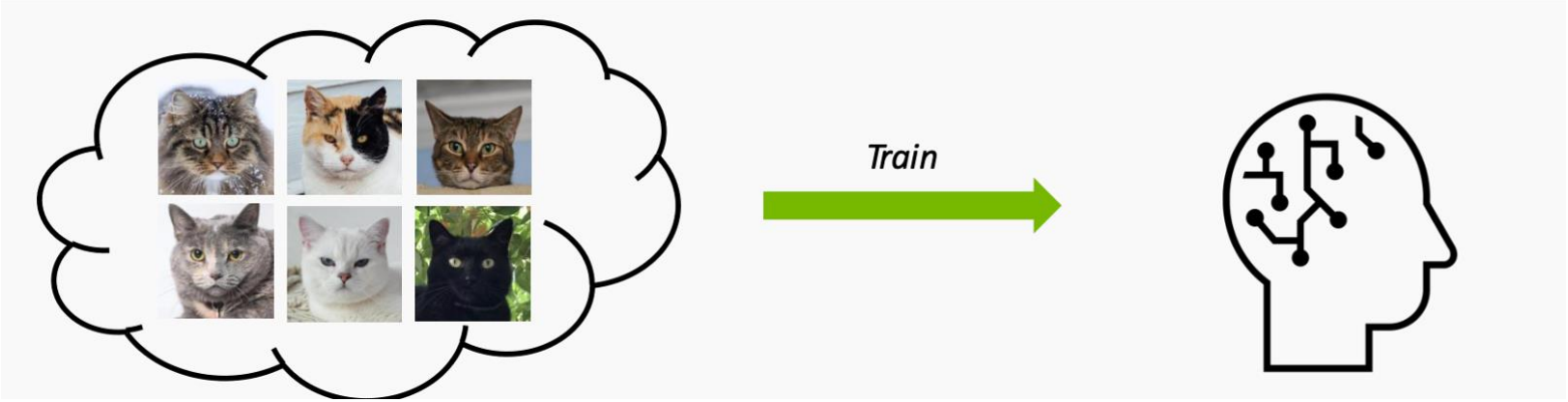
Image to 3D

Shibo Wang

- 2D Diffusion Generative Model
- 3D Generative Model
- Demo Code

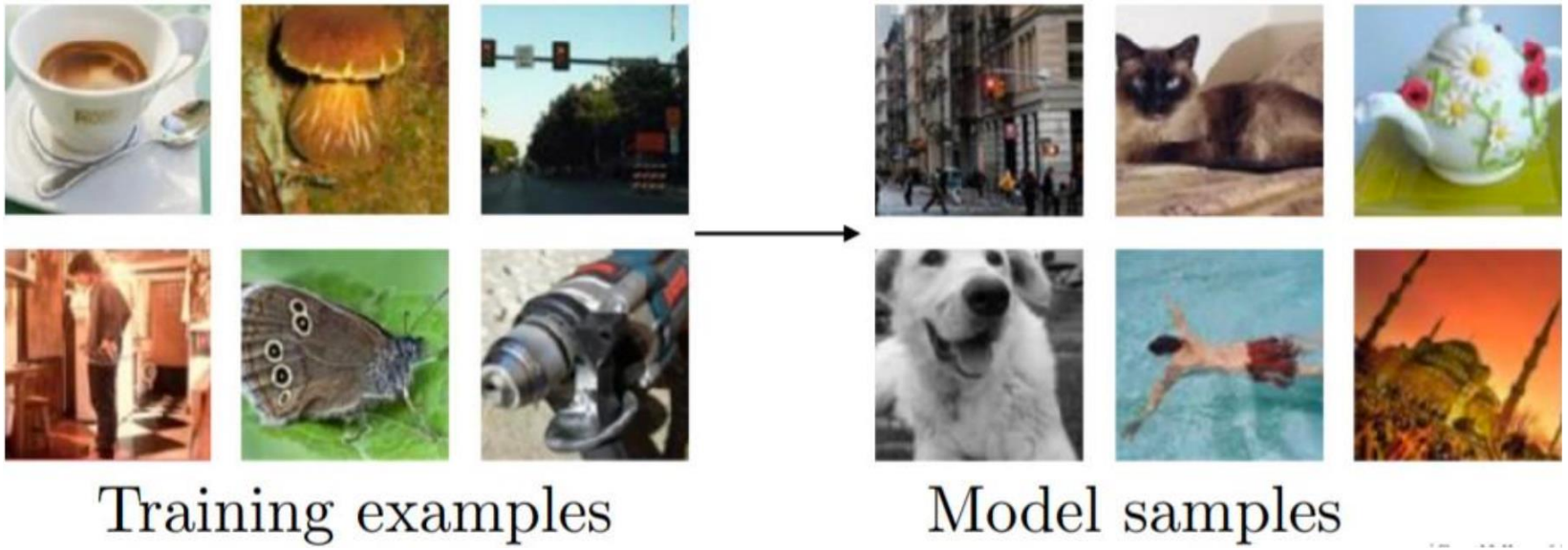
Generative Model

Generative models refer to learn a model to represent distribution p_{model} from a training dataset which is drawn from p_{data}.

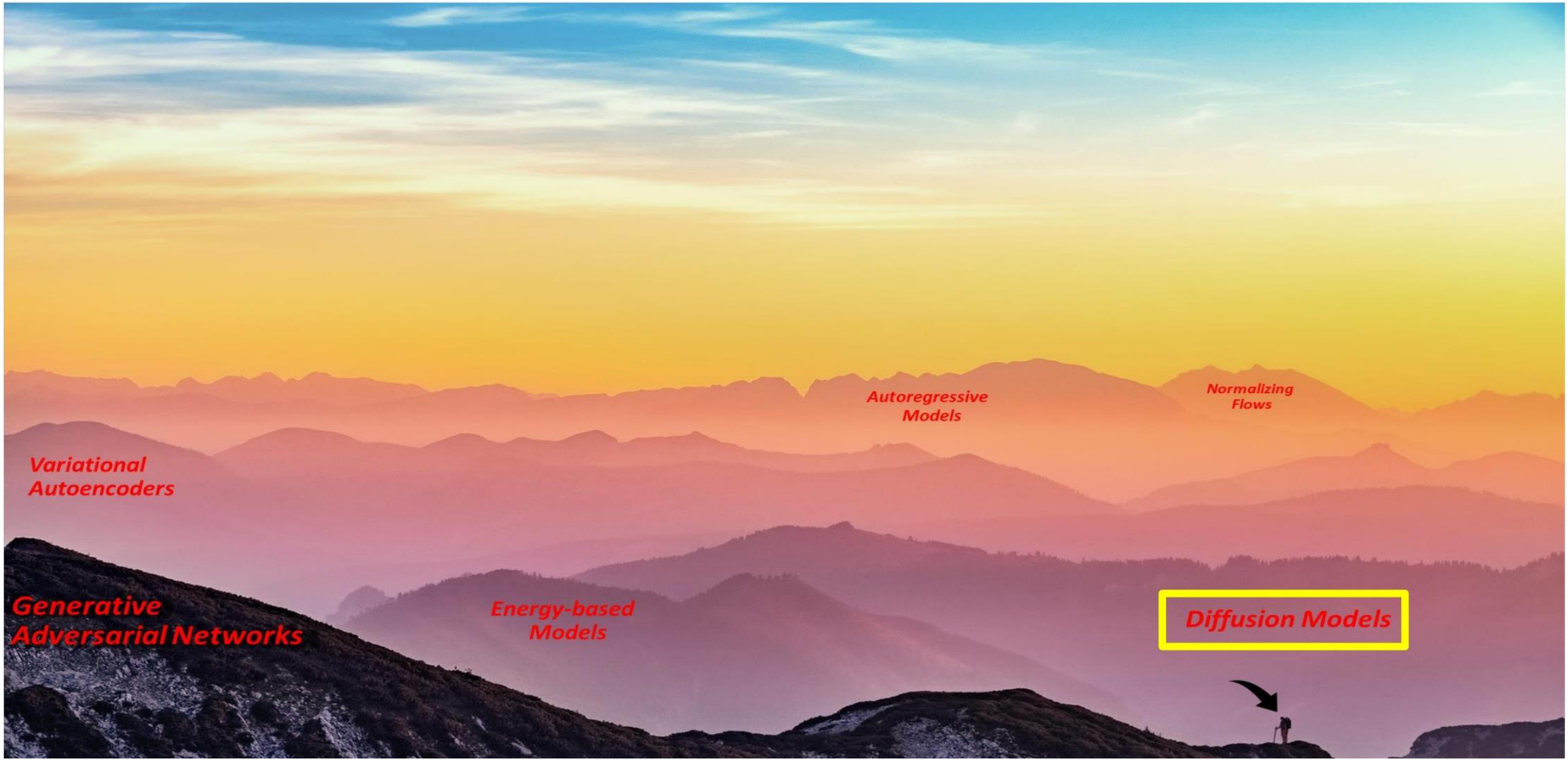


Generative Model

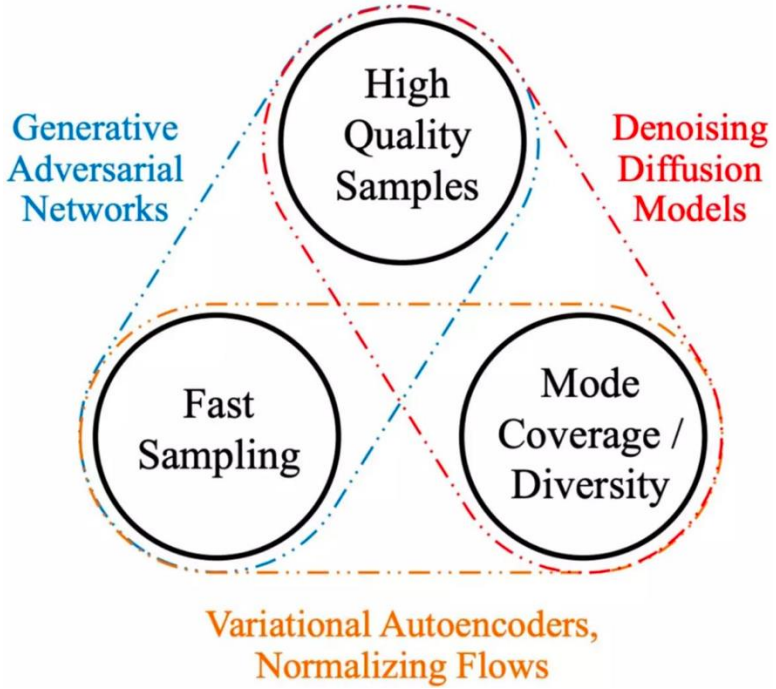
An ideal generative model would be able to train on examples (left) and then **create more examples** from the same distribution (right)

