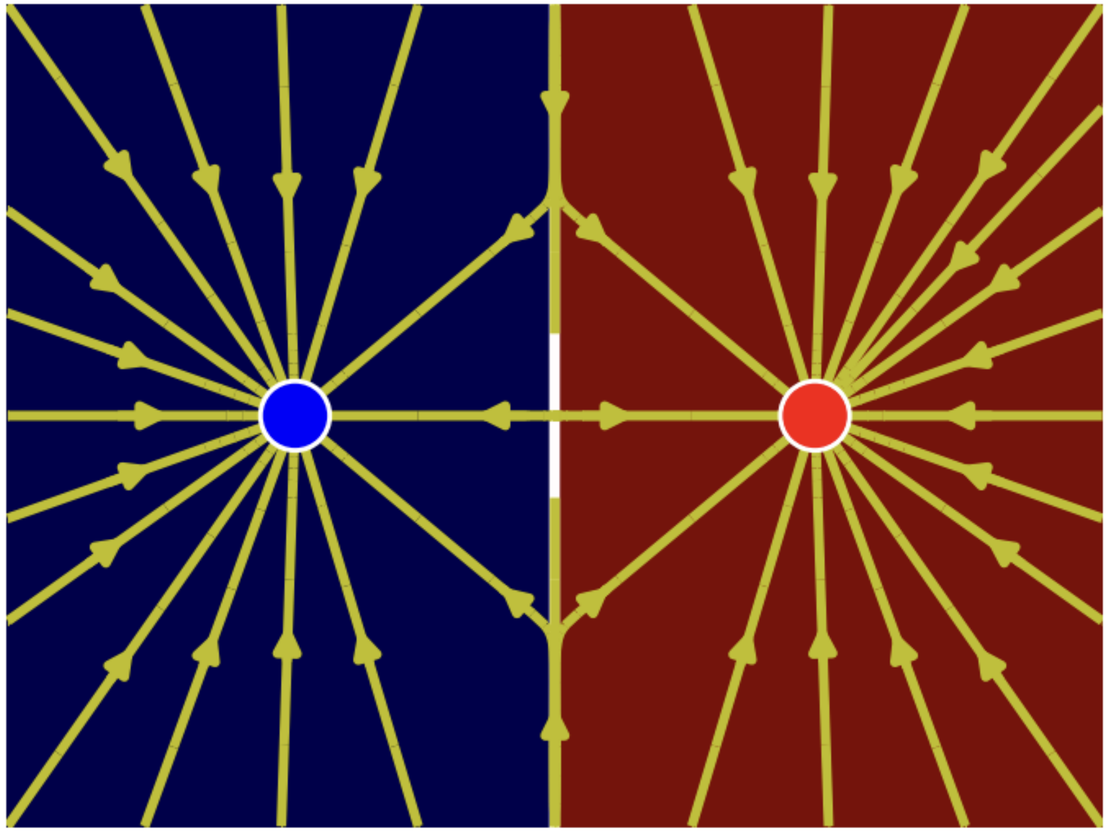
$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

“Optimal Denoisers” or closed-from diffusion

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$

Consider two “training images”
(red and blue)



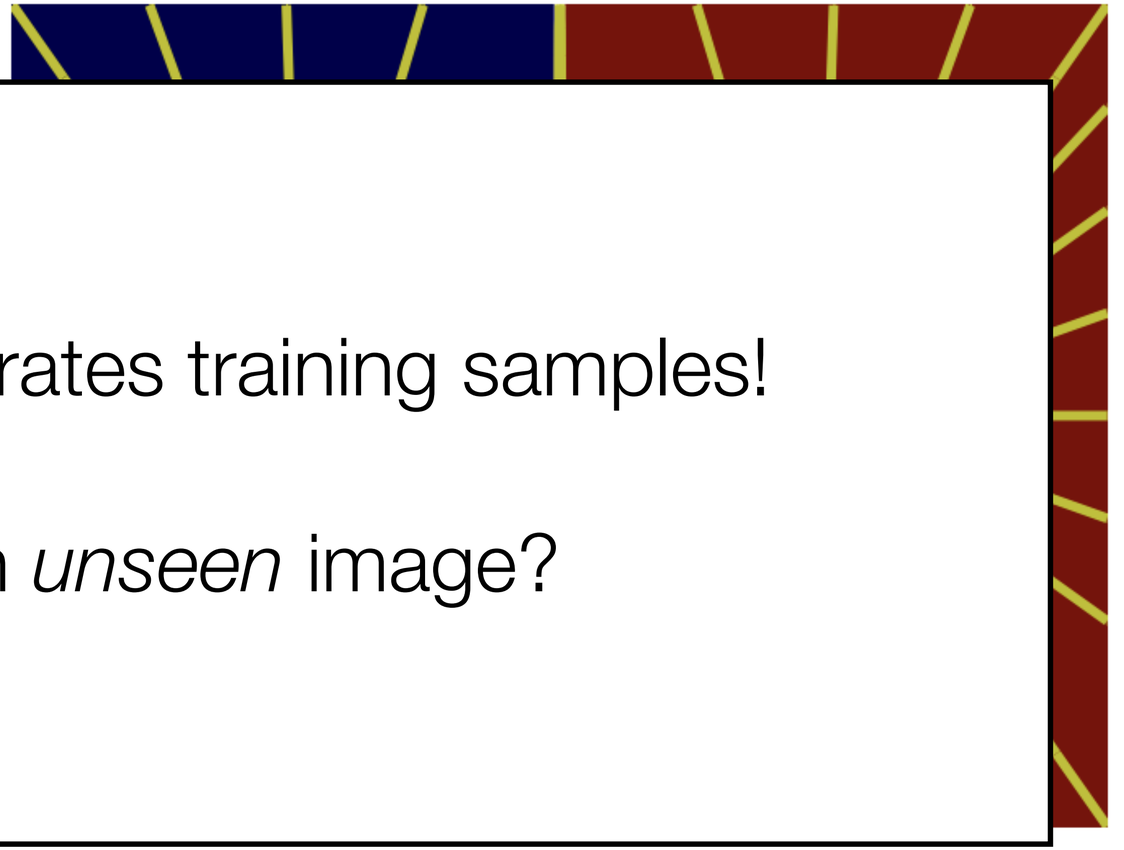
(a) Closed-form score

Diffusion will just retrieve one of them…

[Closed-form Diffusion Models, Scarvelis et al 2023]

“Optimal Denoisers” or closed-from diffusion

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$



The *optimal* denoiser just generates training samples!

How can we generate an *unseen* image?

Diffusion will just retrieve one of them…

(a) Closed-form score

[Closed-form Diffusion Models, Scarvelis et al 2023]

Where does generalization come from…?

Closed-Form Diffusion Models

Christopher Scarvelis *scarv@mit.edu*
MIT CSAIL

Haitz Sáez de Ocáriz Borde *chri6704@ox.ac.uk*
University of Oxford

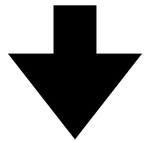
Justin Solomon *jsolomon@mit.edu*
MIT CSAIL

Abstract

Score-based generative models (SGMs) sample from a target distribution by iteratively transforming noise using the score function of the perturbed target. For any finite training set, this score function can be evaluated in closed form, but the resulting SGM memorizes its training data and does not generate novel samples. In practice, one approximates the score by training a neural network via score-matching. The error in this approximation

Idea: Introduce Error Into Score via Smoothing

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$



$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z + \sigma\epsilon_m) - z\right)$$

Idea: Introduce Error Into Score via Smoothing

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$



$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z + \sigma\epsilon_m) - z\right)$$

We are *averaging* the weights k_t over a few different noise perturbations of the current noisy sample z.

Idea: Introduce Error Into Score via Smoothing

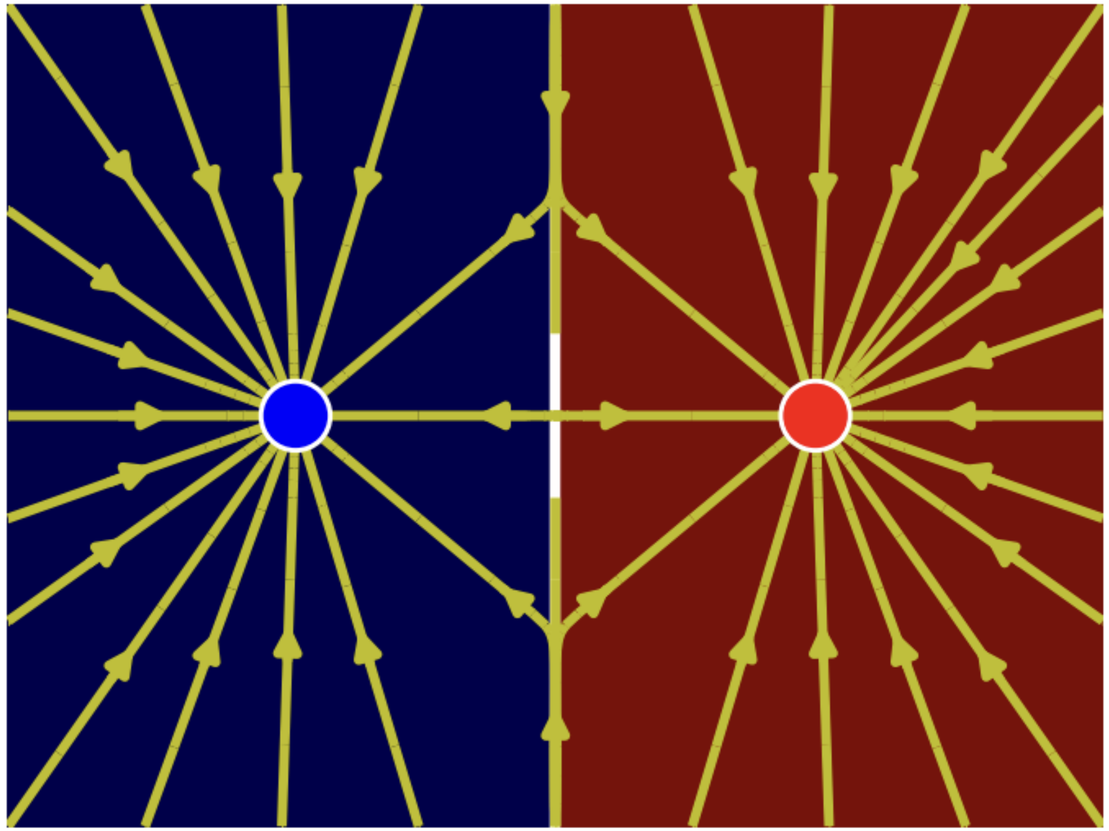
$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z+\sigma\epsilon_m) - z\right)$$

Idea: Introduce Error Into Score via Smoothing

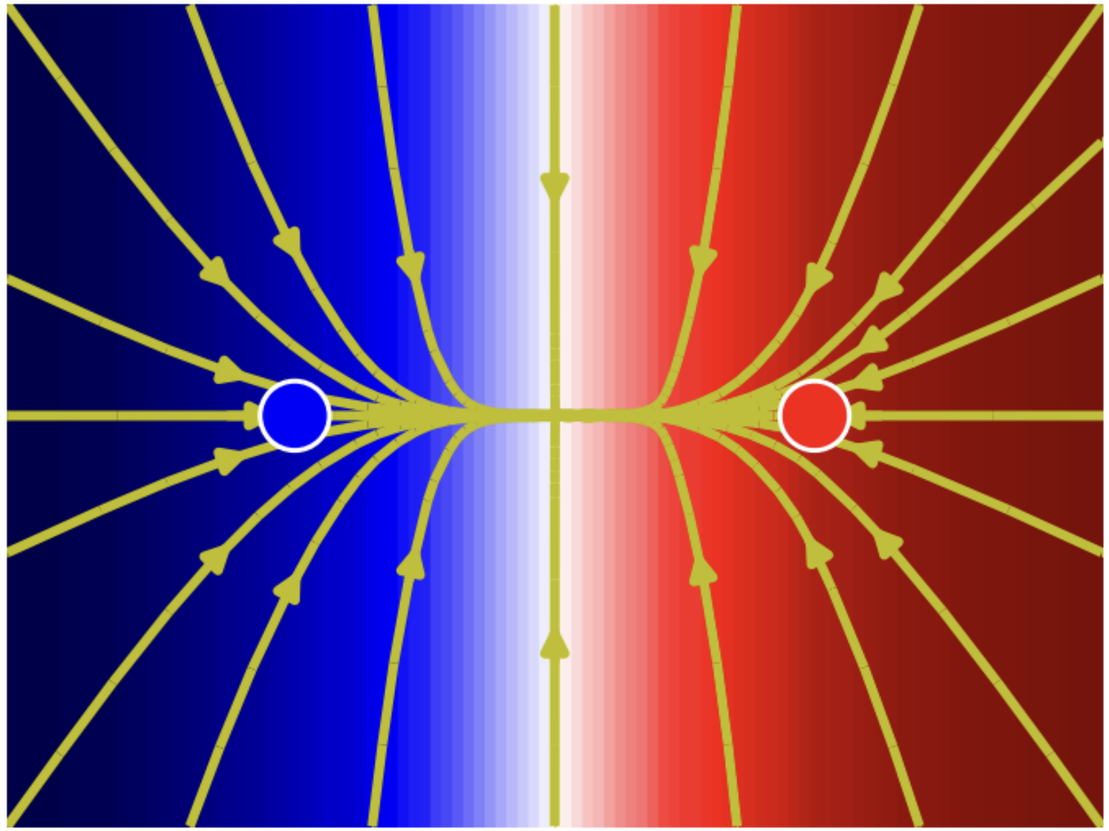
$$s_{\sigma,t}(z) = \frac{1}{(1-t)^2}\left(\frac{1}{M}\sum_{m=1}^{M} k_t(z+\sigma\epsilon_m) - z\right)$$

In [Closed-form Diffusion Models, Scarvelis et al 2023], the authors show that this score points towards *barycenters* of images in the training set.

Idea: Introduce Error Into Score via Smoothing



(a) Closed-form score



(b) Smoothed score

Samples from CelebA with Smoothed Diffusion



(a) Ground truth samples



(b) VAE samples

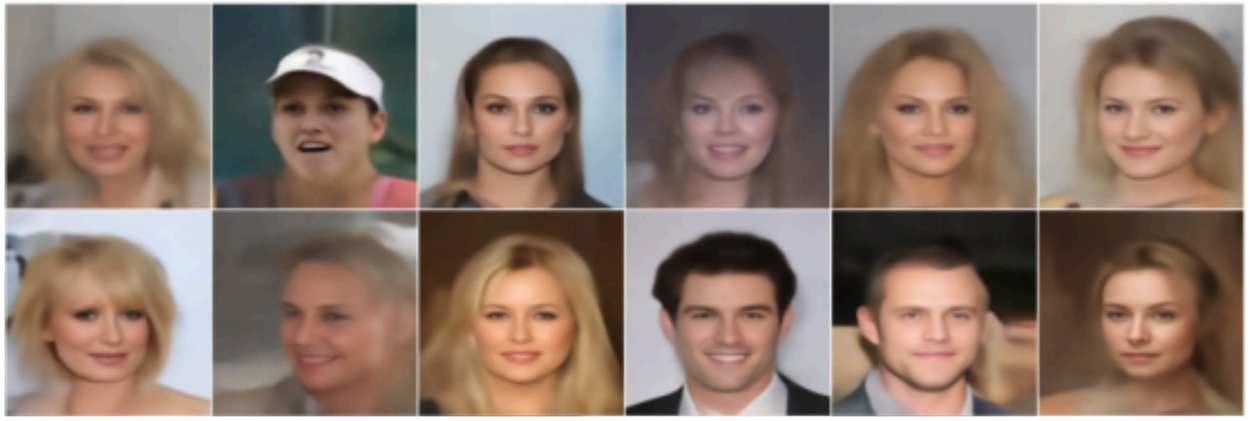


(c) σ-CFDM samples

Samples from CelebA with Smoothed Diffusion

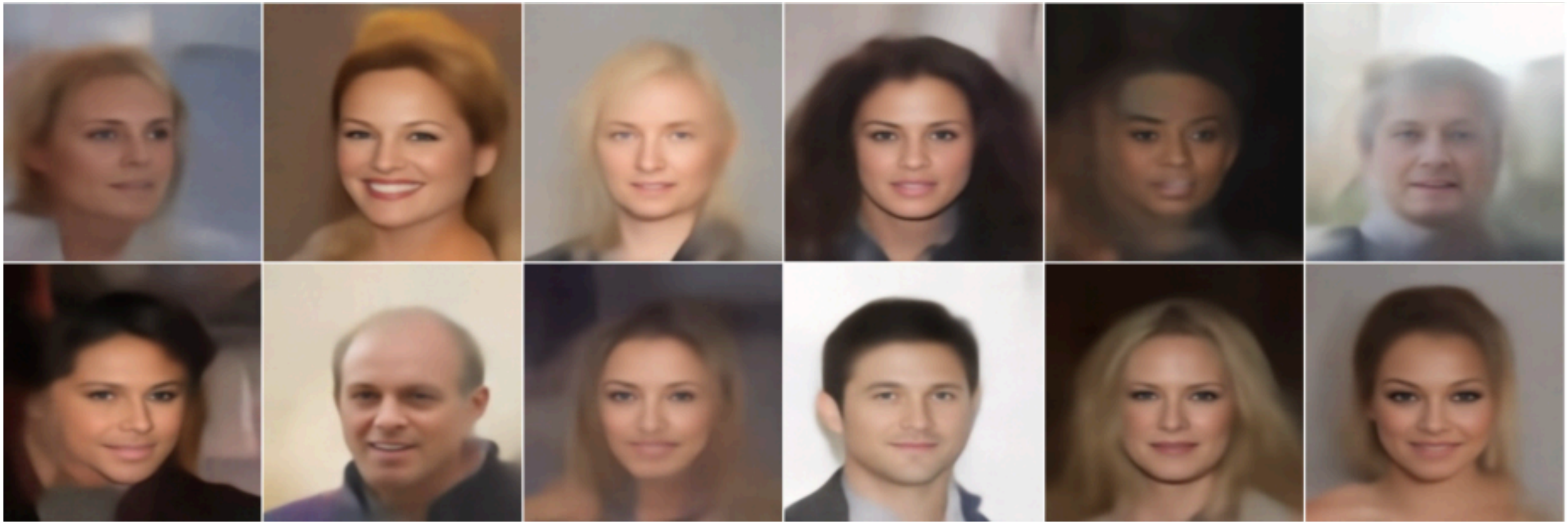


(b) VAE samples



(c) σ-CFDM samples

Samples from CelebA with Smoothed Diffusion



Not amazing quality, but *better* than a VAE, new faces, and *without* a NN altogether!

Flux.1-dev 12B

“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B



“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B



“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B



“a beaver eating a hamburger on a skateboard”



“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B



“a beaver eating a hamburger on a skateboard”



“**a photo of a** beaver eating a hamburger on a skateboard”

“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B



“a beaver eating a hamburger on a skateboard”



“**a photo of a** beaver eating a hamburger on a skateboard”



“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…

“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…



“a beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…



“a beaver eating a hamburger on a skateboard”

“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…



“a beaver eating a hamburger on a skateboard”



“**a photo of a** beaver eating a hamburger on a skateboard”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…



“a beaver eating a hamburger on a skateboard”



“**a photo of a** beaver eating a hamburger on a skateboard”

“a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**”

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…



"a beaver eating a hamburger on a skateboard"



"**a photo of a** beaver eating a hamburger on a skateboard"



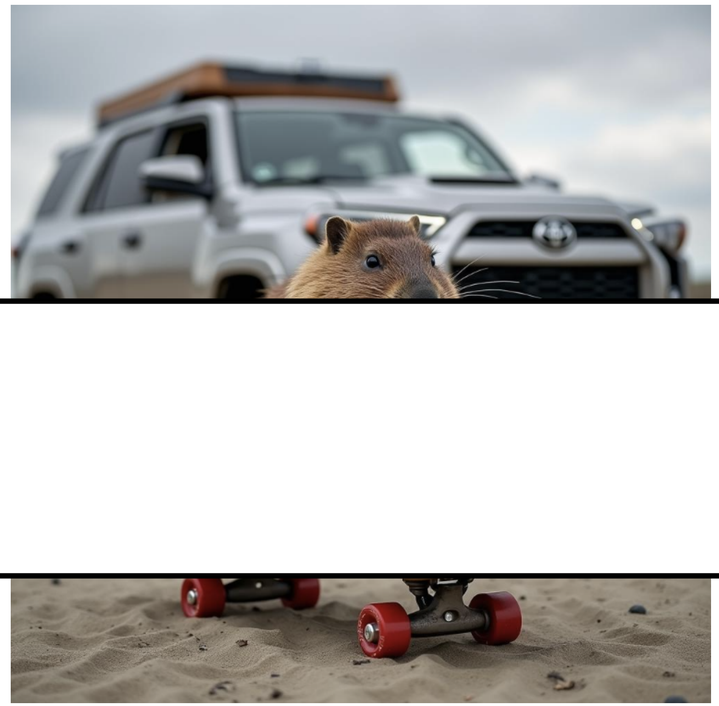
"a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**"

Flux.1-dev 12B

Clearly, these images are *not* just averaging of training images…







What might be missing?

"a beaver eating a hamburger on a skateboard"

"**a photo of a** beaver eating a hamburger on a skateboard"

"a photo of a beaver eating a hamburger on a skateboard **in front of Toyota 4Runner**"

Flux.1-dev 12B

Where does generalization come from…?

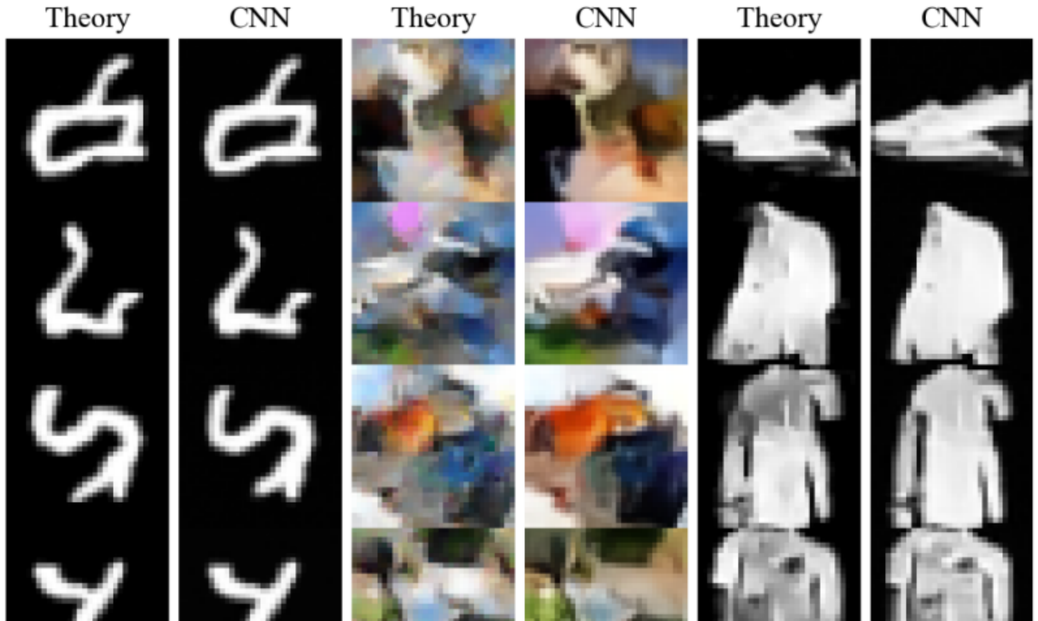
An analytic theory of creativity in convolutional diffusion models

Mason Kamb [1] **Surya Ganguli** [1]

28 Dec 2024

Abstract

We obtain the first analytic, interpretable and predictive theory of creativity in convolutional diffusion models. Indeed, score-based diffusion models can generate highly creative images that lie far from their training data. But optimal score-matching theory suggests that these models should only be able to produce memorized training examples. To reconcile this theory-experiment gap, we identify two simple inductive biases, locality and equivariance, that: (1) induce a form



Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z | \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2} \left(k_t(z) - z \right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax \left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

Diffusion's Analytical Probability **Score**

$$p_t(z) = \frac{1}{|\mathcal{X}|} \sum_i \mathcal{N}(z \,|\, \sqrt{\bar{\alpha}_t} x_i, (1 - \bar{\alpha}_t) I)$$

This has an analytical score !

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2} \left(k_t(z) - z \right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax \left(-\frac{\|z - tX\|^2}{2(1-t)^2} \right)_i tx_i$$

This kernel is computed over the *whole* image.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

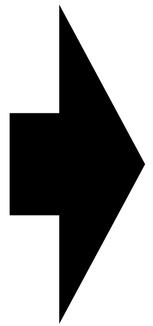
$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$



$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

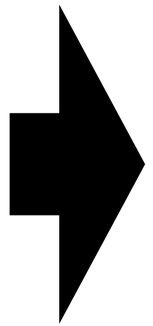
$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

Transition to different notation…

$$\nabla \log p_t^*(z) = \frac{1}{(1-t)^2}\left(k_t(z) - z\right)$$

$$k_t(z) = \sum_{i=1}^{N} softmax\left(-\frac{\|z - tX\|^2}{2(1-t)^2}\right)_i tx_i$$



$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t}\sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

This is to be consistent with the notation in the paper.

