Михаил Степнов

Директор по AdTech BigData МТС

Руководит разработкой Data-продуктов для рекламного бизнеса МТС

Ex-Исполнительный директор SBER-AI

Руководил разработкой ruGPT-3, ruDALL-E и продуктов на их основе

Преподаватель НИУ ВШЭ

Что можно генерировать?

Текст

Изображения

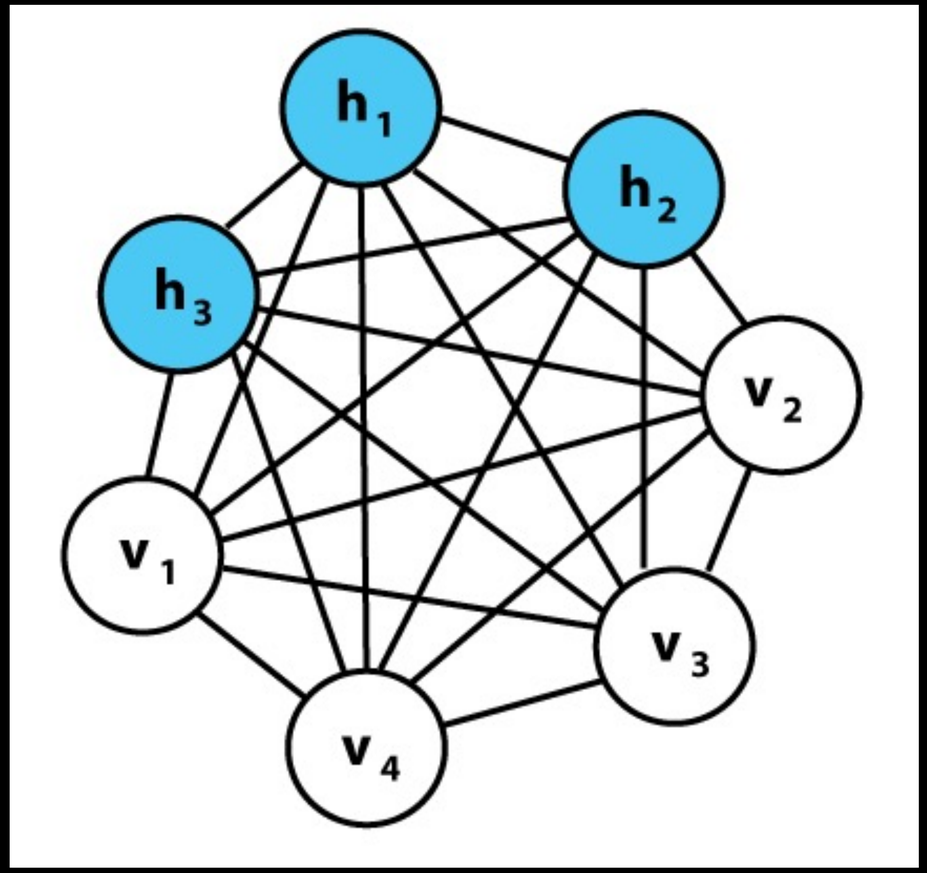
Что можно генерировать?

Видео

Аудио

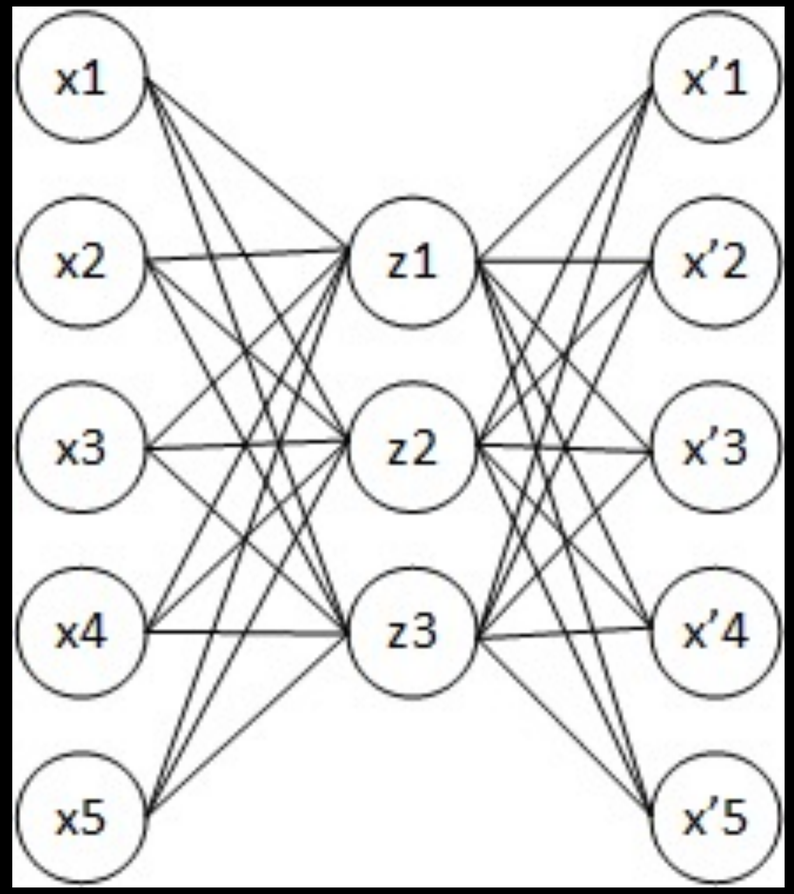
Машины Больцмана

Первая нейронная сеть, способная обучаться внутренним представлениям, решать сложные комбинаторные задачи