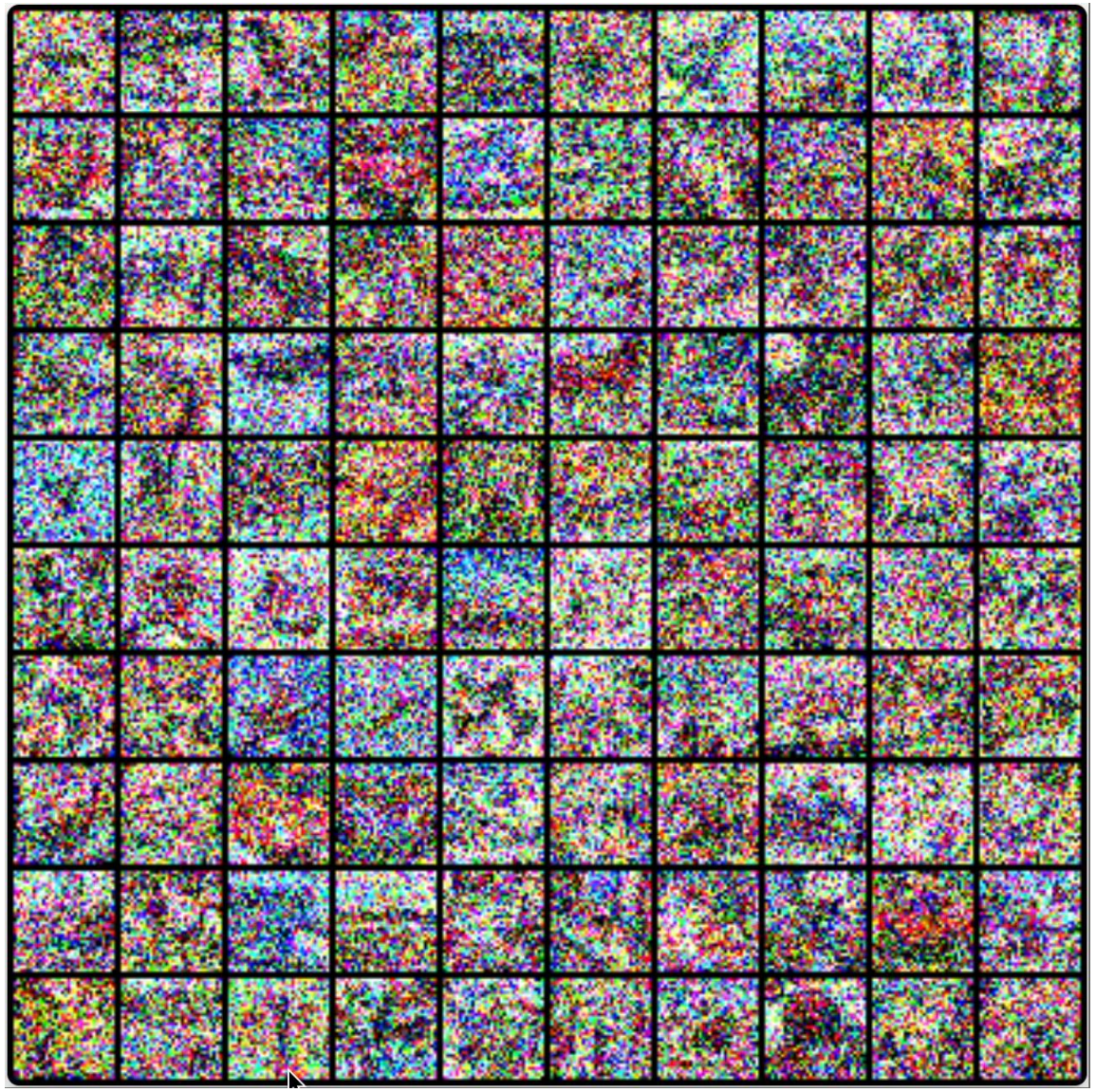


The Landscape of Deep Generative Models



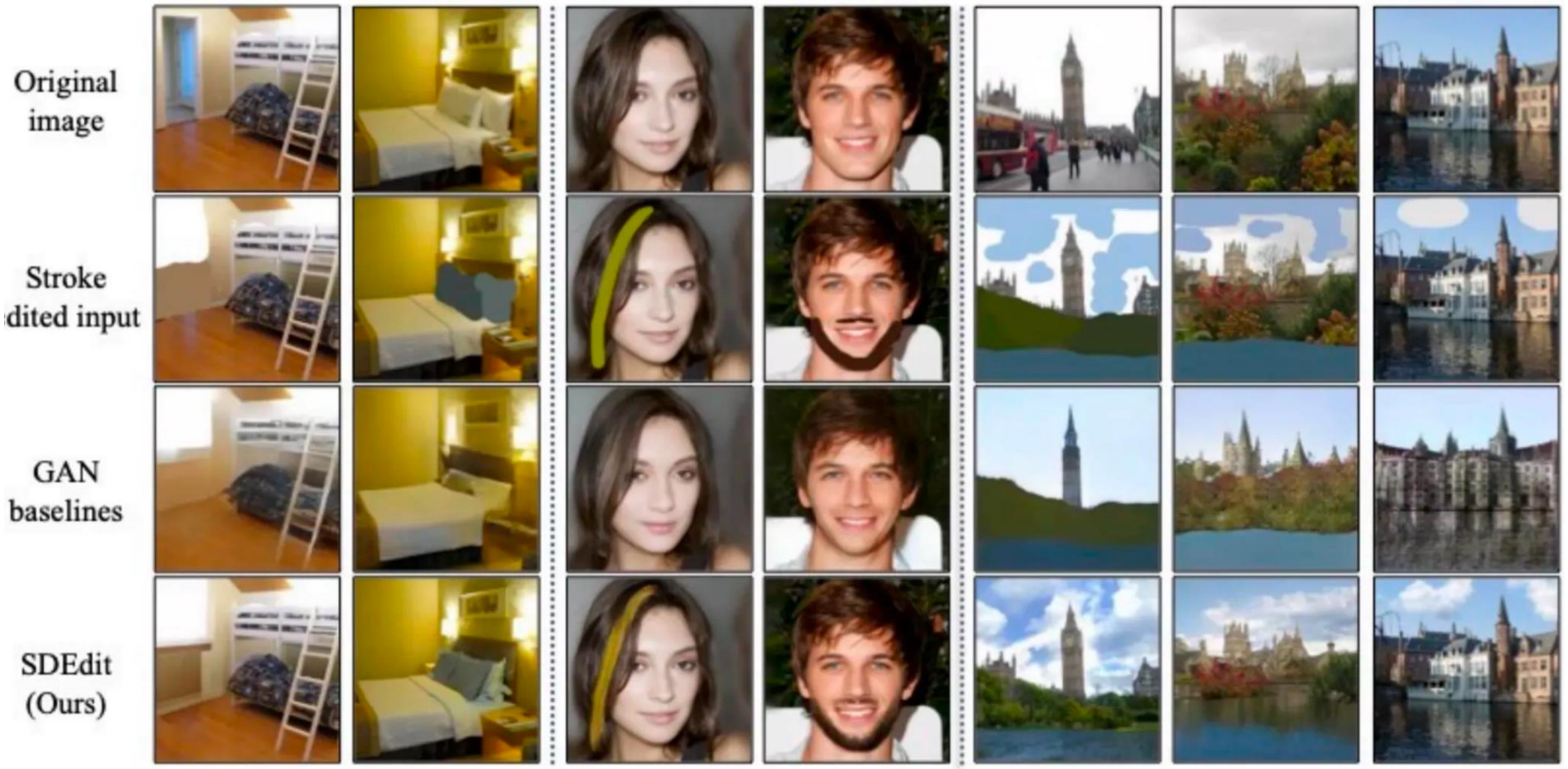
2D Diffusion model





Diffusion model Booms!!

Diffusion model achieves SoTA on image generation

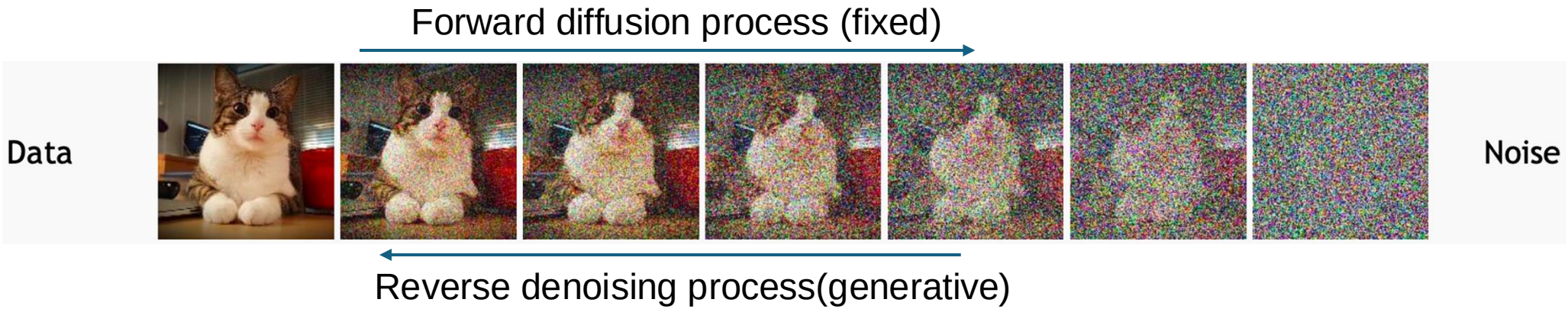


Meng, Chenlin, et al. "Sdedit: Guided image synthesis and editing with stochastic differential equations, ICLR. 2021.

Learning Diffusion Model = Learning to Denoise

Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising

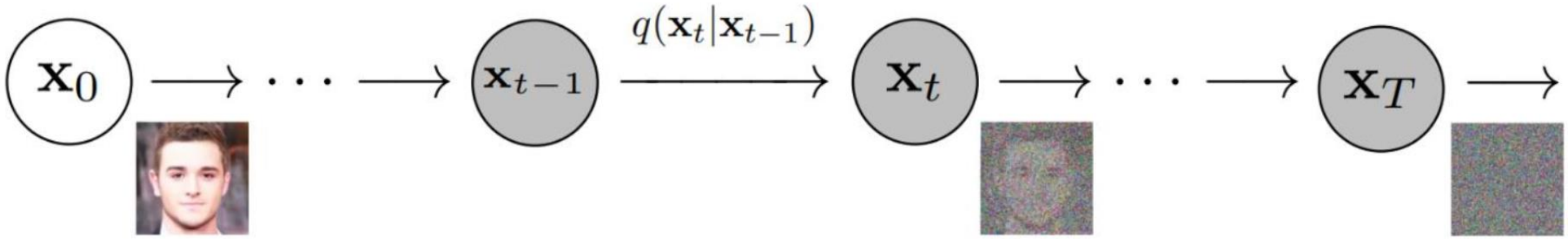


Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021
slide from https://cvpr2022-tutorial-diffusion-models.github.io/

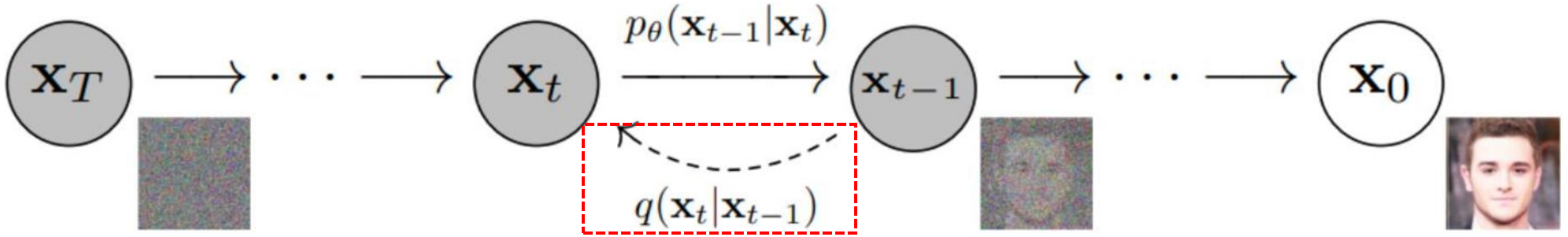
Diffusion Model

1. A **latent** variable model which maps to the latent space using a fixed **Markov** chain

Markov property: probability of next state depends **only** on the present state and not on the previous states $q(X_{n+1} = x \mid X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = q(X_{n+1} = x \mid X_n = x_n)$



2. The **goal** of training a diffusion model is to learn the **reverse** process



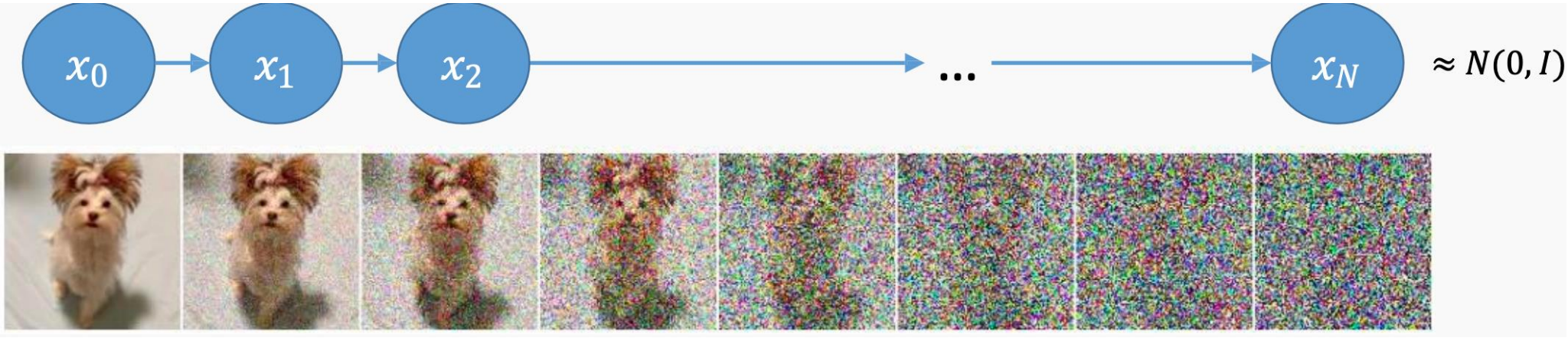
Diffusion Process

- Diffusion process gradually injects noise to data
- Described by a Markov chain: $q(x_0, \dots, x_N) = q(x_0)q(x_1|x_0) \dots q(x_N|x_{N-1})$

Variance (**predefined** noise adding schedule)

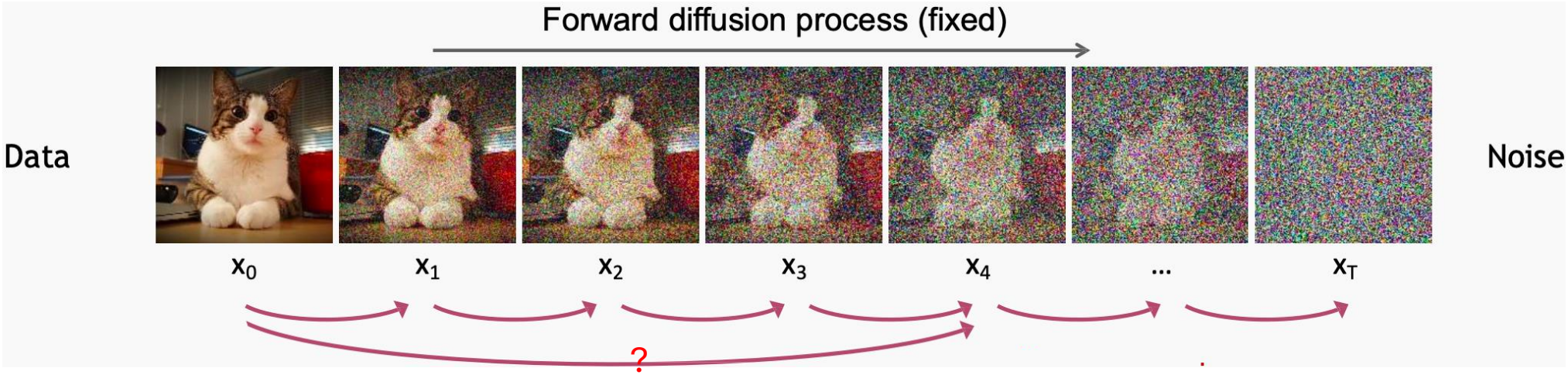
Mean

Transition of diffusion: $q(x_n|x_{n-1}) = N(\sqrt{\alpha_n}x_{n-1}, \beta_n I) \quad \alpha_n = 1 - \beta_n$



Diffusion process: $q(x_0, \dots, x_N) = q(x_0)q(x_1|x_0) \dots q(x_N|x_{N-1})$

Diffusion Kernel



Gaussian property: the product or addition of two independent Gaussian is also a Gaussian

Mean Variance(predefined)

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$$ **(Diffusion Kernel)** Where $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$

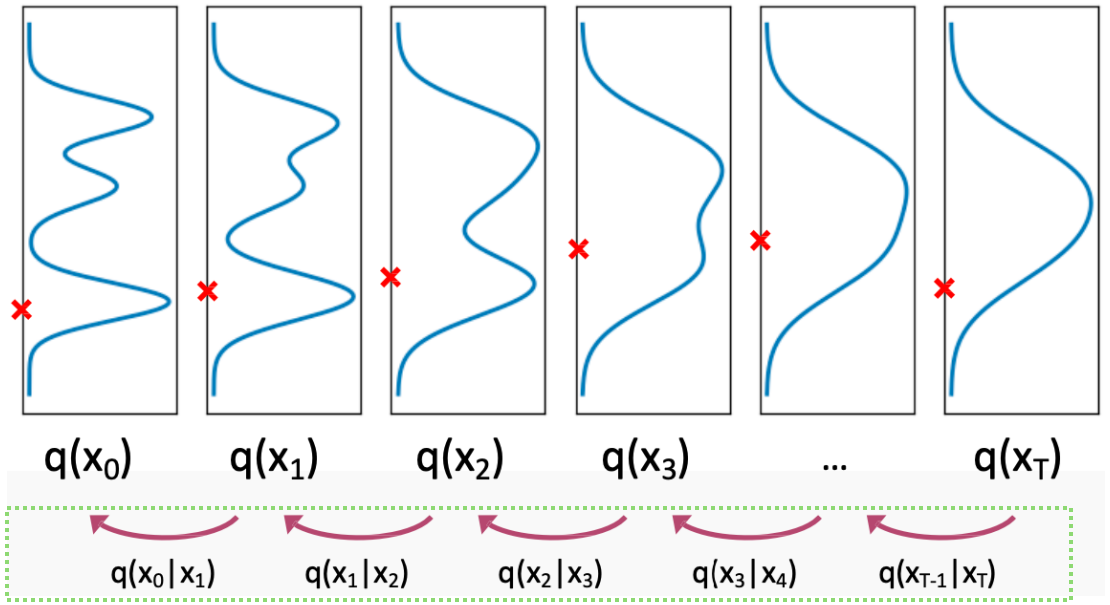
For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$ Where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t the noise schedule which is designed such that $\bar{\alpha}_T \to 0$ And $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Meng, Chenlin, et al. "Sdedit: Guided image synthesis and editing with stochastic differential equations, ICLR. 2021.