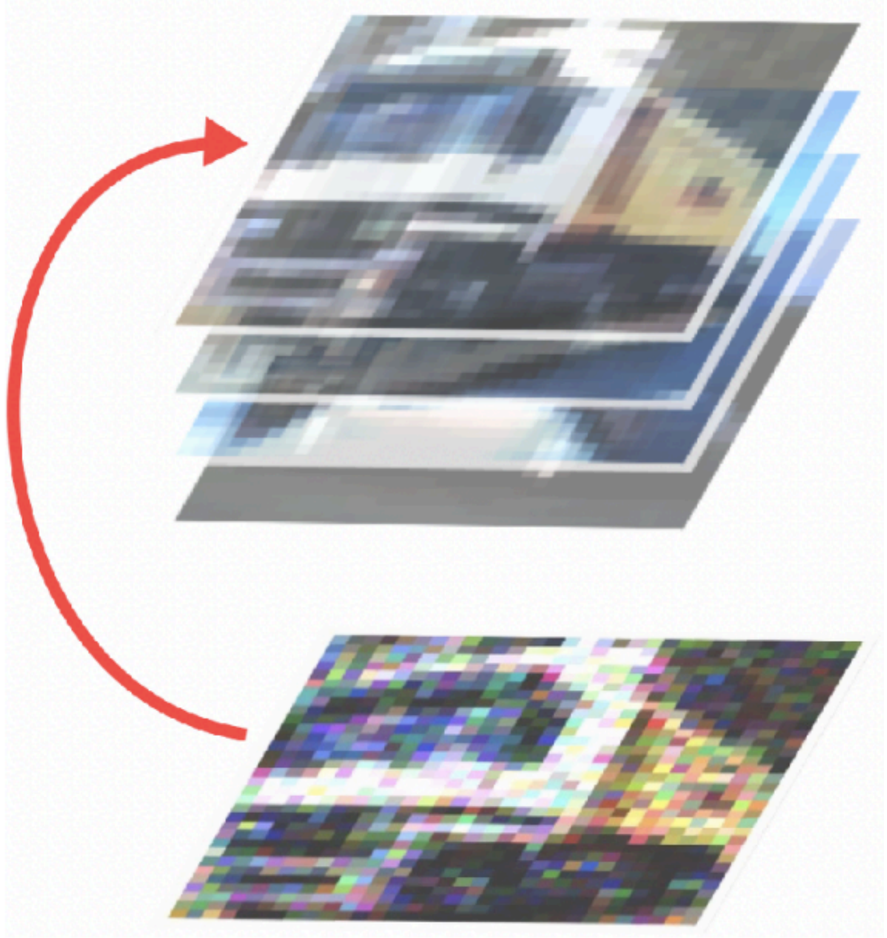
Transition to different notation…

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t} \sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

Time-dependent score function $s_t(\phi)$, current latent ϕ, images in training set φ, noise schedule $\bar{\alpha}_t$, softmax weights W_t

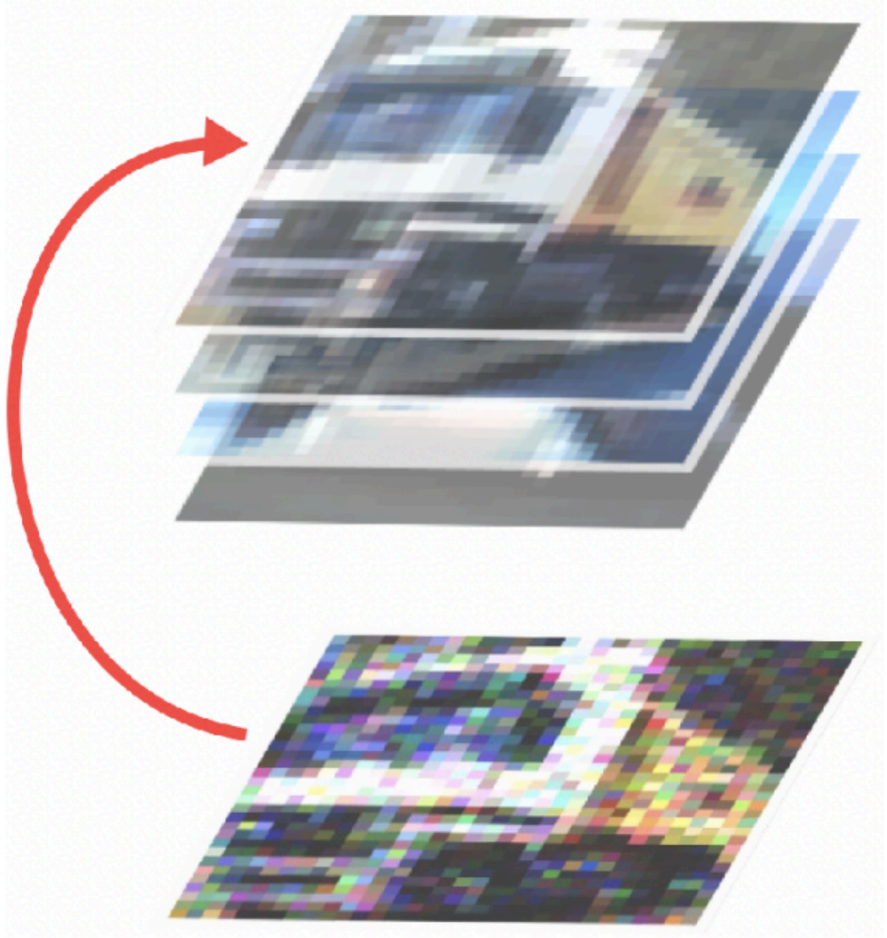
So far: Analytical Score Machine = Weighted Sum of Full Training Images



Weighted sum over *full-image* scores…

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t} \sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

So far: Analytical Score Machine = Weighted Sum of Full Training Images



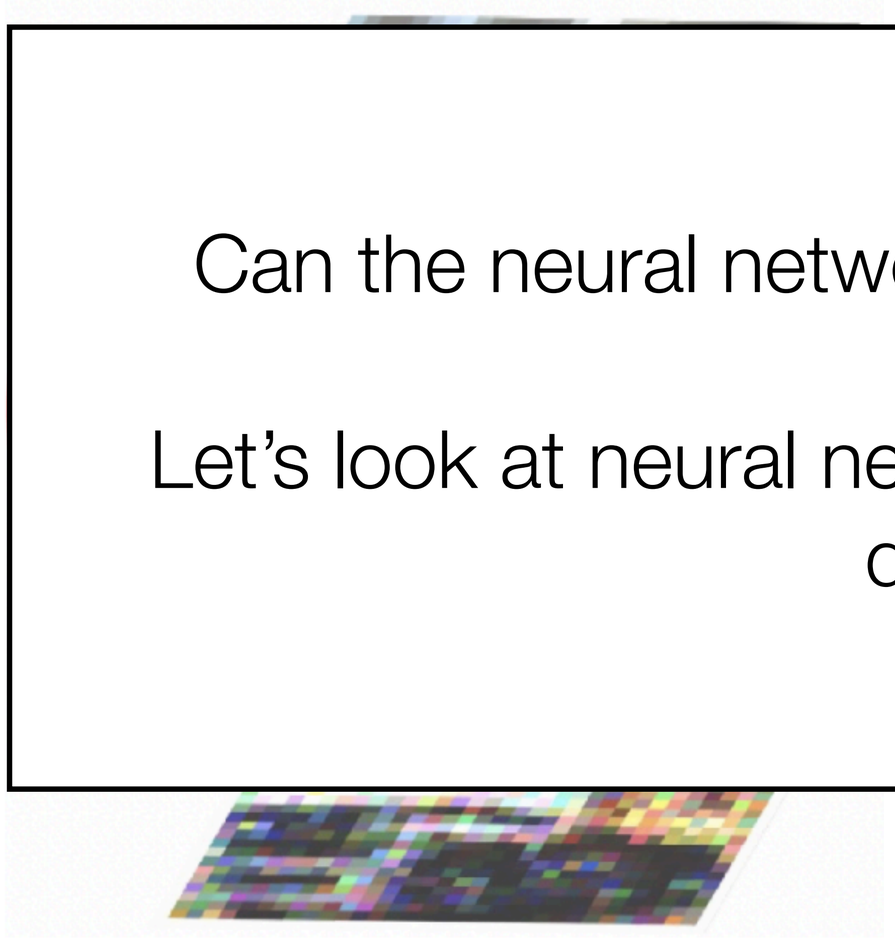
Weighted sum over *full-image* scores…

$$s_t(\phi) = \frac{1}{1-\bar{\alpha}_t} \sum_{\varphi \in \mathcal{D}} (\sqrt{\bar{\alpha}_t}\varphi - \phi) W_t(\varphi|\phi)$$

$$W_t(\varphi|\phi) = \frac{\mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi, (1-\bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi|\sqrt{\bar{\alpha}_t}\varphi', (1-\bar{\alpha}_t)I)}$$

…where weights are computed by comparing *full images.*

So far: Analytical Score Machine = Weighted Sum of Full Training Images

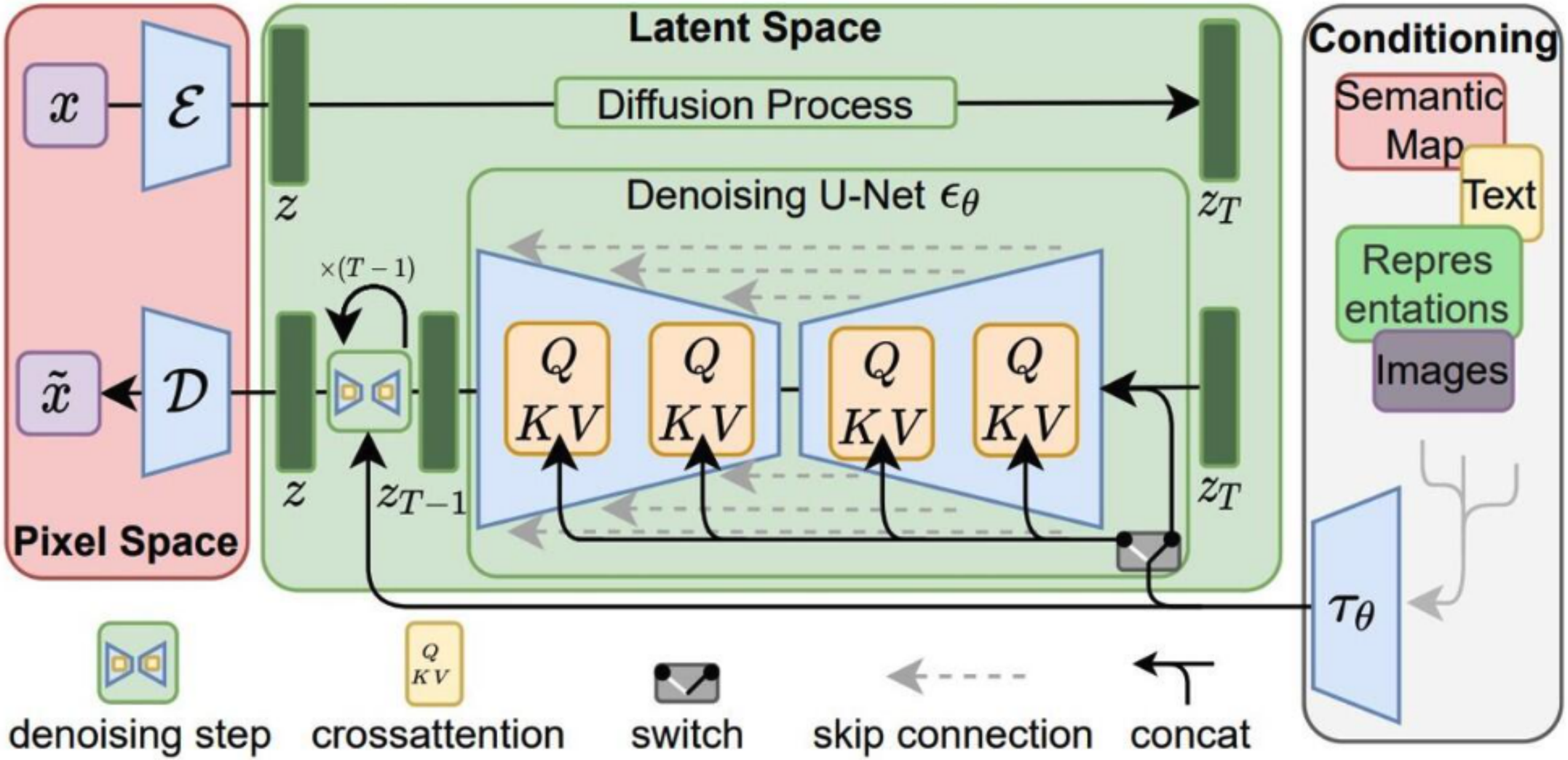


Can the neural network actually *learn* to predict this score?

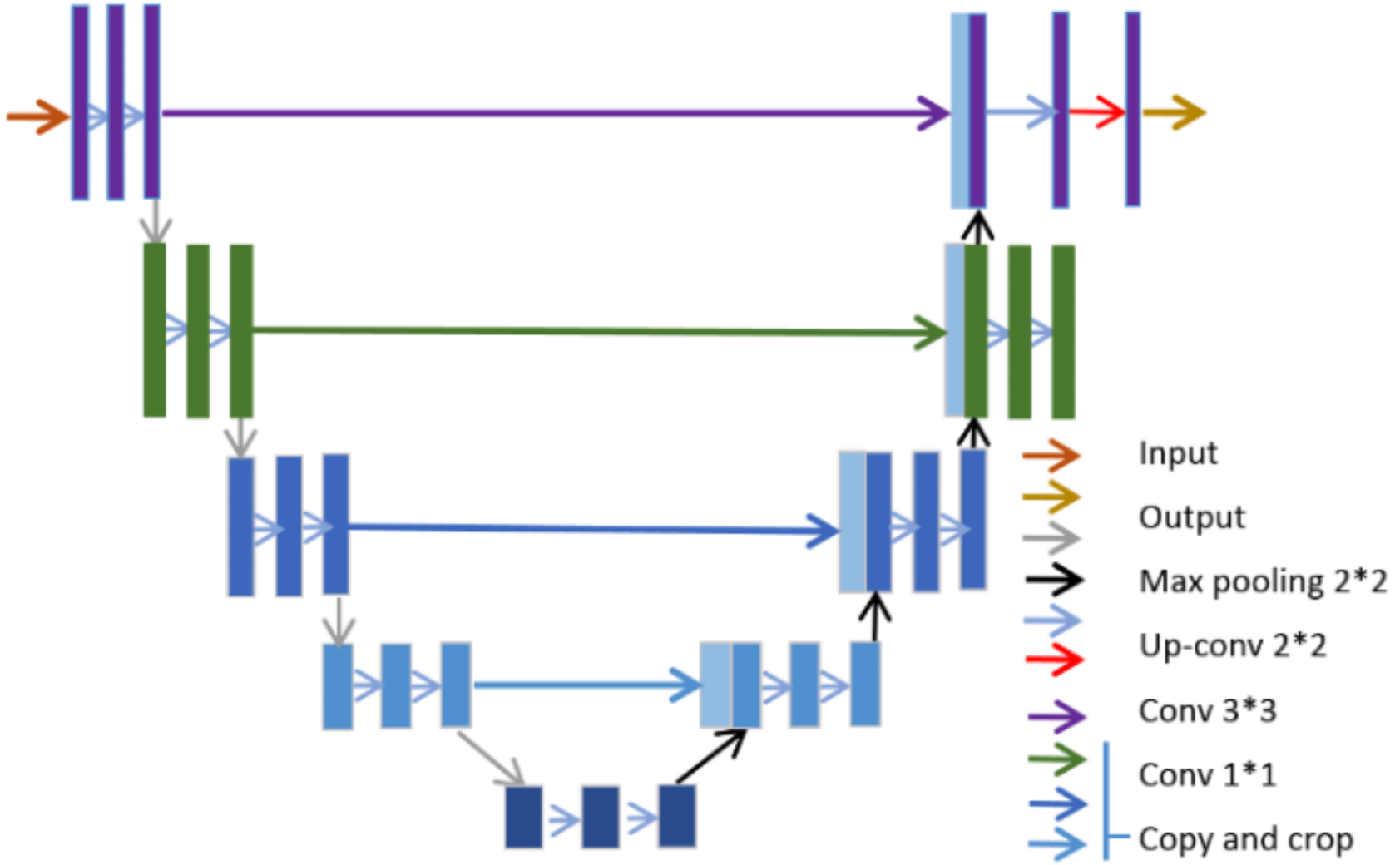
Let's look at neural net architectures that are actually used for diffusion in practice.

…where weights are computed by comparing *full images*.

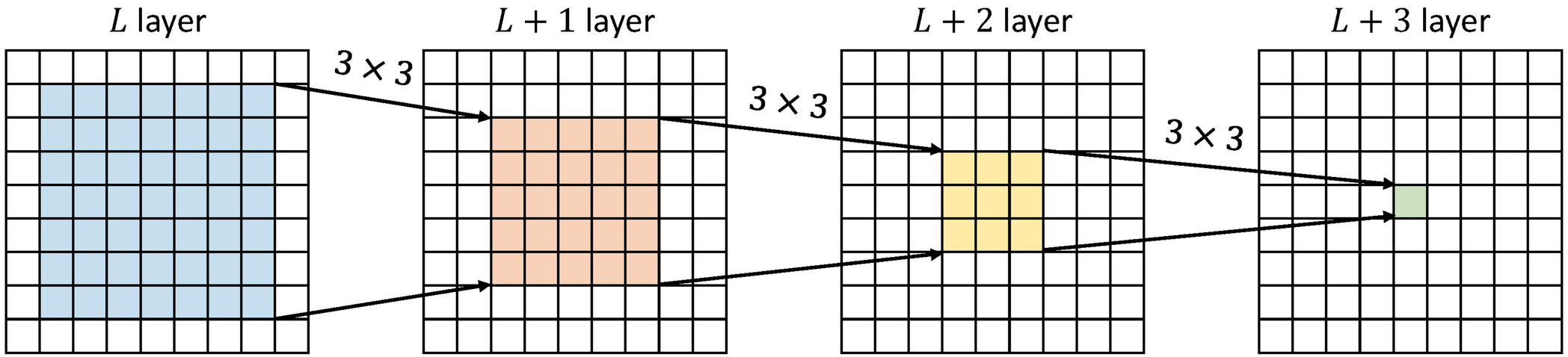
Stable Diffusion: Diffusion U-Nets



U-Nets: A Particular Kind of *Fully Convolutional* Neural Networks

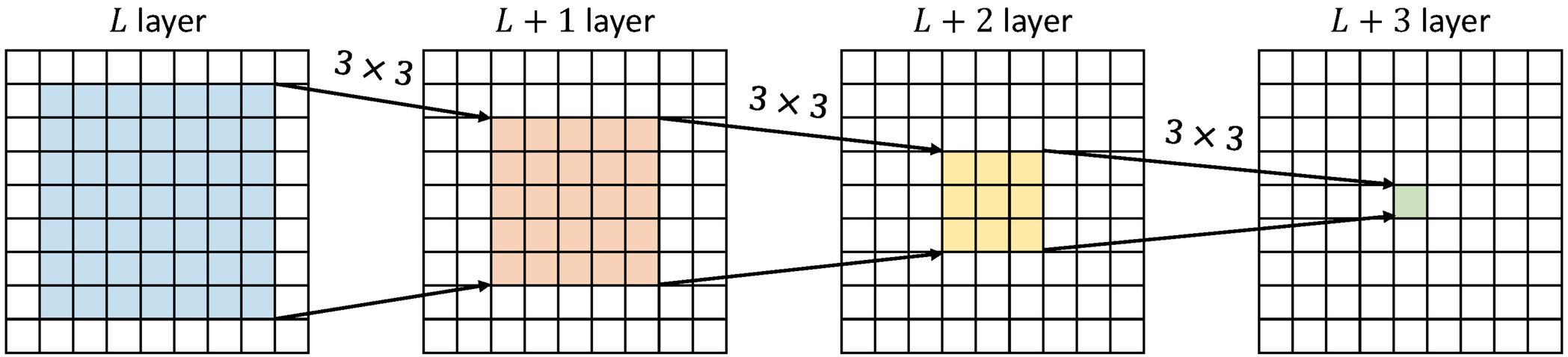


Fully Convolutional Neural Networks have a *Locality Bias.*



[Figure from Multi-Scale Receptive Fields Convolutional Network for Action Recognition, Dong et al. 203]

Fully Convolutional Neural Networks have a *Locality Bias.*



A given pixel in the output is a function of only the input pixels in its *receptive field.*

[Figure from Multi-Scale Receptive Fields Convolutional Network for Action Recognition, Dong et al. 203]

In practice, *Effective Receptive Fields* are smaller

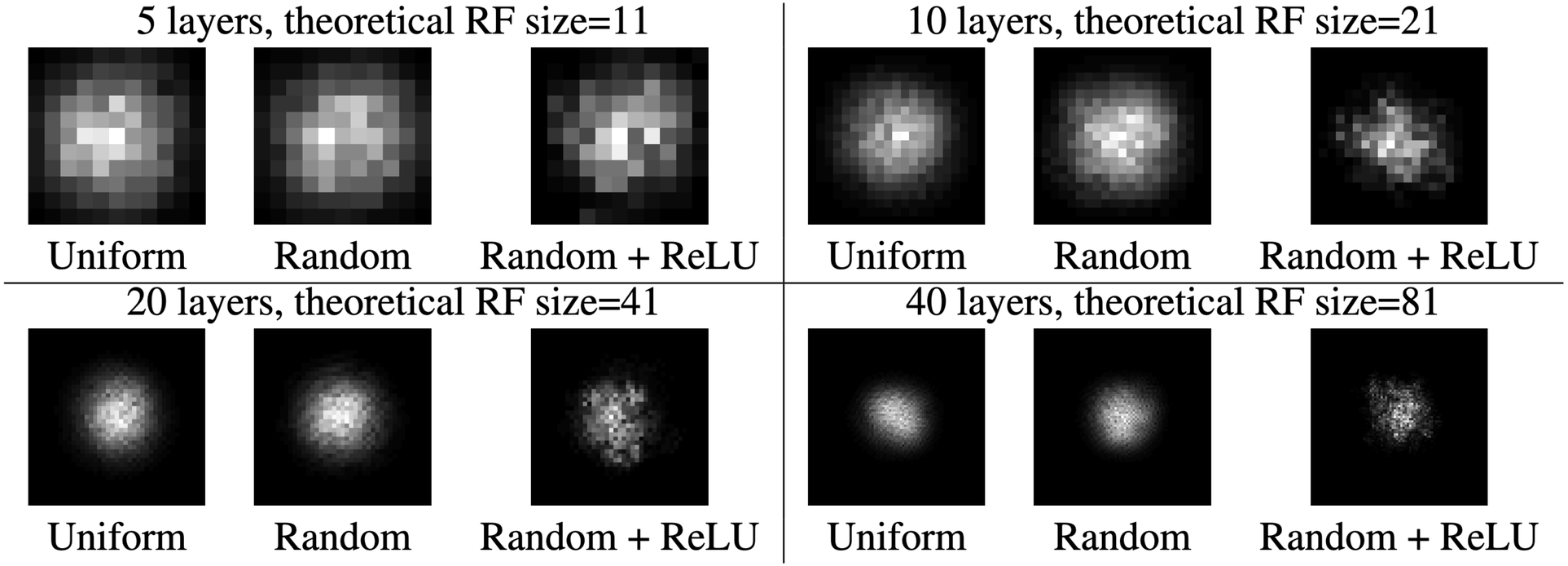
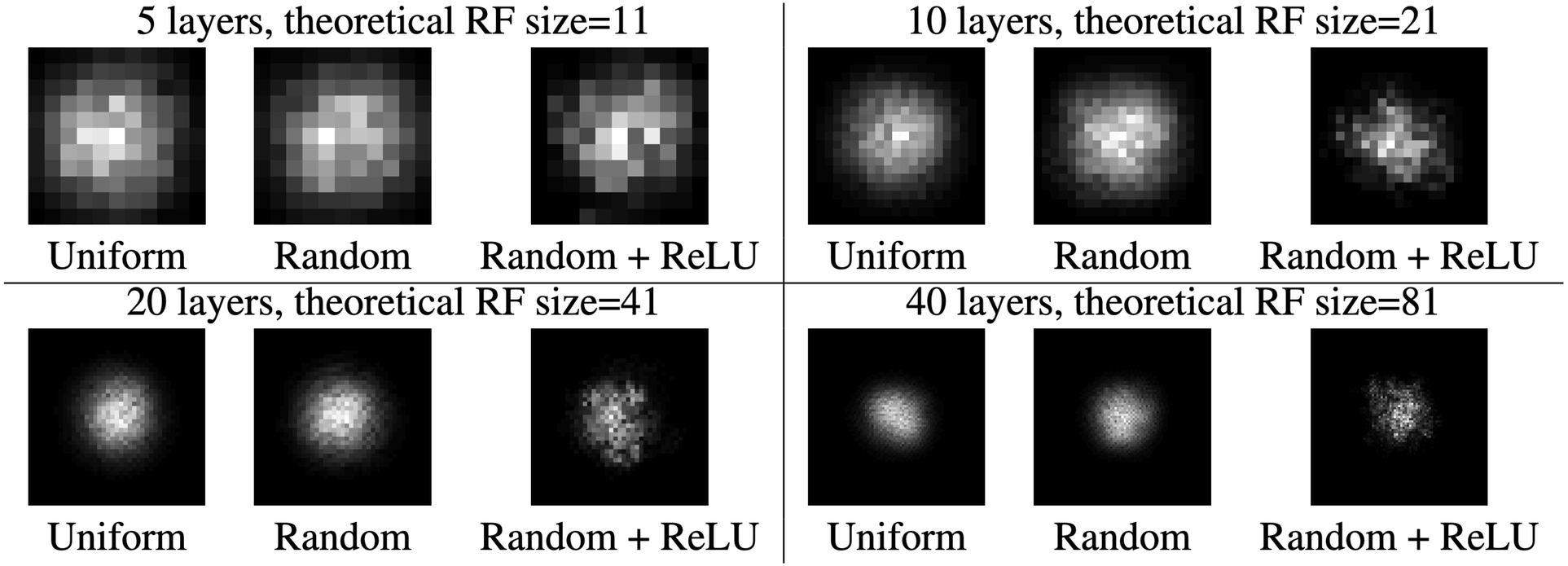