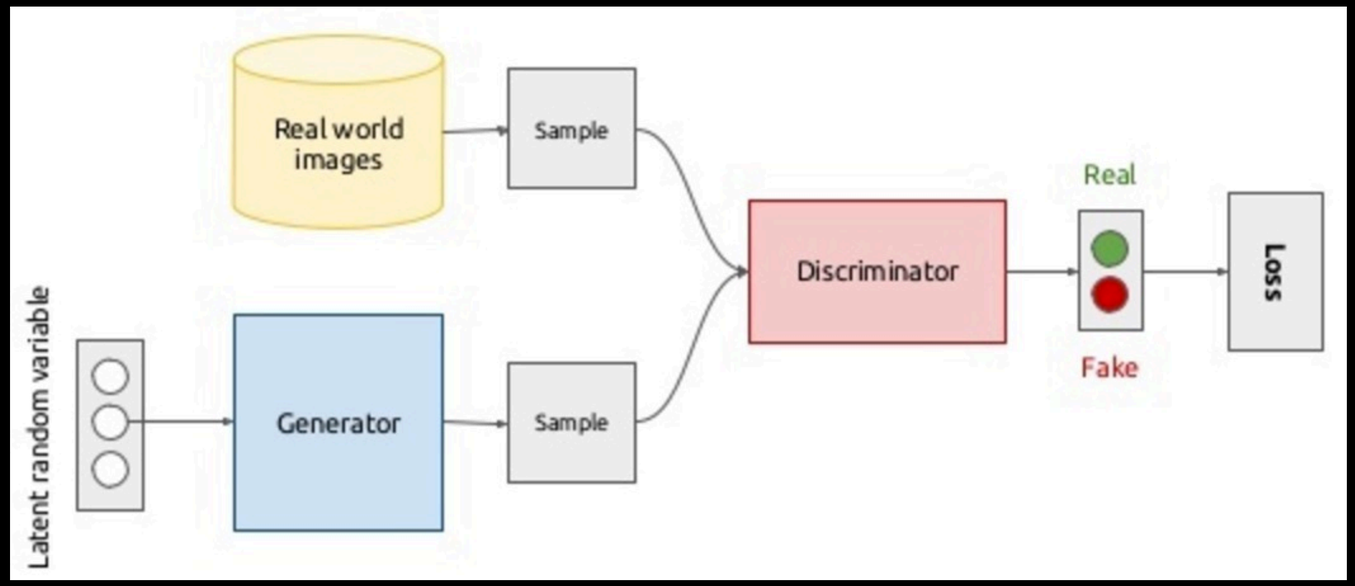
Автоэнкодеры

Специальная архитектура искусственных нейронных сетей, позволяющая применять обучение без учителя[2] при использовании метода обратного распространения ошибки



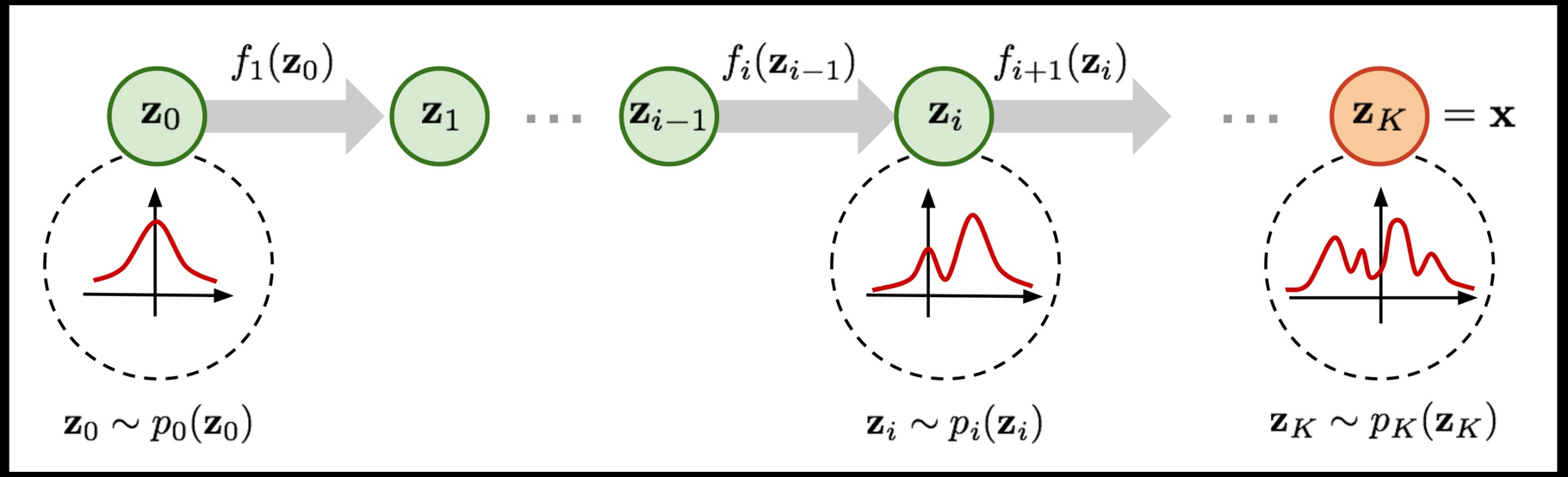
GAN — генеративные состязательные сети

GAN — это модель машинного обучения, в которой две нейронные сети соревнуются друг с другом, чтобы быть более точными в своих прогнозах



Генеративные модели на основе потоков

Модель, используемая в машинном обучении, которая явно моделирует распределение вероятностей, используя нормализующий поток (статистический метод, использующий закон изменения вероятностей для преобразования простого распределения в сложное)



Диффузионные нейронные сети

Диффузионные модели - подкатегория глубоких генеративных моделей, которые состоят из этапов прямой и обратной диффузии, генерируют данные, аналогичные тем, на которых они обучаются.