Generative Learning by Denoising

Recall: the diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$



Sample from: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}$

True Denoising Dist.

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

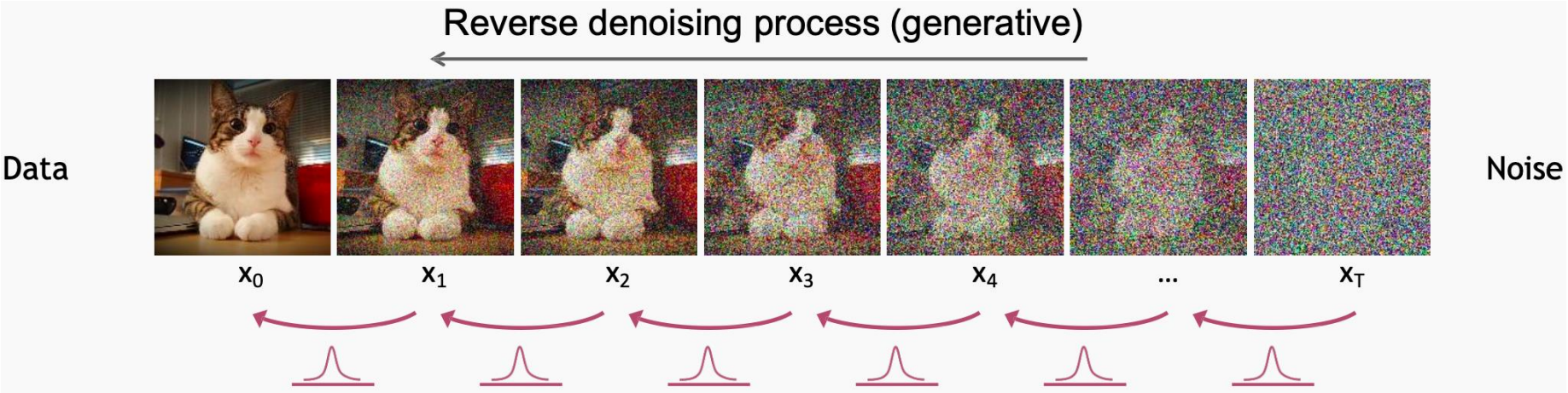
From Bayes rule, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable due to **unkown** $q(\mathbf{x}_{t-1})$

Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$?

Yes, we can use a Gaussian distribution if is small in each forward diffusion step.

Reverse Denoising Process

Approximate each reverse step with Gaussian distribution :



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

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}, \sigma_t^2\mathbf{I})$$

$$\sigma_t^2 = \beta_t$$

Trainable network (UNet)

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

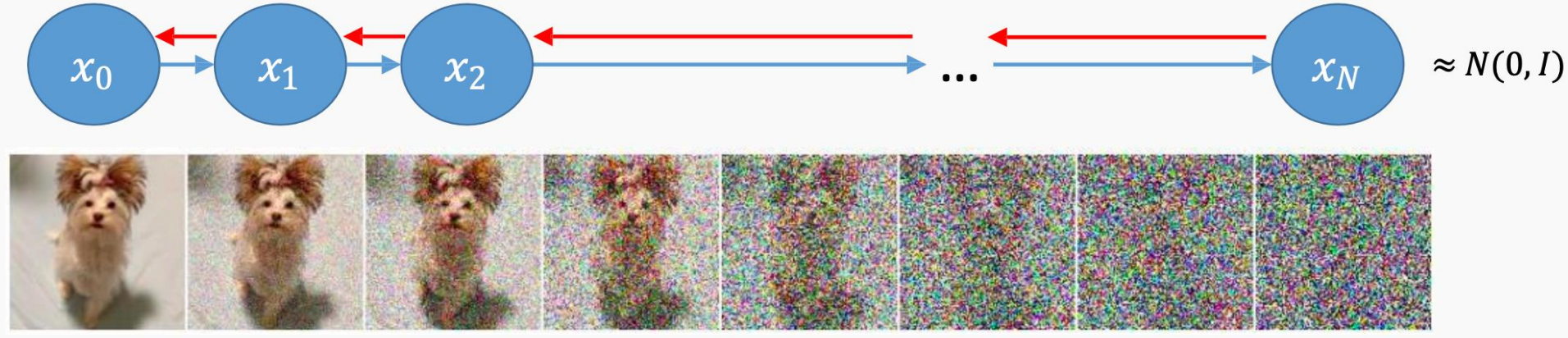
Approximate Diffusion Model

Approximate distribution by Gaussian

Model transition: $p(x_{n-1}|x_n) = N(\mu_n(x_n), \Sigma_n(x_n))$

How to approximate?

Transition of denoising: $q(x_{n-1}|x_n) = ?$



Diffusion process:

$$q(x_0, \dots, x_N) = q(x_0)q(x_1|x_0) \dots q(x_N|x_{N-1})$$
$$= q(x_0|x_1) \dots q(x_{N-1}|x_N)q(x_N)$$

Approximate Diffusion Process (Read this book if interested)

We hope $q(x_0, \ldots, x_N) \approx p(x_0, \ldots, x_N)$ $\quad p(x_{n-1}|x_n) = N(\mu_n(x_n), \Sigma_n(x_n))$

Achieved by minimizing their KL divergence (i.e., maximizing the ELBO)

min KL $\qquad\qquad$ max ELBO

$$\min_{\mu_n, \Sigma_n} KL(q(x_{0:N})||p(x_{0:N})) \Leftrightarrow \max_{\mu_n, \Sigma} \mathrm{E}_q \log \frac{p(x_{0:N})}{q(x_{1:N}|x_0)}$$

Kullback–Leibler (KL) divergence measure a distance between two probability distribution.

Evidence lower bound(ELBO) guarantee on the worst-case for the log-likelihood of a distribution.

What is the optimal solution?

Bishop et al. Deap Learning: Foundation and Concepts, 2023.

Theoretical Analysis (Read this book if interested)

max ELBO

$$\max_{\mu_n, \Sigma} \mathbb{E}_q \log \frac{p(x_{0:N})}{q(x_{1:N} \mid x_0)}$$

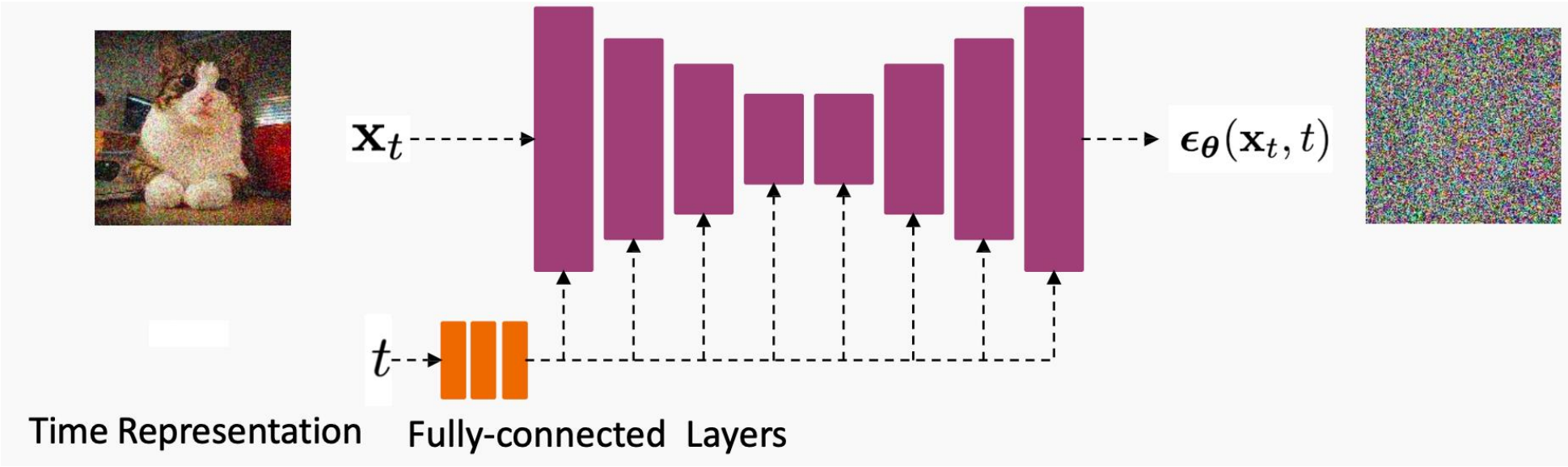
min noise difference

$$\min_{\mathbf{w}} \sum_{t-1}^{T} \left\| g\left(\sqrt{\alpha_t}\mathbf{x_0} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_t, \mathbf{w}, t\right) - \boldsymbol{\epsilon}_t \right\|^2$$

Bishop et al. Deap Learning: Foundation and Concepts, 2023.

Network Architectures

Diffusion models often use U-Net architectures to represent $\epsilon_\theta(\mathbf{x}_t, t)$, due to U-Net have the same dimension of input and output



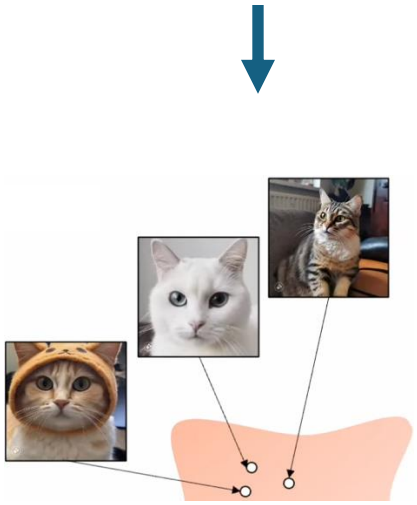
Time representation: sinusoidal positional embeddings or random Fourier features.

Condition Denoising

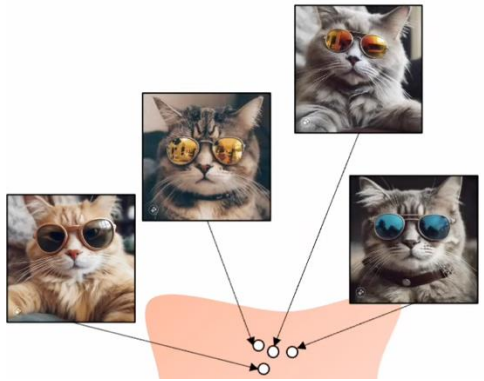
In practice, we generate a sample from a **condition y** distribution p(x | y, t) instead of random sampling from p(x|y).

y can be a label, a text prompt, or an image

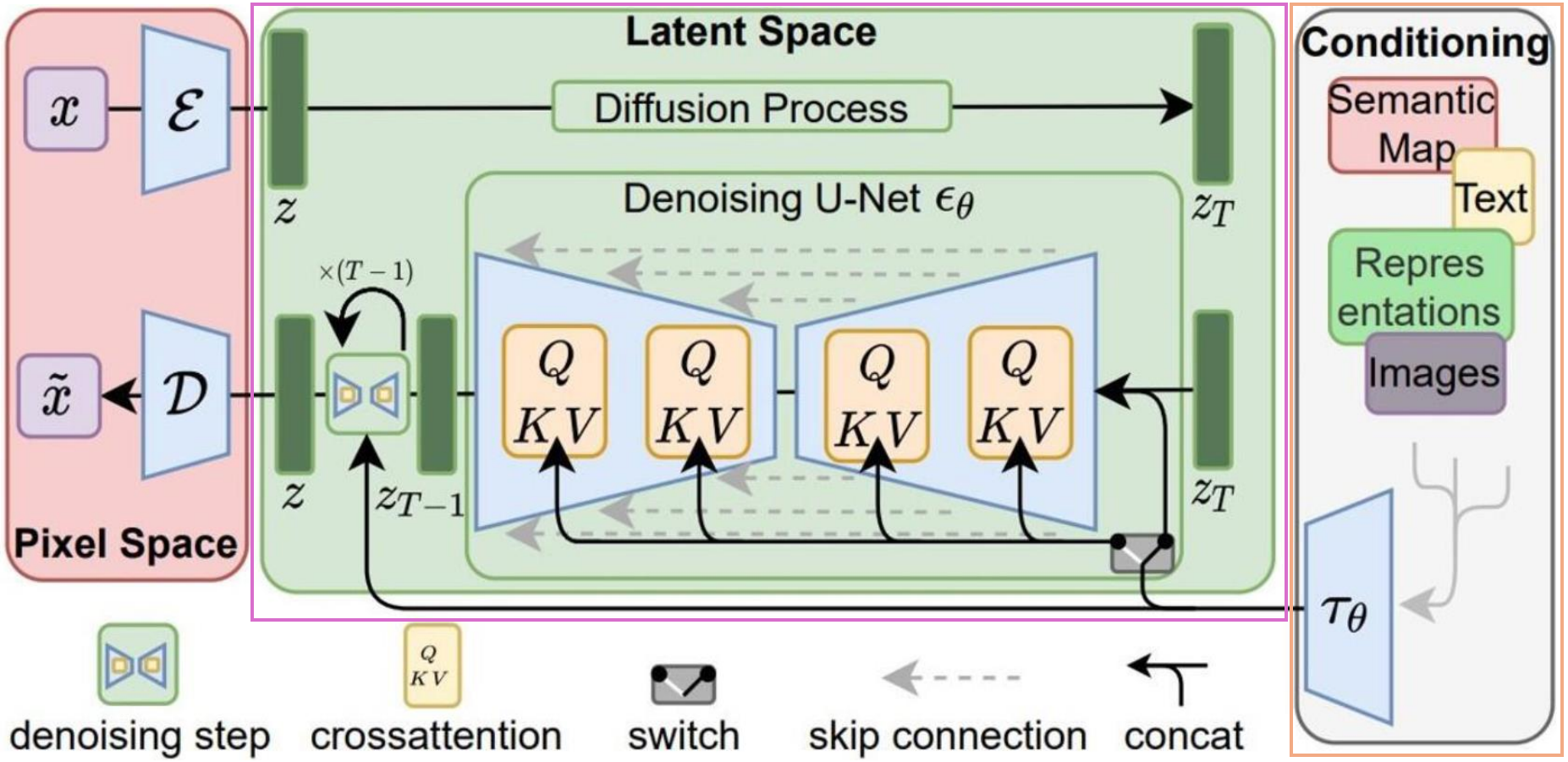
$y =$ 'CAT'