Figure from [Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, Luo, Li et al. 2016]

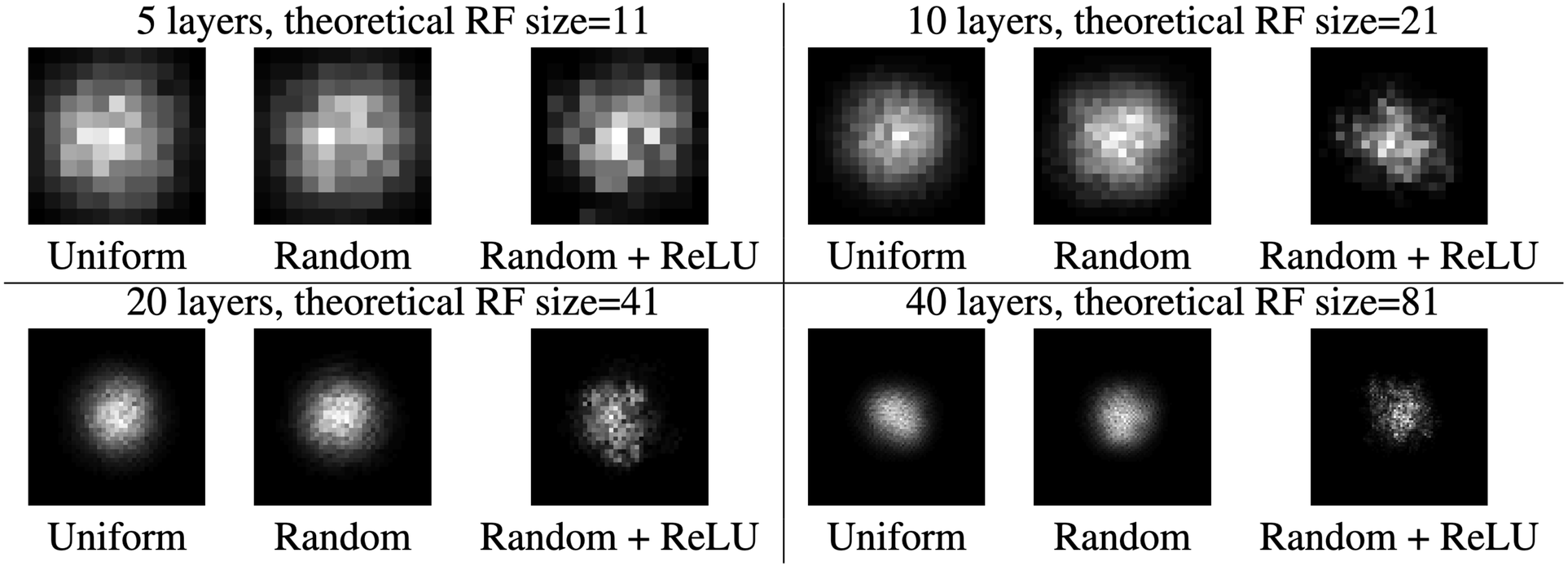
In practice, *Effective Receptive Fields* are smaller



Sensitivies (=magnitudes of gradients) of output pixel wrt input pixels.

Figure from [Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, Luo, Li et al. 2016]

In practice, *Effective Receptive Fields* are smaller

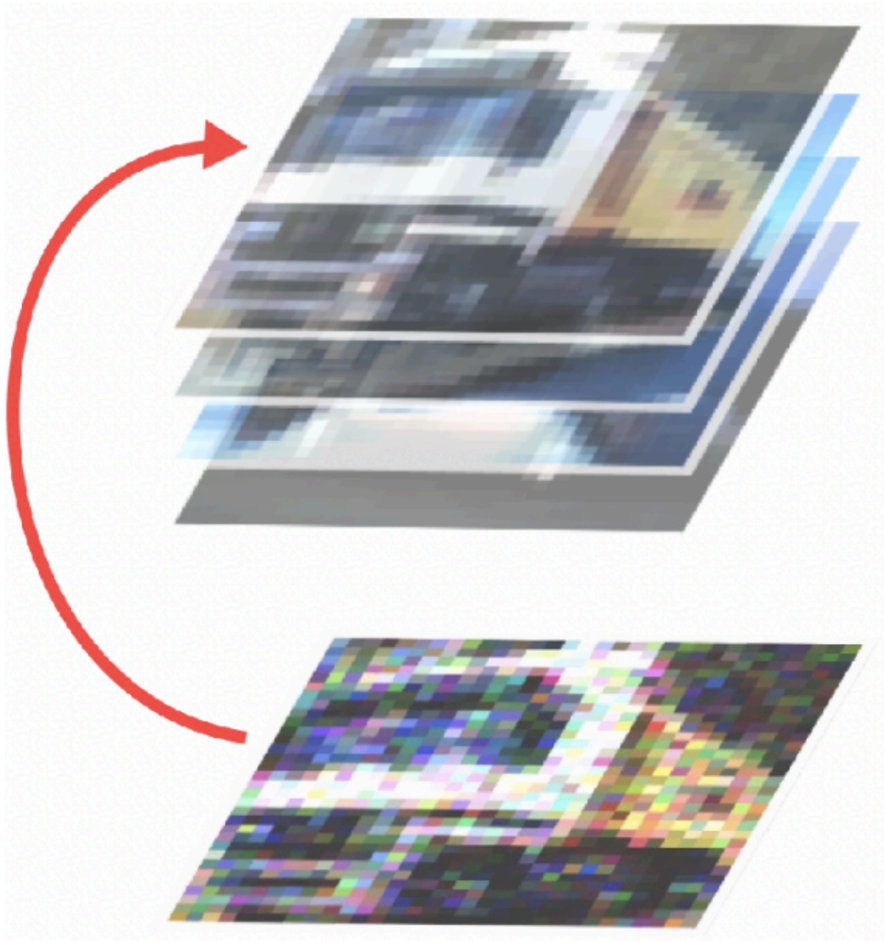


Sensitivies (=magnitudes of gradients) of output pixel wrt input pixels.

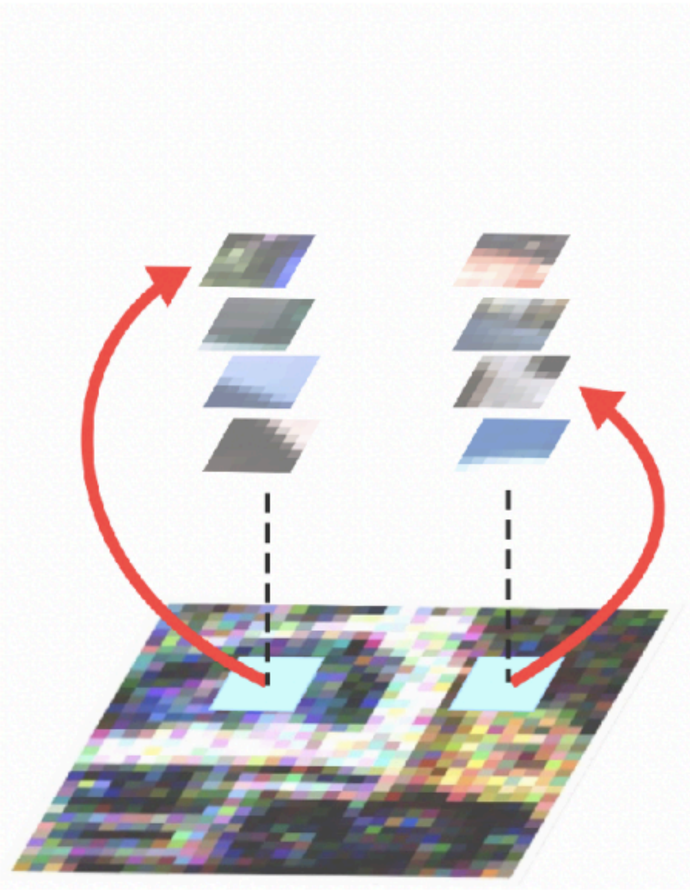
Denote effective receptive field as Ω_x.

Figure from [Understanding the Effective Receptive Field in Deep Convolutional Neural Networks, Luo, Li et al. 2016]

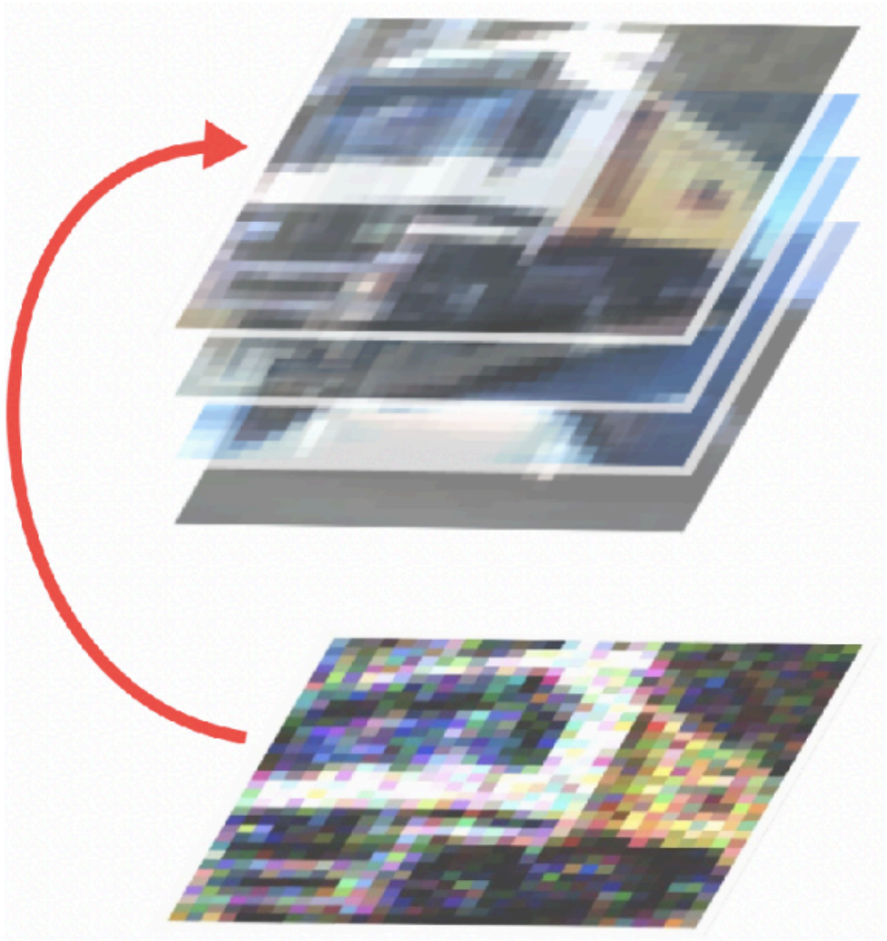
Idea: Make Score Machine *Local*



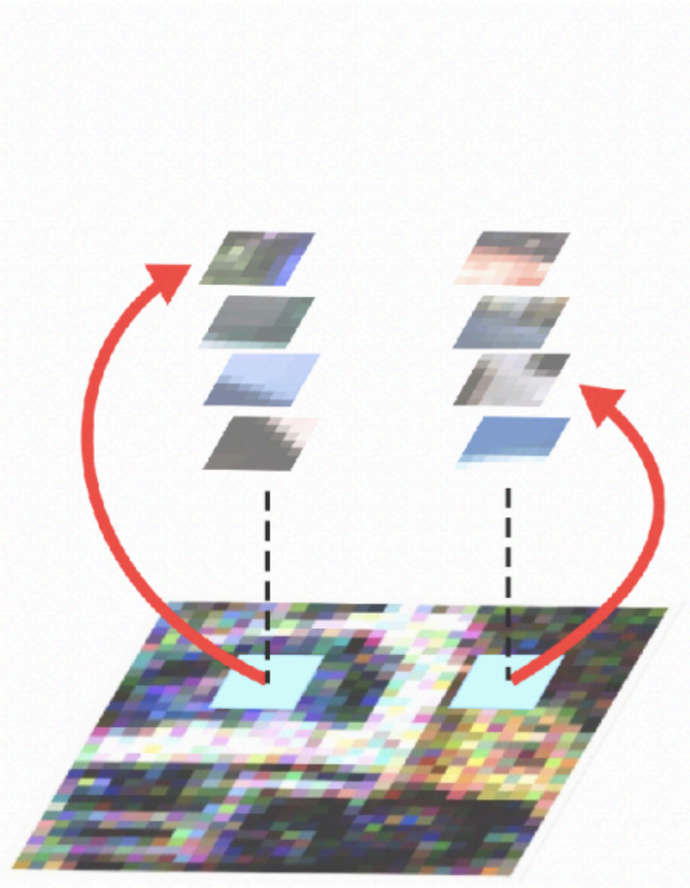




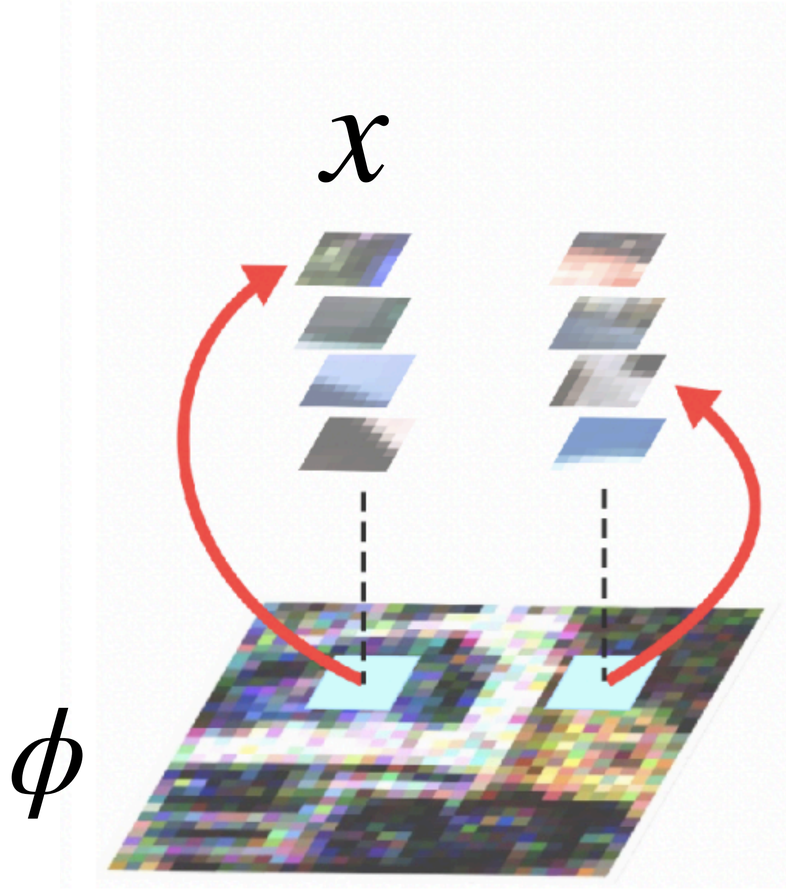
Idea: Make Score Machine *Local*



Each *patch* in the diffused image is a weighted sum of *patches* in training set!



Idea: Make Score Machine *Local*



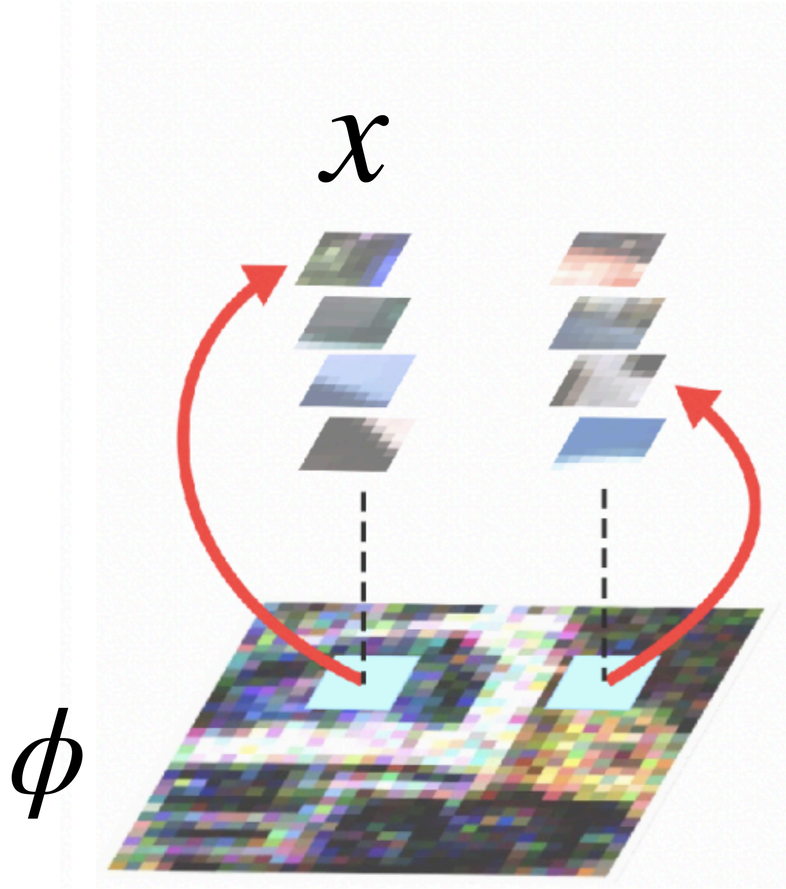
Idea: Make Score Machine *Local*

Denote:

Image as ϕ

Pixel coordinate as x

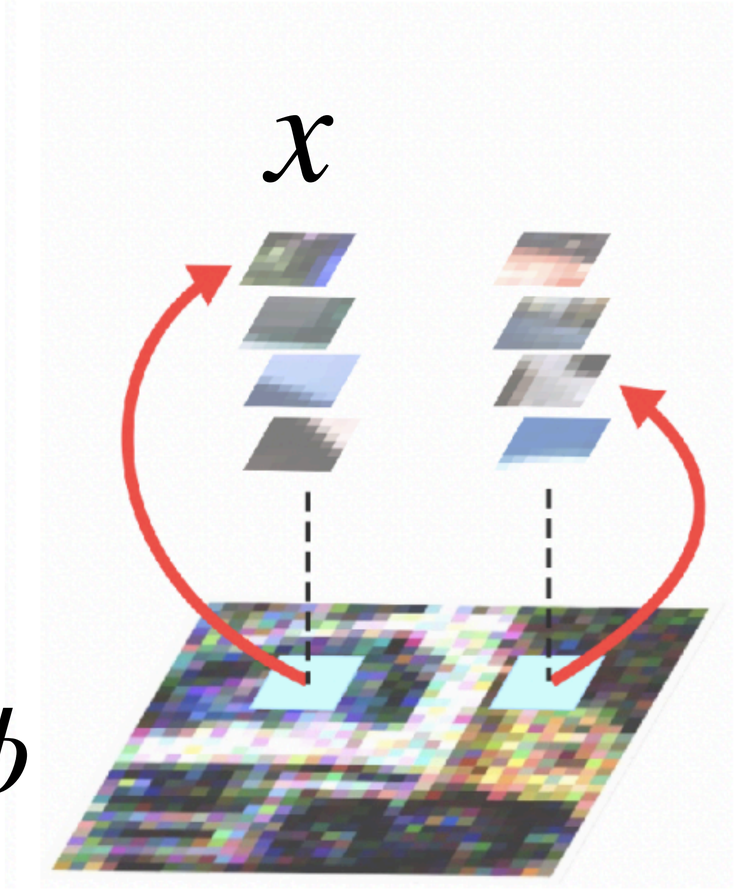
And score for pixel at x
in image ϕ as $M_t[\phi](x)$



Idea: Make Score Machine *Local*

$$M_t[\phi](x) = \sum_{\varphi \in \mathcal{D}} \frac{(\sqrt{\bar{\alpha}_t}\varphi(x) - \phi(x))}{1 - \bar{\alpha}_t} W_t(\varphi_{\Omega_x} | \phi_{\Omega_x})$$

$$W_t(\varphi_{\Omega_x} | \phi_{\Omega_x}) = \frac{\mathcal{N}(\phi_{\Omega_x} | \sqrt{\bar{\alpha}_t}\varphi_{\Omega_x}, (1 - \bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi_{\Omega_x} | \sqrt{\alpha}_t \varphi'_{\Omega_x}, (1 - \bar{\alpha}_t)I)}$$

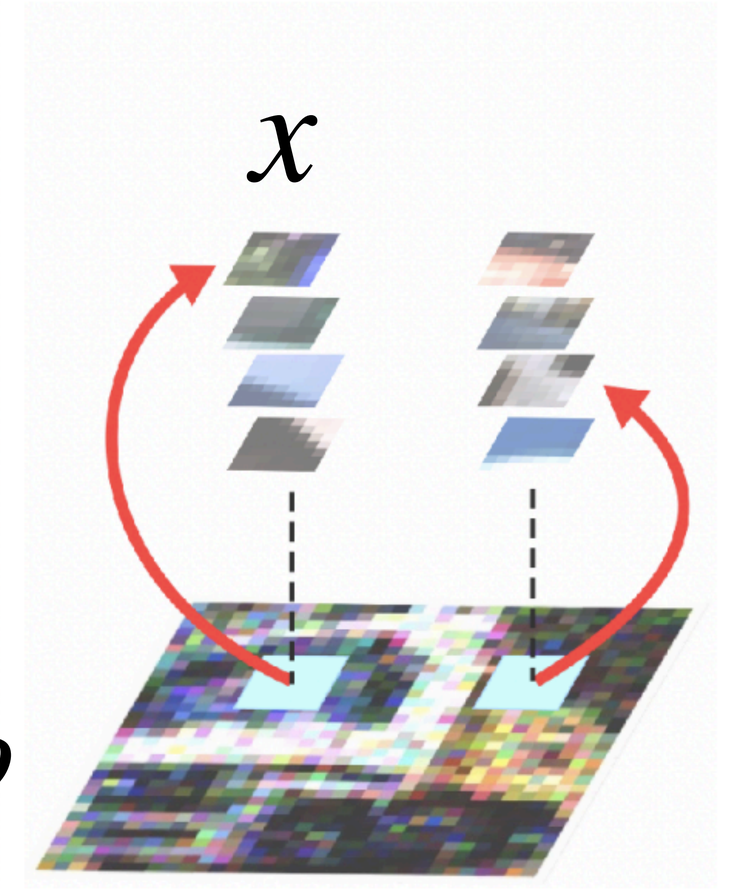


Idea: Make Score Machine *Local*

$$M_t[\phi](x) = \sum_{\varphi \in \mathcal{D}} \frac{(\sqrt{\bar{\alpha}_t}\varphi(x) - \phi(x))}{1 - \bar{\alpha}_t} W_t(\varphi_{\Omega_x} | \phi_{\Omega_x})$$

$$W_t(\varphi_{\Omega_x} | \phi_{\Omega_x}) = \frac{\mathcal{N}(\phi_{\Omega_x} | \sqrt{\bar{\alpha}_t}\varphi_{\Omega_x}, (1 - \bar{\alpha}_t)I)}{\sum_{\varphi' \in \mathcal{D}} \mathcal{N}(\phi_{\Omega_x} | \sqrt{\alpha}_t \varphi'_{\Omega_x}, (1 - \bar{\alpha}_t)I)}$$

Pixel at x in image ϕ is pulled towards training set pixel values $\varphi(x)$ according to similarities of effective receptive fields Ω_x.