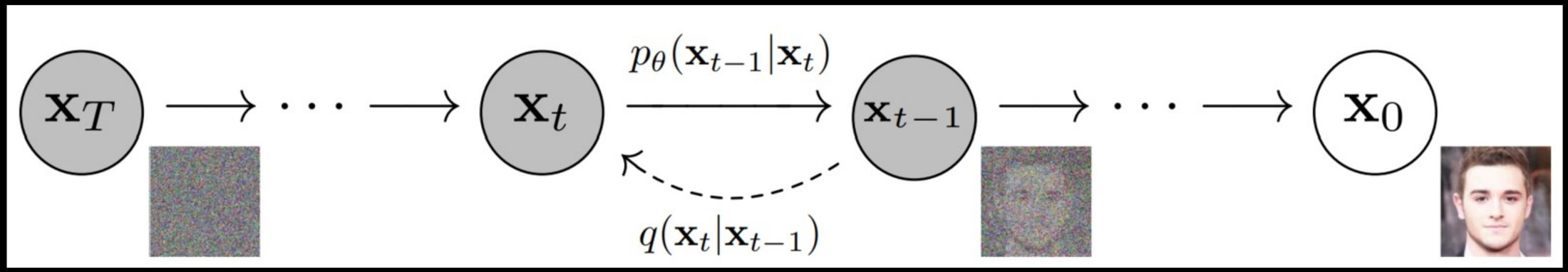
stability.ai

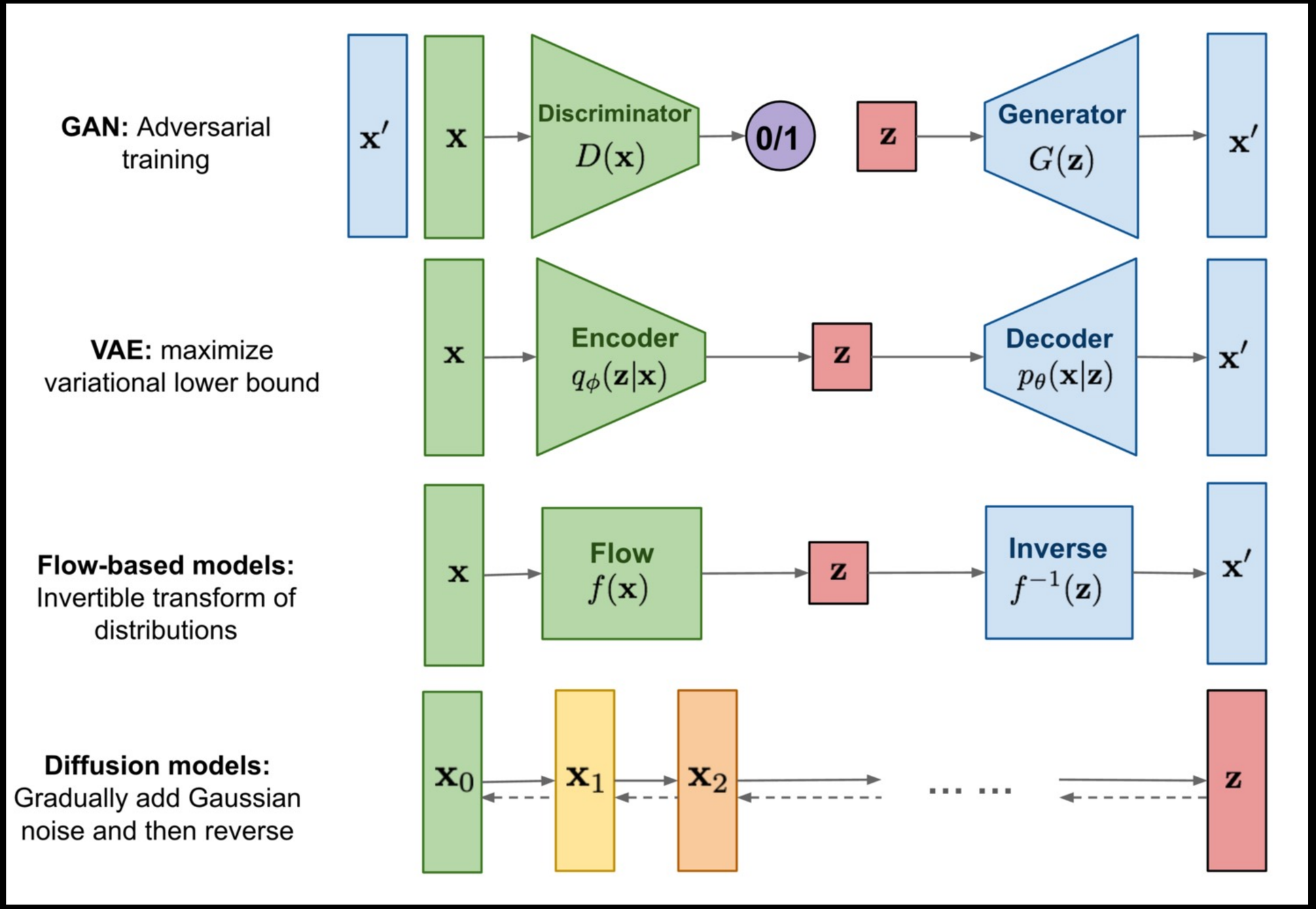
DALL·E 2

Midjourney

Imagen

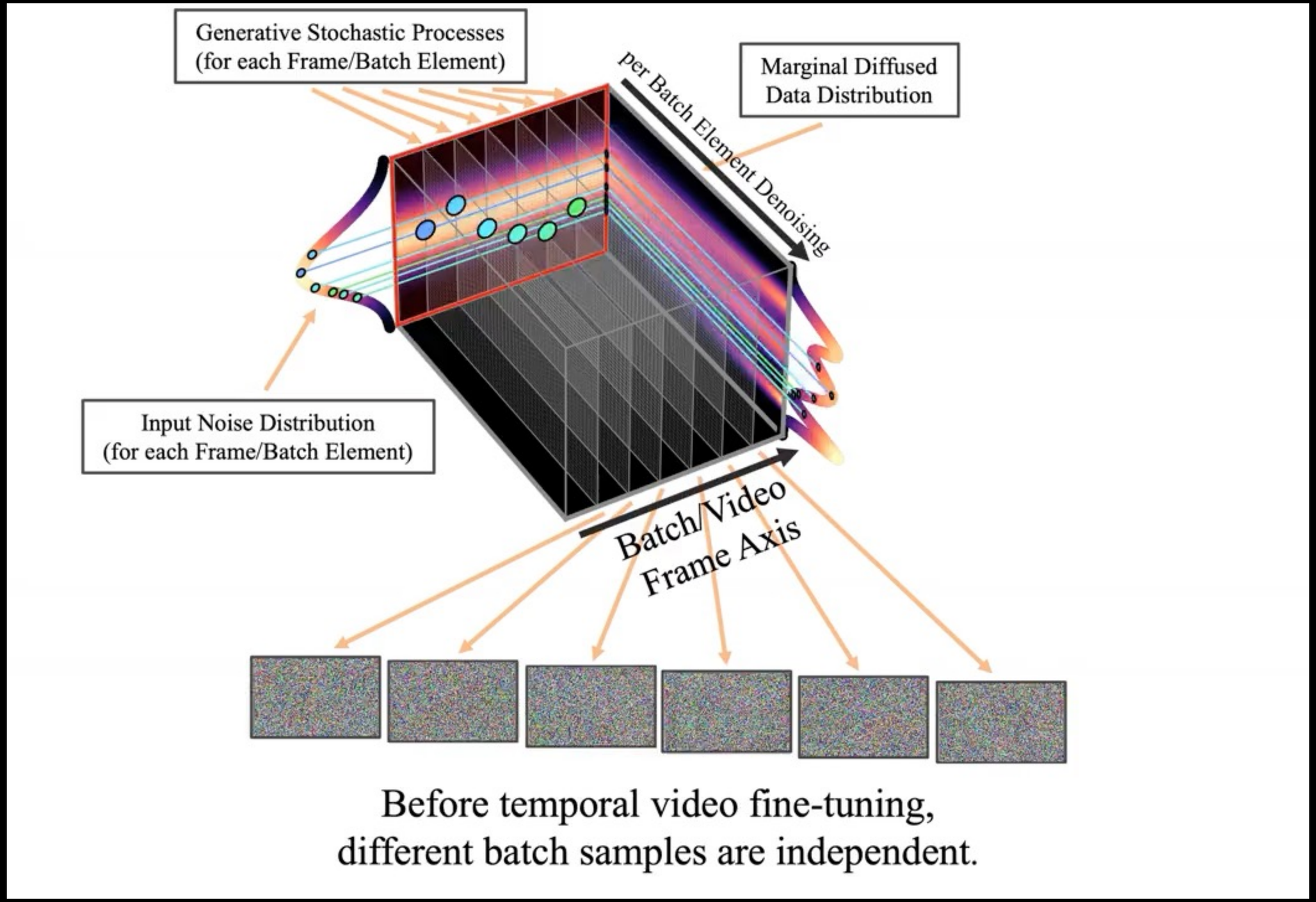
Kandinsky 2.2




GAN: Adversarial training
$\mathbf{x}'$
$\mathbf{x}$
Discriminator
$D(\mathbf{x})$
0/1
$\mathbf{z}$
Generator
$G(\mathbf{z})$
$\mathbf{x}'$
VAE: maximize variational lower bound
$\mathbf{x}$
Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$
Flow-based models: Invertible transform of distributions
$\mathbf{x}$
Flow
$f(\mathbf{x})$
$\mathbf{z}$
Inverse
$f^{-1}(\mathbf{z})$
$\mathbf{x}'$
Diffusion models: Gradually add Gaussian noise and then reverse
$\mathbf{x}_0$
$\mathbf{x}_1$
$\mathbf{x}_2$
... ...
$\mathbf{z}$

Генерация видео

VideoLDM



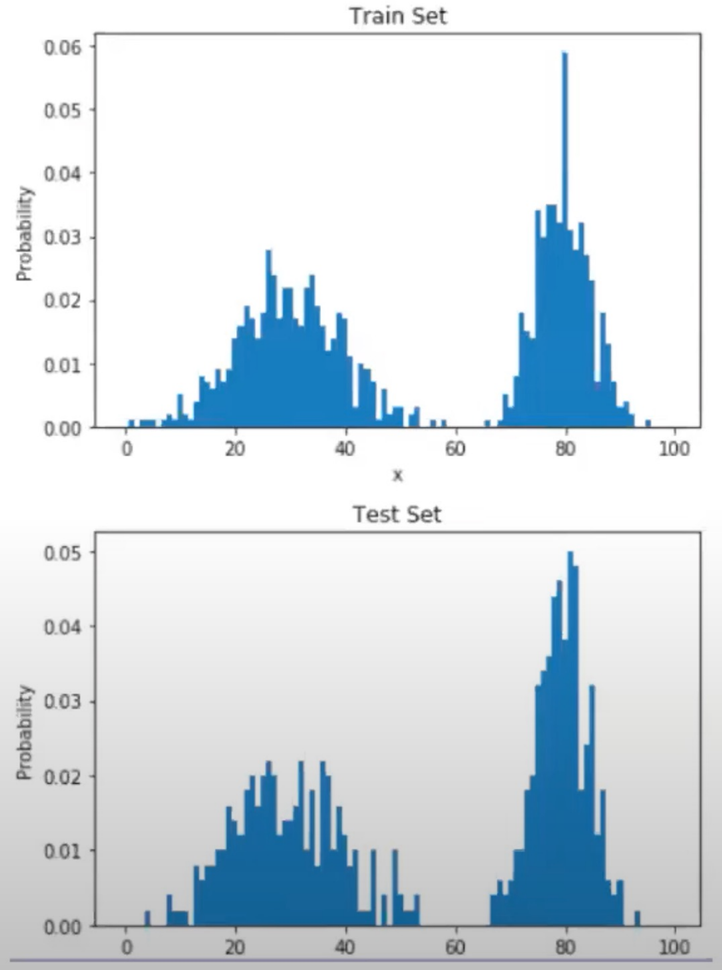
Before temporal video fine-tuning,
different batch samples are independent.

Diffusion Models

Density Modeling for Data Synthesis

Assume that all data comes from a distribution $p_{data}(x)$:

- The goal of generative machine learning models is to *learn* this distribution to the best of their ability — the distribution approximated by the model is denoted as $p_\theta(x)$
- We generate new data by *sampling* from the learned distribution
- In practice, train models to maximize the expected log likelihood of $p_\theta(x)$ (or minimizing negative log likelihood)/minimize divergence between $p_\theta(x)$ and $p_{data}(x)$



Prior Methods

VAE:

$$
\begin{aligned}
L_{\mathrm{VAE}}(\theta, \phi) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\
&= -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \\
\theta^*, \phi^* &= \arg\min_{\theta, \phi} L_{\mathrm{VAE}}
\end{aligned}
$$