$y =$ "A cat wearing sunglasses"



Stable Diffusion

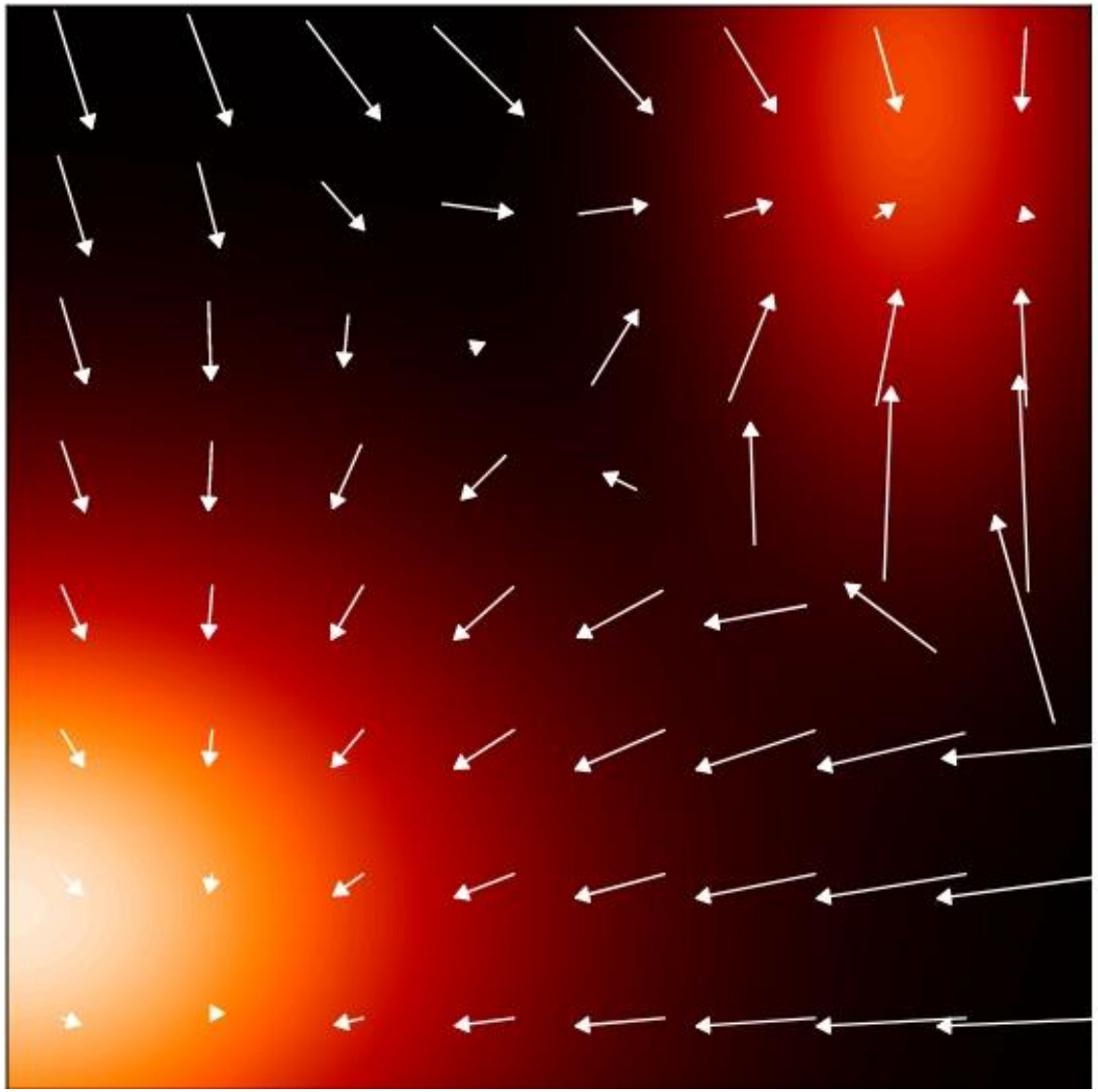


- Use Unet as to model image to image mapping
- Modulate the Unet with condition (Text, image)

Score Matching

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

Score Function: gradient of log likelihood



A heat map illustration of the score function, showing a distribution in two dimensions

Theoretical Analysis (Read this book if interested)

Score Function

$$\nabla_{x_t} \ln q(\mathbf{x_t}|\mathbf{x_0}, \sigma) = -\frac{1}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}$$

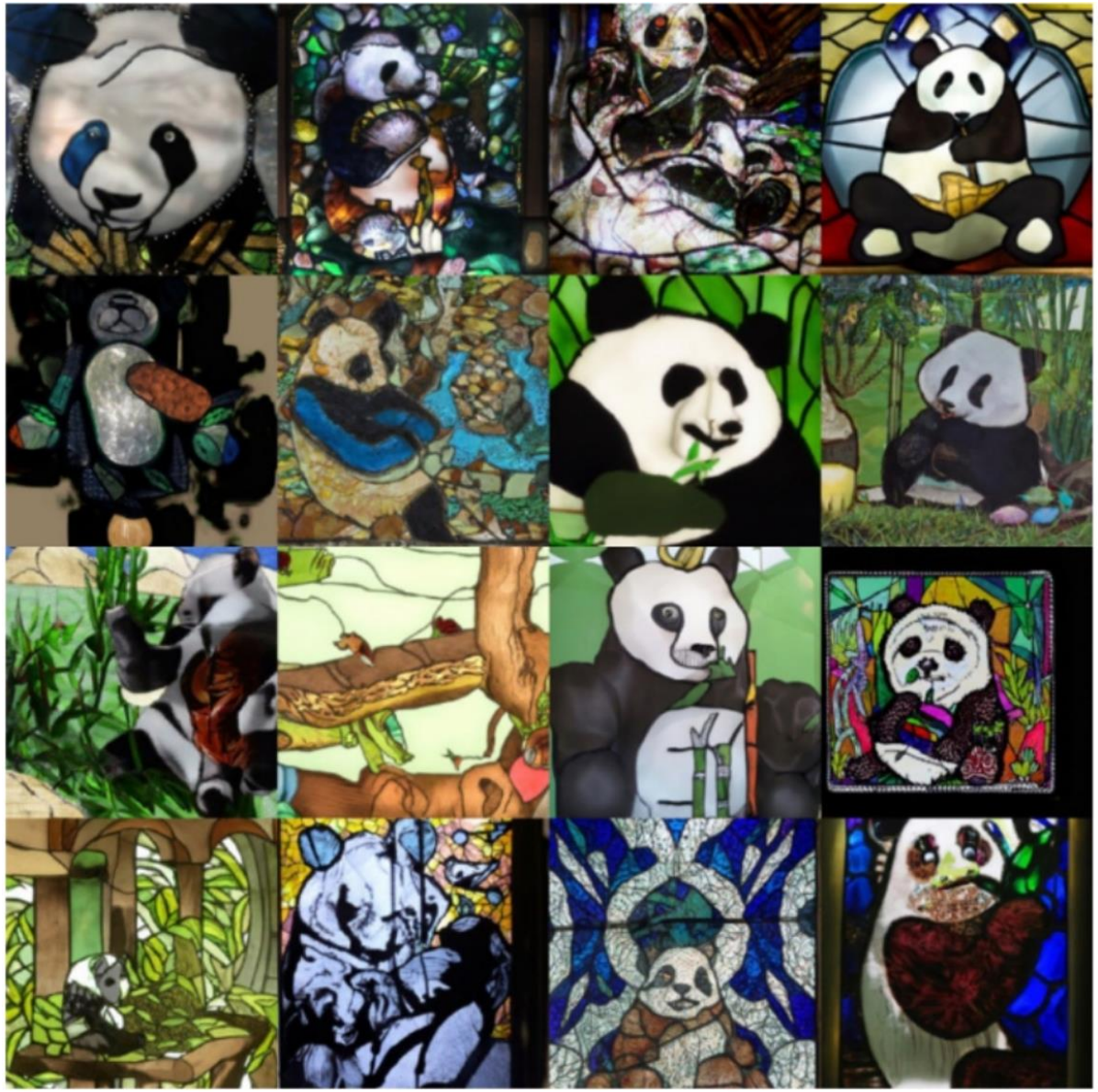
Noise predicting network

$$\boldsymbol{\epsilon} = g(\mathbf{x_t}, \mathbf{w})$$

Just a scale between score function and predicting network

Bishop et al. Deap Learning: Foundation and Concepts, 2023.

Conditional 2D diffusion model



conditioning text “A stained glass window of a panda eating bamboo”.

Classifier Guidance

2D diffusion model sometimes is not stable.

Can we utilize annotations in dataset to train a classifier diffusion model ?

- Assume pairs of data (x, c). A classifier guidance diffusion model consists of
  - A trained conditional diffusion model $p_\theta(\mathbf{x}_t|\mathbf{c})$
  - A trained classifier model on noisy data $p_\phi(\mathbf{c}|\mathbf{x}_t)$
- During sampling, at each denoising step, modify the score function to

$$\nabla_{\mathbf{x}_t} \log \tilde{p}_{\theta,\phi}(\mathbf{x}_t|\mathbf{c}) = \nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|\mathbf{c}) + \omega \nabla_{\mathbf{x}_t} \log p_\phi(\mathbf{c}|\mathbf{x}_t).$$

From the conditional diffusion model

From the classifier model

- Upweight samples that the classifier assigns high probability with, better alignment with c.

- Cons: need to train an additional classifier. Increase model complexity.

Classifier Free Guidance

- Recall: Assume there're two diffusion models:
  - one conditional model $p_\theta(\mathbf{x}_t|\mathbf{c})$
  - one classifier model. $p_\phi(\mathbf{c}|\mathbf{x}_t)$

- By Bayes' rule, we can define an implicit classifier $p_\theta(\mathbf{c}|\mathbf{x}_t) \propto p_\theta(\mathbf{x}_t|\mathbf{c})/p_\theta(\mathbf{x}_t).$
- The modified score function during sampling then becomes

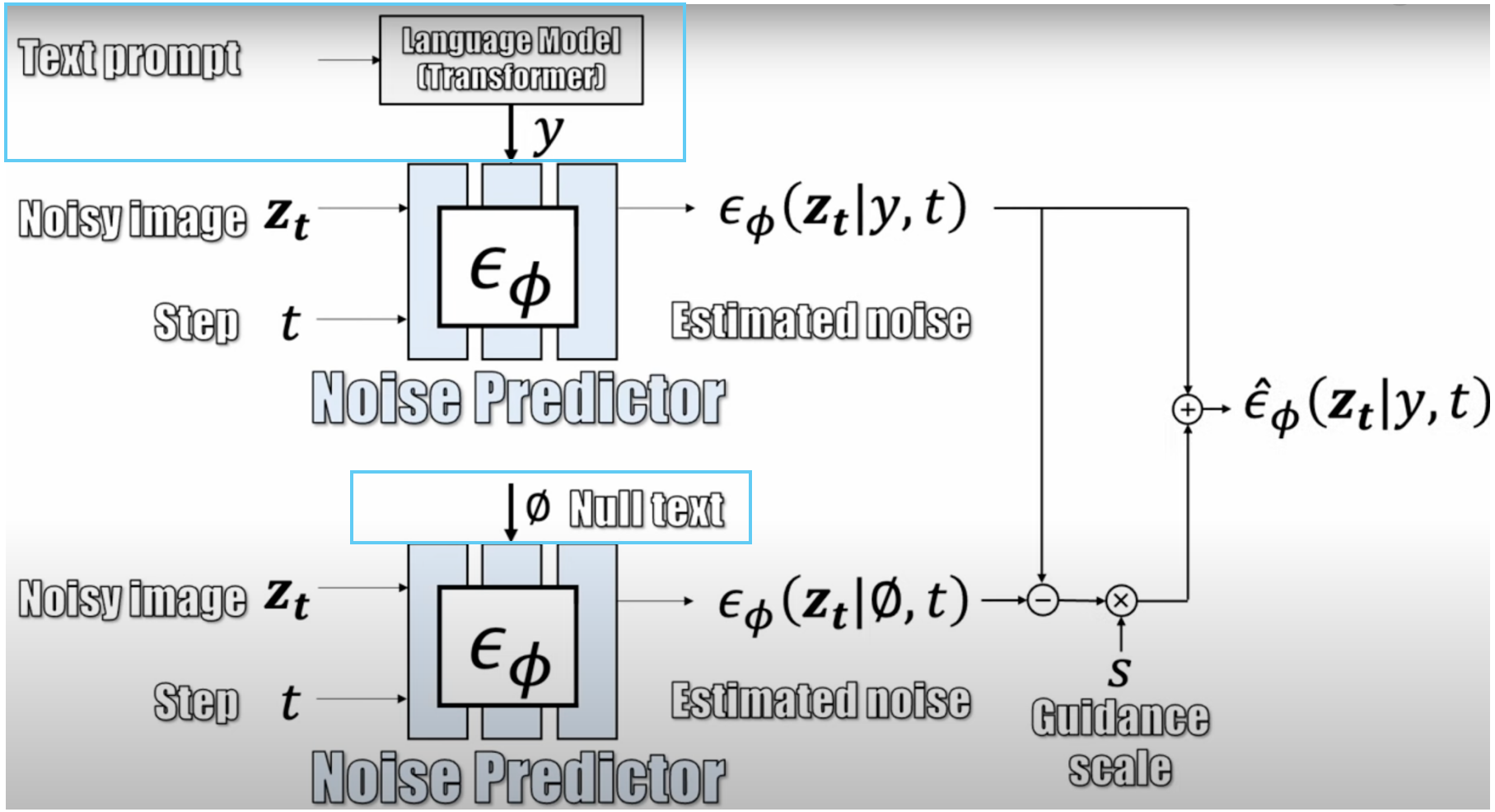
$$\nabla_{\mathbf{x}_t} \log \tilde{p}_\theta(\mathbf{x}_t|\mathbf{c}) = (1+\omega)\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t|\mathbf{c}) - \omega\nabla_{\mathbf{x}_t} \log p_\theta(\mathbf{x}_t).$$

From the conditional diffusion model

From the unconditional diffusion model

- The two models can share weights, with the unconditional model taking a null class label c.