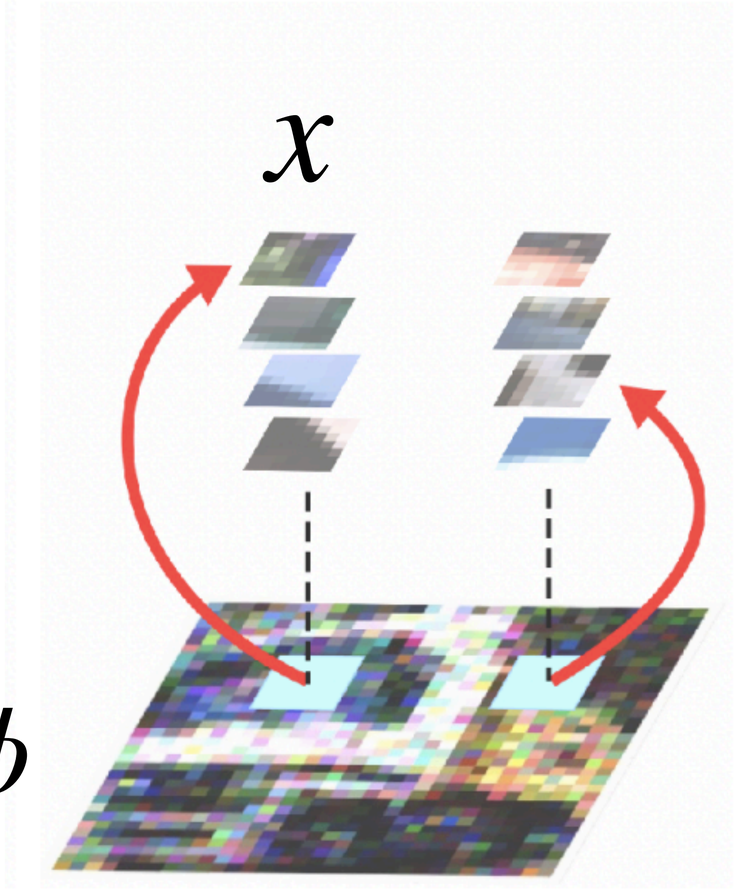


Idea: Make Score Machine *Local*

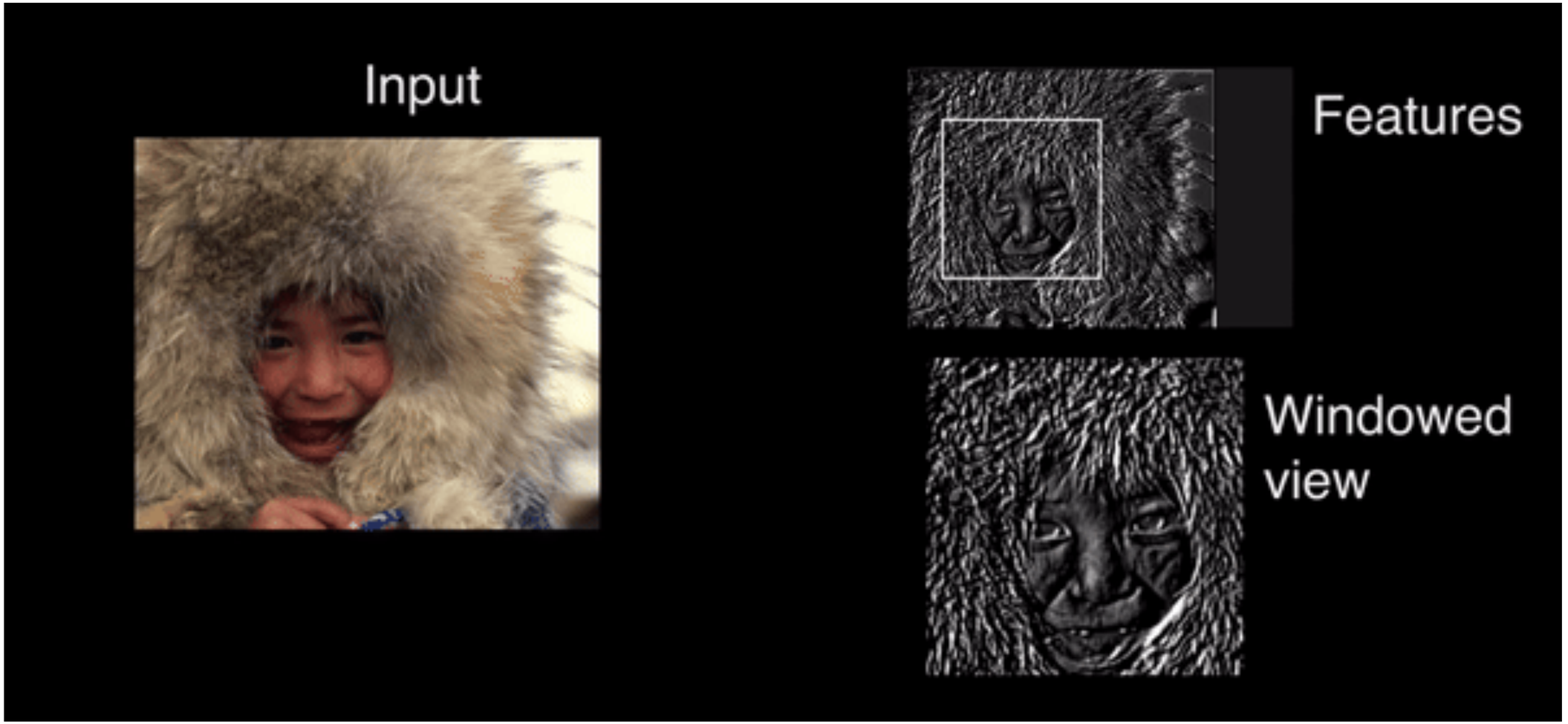
$$M_t[\phi](x) = \sum_{\varphi \in \mathcal{D}} \frac{(\sqrt{\bar{\alpha}_t}\varphi(x) - \phi(x))}{1 - \bar{\alpha}_t} W_t(\varphi_{\Omega_x} | \phi_{\Omega_x})$$

What has changed?

The softmax weights can now be *different* for each pixel in the image!

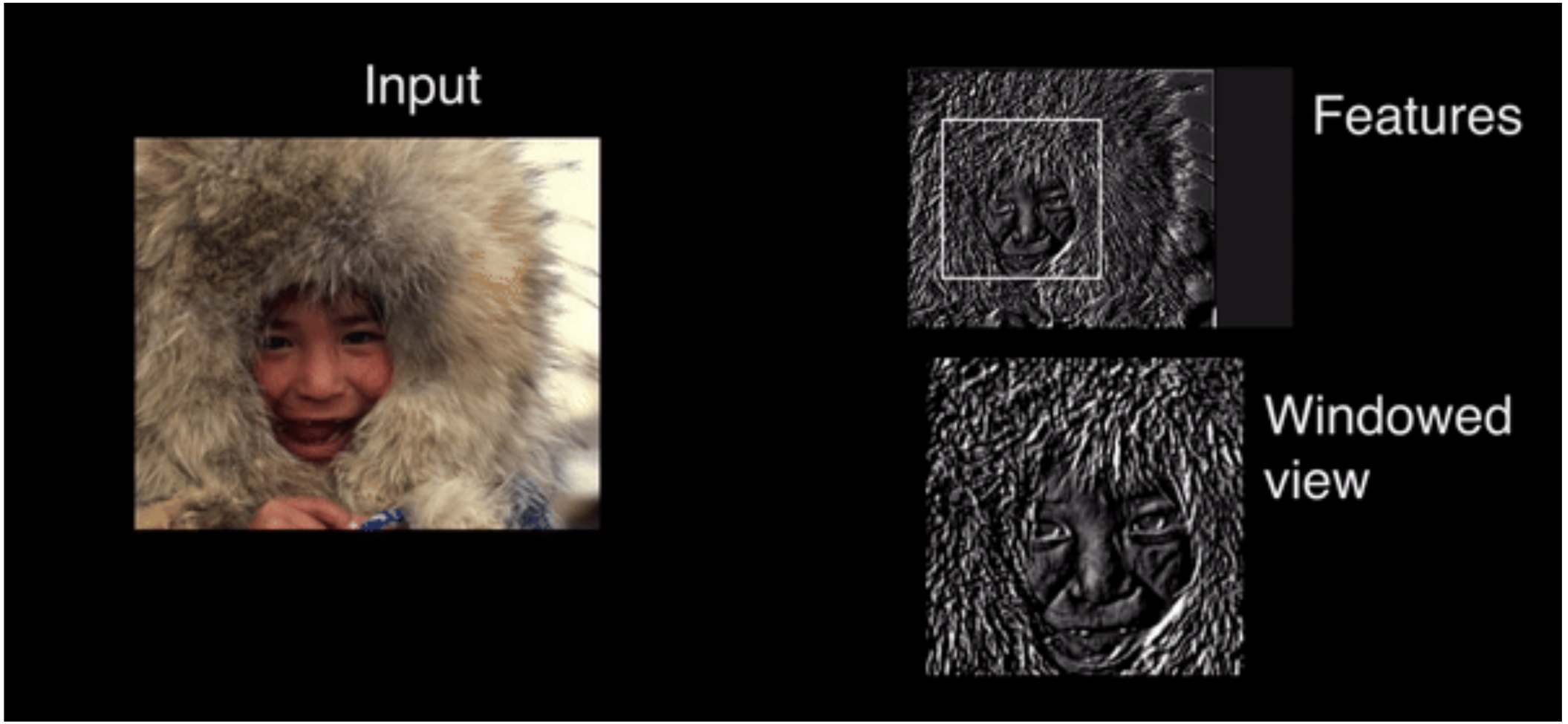


CNNs are *shift-equivariant!*



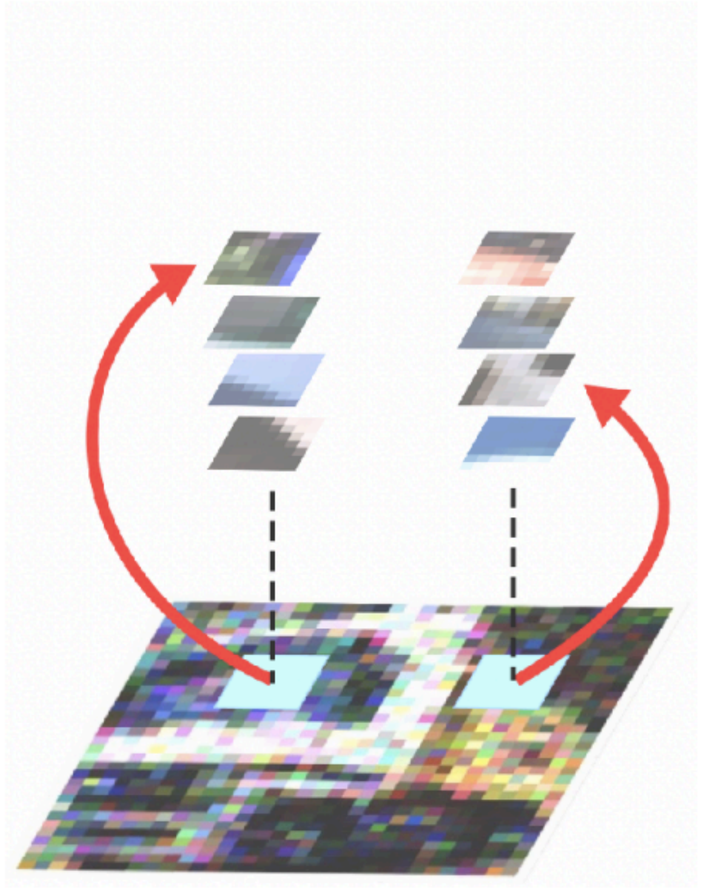
*** CNNs and Equivariance, Fabian Fuchs**

CNNs are *shift-equivariant!*

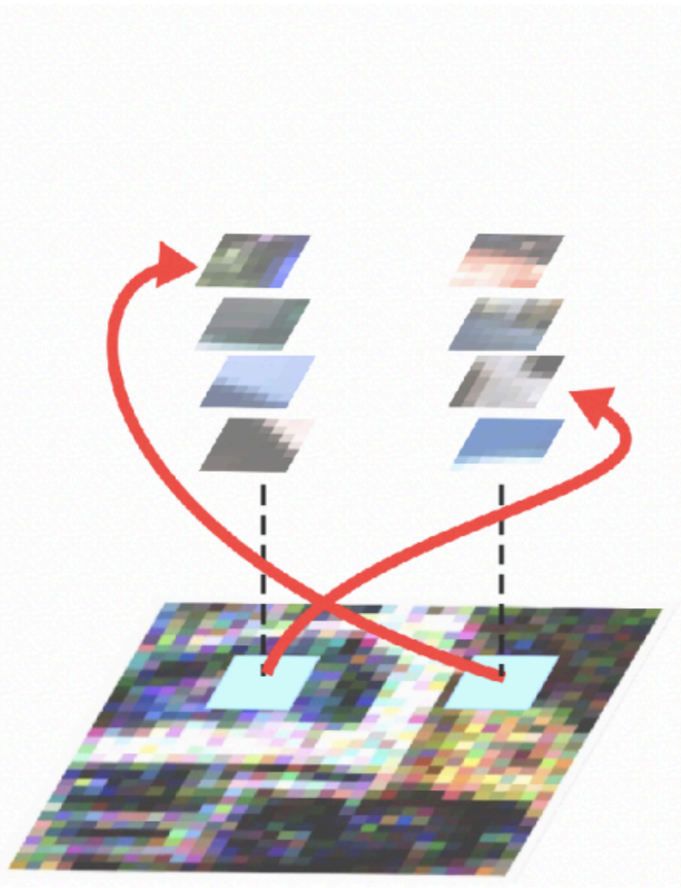


*** CNNs and Equivariance, Fabian Fuchs**

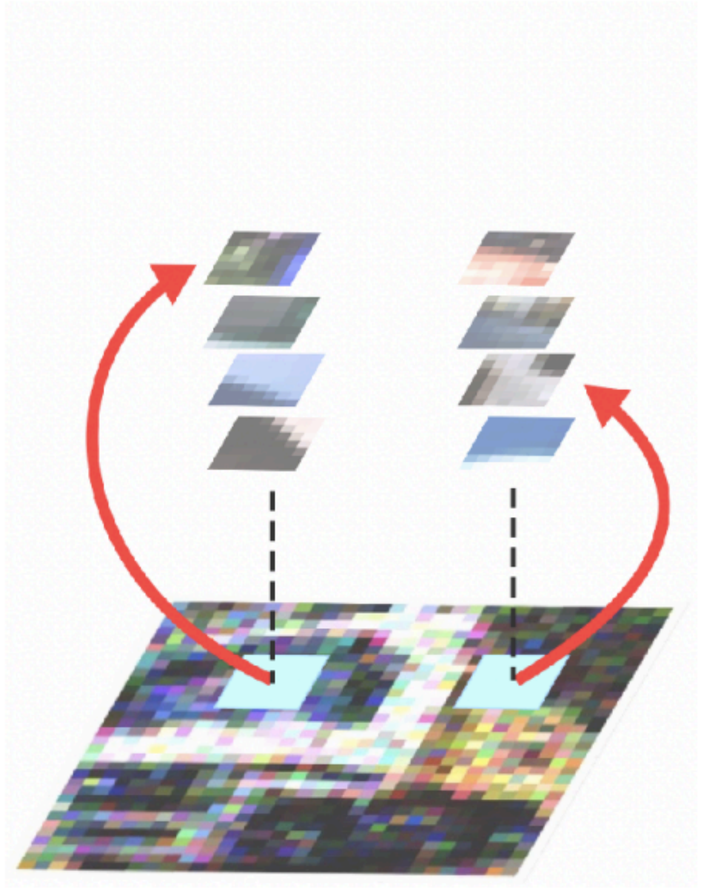
Idea: Make Local Score Machine
Shift-Equivariant



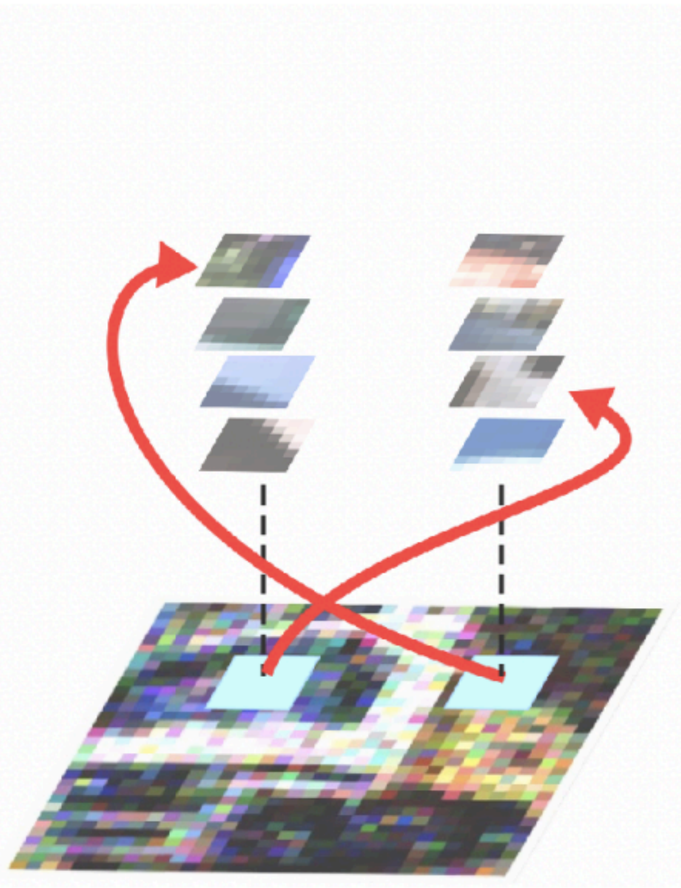


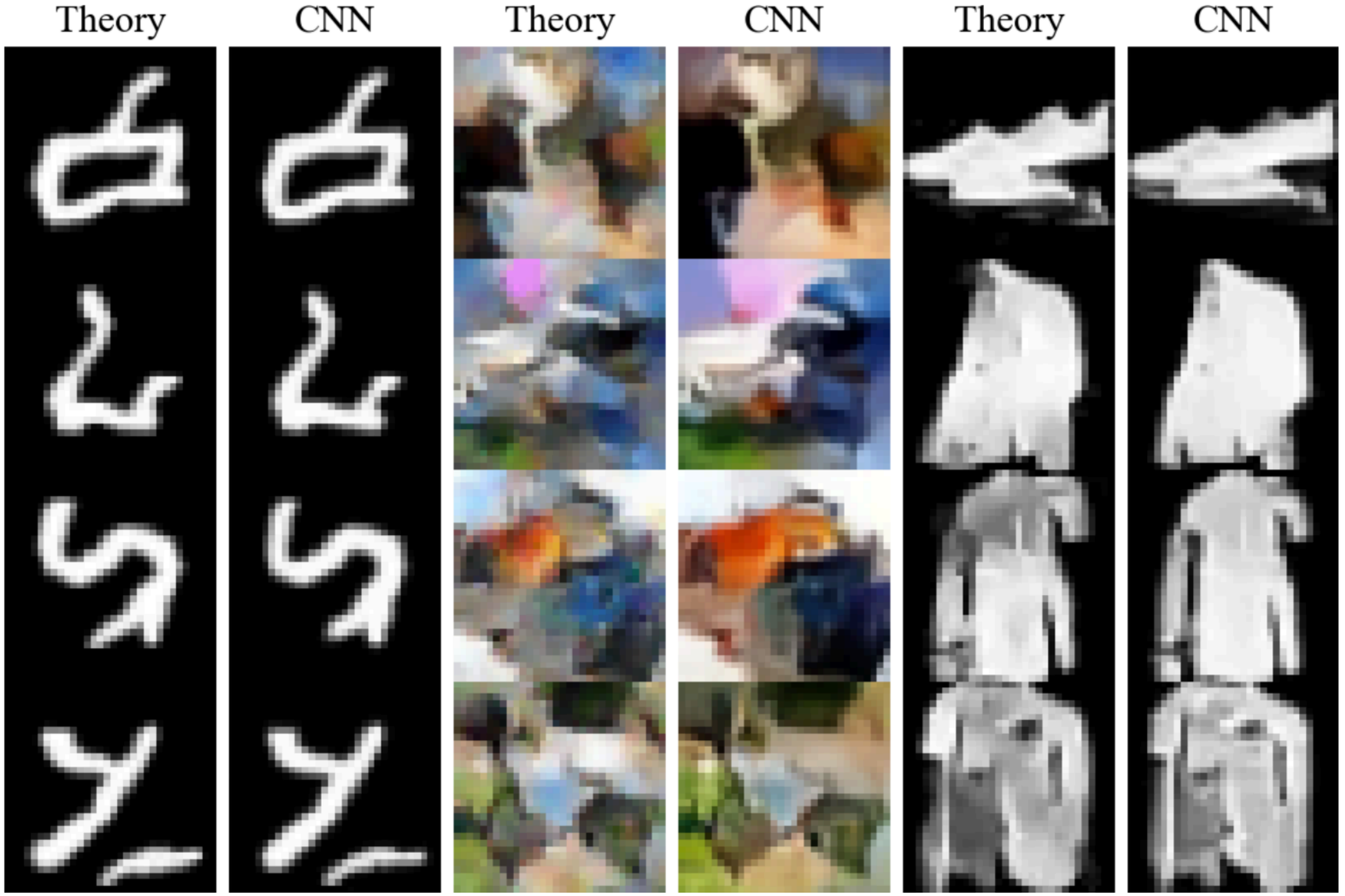


Idea: Make Local Score Machine *Shift-Equivariant*



The diffusion model doesn't know where in the image the patches came from!





In many ways images are *worse* than the prior ones on CelebA, but impressively, *they predict the behavior of a trained U-Net Diffusion Model.*



An analytical, “optimal” generative model exists for most generative models.

In practice, theories of generalization: that problem is *as hard* as explaining generalization in *any other* neural net

This can only serve as a “mental model” or as intuition

The “real magic” is deep learning and not understood.

In many ways images are *worse* than the prior ones on CelebA, but impressively, *they predict the behavior of a trained U-Net Diffusion Model.*

Locality in Image Diffusion Models Emerges from Data Statistics

Artem Lukoianov
Massachusetts Institute of Technology
artem1@mit.edu

Chenyang Yuan
Toyota Research Institute
chenyang.yuan@tri.global

Justin Solomon
Massachusetts Institute of Technology
jsolomon@mit.edu

Vincent Sitzmann
Massachusetts Institute of Technology
sitzmann@mit.edu

https://locality.lukoianov.com

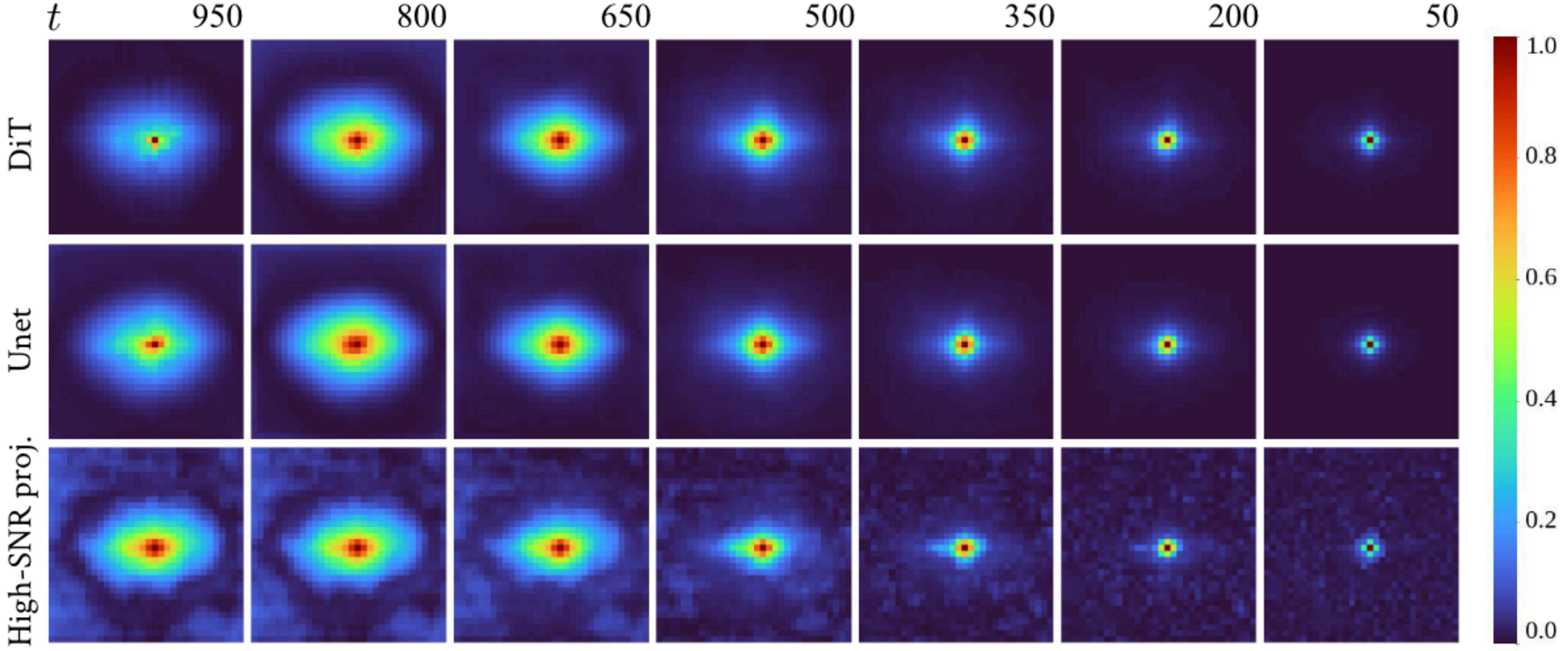


Figure 2: Comparison of sensitivity fields of deep denoisers and the projection operators to high-SNR data's components (i.e. the Wiener filter) on CIFAR-10 dataset. Sensitivity is measured at the center pixel w.r.t. x_0 prediction and throughout a 1000-step DDIM denoising process. Each image is averaged across 32 samples and normalized to [0,1].