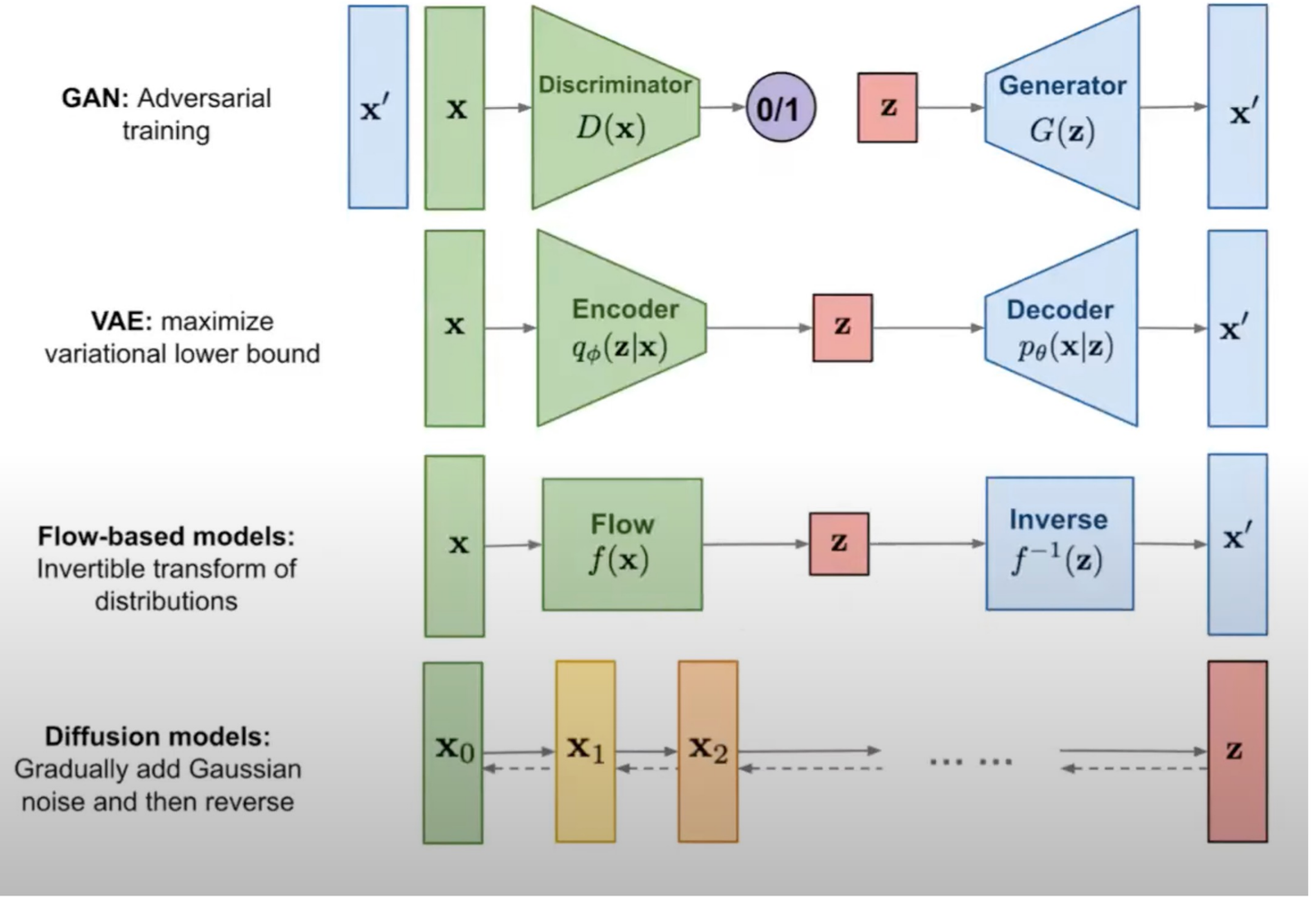
$$
\boxed{-L_{\mathrm{VAE}} = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x})}
$$

GAN:

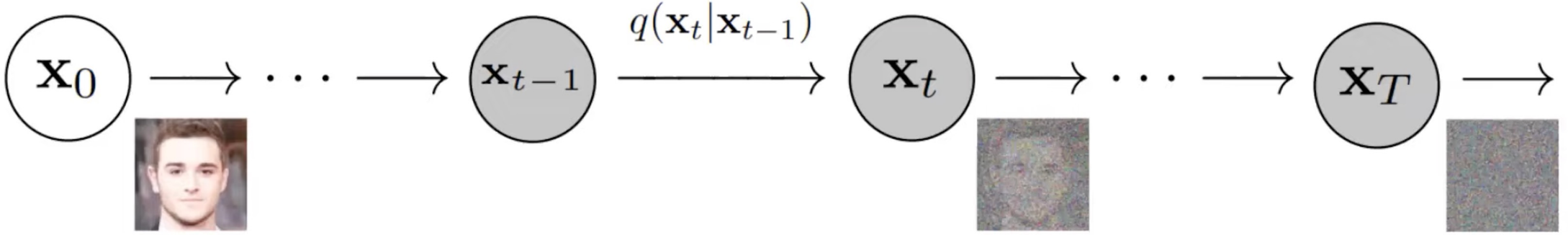
$$
\begin{aligned}
\min_G \max_D L(D, G) &= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \\
&= \mathbb{E}_{x \sim p_r(x)}[\log D(x)] + \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x)]
\end{aligned}
$$

$$
\boxed{L(G, D^*) = 2D_{JS}(p_r \| p_g) - 2\log 2}
$$

Sampling from Noise



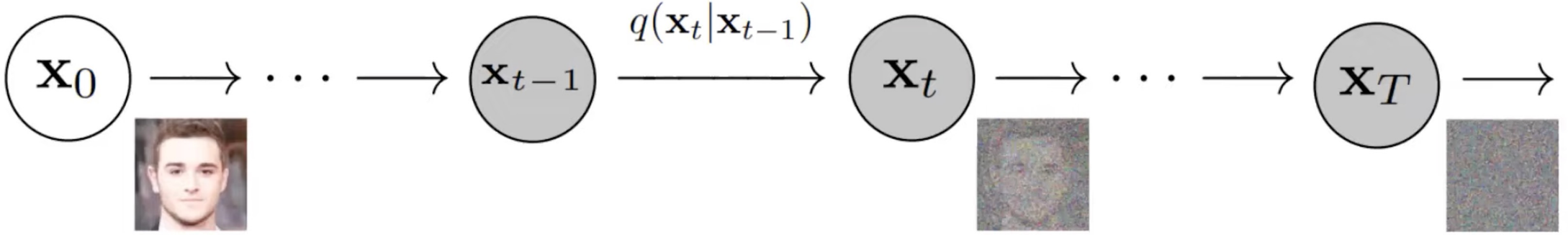
Forward Process



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad \{\beta_t \in (0,1)\}_{t=1}^{T}$$

- Take a datapoint x_0 and keep gradually adding very small amounts of Gaussian noise to it
  - Vary the parameters of the Gaussian according to a *noise schedule* controlled by beta_t
- Repeat this process for T steps — as the timesteps increase, the more features of the original input are destroyed
- You can prove with some math that as T approaches infinity, you eventually end up with an Isotropic Gaussian (i.e. pure random noise)

Forward Process



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad \{\beta_t \in (0,1)\}_{t=1}^{T}$$

- Take a datapoint x_0 and keep gradually adding very small amounts of Gaussian noise to it
  - Vary the parameters of the Gaussian according to a *noise schedule* controlled by beta_t
- Repeat this process for T steps — as the timesteps increase, the more features of the original input are destroyed
- You can prove with some math that as T approaches infinity, you eventually end up with an Isotropic Gaussian (i.e. pure random noise)

A neat (reparametrization) trick!