Classifier Free Guidance



- Large classifier-free guidance weights → better text alignment, worse image fidelity

Classifier Free Guidance

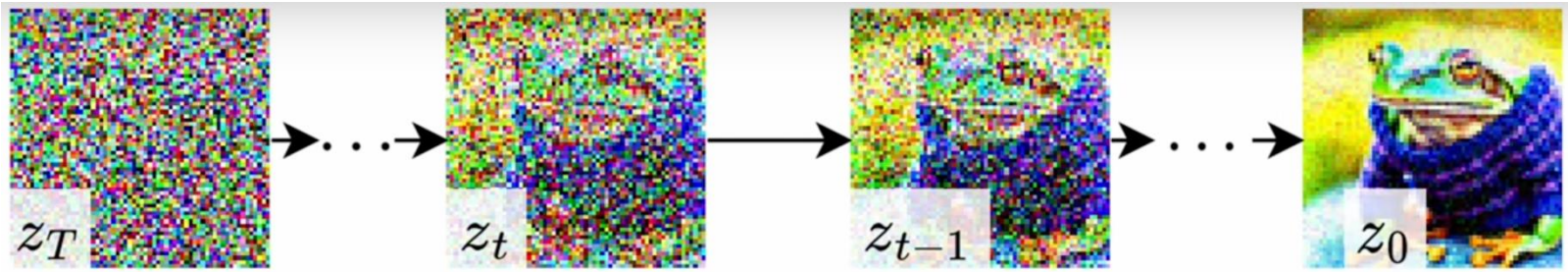
conditioning text “A stained glass window of a panda eating bamboo”.

Guidance weight 0.

Guidance weight 3.

- 2D Diffusion Generative Model
- 3D Generative Model
- Demo Code

Text to 3D



Update samples in **pixel space**, but it only works in 2D diffusion model.

3D representation with θ

$$\mathbf{x} = g(\theta)$$

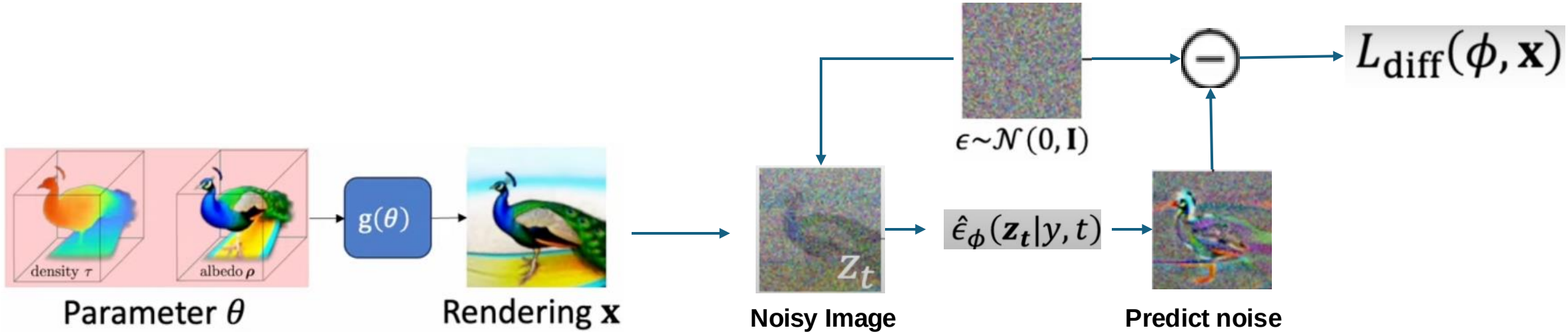
Parameter

Rendered image

Differentiable rendering

Update in **parameter space?**

Text to 3D



z_t the same as x_t in 2D diffusion model

$$L_{\text{diff}}(\phi, \mathbf{x}) = \mathbb{E}_{t\sim\mathcal{U}(0,1),\epsilon\sim\mathcal{N}(0,\mathbf{I})}\left[w(t)||\epsilon_\phi(\boldsymbol{z_t}|y,t) - \epsilon||_2^2\right]$$

$$z_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$$

$$\nabla_\theta L_{\text{Diff}}(\phi, \mathbf{x} = g(\theta)) = \mathbb{E}_{t,\epsilon}\left[w(t) \ \left(\hat{\epsilon}_\phi(\boldsymbol{z_t}|y,t) - \epsilon\right) \ \frac{\partial \hat{\epsilon}_\phi(\boldsymbol{z_t}|y,t)}{\partial z_t} \ \frac{\partial \mathbf{x}}{\partial \theta}\right]$$

Noise Residual — U-Net Jacobian — Generator Jacobian

Computational Expensive

Text to 3D

$$\nabla_\theta L_{\text{Diff}}(\phi, \mathbf{x} = g(\theta)) = \mathbb{E}_{t,\epsilon}\left[w(t) \left(\hat{\epsilon}_\phi(\mathbf{z}_t|y,t) - \epsilon\right) \frac{\partial \hat{\epsilon}_\phi(\mathbf{z}_t|y,t)}{\partial \mathbf{z}_t} \frac{\partial \mathbf{x}}{\partial \theta} \right]$$

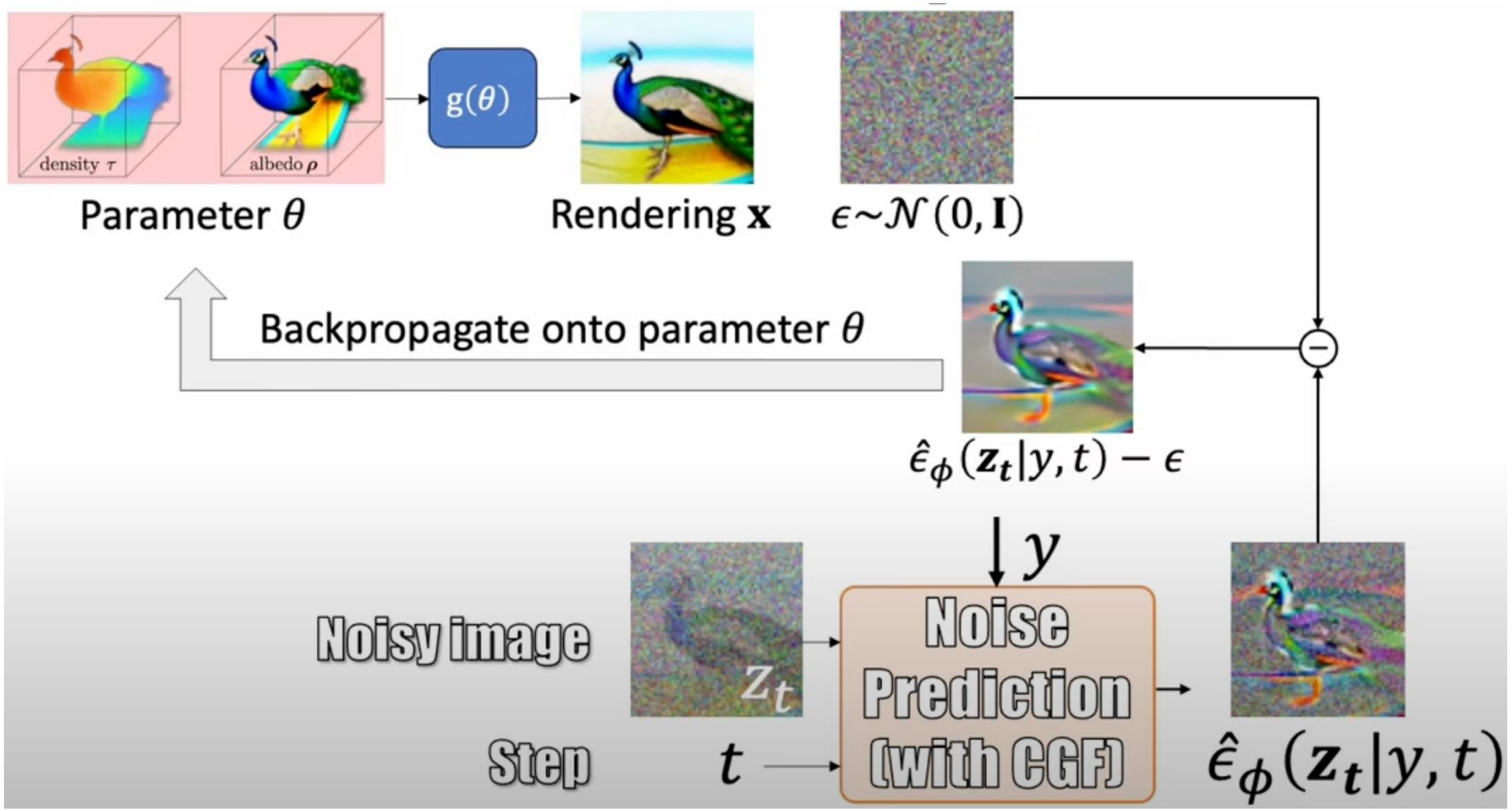
Noise Residual — U-Net Jacobian — Generator Jacobian

Optimize more efficient and works better **by removing U-Net Jacobia**

$$\nabla_\theta L_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) = \mathbb{E}_{t,\epsilon}\left[w(t) \left(\hat{\epsilon}_\phi(\mathbf{z}_t|y,t) - \epsilon\right) \frac{\partial \mathbf{x}}{\partial \theta} \right]$$

Score Distillation Sampling (SDS)

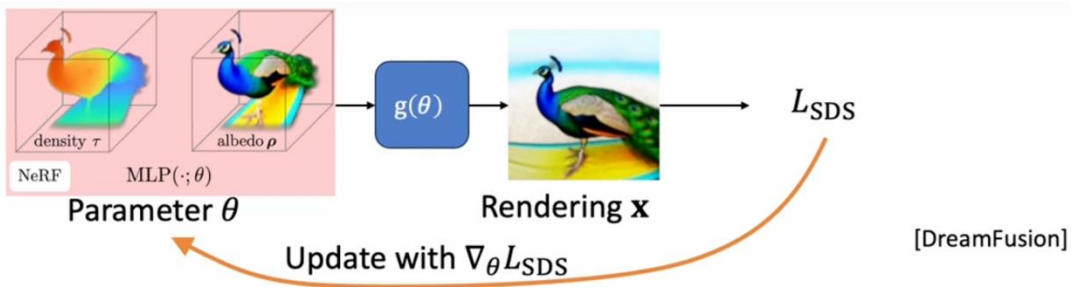
SDS Loss Implementation



[DreamFusion]

Poole et al. DreamFusion: Text-to-3D using 2D Diffusion, 2022.

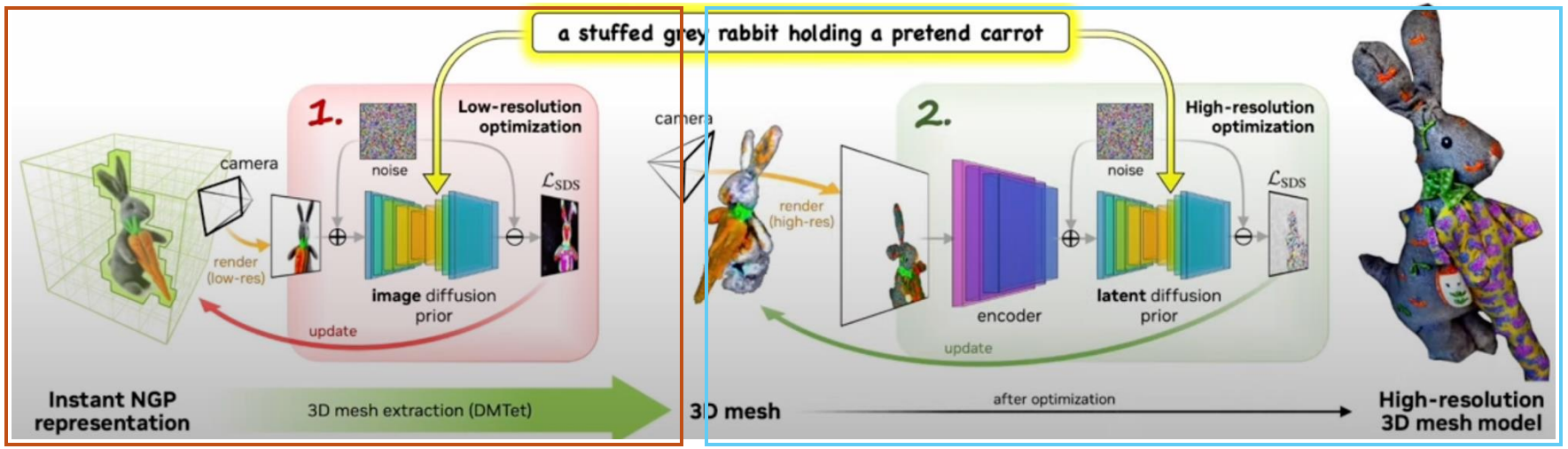
Low- and High-Resolution Optimization



[DreamFusion]

$$\nabla_\theta L_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) = \mathbb{E}_{t,\epsilon}\left[w(t)(\hat{\epsilon}_\phi(\boldsymbol{z_t}|y,t) - \epsilon)\frac{\partial \mathbf{x}}{\partial \theta}\right]$$

For SDS loss, Rendering in high resolution radiance field is **computation expensive**