Instead of the optimal *non-parametric* denoiser, let's look at the optimal *linear* denoiser

$$W_t = \frac{1}{\sqrt{\alpha_t}} U \operatorname{diag}\left(\frac{\lambda_i^2}{\lambda_i^2 + \sigma_t^2}\right) U^T.$$

Wiener filter: optimal linear denoiser

U: Principal Component Matrix of dataset

λ: Eigenvalue of denoised datapoint

Instead of the optimal *non-parametric* denoiser, let's look at the optimal *linear* denoiser

$$W_t = \frac{1}{\sqrt{\alpha_t}} U \operatorname{diag}\left(\frac{\lambda_i^2}{\lambda_i^2 + \sigma_t^2}\right) U^T.$$

Wiener filter: optimal linear denoiser

U: Principal Componen Matrix

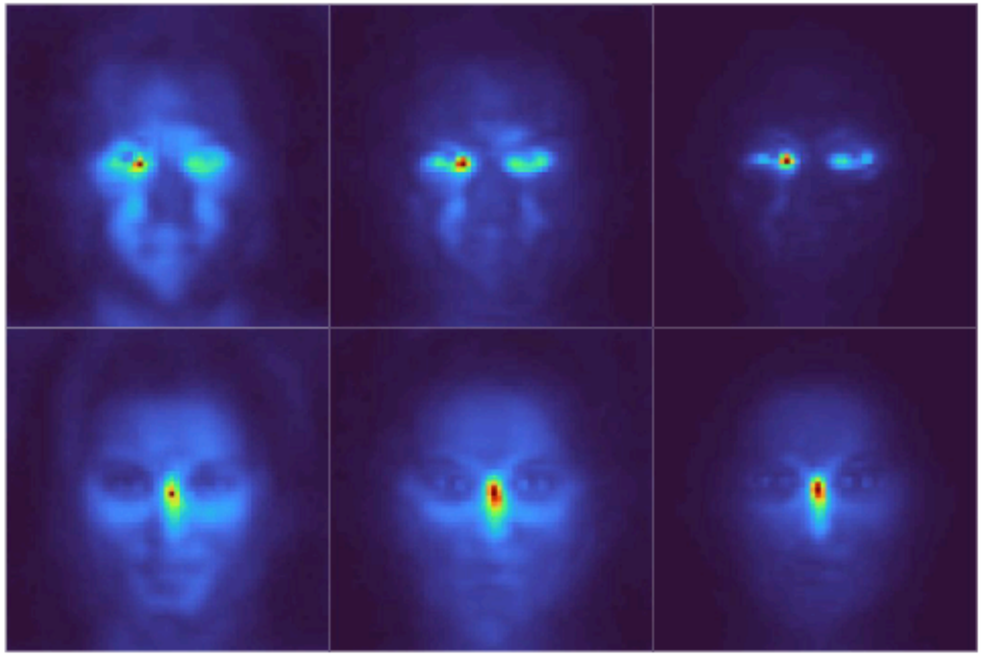
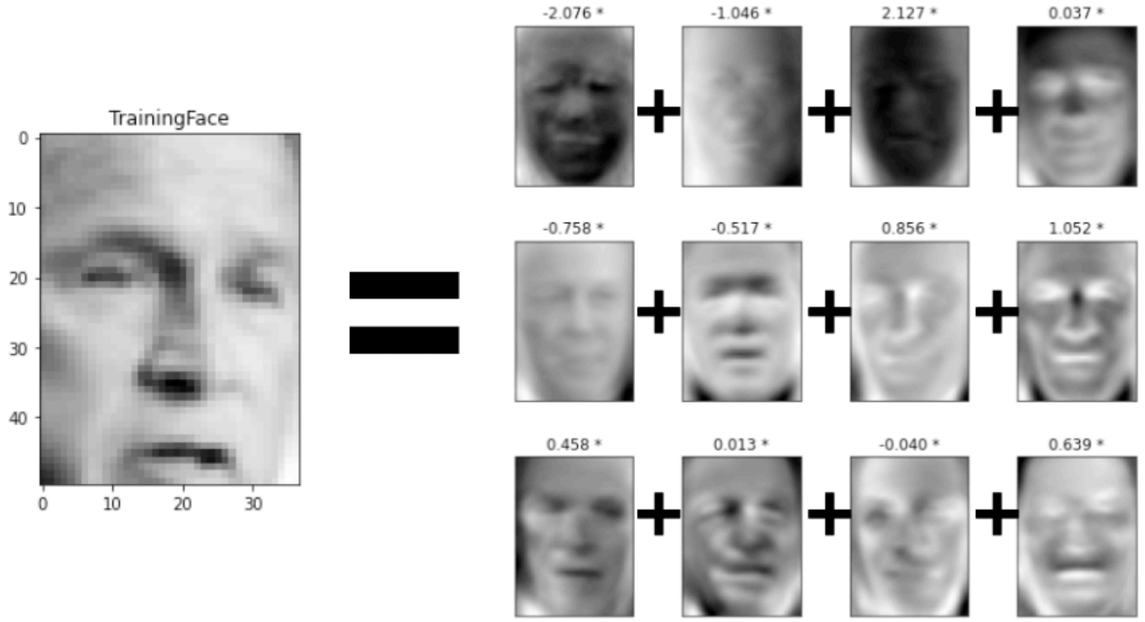


Figure 3: Average sensitivity fields of a trained DDPM on the CelebA-HQ dataset. The top row corresponds to an output pixel located near the left eye; the bottom row corresponds to an output pixel near the image center. Left to right: different noise levels corresponding to t of 600, 400, 200.

Instead of the optimal *non-parametric* denoiser, let's look at the optimal *linear* denoiser



Eigenfaces, Turk & Pentland 1991

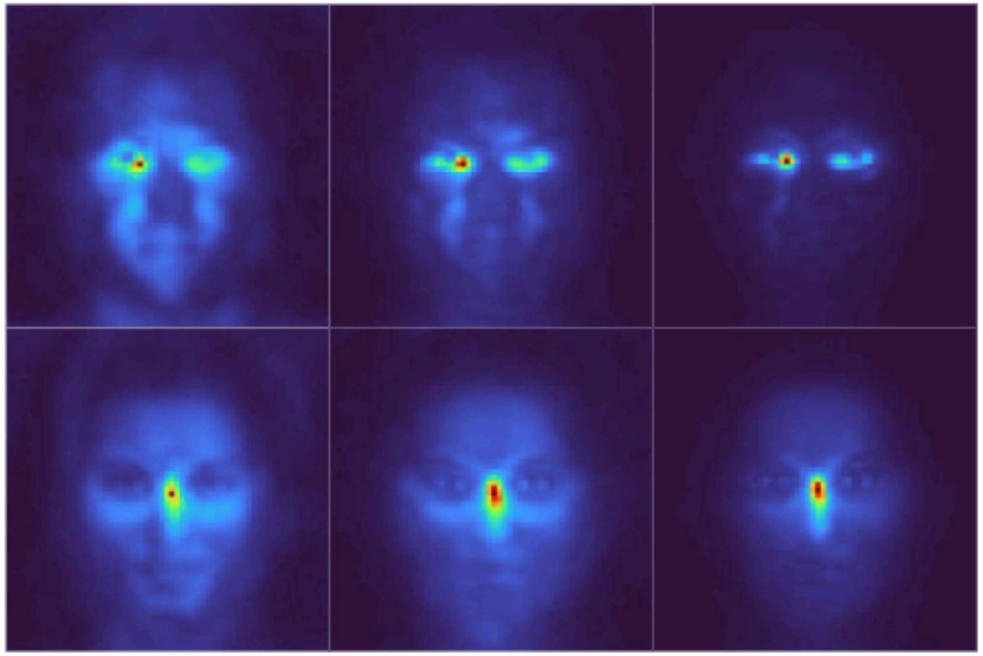
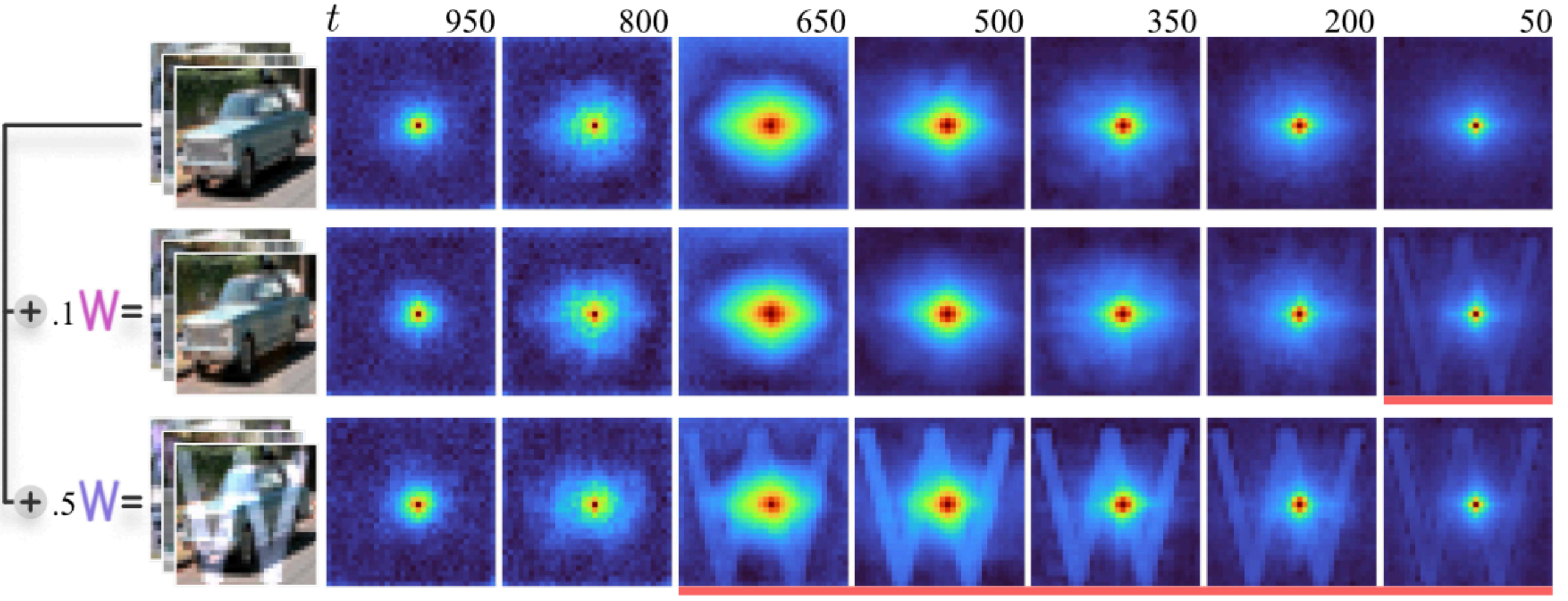


Figure 3: Average sensitivity fields of a trained DDPM on the CelebA-HQ dataset. The top row corresponds to an output pixel located near the left eye; the bottom row corresponds to an output pixel near the image center. Left to right: different noise levels corresponding to t of 600, 400, 200.


t
950
800
650
500
350
200
50
+ .1 W =
+ .5 W =


CelebA-HQ
CIFAR10
AFHQv2
MNIST
FashionMNIST
NN2
NN1
Ours
Wiener
Kamb
Dataset

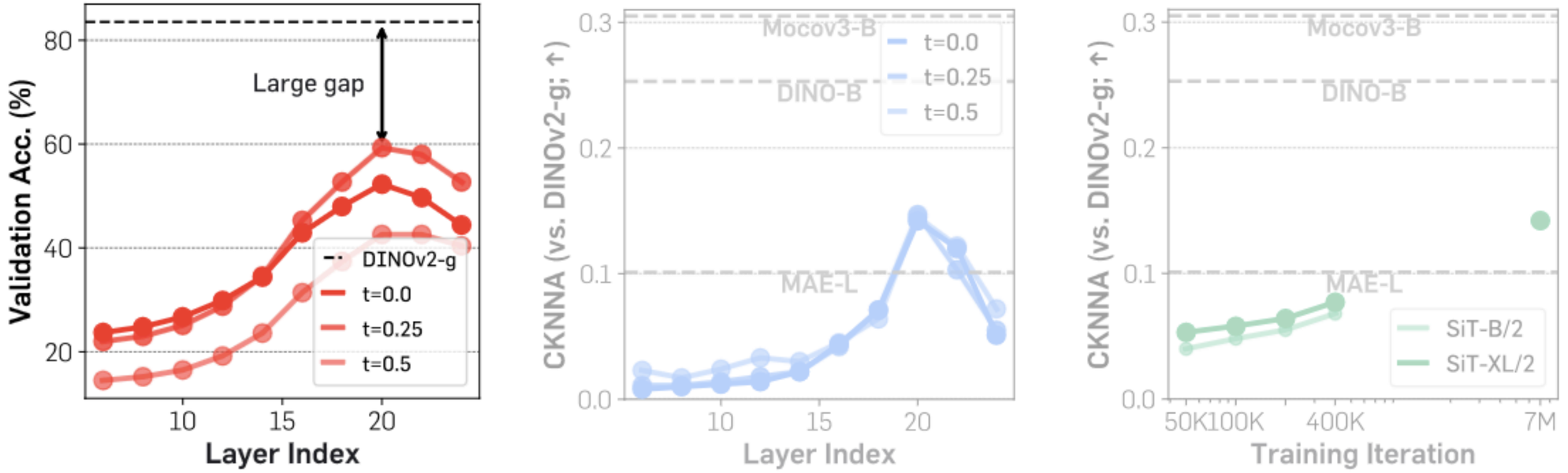
The Interplay of Generation & Representation Learning

Published as a conference paper at ICLR 2025

REPRESENTATION ALIGNMENT FOR GENERATION: TRAINING DIFFUSION TRANSFORMERS IS EASIER THAN YOU THINK

Sihyun Yu[1] **Sangkyung Kwak**[1,3] **Huiwon Jang**[1]
Jongheon Jeong[2] **Jonathan Huang**[3] **Jinwoo Shin**[1*] **Saining Xie**[4*]
[1]KAIST [2]Korea University [3]Scaled Foundations [4]New York University

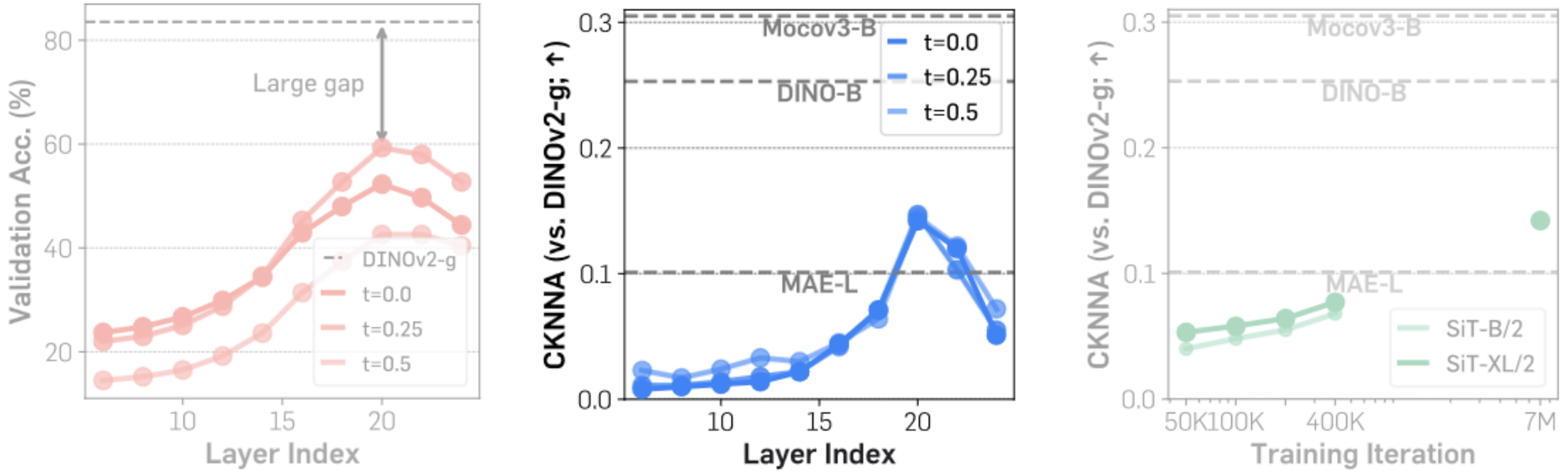
Observation 1: Diffusion Models Bad at Representation Learning



(a) Semantic gap: Linear probing (b) Alignment to DINOv2-g (c) Alignment progression

Diffusion transformers exhibit a significant semantic gap from state-of-the-art visual encoders. As shown in Figure 2a, we observe that the hidden state representation of the pretrained diffusion transformer, in line with prior works (Xiang et al., 2023; Chen et al., 2024c), achieves a reasonably high linear probing peak at layer 20. However, its performance remains well below that of DINOv2, indicating a substantial semantic gap between the two representations. Additionally, we find that after reaching this peak, linear probing performance quickly declines, suggesting that the diffusion transformer must shift away from focusing solely on learning semantically-rich representations in order to generate images with high-frequency details.

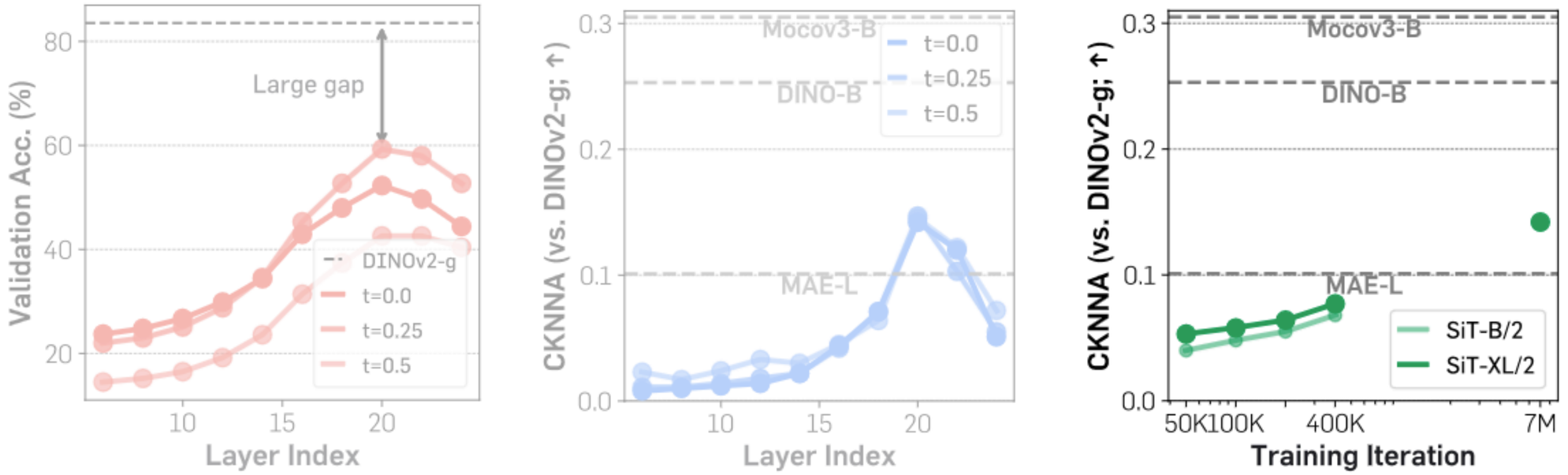
Observation 2: Diffusion *Does* Lead to Comparable Representations, but Alignment Remains Weak



(a) Semantic gap: Linear probing (b) Alignment to DINOv2-g (c) Alignment progression

Diffusion representations are already (weakly) aligned with other visual representations. In Figure 2b, we report representational alignments between SiT and DINOv2 using CKNNA. In particular, the SiT model representation already shows better alignment than MAE (He et al., 2022), which is also a self-supervised learning approach based on the reconstruction of masked patches. However, the absolute alignment score remains lower than that observed between other self-supervised learning methods (*e.g.*, MoCov3 (Chen et al., 2021) *vs.* DINOv2). These results suggest that while diffusion transformer representations exhibit some alignment with self-supervised visual representations, the alignment remains weak.

Observation 3: The Better the Model, the Better the Alignment



(a) Semantic gap: Linear probing (b) Alignment to DINOv2-g (c) Alignment progression

Alignment improves with larger models and extended training. We also measure CKNNA values across different model sizes and training iterations. As depicted in Figure 2c, we observe improved alignment with larger models and extended training. However, the absolute alignment remains low and does not reach the levels observed between other self-supervised visual encoders (*e.g.*, MoCov3 and DINOv2), even after extensive training of 7M iterations.

Idea: *Align* Features Between DINO and Diffusion Model

In particular, REPA achieves alignment through a maximization of patch-wise similarities between the pretrained representation $\mathbf{y}_*$ and the hidden state $\mathbf{h}_t$:

$$\mathcal{L}_{\text{REPA}}(\theta, \phi) := -\mathbb{E}_{\mathbf{x}_*, \boldsymbol{\epsilon}, t}\left[\frac{1}{N}\sum_{n=1}^{N} \text{sim}(\mathbf{y}_*^{[n]}, h_\phi(\mathbf{h}_t^{[n]}))\right], \tag{8}$$

where n is a patch index and $\text{sim}(\cdot, \cdot)$ is a pre-defined similarity function.

In practice, we add this term to the original diffusion-based objectives described in Section 2 and Appendix A. For instance, for the training of a velocity model in Eq. (4), the objective becomes:

$$\mathcal{L} := \mathcal{L}_{\text{velocity}} + \lambda \mathcal{L}_{\text{REPA}} \tag{9}$$

Idea: *Align* Features Between DINO and Diffusion Model

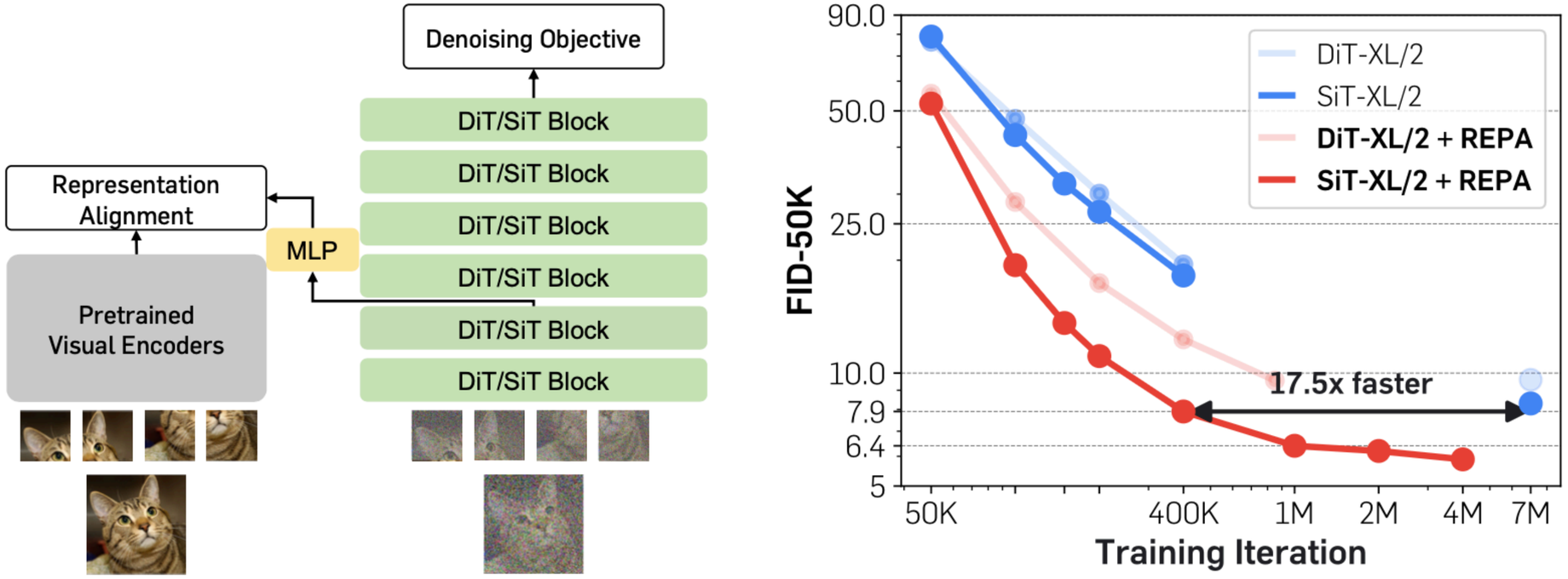
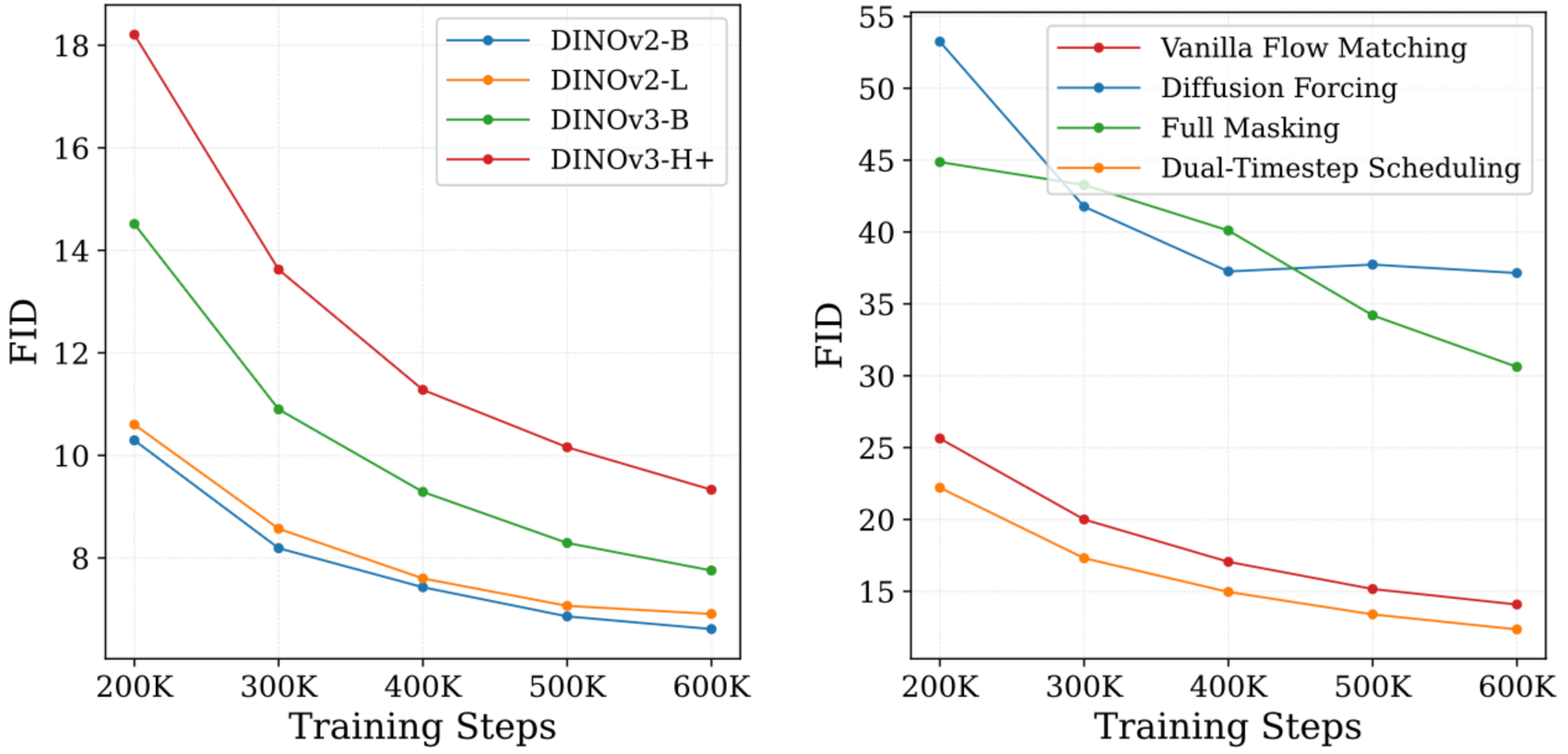


Figure 1: **Representation alignment makes diffusion transformer training significantly easier.** Our framework, REPA, explicitly aligns the diffusion model representation with powerful pretrained visual representation through a simple regularization. Notably, model training becomes significantly more efficient and effective, and achieves $>17.5\times$ faster convergence than the vanilla model.