Define

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

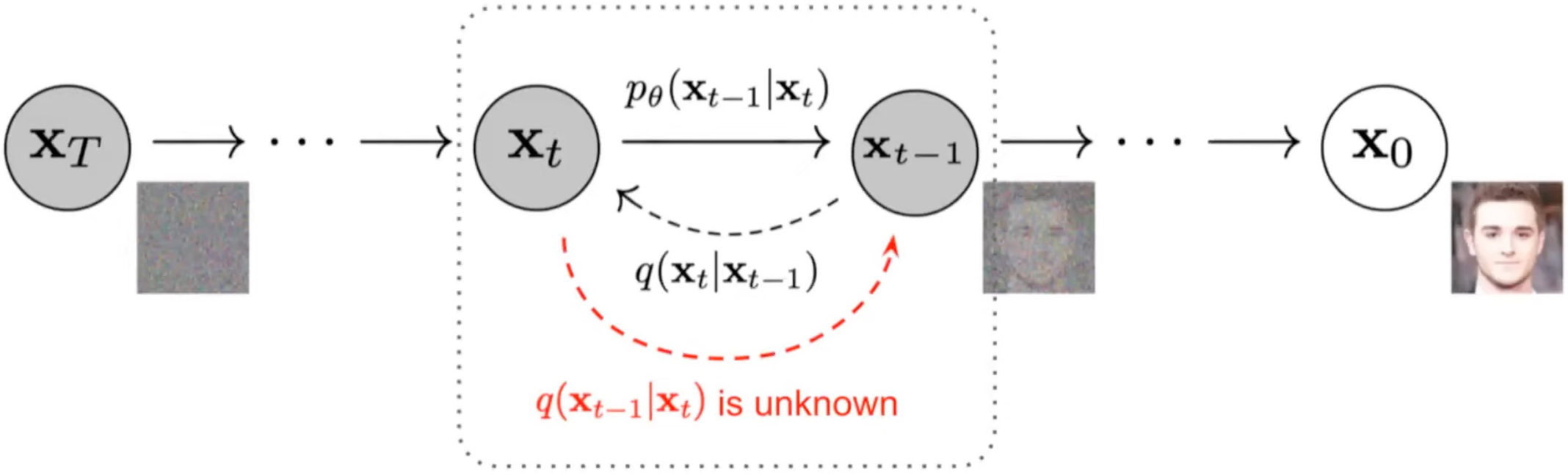
Then

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

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{1-\beta_t}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \\
&= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\epsilon \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon
\end{aligned}$$

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0,\ (1-\bar{\alpha}_t)\mathbf{I}\right)$$

Reverse Process



- The goal of a diffusion model is to **learn** the reverse *denoising* process to iteratively **undo** the forward process
- In this way, the reverse process appears as if it is generating new data from random noise!

Finding the exact distribution is hard

$$f(\theta \mid x) = \frac{f(\theta, x)}{f(x)} = \frac{f(\theta)\, f(x \mid \theta)}{f(x)} \quad \Longrightarrow \quad q(x_{t-1} \mid x_t) = q(x_t \mid x_{t-1}) \frac{q(x_{t-1})}{q(x_t)}$$

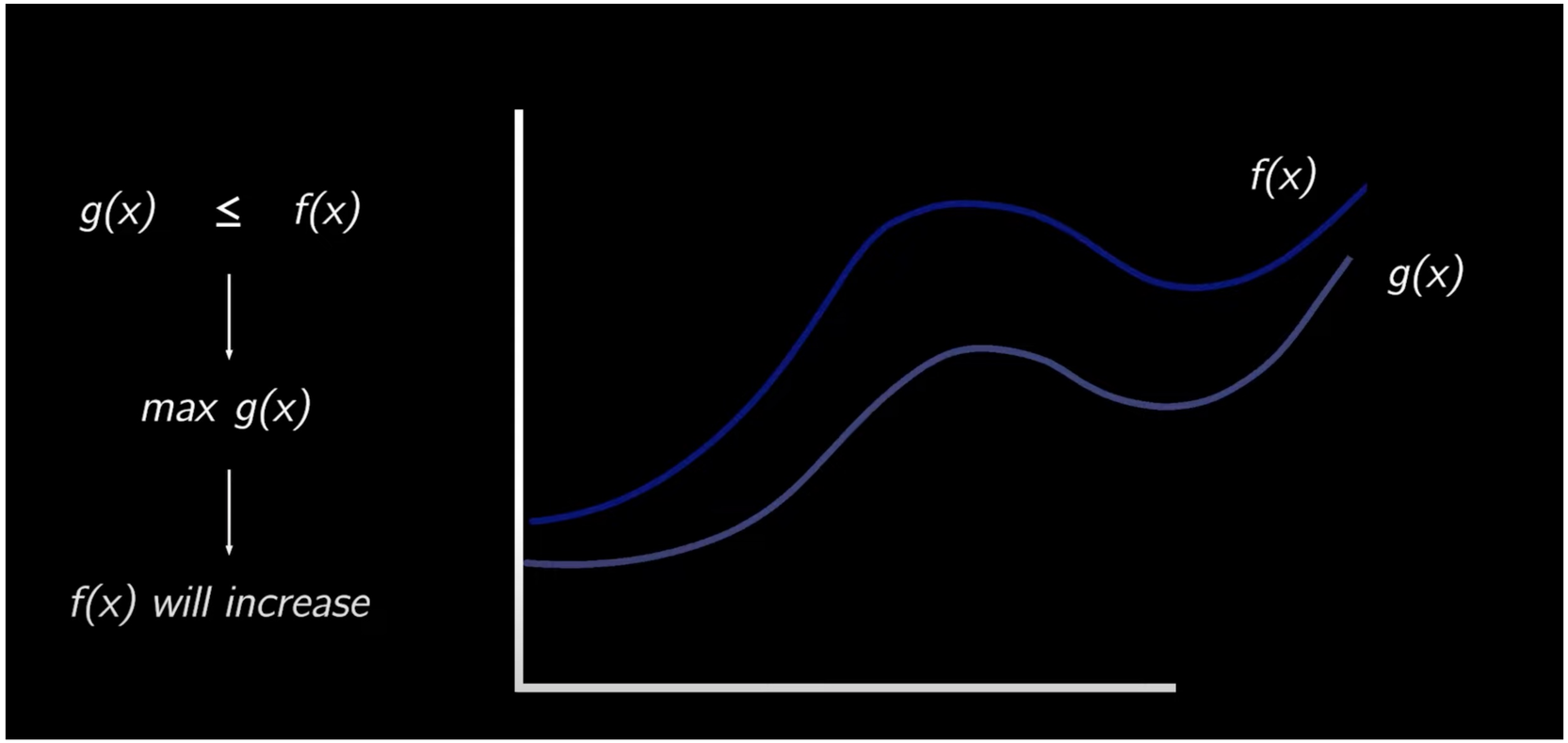
$$q(x_t) = \int q(x_t \mid x_{t-1}) q(x_{t-1}) \mathrm{d}x$$