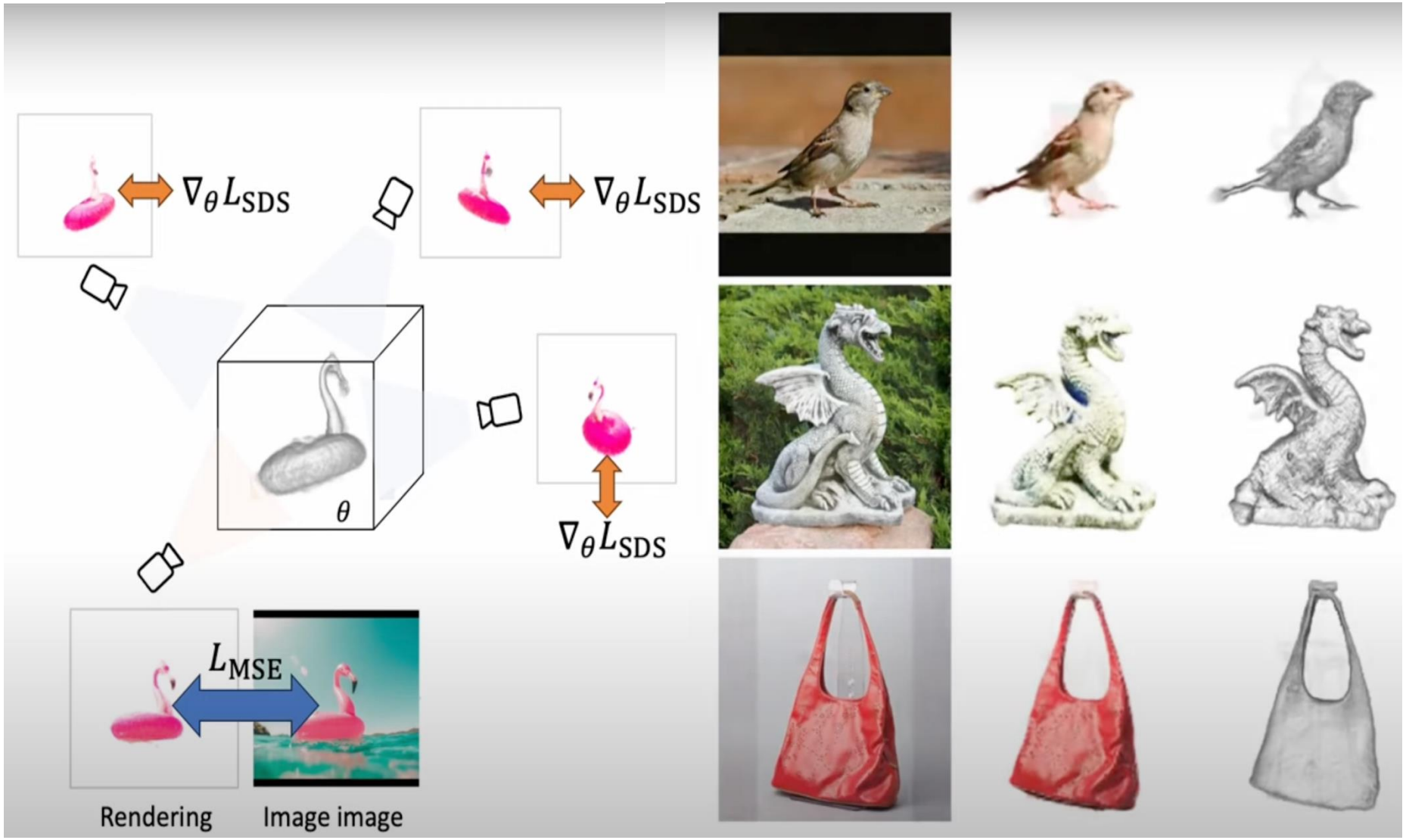


[Magic3D]

- First train on a low-resolution field to get a 3D Mesh
- Then train 3D Mesh to refine with SDS optimization to get high resolution 3D model

Lin et al. Magic3D: High-Resolution Text-to-3D Content Creation, 2023 CVPR.

Single-Image to 3D

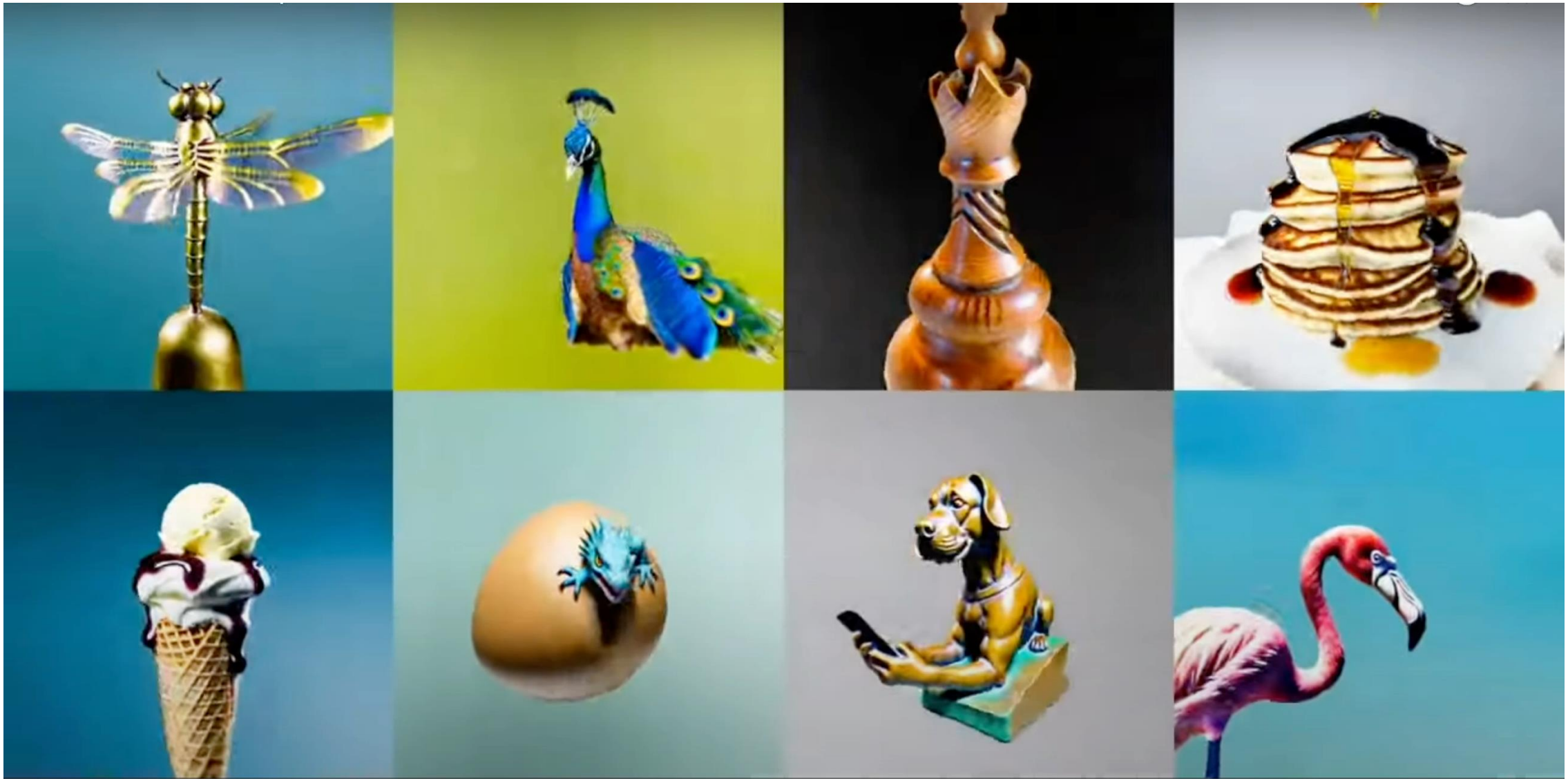


[RealFusion]

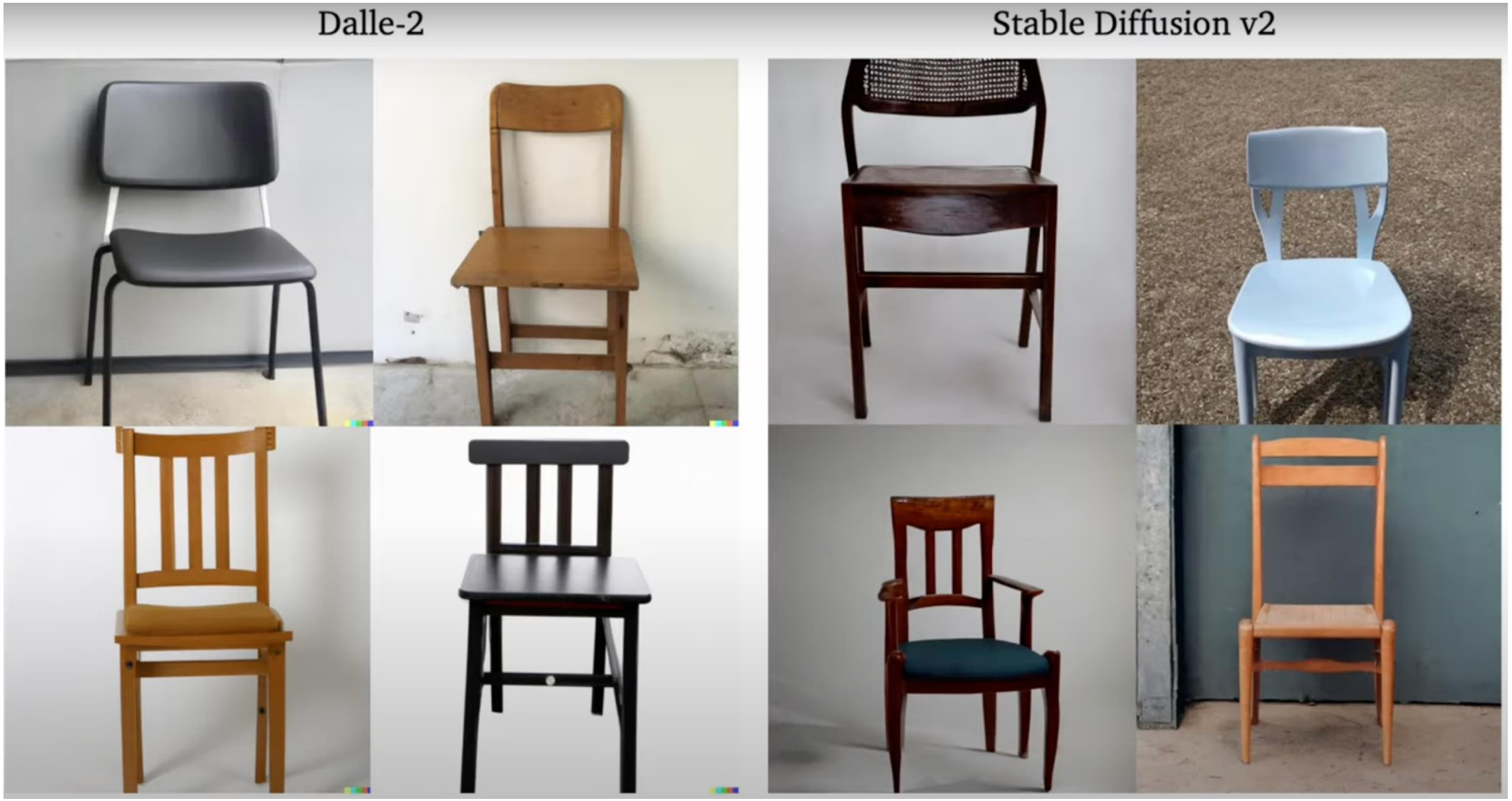
Condition input change to image instead of text prompt

Melas-Kyriazi. RealFusion: 360 Reconstruction of Any Object from a Single Image, 2023 CVPR.