Self-Supervised Flow Matching for Scalable Multi-Modal Synthesis

Hila Chefer [*1] **Patrick Esser** [*1]
Dominik Lorenz [1] **Dustin Podell** [1] **Vikash Raja** [1] **Vinh Tong** [1] **Antonio Torralba** [1 2] **Robin Rombach** [1]

https://bfl.ai/research/self-flow



(a) REPA scaling *(b)* Noise scheduler comparison

Figure 2. **Motivation.** (a) Scaling DINO (v2-B< v2-L< v3-H+) paradoxically degrades the generations quality using REPA. (b) Diffusion forcing and full masking create a train-inference gap which degrades generations. Our Dual-Timestep Scheduling improves generation even without a self-supervised objective.

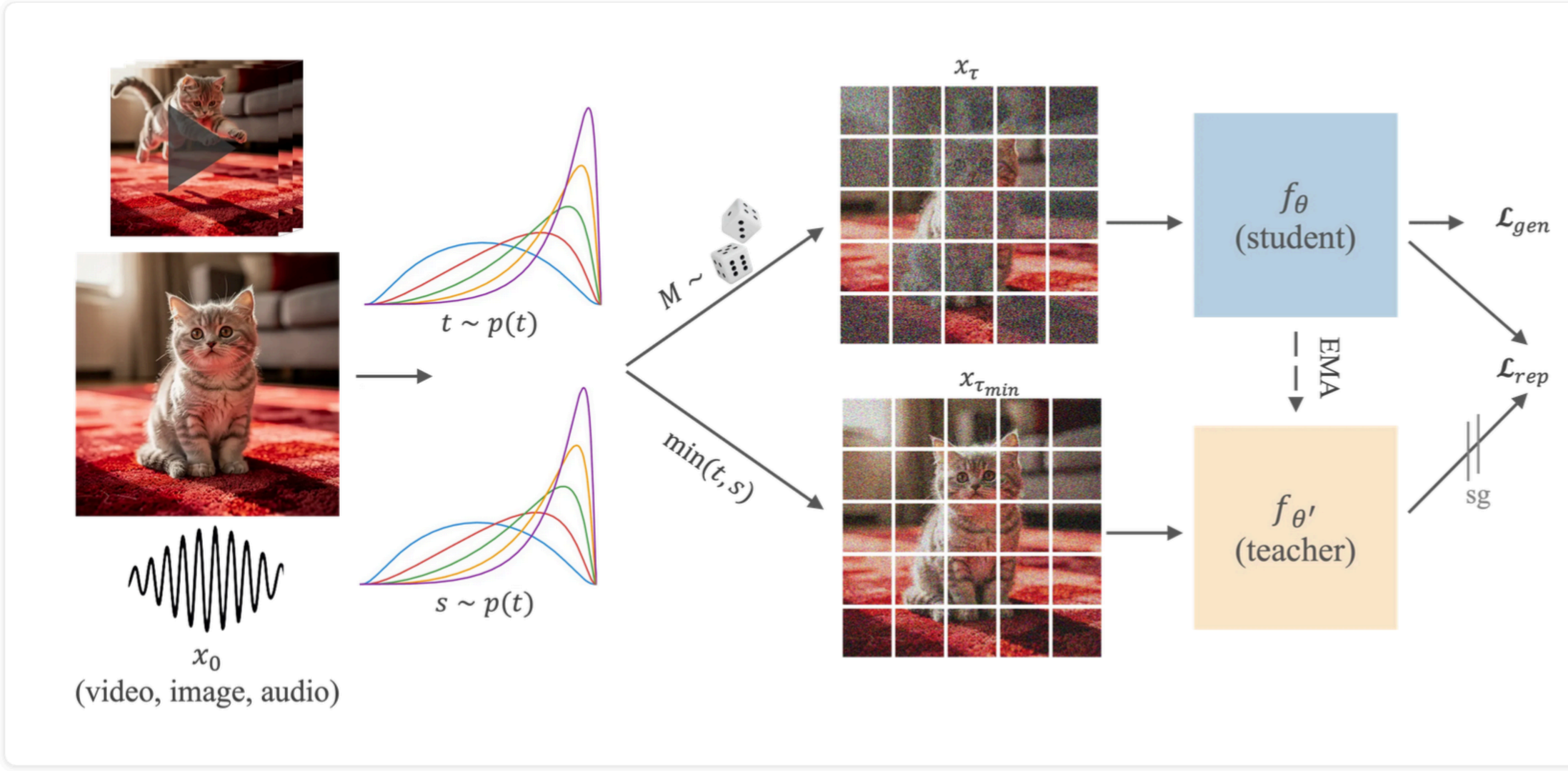


Illustration of our method. Given a clean input x_0, we draw two timesteps t, s, and a random mask M, then noise each token according to its assigned timestep. The teacher input is noised with $\tau_{\text{min}}=\min\{t,s\}$, creating an information asymmetry compared to the student. The student is trained to simultaneously denoise the input and reconstruct the teacher's features given its mixed-noised view.

High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach[1] * Andreas Blattmann[1] * Dominik Lorenz[1] Patrick Esser[R] Björn Ommer[1]

[1]Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany [R]Runway ML

https://github.com/CompVis/latent-diffusion

Problem: Images very high-resolution.
That makes pixel-space diffusion expensive.

Solution: Pre-train a down-sampling VAE with perceptual & patch-based adversarial loss.

Pre-trained VAE can be seen as a
tokenizer, just as tokenizers in NLP.

High-Resolution Image Synthesis with Latent Diffusion Models

Robin Rombach[1] * Andreas Blattmann[1] * Dominik Lorenz[1] Patrick Esser[R] Björn Ommer[1]

[1]Ludwig Maximilian University of Munich & IWR, Heidelberg University, Germany [R]Runway ML

https://github.com/CompVis/latent-diffusion



"Since diffusion models offer excellent inductive biases for spatial data, we do not need the heavy spatial downsampling of related generative models in latent space, but can still greatly reduce the dimensionality of the data via suitable autoencoding models"

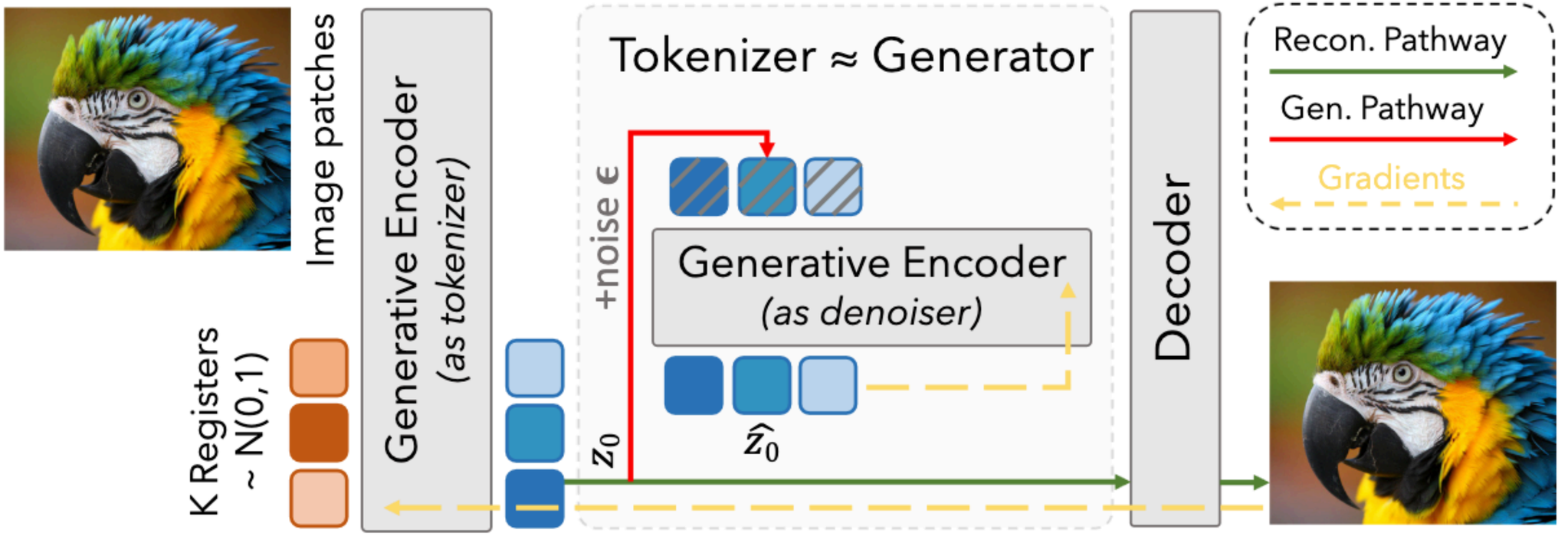
End-to-End Training for Unified Tokenization and Latent Denoising

Shivam Duggal [1 *] **Xingjian Bai** [1 *] **Zongze Wu** [2] **Richard Zhang** [2] **Eli Shechtman** [2]
Antonio Torralba [1] **Phillip Isola** [1] **William T. Freeman** [1]

> Latent diffusion models (LDMs) enable high-fidelity synthesis by operating in learned latent spaces. However, training state-of-the-art LDMs requires complex staging: a tokenizer must be trained first, before the diffusion model can be trained in the frozen latent space.

End-to-End Training for Unified Tokenization and Latent Denoising

Shivam Duggal [1 *] **Xingjian Bai** [1 *] **Zongze Wu** [2] **Richard Zhang** [2] **Eli Shechtman** [2]
Antonio Torralba [1] **Phillip Isola** [1] **William T. Freeman** [1]



Emerging Alternatives to Diffusion

Mean Flows for One-step Generative Modeling

Zhengyang Geng[1*] **Mingyang Deng**[2] **Xingjian Bai**[2] **J. Zico Kolter**[1] **Kaiming He**[2]

[1]CMU [2]MIT

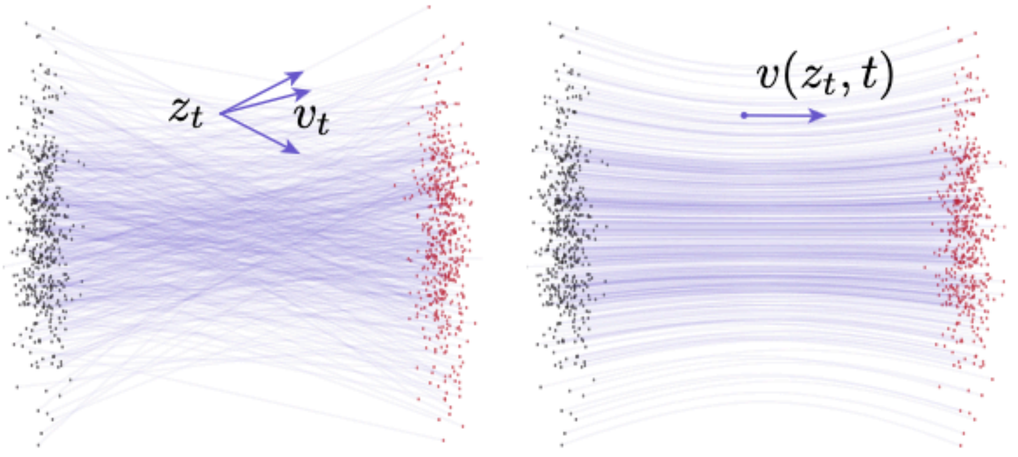


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal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

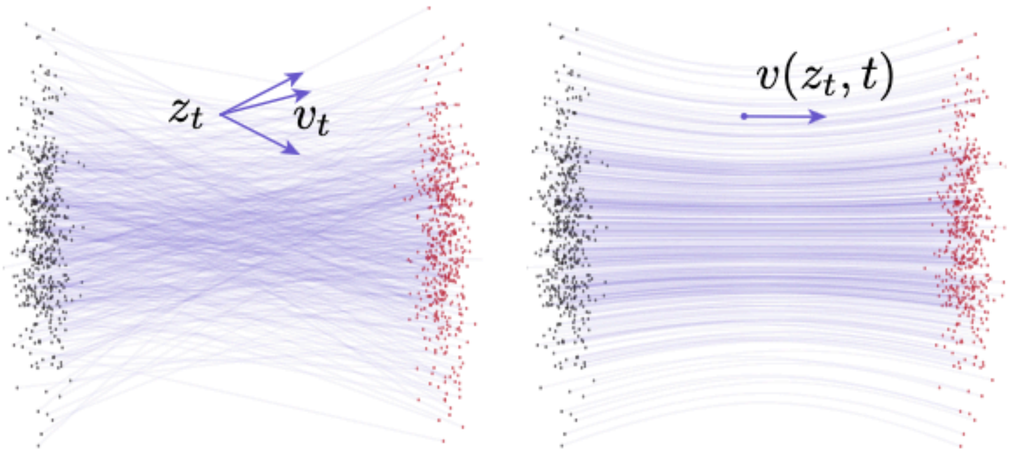


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal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

Because a given z_t and its v_t can arise from different x and ϵ, Flow Matching essentially models the expectation over all possibilities, called the *marginal* velocity [28] (Fig. 2 right):

$$v(z_t, t) \triangleq \mathbb{E}_{p_t(v_t|z_t)}[v_t]. \tag{1}$$

A neural network v_θ parameterized by θ is learned to fit the marginal velocity field: $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t,p_t(z_t)}\|v_\theta(z_t, t) - v(z_t, t)\|^2$.

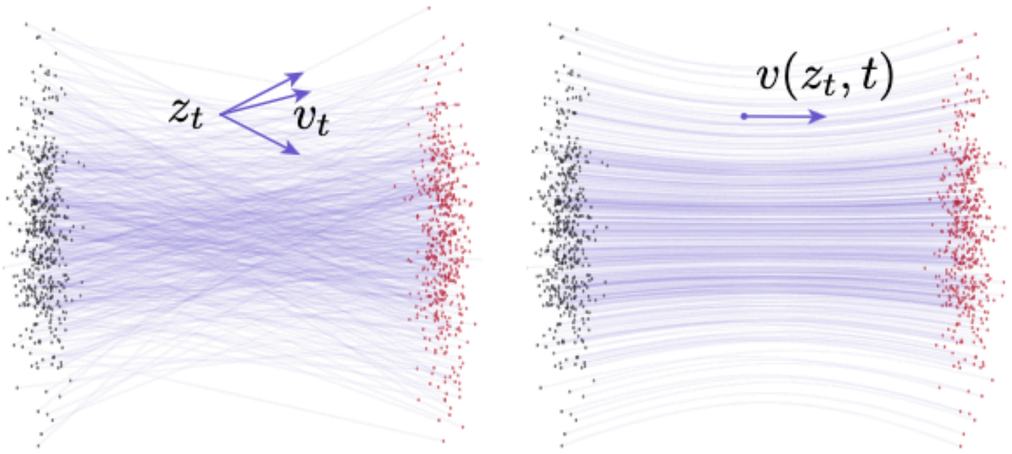


Figure 2: **Velocity fields in Flow Matching** [28]. **Left**: *conditional* flows [28]. A given z_t can arise from different (x, ϵ) pairs, resulting in different conditional velocities v_t. **Right**: *marginal* flows [28], obtained by marginalizing over all possible conditional velocities. The marginal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

Question of this (and many other recent) paper(s):

Are there any *other* flow fields that map noise to data that we can approximate with our neural net?

Average Velocity. We define average velocity as the displacement between two time steps t and r (obtained by integration) divided by the time interval. Formally, the average velocity u is:

$$u(z_t, r, t) \triangleq \frac{1}{t-r} \int_r^t v(z_\tau, \tau) d\tau. \tag{3}$$

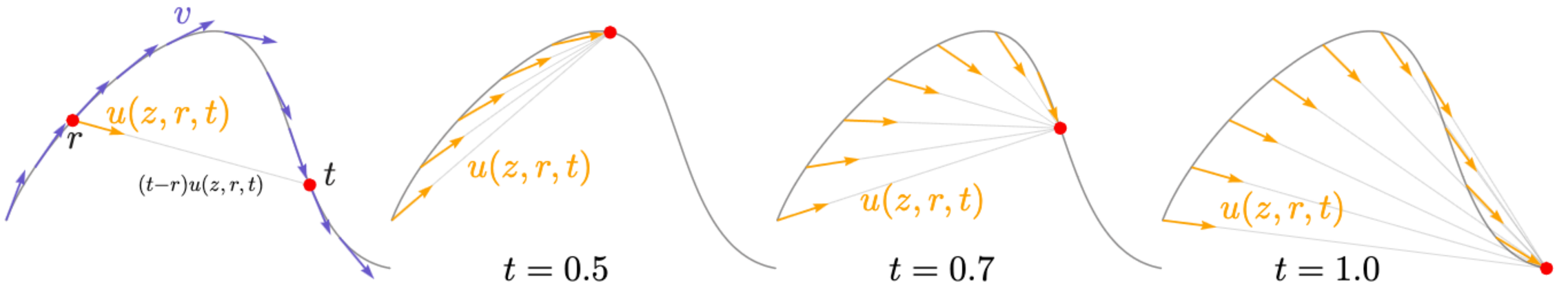


Figure 3: **The field of *average velocity*** $u(z, r, t)$. **Leftmost**: While the *instantaneous velocity* v determines the tangent direction of the path, the average velocity $u(z, r, t)$, defined in Eq. (3), is generally not aligned with v. The average velocity is aligned with the *displacement*, which is $(t - r)u(z, r, t)$. **Right three subplots**: The field $u(z, r, t)$ is conditioned on both r and t, and is shown here for $t = 0.5$, 0.7, and 1.0.

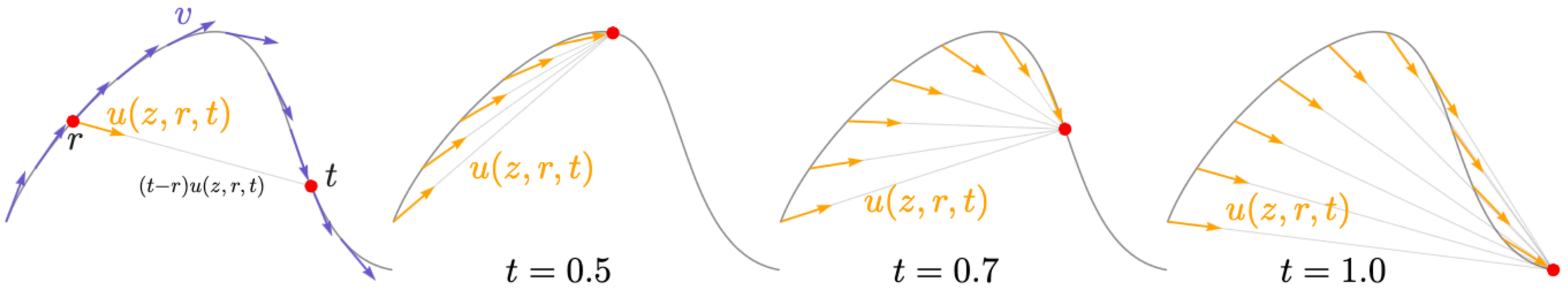


Figure 3: **The field of *average velocity*** $u(z, r, t)$. **Leftmost**: While the *instantaneous velocity* v determines the tangent direction of the path, the average velocity $u(z, r, t)$, defined in Eq. (3), is generally not aligned with v. The average velocity is aligned with the *displacement*, which is $(t - r)u(z, r, t)$. **Right three subplots**: The field $u(z, r, t)$ is conditioned on both r and t, and is shown here for $t = 0.5, 0.7$, and 1.0.

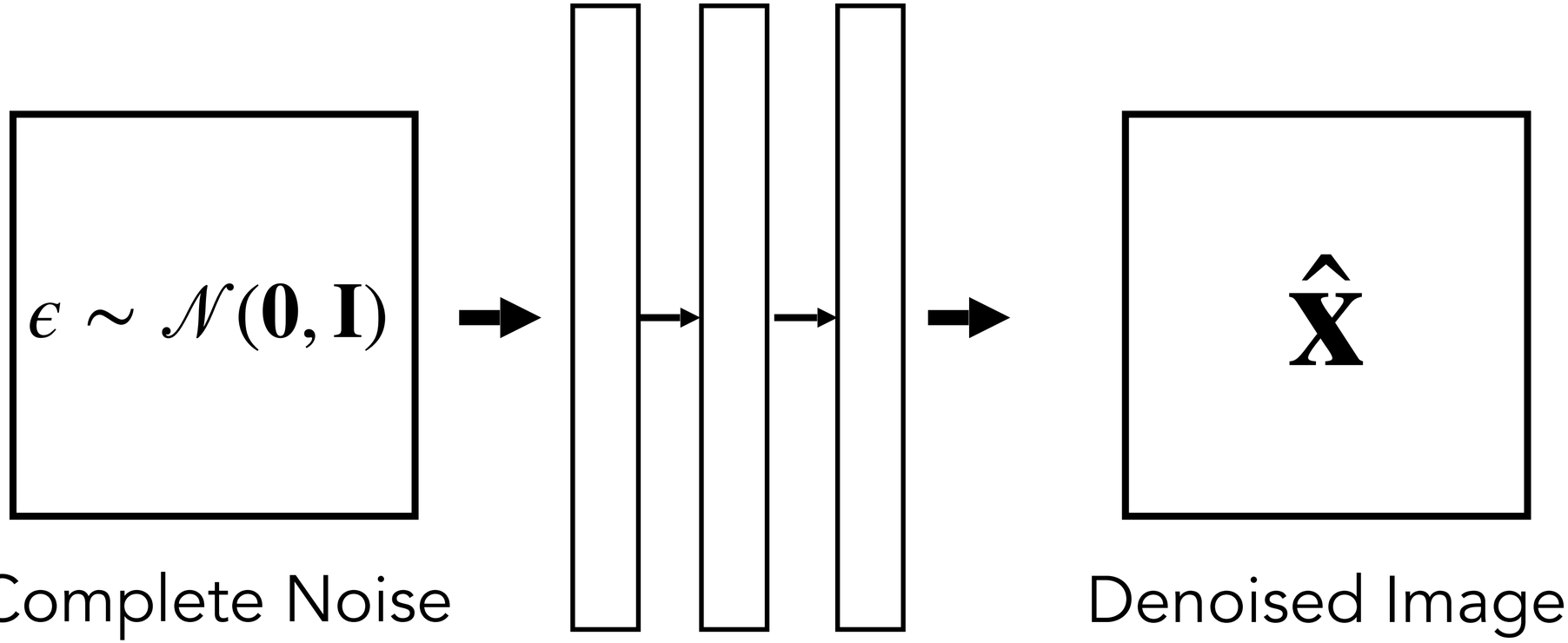
Problem: in contrast to instantaneous velocity field, this field is not tractable from the data directly…

Alas not enough time: pls refer to paper how we actually *train* the neural network to approximate this field.