The distribution of each timestep and q(x_t | x_{t-1}) depends on the entire data distribution:

- Computing this is computationally intractable (where else have we seen this dilemma?)
- However, we still need those to describe the reverse process. Can we approximate them somehow?

Оставим математику профессионалам:

Variational Lower Bound



Training

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on

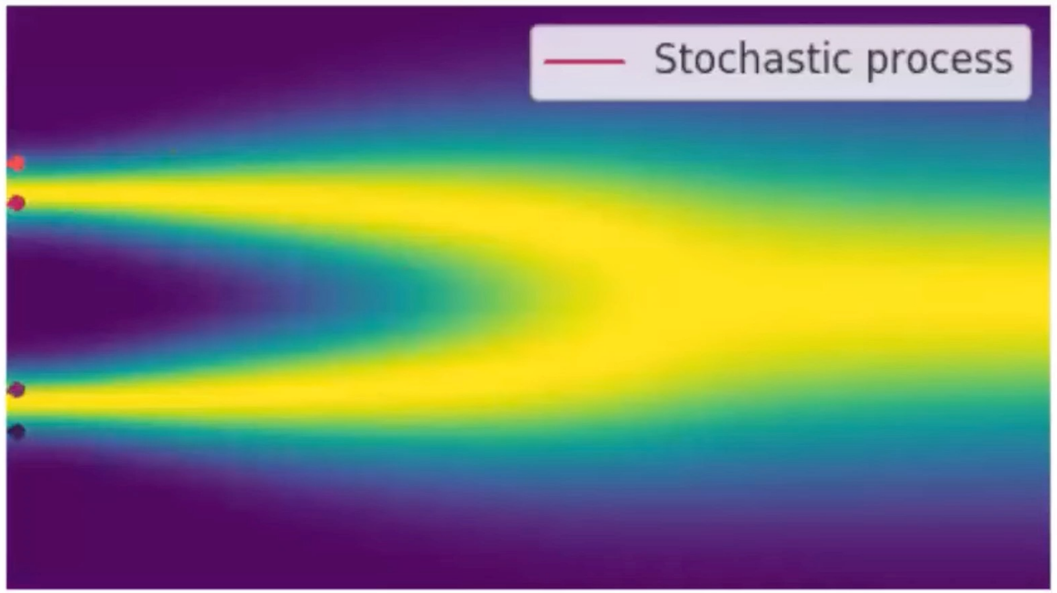
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$