Janus (multi-face) problem

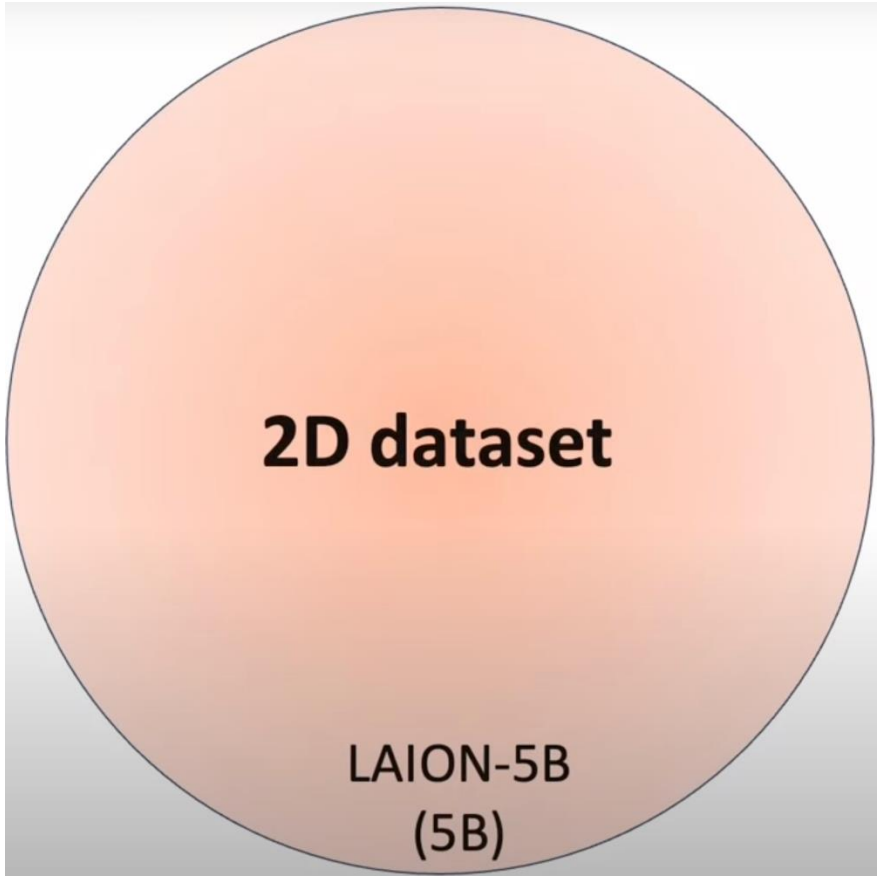


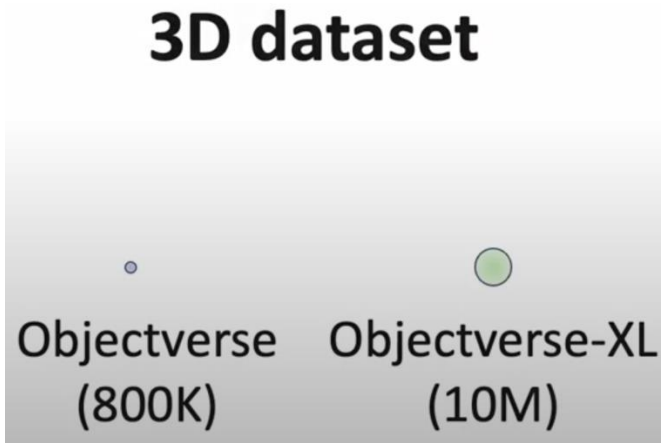
2D Dataset



Front view bias

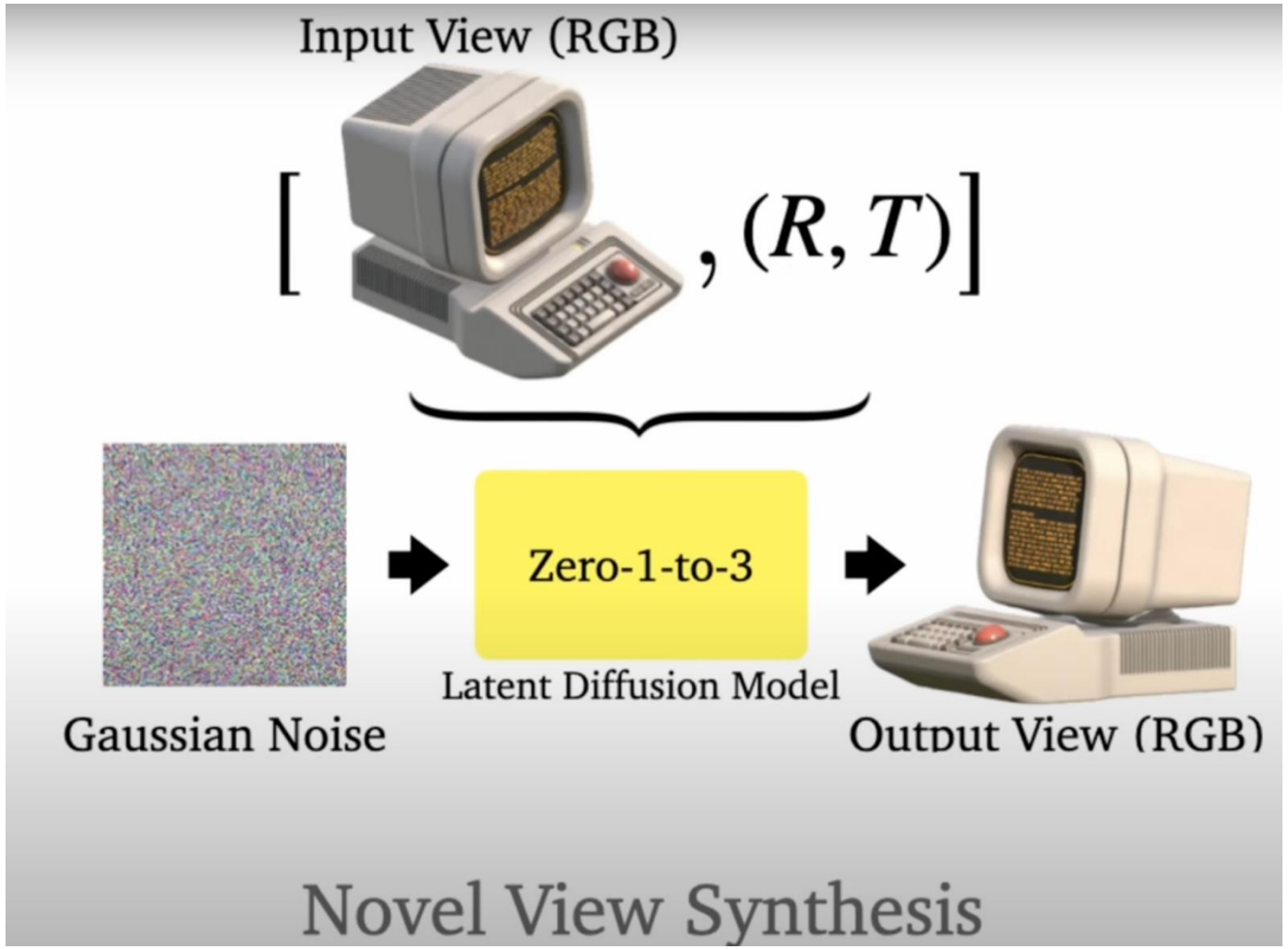
3D Dataset





Can we **finetune** existed 2D diffusion model with 3D datase to get a more consistent novel view generation model?

Novel View Synthesis



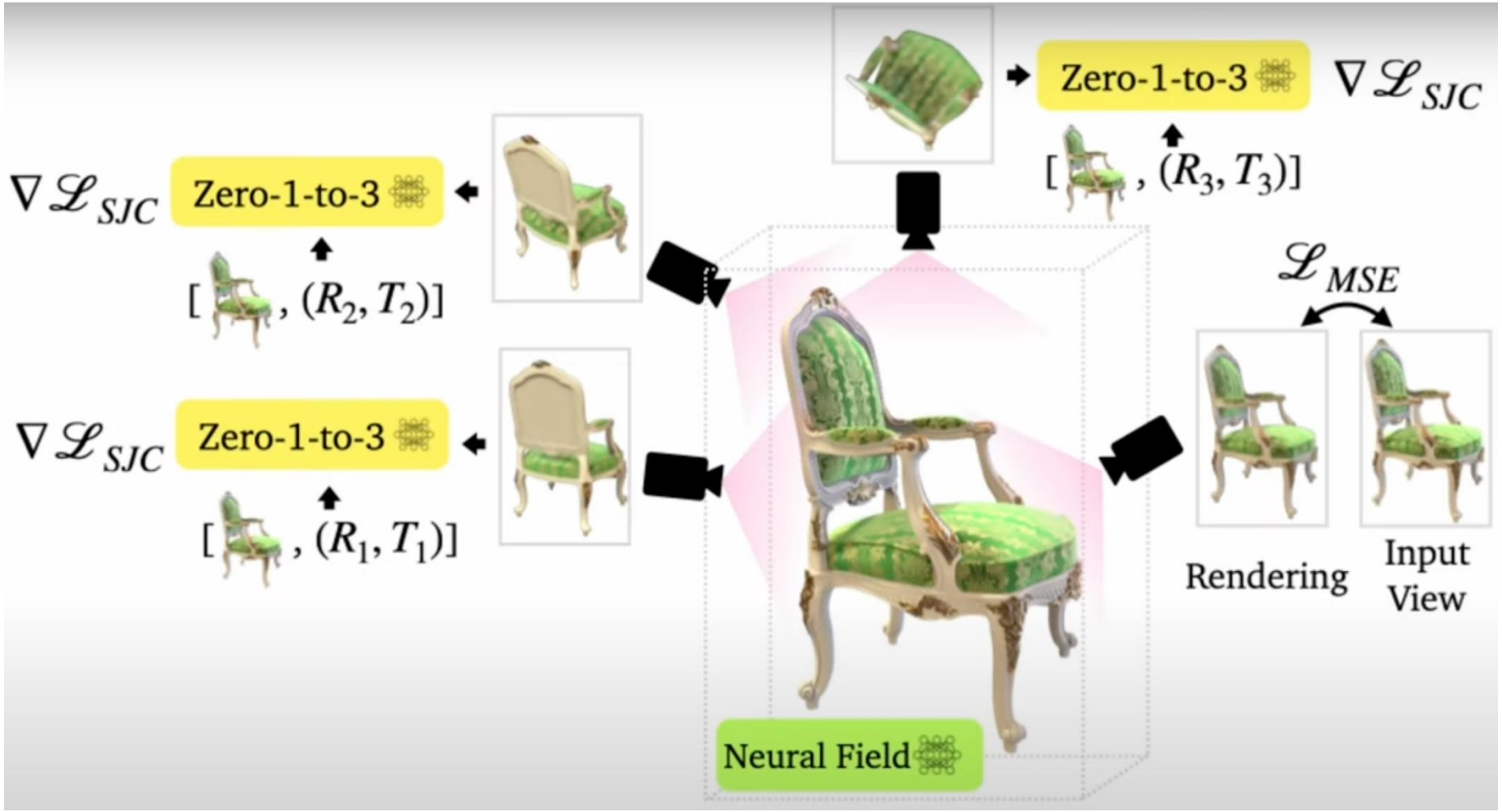
$p(z \mid y, t)$

Add pose condition

$p(z \mid R, T, y, t)$

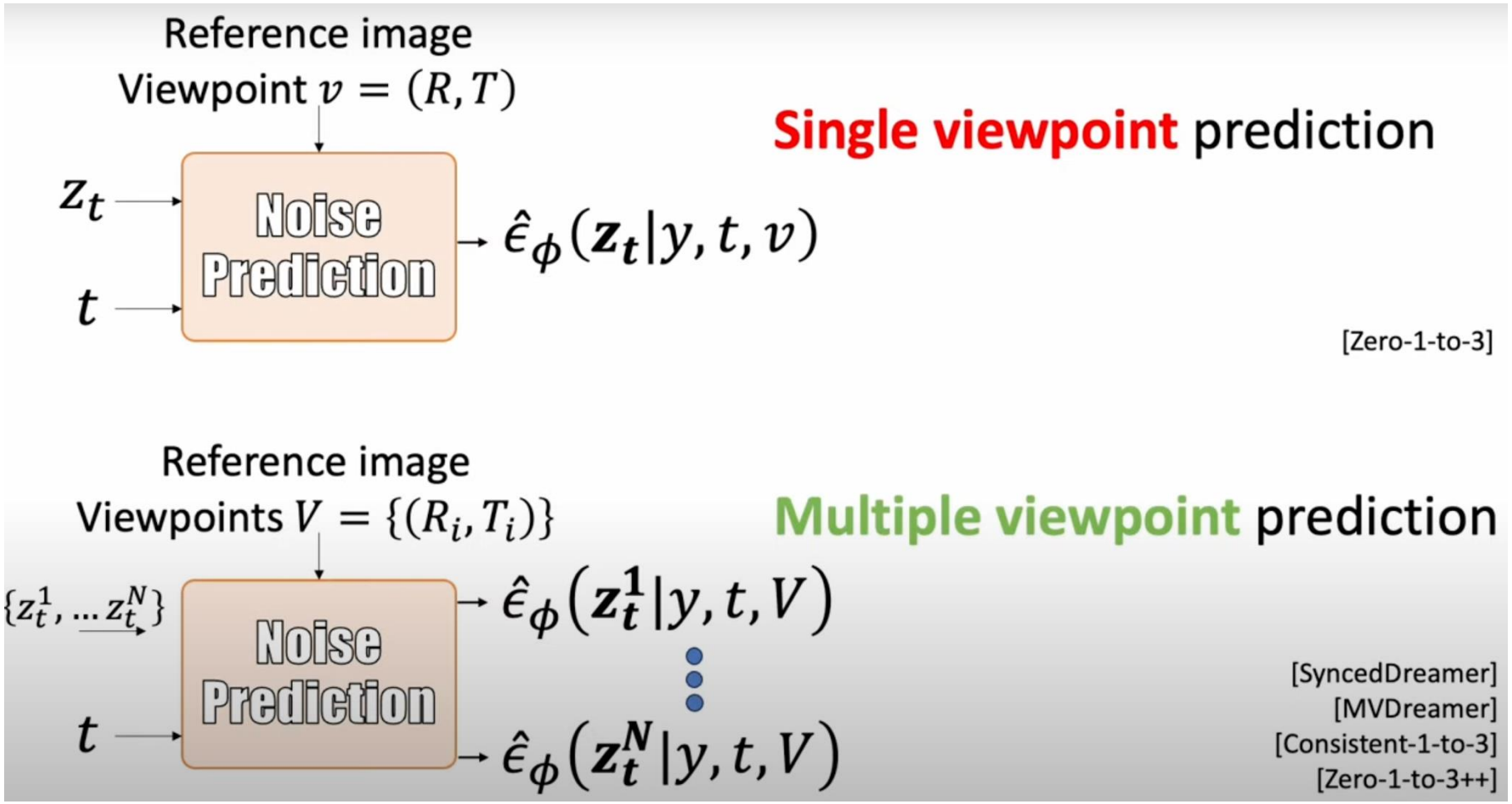
Ruoshi Liu et al. Zero-1-to-3: Zero-shot One Image to 3D Object, 2023 ICCV.

Novel View Synthesis



Ruoshi Liu et al. Zero-1-to-3: Zero-shot One Image to 3D Object, 2023 ICCV.