Generative Modeling via Drifting

Mingyang Deng [1] **He Li** [1] **Tianhong Li** [1] **Yilun Du** [2] **Kaiming He** [1]

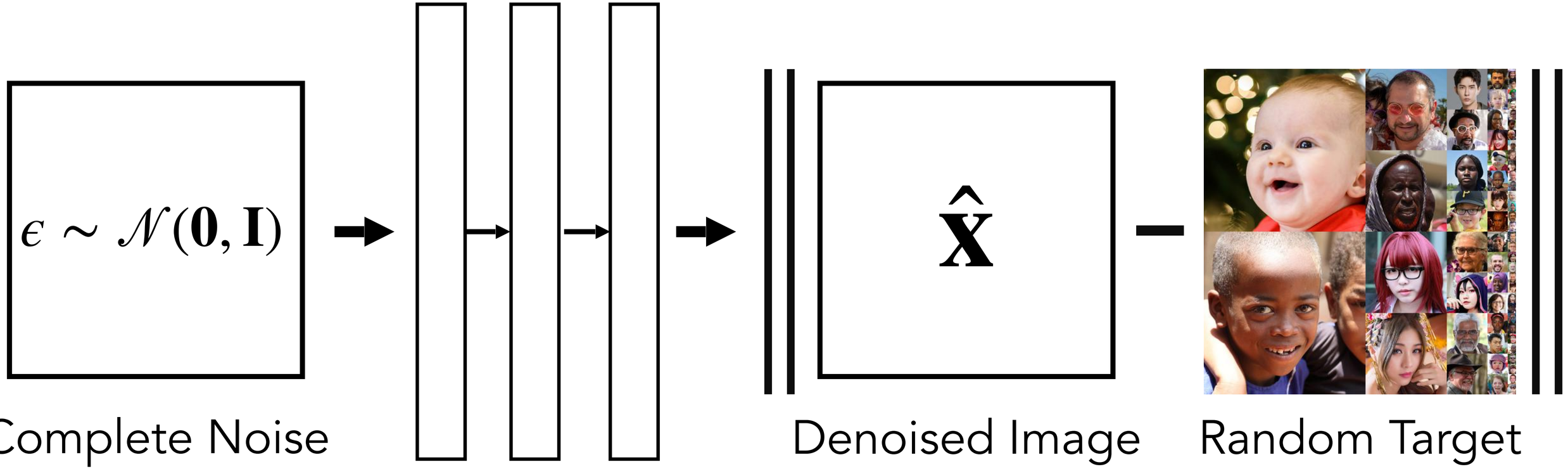
A naive but valuable perspective on GenAI



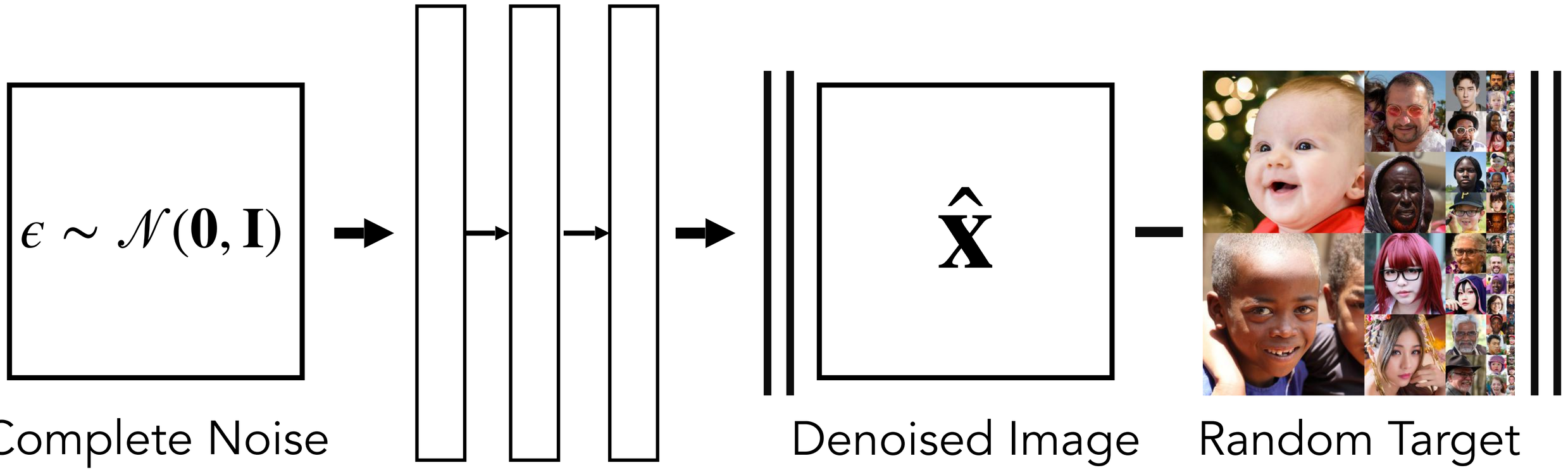
Complete Noise

Denoised Image

A naive but valuable perspective on GenAI

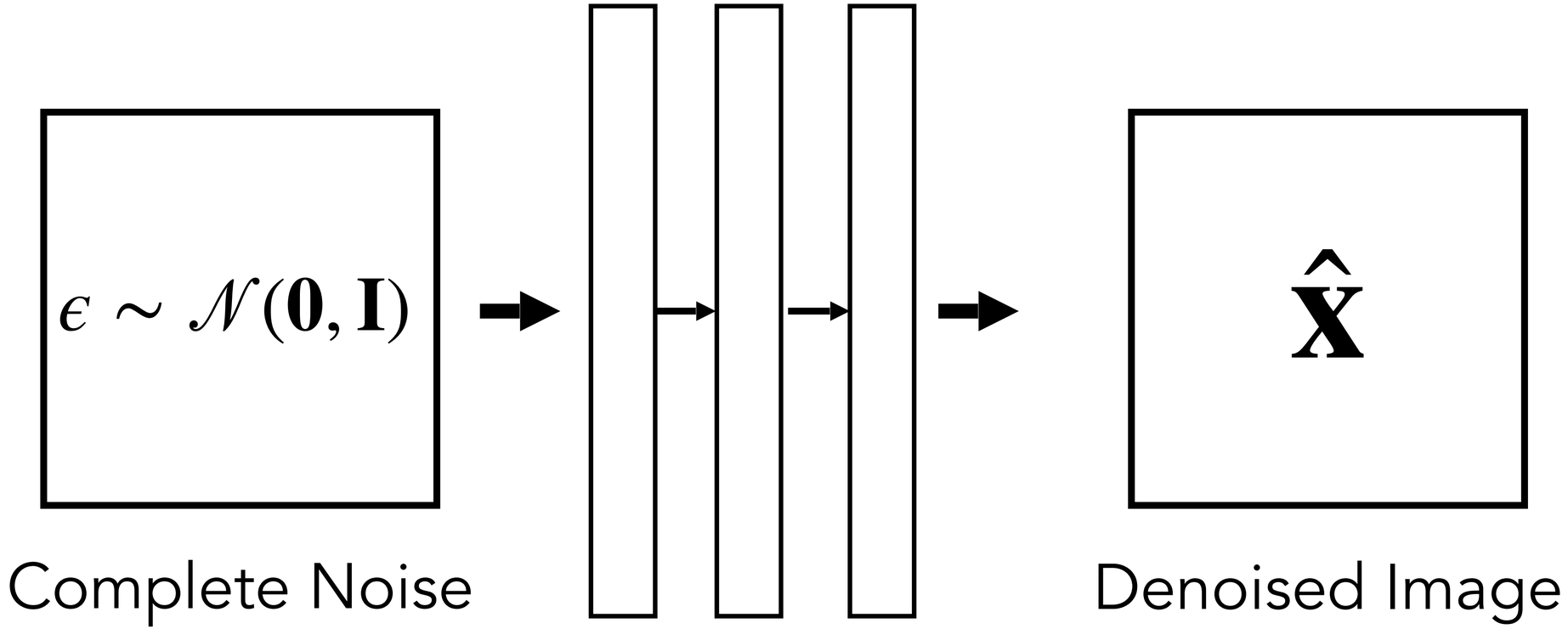


A naive but valuable perspective on GenAI

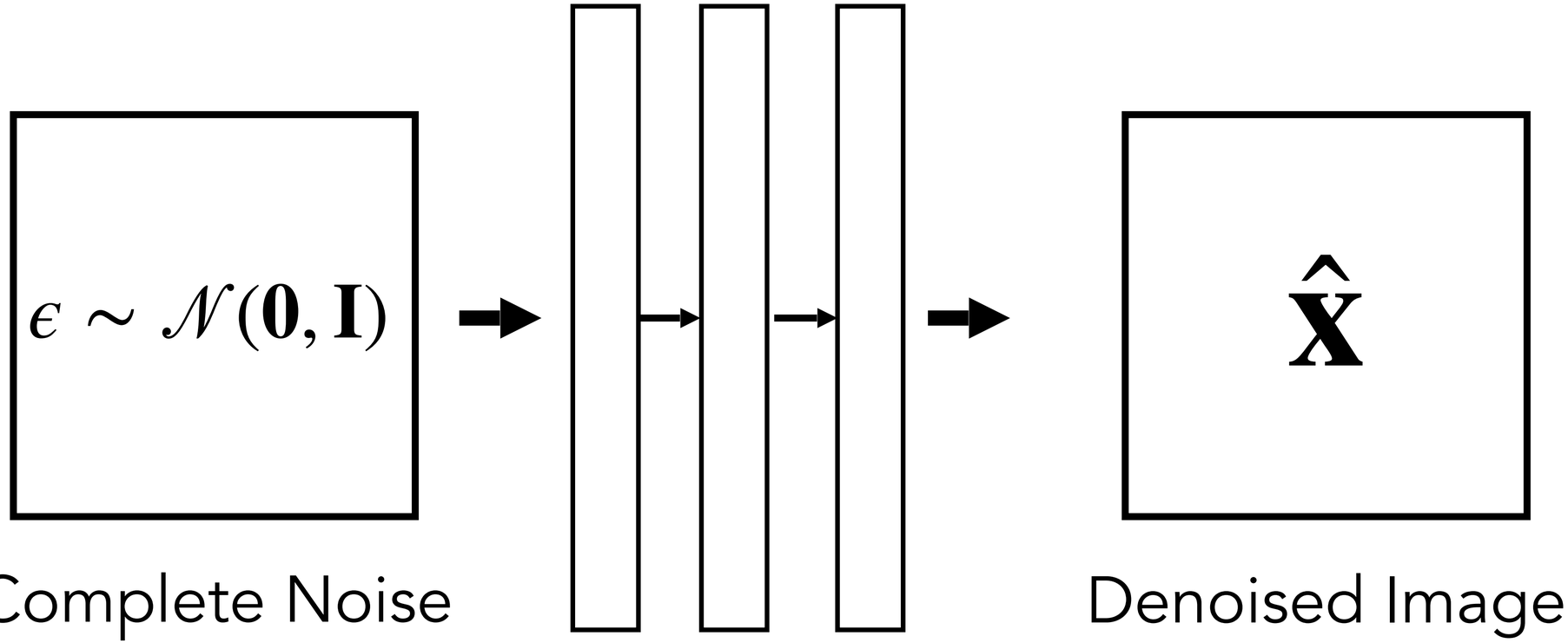


What will happen?

A naive but valuable perspective on GenAI

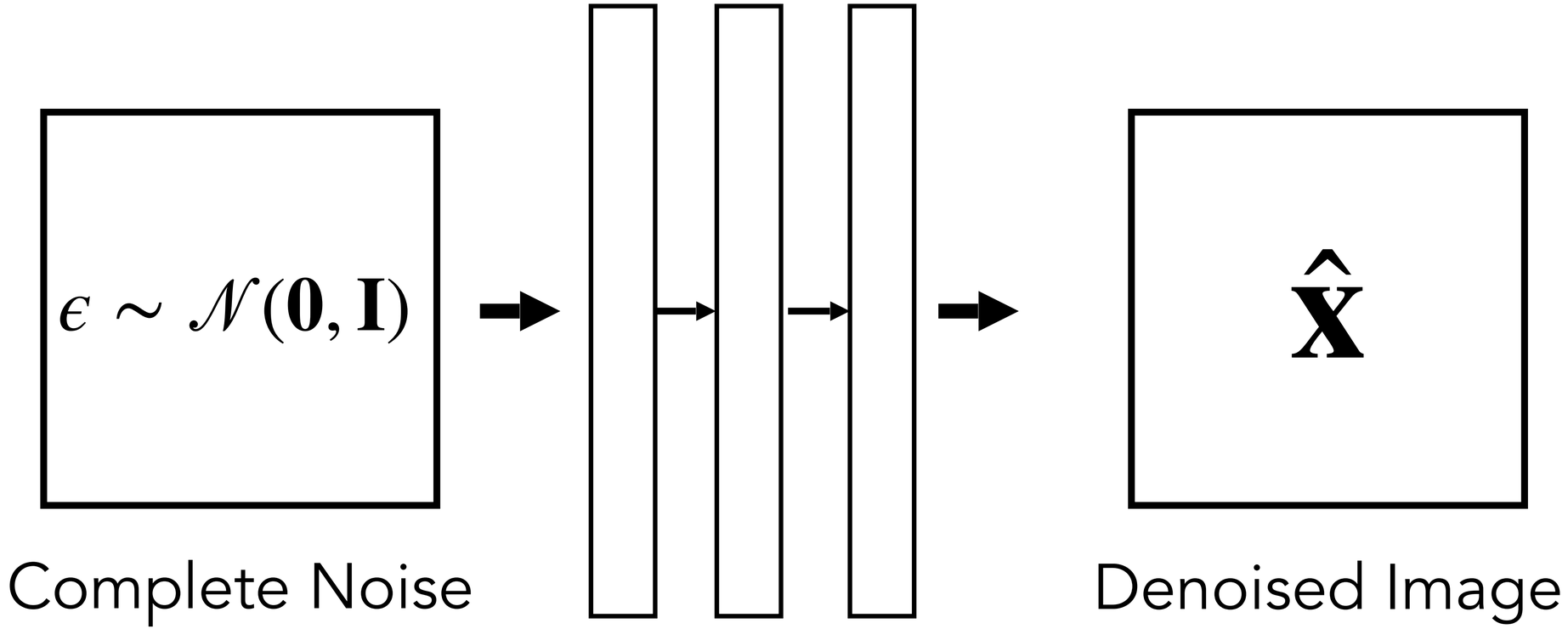


A naive but valuable perspective on GenAI

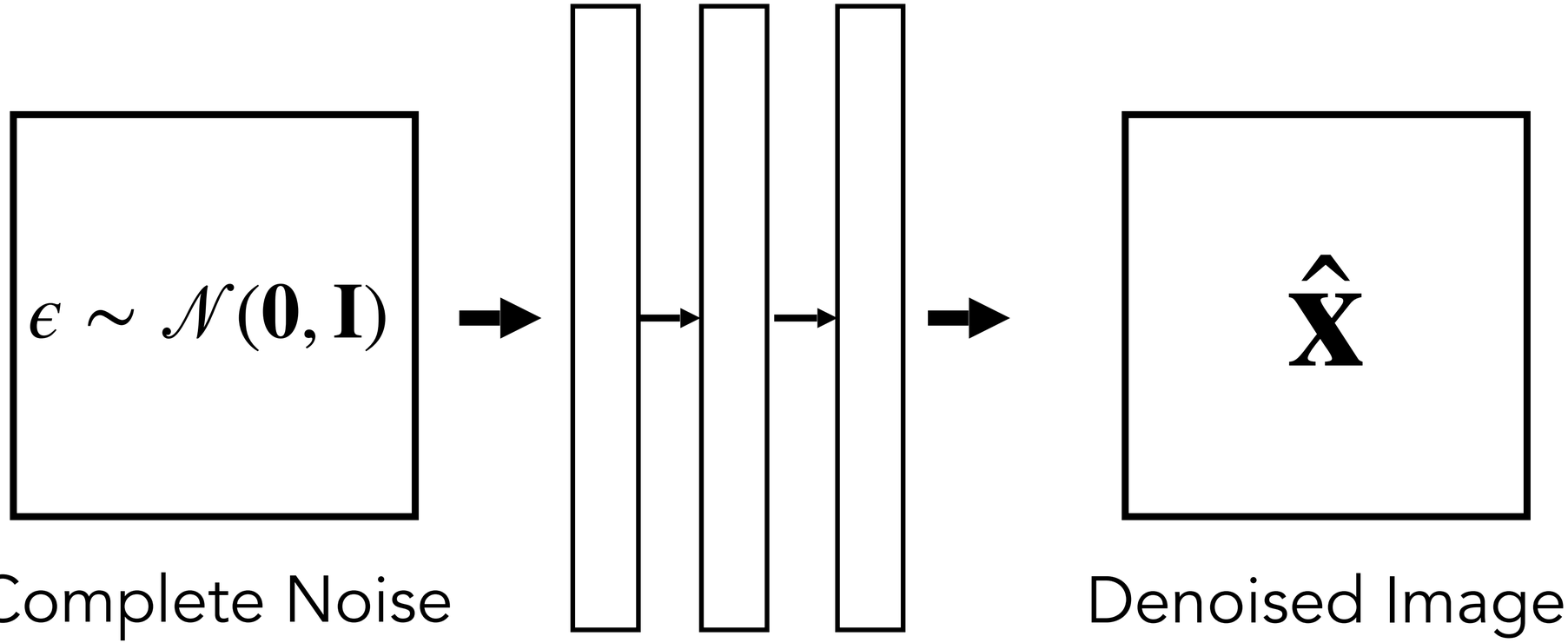


Intuition: For any generative model of this form,
when training starts, the model is *bad*.
Only throughout training does the map from noise to image become good.

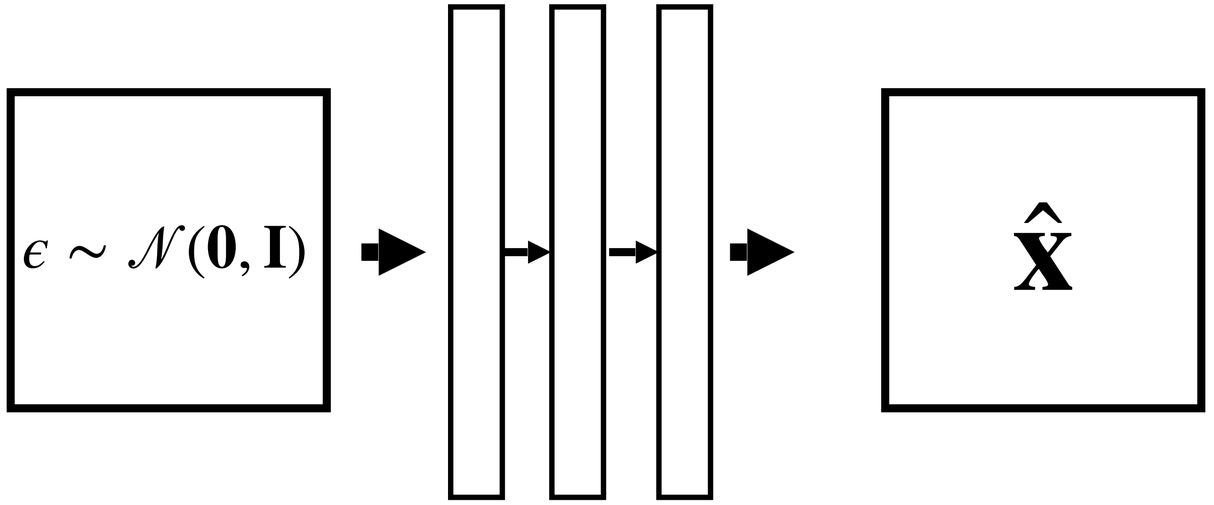
A naive but valuable perspective on GenAI

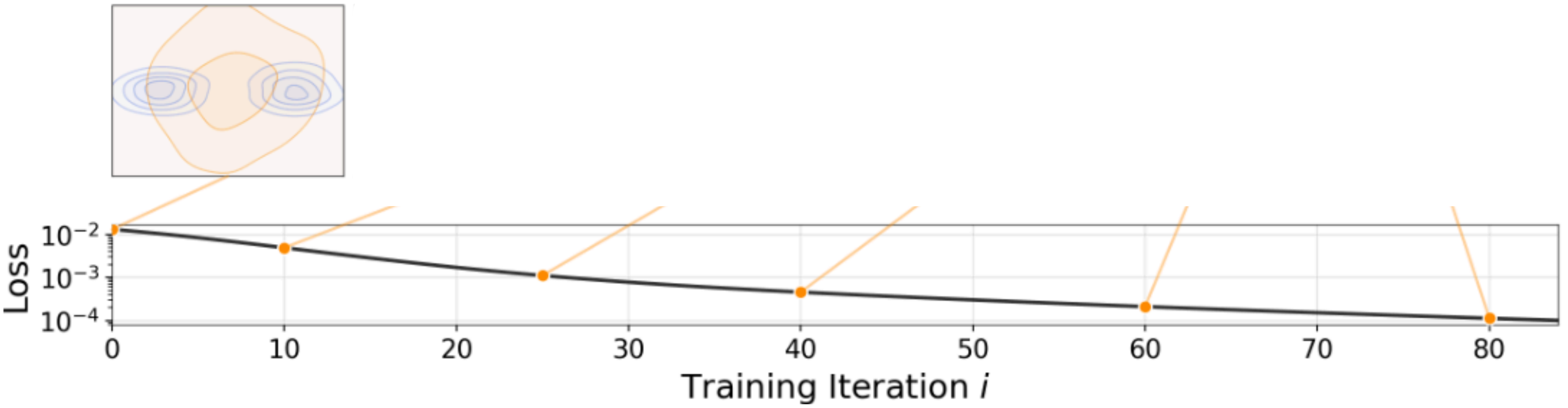


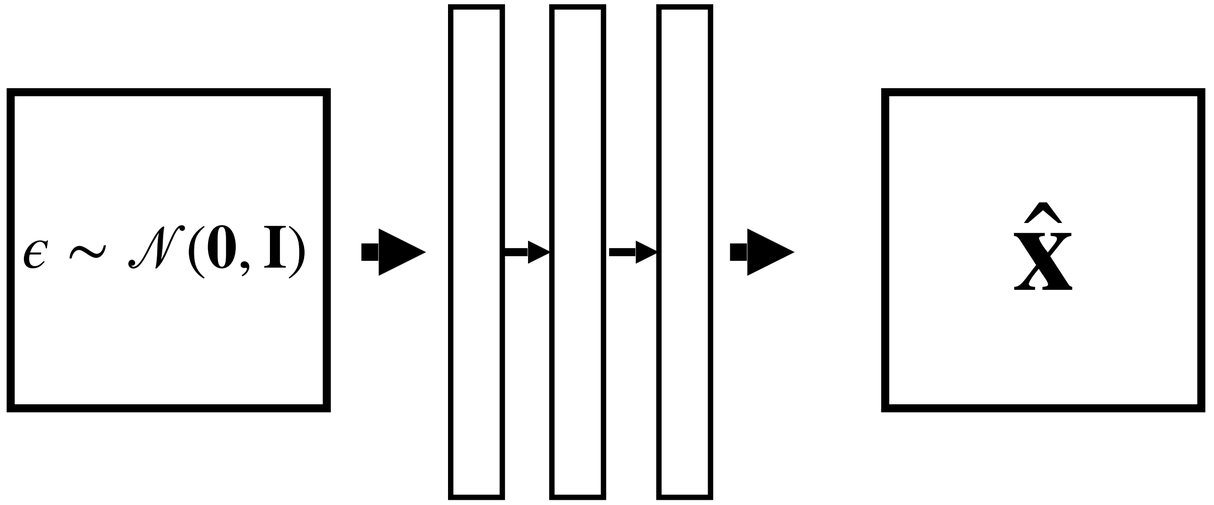
A naive but valuable perspective on GenAI