6: **until** converged

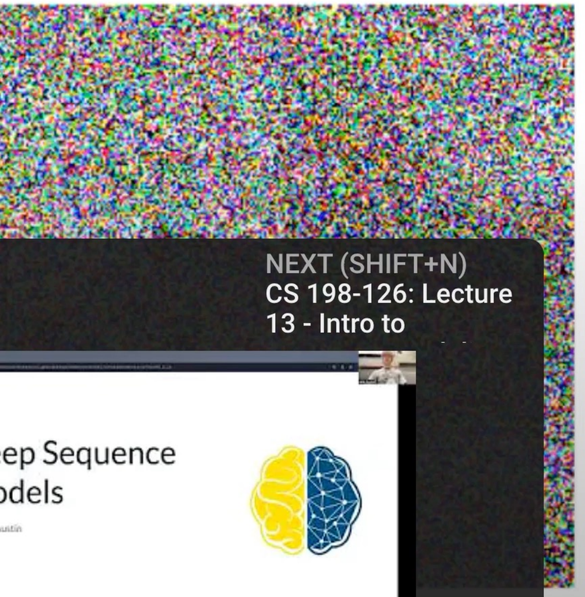
Sampling

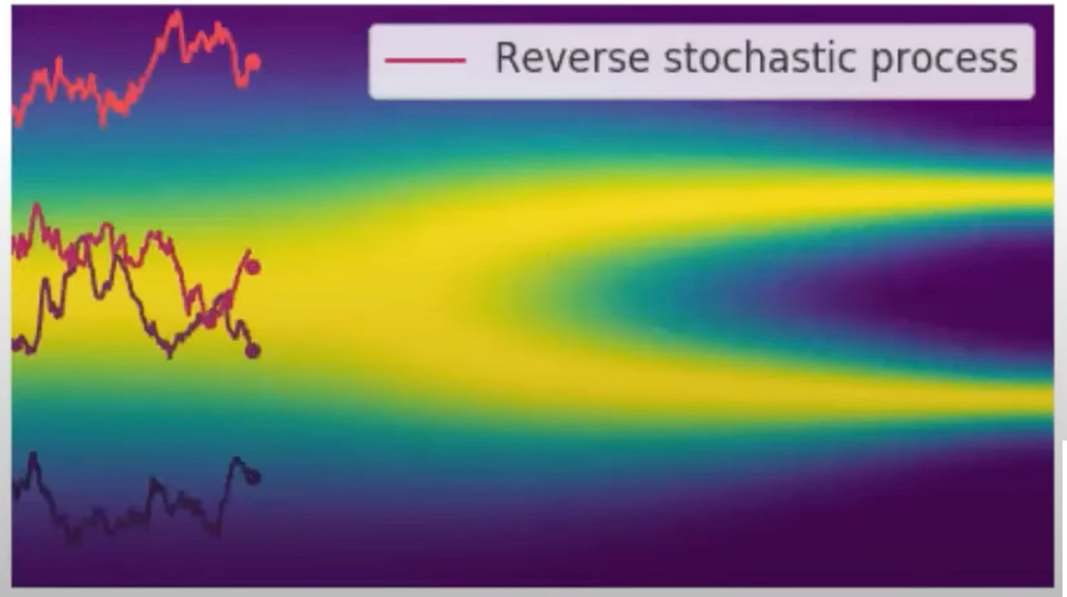
Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$



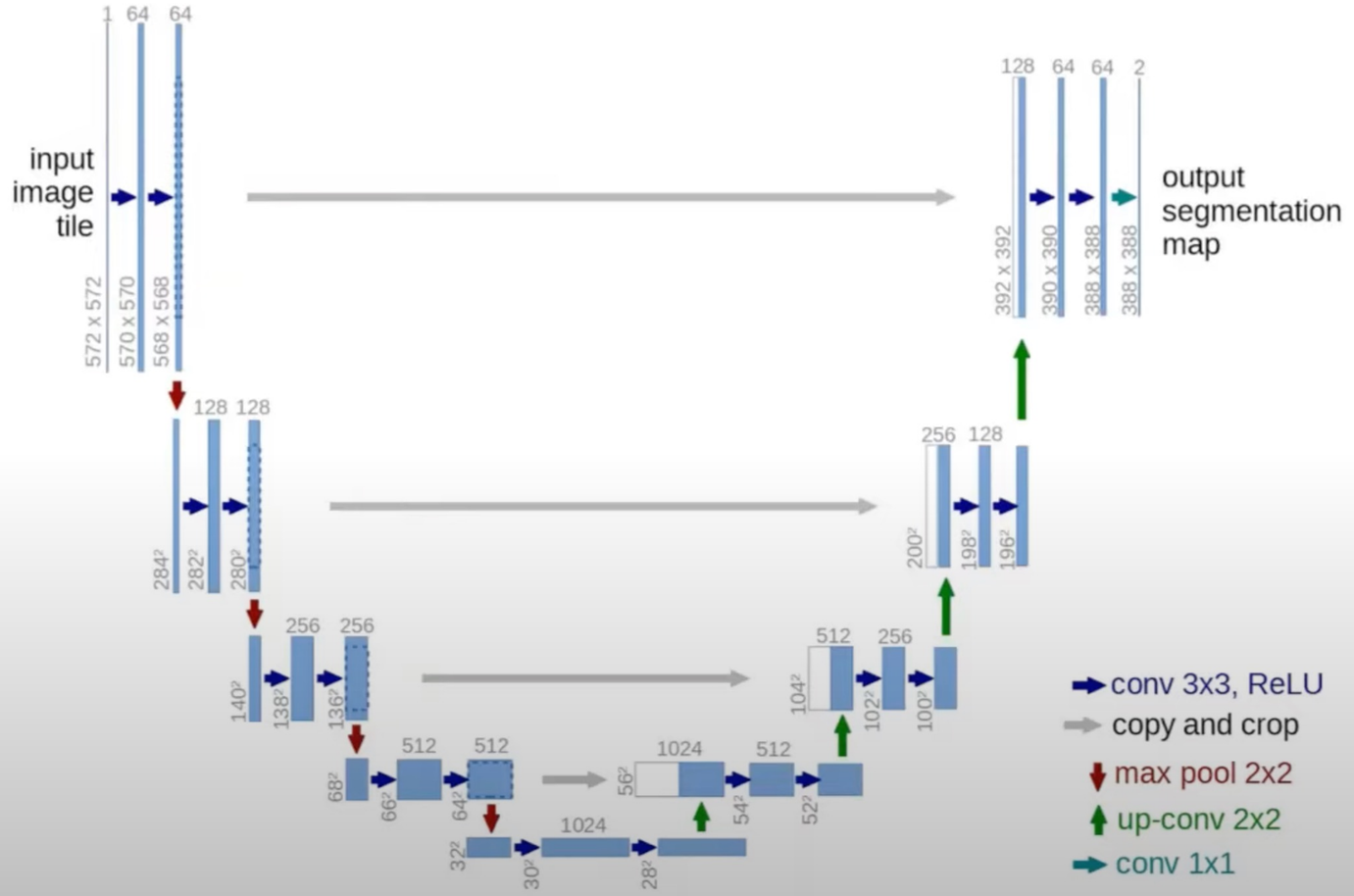
Forward process: converting the image distribution to pure noise





Reverse process: sampling from the image distribution, starting with pure noise

UNet + Other Stuff



Diffusion models typically use a U-Net on steroids as the noise predictive model — you take the good ol' model that you are already familiar with and add:

- Positional Embeddings
- ResNet Blocks
- ConvNext Blocks
- Attention Modules
- Group Normalization
- Swish and GeLU

It's a massive kitchen sink of modern CV tricks

Linear vs Cosine Schedule

- A linear noise schedule converts initial data to noise really quickly, making the reverse process harder for the model to learn.
- Researchers hypothesized that a cosine-like function that is changing relatively gradually near the endpoints might work better
    - Note: It did end up working better but this choice of cosine was completely arbitrary

Linear (top) vs Cosine (bottom)

Architecture Improvements

Nichol and Dhariwal proposed several architectural changes that seem to help diffusion training:

1. Increasing model depth vs width (not both): both help but increasing width is computationally cheaper while providing similar gains as increased depth
2. Increasing number of attention heads and applying it to multiple resolutions
3. Stealing BigGAN residual blocks for upsampling and downsampling
4. *Adaptive Group Normalization* — hopes to better incorporate timestep (and potentially class) information during the training/reverse process