Multi View to 3D

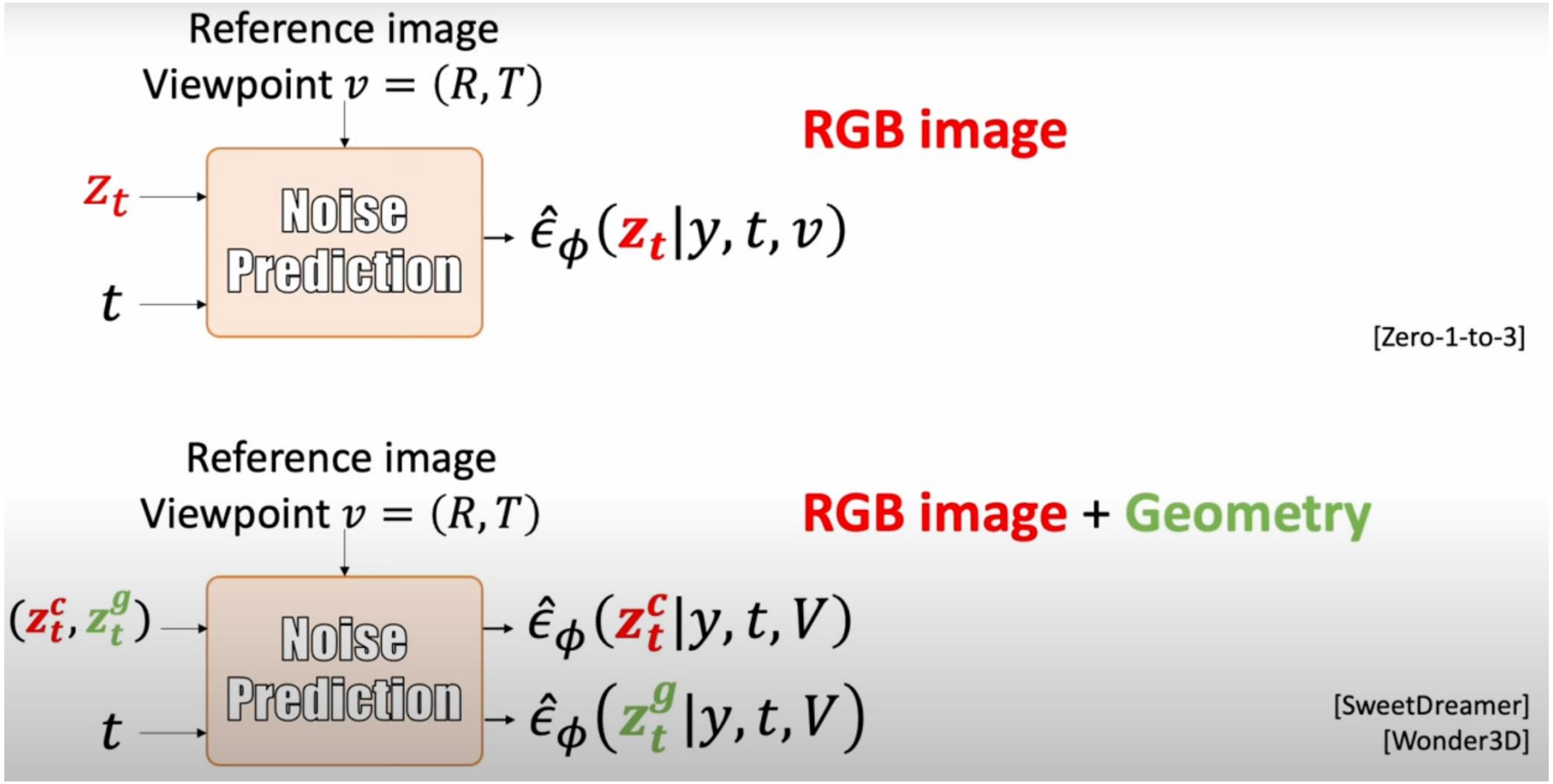


p(z | R, T, y, t)

Multiview prediction

p(z^1, …, z^N | R, T, y, t)

Rendering Geometry



p(z | R, T, y, t)

Geometry prediction

p(z^c, z^g | R, T, y, t)

References

- Bishop et al. Deap Learning: Foundation and Concepts, 2023.
- Jia-Bin Hunag: How I Understand Diffusion Models
- Jia-Bin Hunag: AI 3D Generation, explained
- Tutorial on Denoising Diffusion-based Generative Modeling: Foundations and Applications

- 2D Diffusion Generative Model
- 3D Generative Model
- Demo Code

Training Code of 2D Diffusion Model

Input: Training data $\mathcal{D} = \{x_n\}$

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

Output: Network parameters 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

$x_0 \sim \mathcal{D}$ // Sample a data point

$t \sim \{1, \ldots, T\}$ // Sample a point along the Markov chain

$\epsilon \sim \mathcal{N}(\epsilon|0, I)$ // Sample a noise vector

$x_t \leftarrow \sqrt{\alpha_t}x_0 + \sqrt{1 - \alpha_t}\epsilon$ // Evaluate noisy latent variable

$\mathcal{L}(w) \leftarrow \|\mathbf{g}(x_t, w, t) - \epsilon\|^2$ // Compute loss term

Take optimizer step

until converged

return w

$$\min_{\mathbf{w}} \sum_{t-1}^{T} \left\| g\left(\sqrt{\alpha_t}\mathbf{x_0} + \sqrt{1 - \alpha_t}\boldsymbol{\epsilon}_t, \mathbf{w}, t\right) - \boldsymbol{\epsilon}_t \right\|^2$$

Sampling Code of 2D Diffusion Model

Input: Trained denoising network $g(x_t, w, t)$

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

Output: Sample vector x in data space

$x_T \sim \mathcal{N}(x|0, I)$ // 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

$\mu(x_t, w, t) \leftarrow \frac{1}{\sqrt{1-\beta_t}}\left\{x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} g(x_t, w, t)\right\}$

$\epsilon \sim \mathcal{N}(\epsilon|0, I)$ // Sample a noise vector

$x_{t-1} \leftarrow \mu(x_t, w, t) + \sqrt{\beta_t}\epsilon$ // Add scaled noise

end for

$x_0 = \frac{1}{\sqrt{1-\beta_1}}\left\{x_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}} g(x_1, w, t)\right\}$ // Final denoising step

return x_0

$$\mathbf{x}_{t-1} = \mu(\mathbf{x}_t, \mathbf{w}, t) + \sqrt{\beta_t}\boldsymbol{\epsilon}$$

$$\mu(\mathbf{x}_t, \mathbf{w}, t) = \frac{1}{\sqrt{1-\beta_t}}\left\{\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\mathbf{g}(\mathbf{x}_t, \mathbf{w}, t)\right\}$$

Image to 3D

Input: Classifier Free Guidance δ

Trained condition network $c(elevation, azimuth, distance)$

Trained denoising network $g(x_t, w, t, cond)$

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

Output: Update vector θ in parameter space

---

repeat

$x \leftarrow f(\theta, elevation, azimuth, distance)$ // Sample a novel view

$t \sim \{1, \ldots, T\}$ // Sample a point along the Markov chain

$\epsilon \sim \mathcal{N}(\epsilon|0, I)$ // Sample a noise vector

$x_t \leftarrow \sqrt{\alpha_t} x_0 + \sqrt{1-\alpha_t}\epsilon$ // Evaluate noisy latent variable

$cond \leftarrow c(elevation, azimuth, distance)$ // Compute condition

$\hat{\epsilon}\_uncond \leftarrow g(x_t, w, t, cond)$ // Obtain condition noise prediction

$\hat{\epsilon}\_uncond \leftarrow g(x_t, w, t, null)$ // Obtain uncondition noise prediction

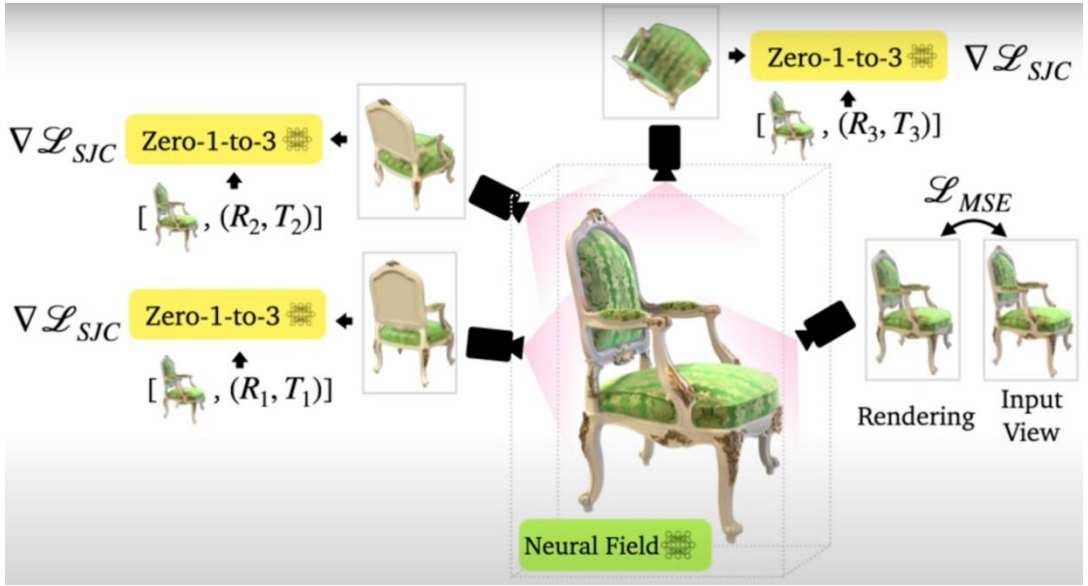
$\hat{\epsilon} = \hat{\epsilon}\_uncond + \delta(\hat{\epsilon}\_cond - \hat{\epsilon}\_uncond)$ // Obtain final noise prediction

$\mathcal{L}(\theta) \leftarrow (1-\alpha_t)\|\hat{\epsilon} - \epsilon\|^2$ // Compute SDS loss

Take optimizer step

until converged

return θ

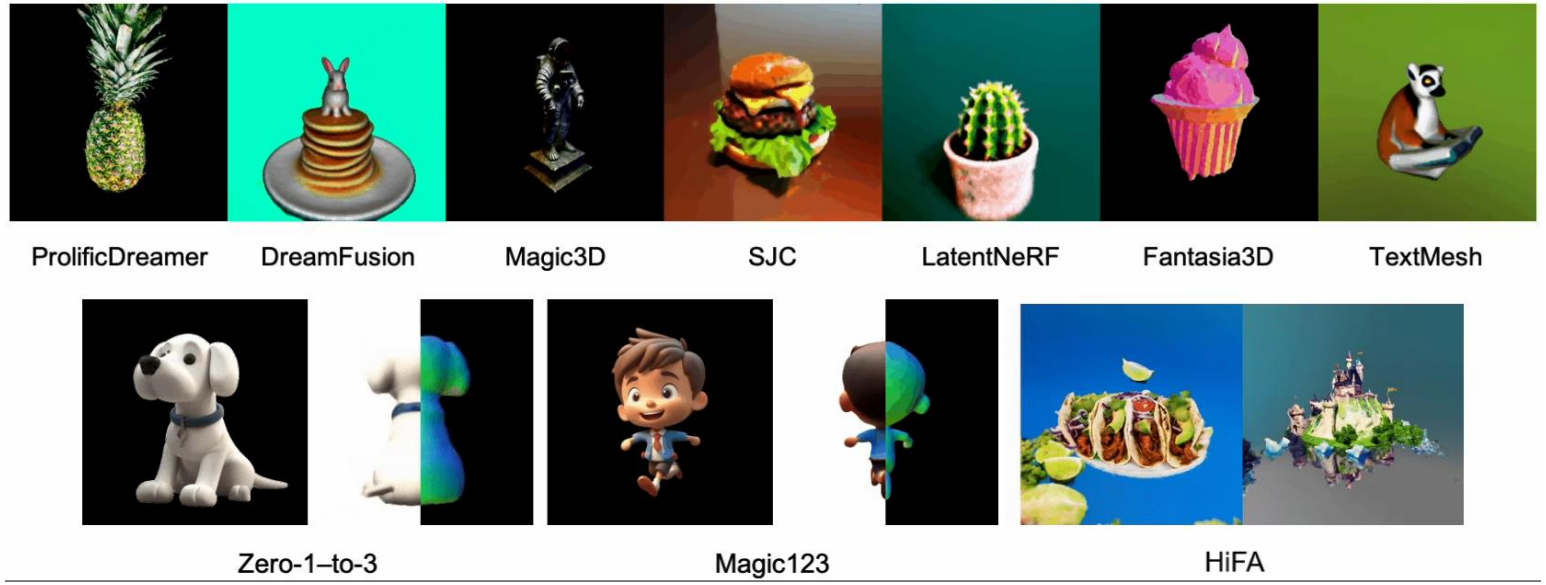


$$\nabla_\theta L_{\text{SDS}}(\phi, \mathbf{x} = g(\theta)) = \mathbb{E}_{t,\epsilon}\left[w(t)\left(\hat{\epsilon}_\phi(\boldsymbol{z_t}|y, t) - \epsilon\right)\frac{\partial \mathbf{x}}{\partial \theta}\right]$$

Project: Image to 3D

threestudio

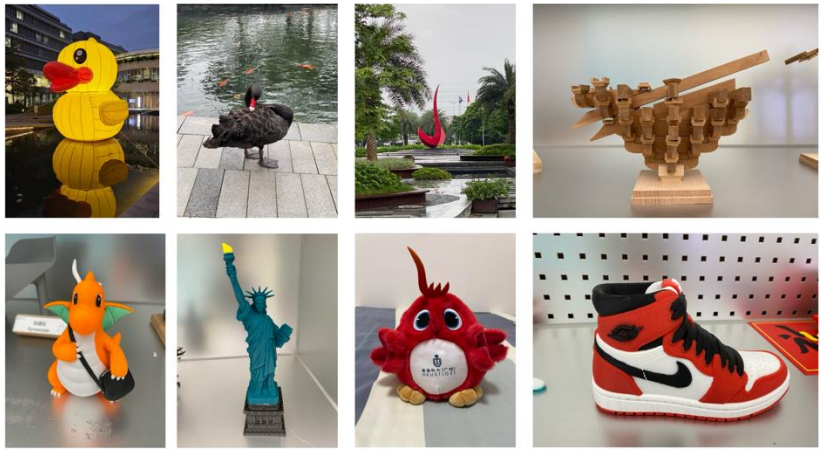
threestudio is a unified framework for 3D content creation from text prompts, single images, and few-shot images, by lifting 2D text-to-image generation models.



1. Base-code is based on Threestudio
2. You can select open-source methods from Threestudio

Project: Image to 3D

- **2D priors with Score Distillation Sampling** [DreamFusion]
- **2D prior Improvements**
  - Higher resolution [Magic3D]
  - Richer appearance [Fantasia3D]
  - Faster speed [DreamGaussian]
  - Single-view 3D [RealFusion] [NeRDi] [NeuralLift-360]
  - Photorealistic appearance [NFSD][ProlificDreamer]
- **3D priors**
  - View-conditioned diffusion [Zero-1-to-3] [Magic-123] [DreamCraft3D]
  - Multi-view diffusion [SyncedDreamer] [MVDreamer] [Consistent-1-to-3] [Zero-1-to-3++]
  - View-conditioned geometry diffusion [Wonder3D] [SweetDreamer]
- **Generalizable methods** (without per-instance optimization)
  - Single-image 3D [LRM]
  - Multi-view diffusion + LRM [Instant3D]
  - LRM for multi-view diffusion [DMV3D]



- On evaluation dataset
- At least **three** open-source methods
- At least **two** close-source methods
- Try different hyperparameters
- Summarize some conclusions

Thank you for listening!