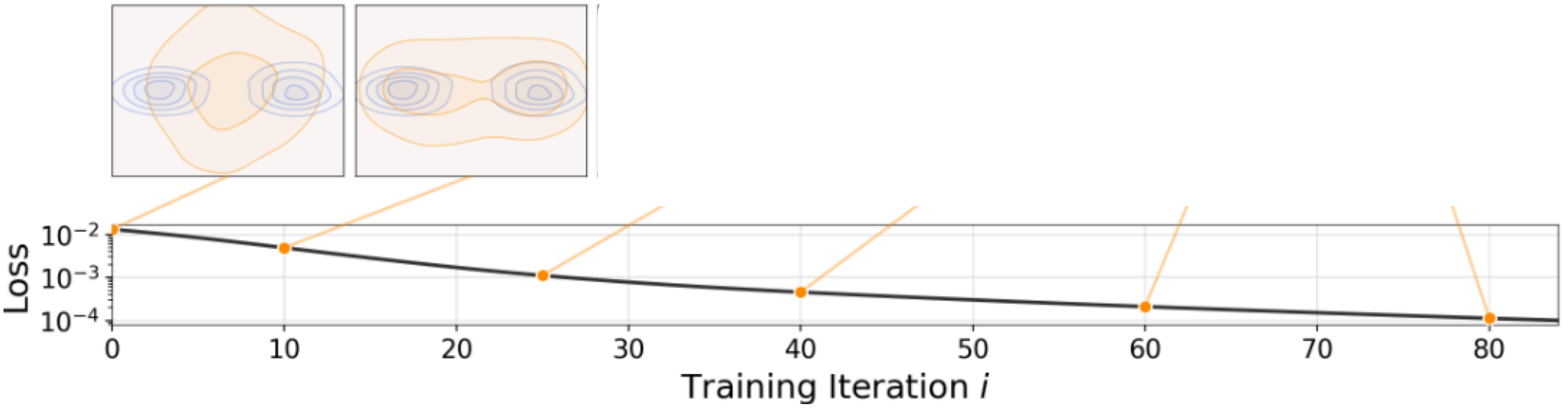


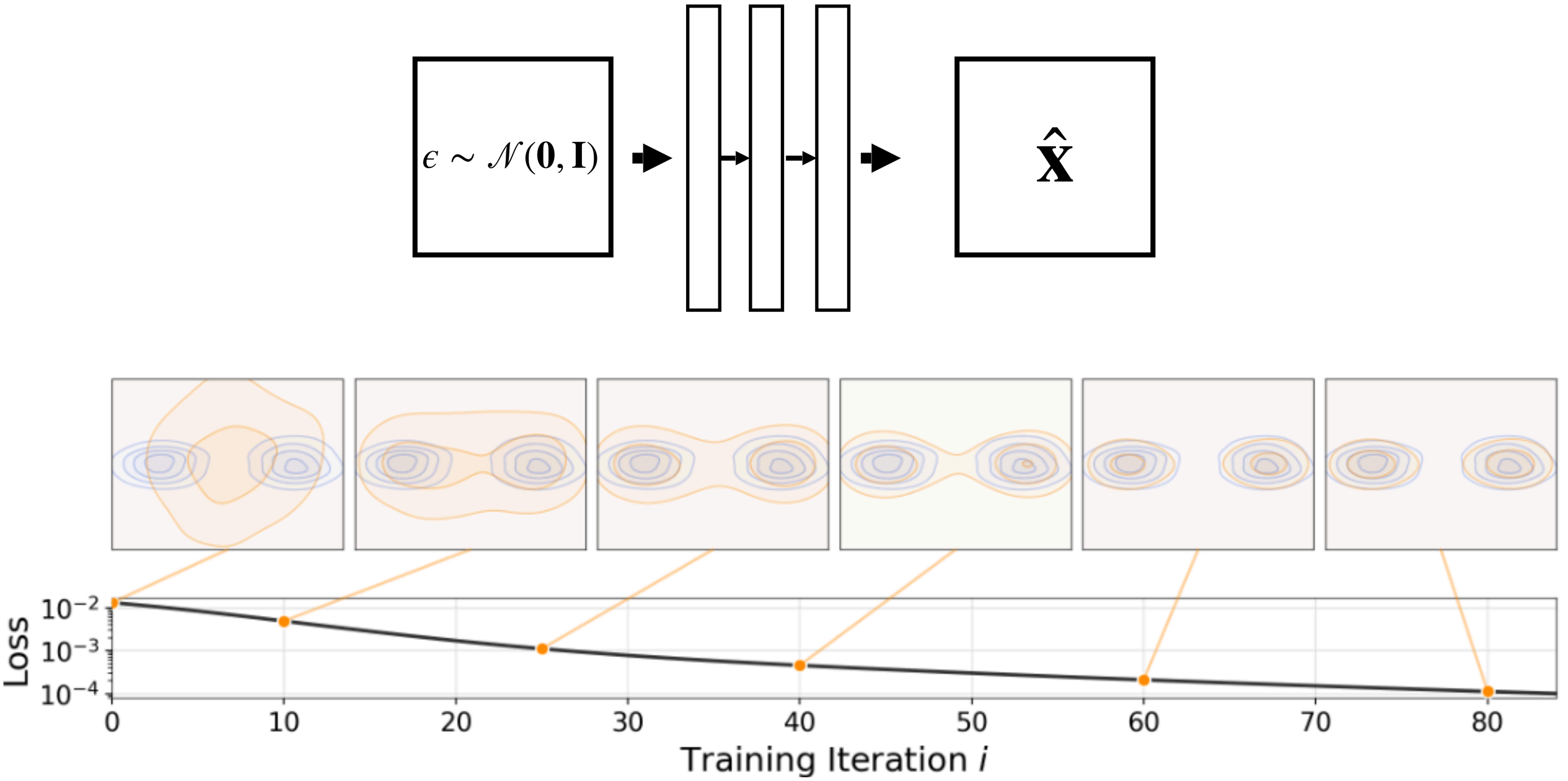
Idea: let's build an objective that tries to improve (*drift)* the initially bad samples of the model a little closer to good images at every training step!

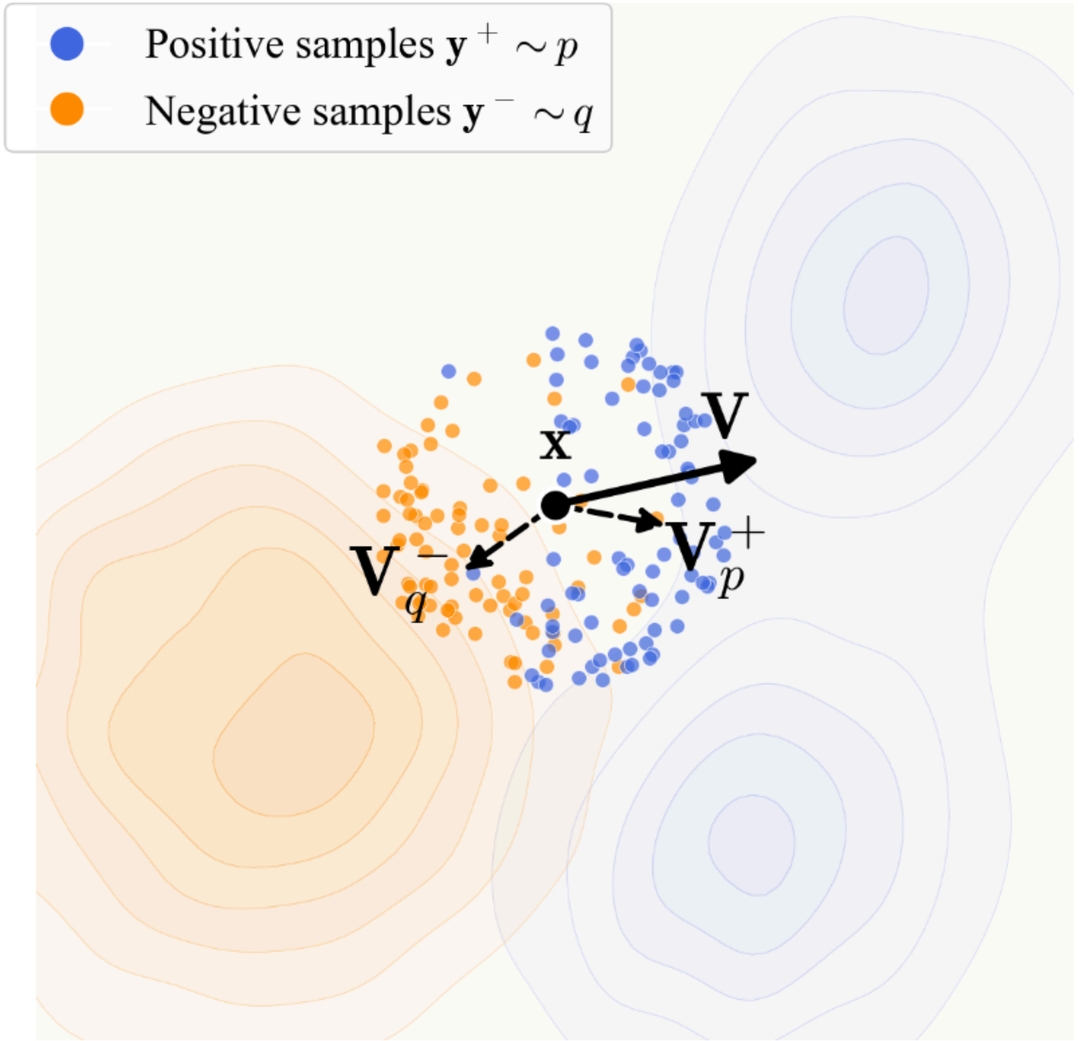

$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\hat{\mathbf{x}}$

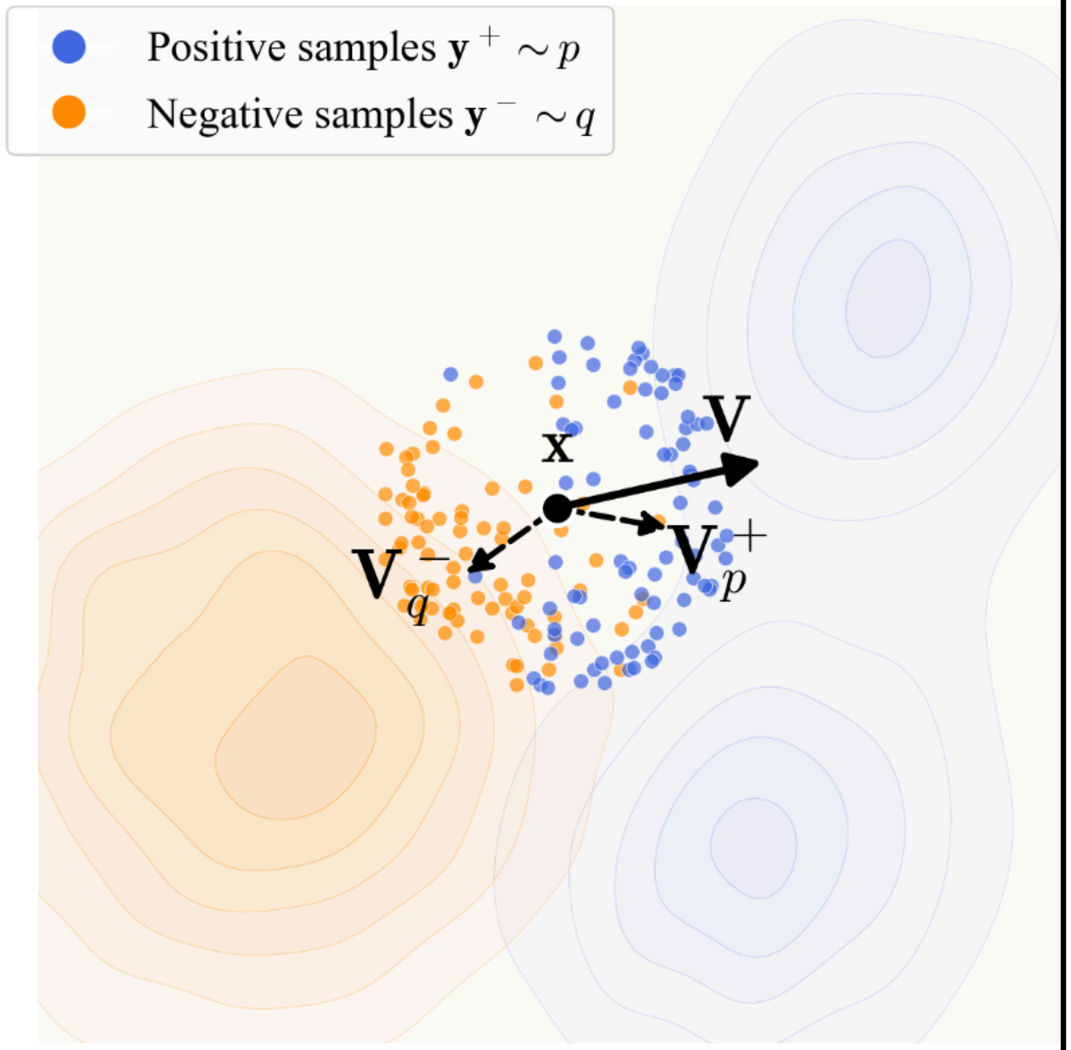

Loss
10^{-2}
10^{-3}
10^{-4}
0
10
20
30
40
50
60
70
80
Training Iteration i


$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\hat{\mathbf{x}}$


Loss
10^{-2}
10^{-3}
10^{-4}
0
10
20
30
40
50
60
70
80
Training Iteration i


$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
$\hat{\mathbf{x}}$
Loss
10^{-2}
10^{-3}
10^{-4}
0
10
20
30
40
50
60
70
80
Training Iteration i


Positive samples $\mathbf{y}^+ \sim p$
Negative samples $\mathbf{y}^- \sim q$
$\mathbf{x}$
$\mathbf{V}$
$\mathbf{V}_p^+$
$\mathbf{V}_q^-$

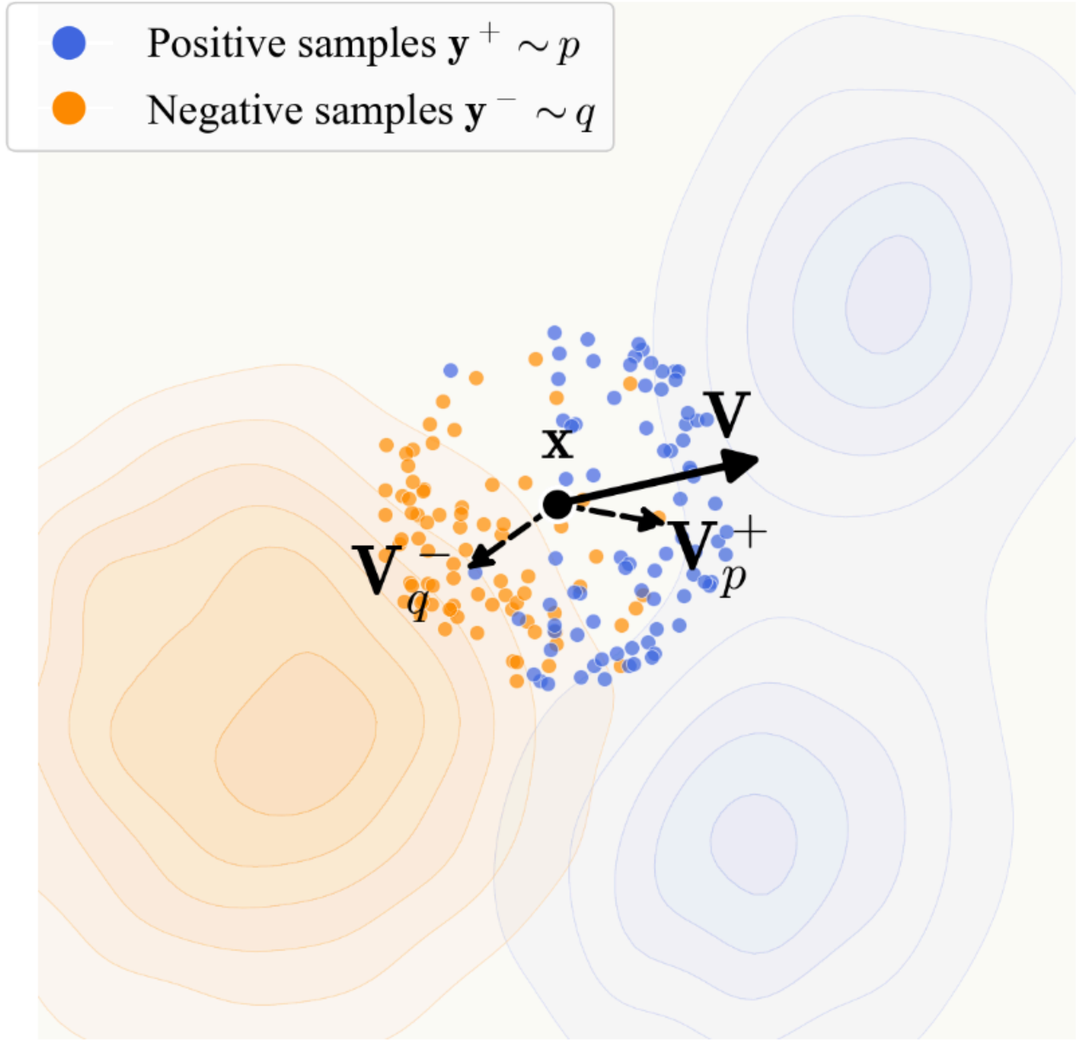


Core idea: real images are “positive samples”.

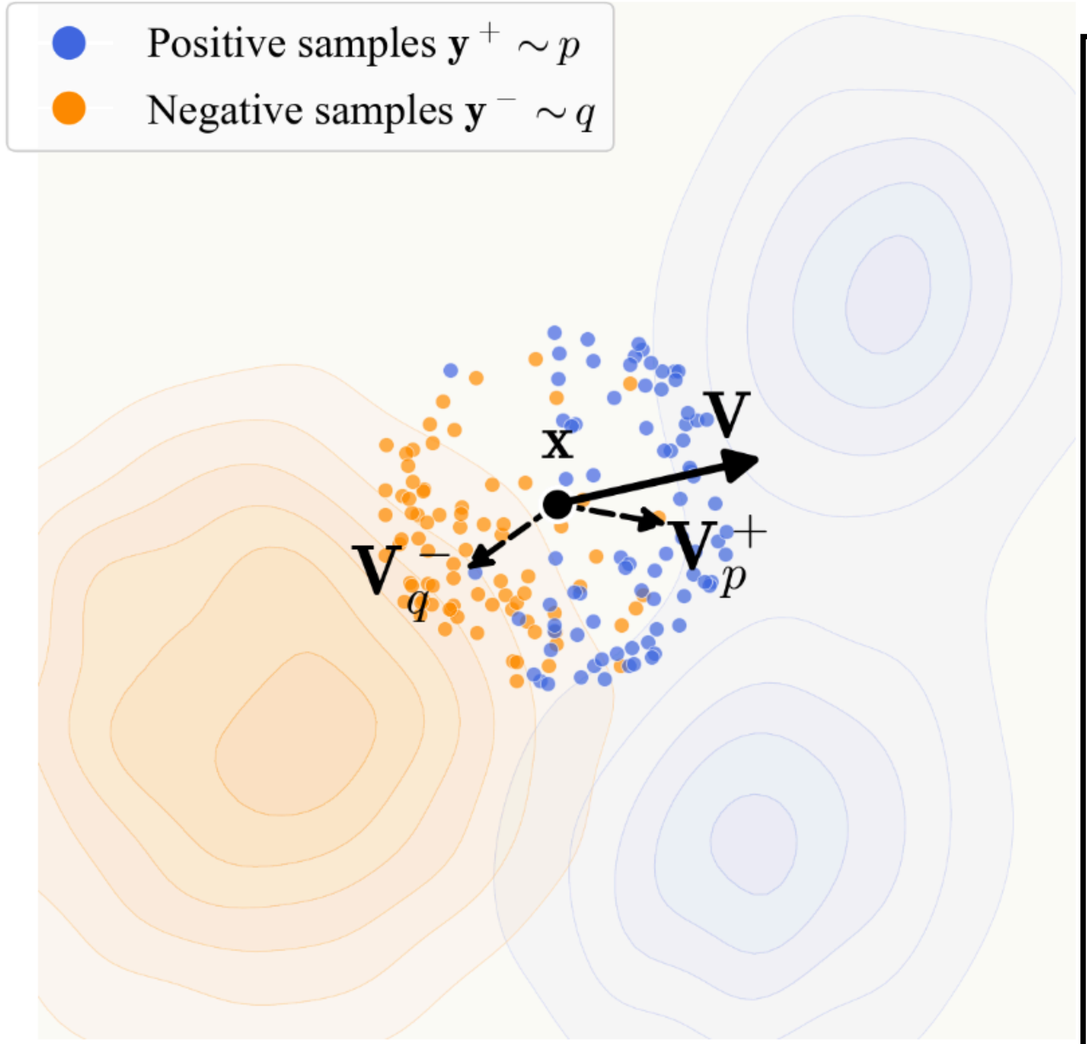
Current samples are “negative samples”.

At every training step, say “each sample in the current batch should move closer to the closest image in a test batch” while being “pushed away” from itself and other samples

$$\mathbf{V}_{p,q}(\mathbf{x}) = \frac{1}{Z_p Z_q} \mathbb{E}_{p,q}\left[k(\mathbf{x}, \mathbf{y}^+)k(\mathbf{x}, \mathbf{y}^-)(\mathbf{y}^+ - \mathbf{y}^-)\right].$$



$$\mathbf{V}_{p,q}(\mathbf{x}) = \frac{1}{Z_p Z_q} \mathbb{E}_{p,q}\left[k(\mathbf{x}, \mathbf{y}^+)k(\mathbf{x}, \mathbf{y}^-)(\mathbf{y}^+ - \mathbf{y}^-)\right].$$

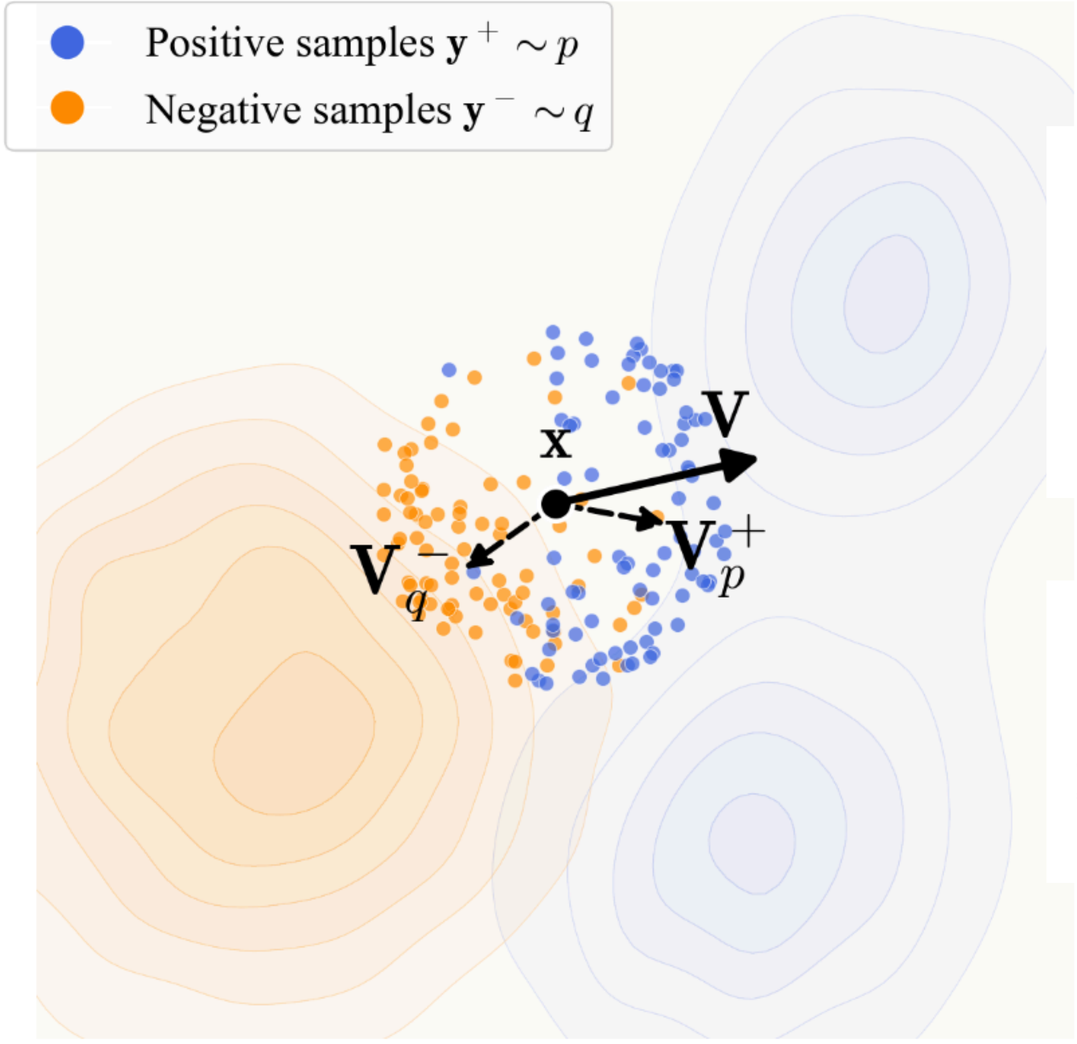


Core idea: real images are “positive samples”.

Current samples are “negative samples”.

At every training step, say “each sample in the current batch should move closer to the closest image in a test batch” while being “pushed away” from itself and other samples

$$\mathbf{V}_{p,q}(\mathbf{x}) = \frac{1}{Z_p Z_q} \mathbb{E}_{p,q}\left[k(\mathbf{x}, \mathbf{y}^+)k(\mathbf{x}, \mathbf{y}^-)(\mathbf{y}^+ - \mathbf{y}^-)\right].$$



Kernel. The kernel $k(\cdot, \cdot)$ can be a function that measures the similarity. In this paper, we adopt:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{1}{\tau}\|\mathbf{x} - \mathbf{y}\|\right), \tag{12}$$

In practice, we implement $\tilde{k}$ using a *softmax* operation, with logits given by $-\frac{1}{\tau}\|\mathbf{x}-\mathbf{y}\|$, where the softmax is taken over $\mathbf{y}$. This softmax operation is similar to that of InfoNCE (Oord et al., 2018) in contrastive learning.