# Recent neural architectures for vision II: Generative models

AI602: Recent Advances in Deep Learning

Lecture 2

Jinwoo Shin

KAIST AI

## Table of Contents

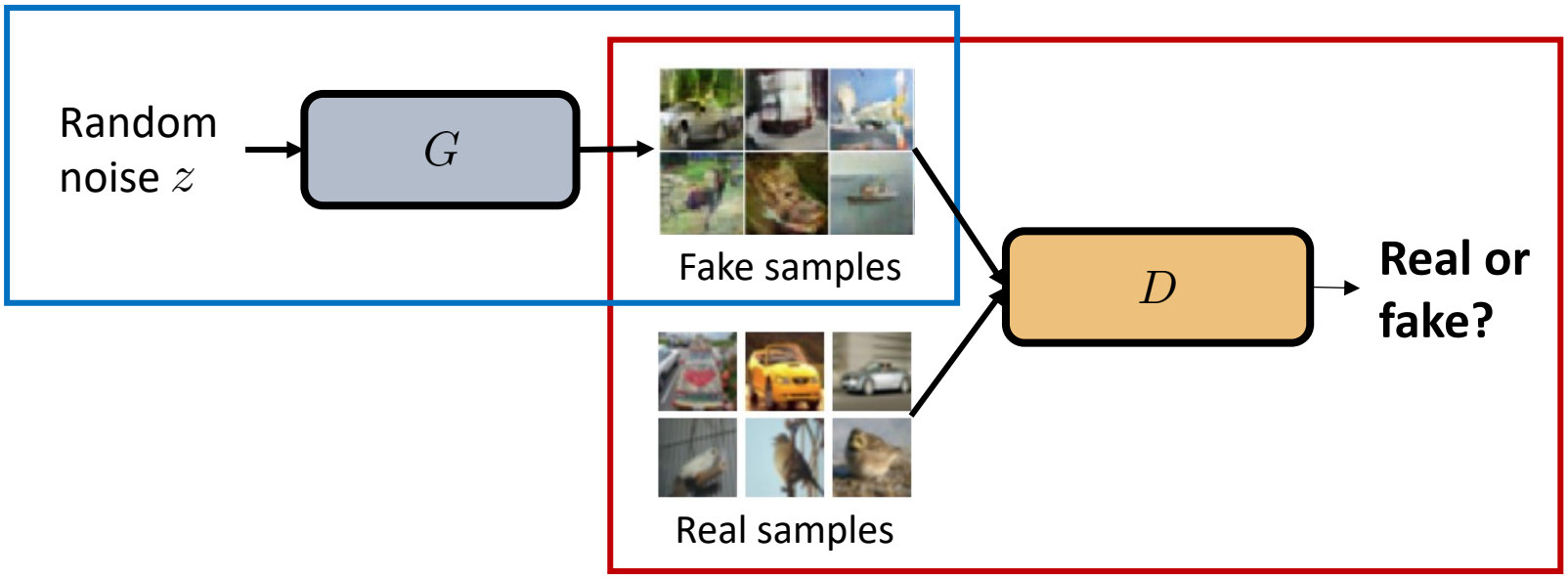
## Table of Contents

Classical (usually explicit) generative methods struggle on complex data

- Sampling from **high-dimensional, complex** distributions can be intractable

GANs [Goodfellow, et. al., 2014] do not explicitly model $p_{\text{model}}(\mathbf{x})$

- **Two player game** between discriminator network $D$ and generator network $G$
- $D$ **tries to discriminate** real data and samples generated by $G$ ("fake" samples)
- $G$ **tries to fool** $D$ by generating more "realistic" images
- GAN utilizes **neural networks** to model the sampling function itself

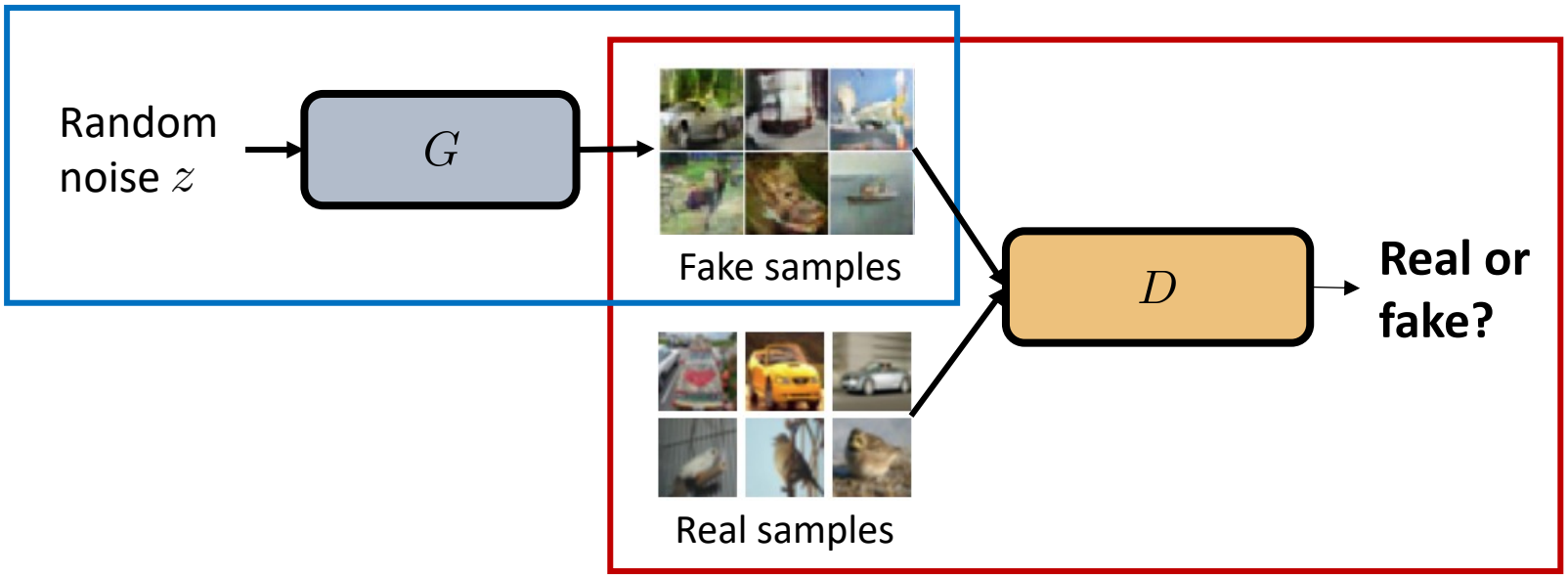


**Two player game** between discriminator network $D$ and generator network $G$

**Training objective**:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data}} + \mathbb{E}_{z \sim p_z} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data}}) \right]$$

- $D$ maximizes the objective: $D(x) \to 1$ and $D(G(z)) \to 0$
- $G$ minimizes the objective: $D(G(z)) \to 1$



**Early GANs suffer from several limitations:**

**Limitation 1**. Limited scalability with respect to data dimension

- GANs had been difficult to be trained on high-resolution data
- Will introduce recent crucial works to mitigate this problem (with controllability)

**Limitation 2**. Limited scalability with respect to dataset complexity

- Due to "mode collapse" problem, had been difficult to trained on complex data
- Will introduce recent works to scale-up GANs (even zero-shot text-to-image scale)

**Limitation 3**. Training instability

- GAN training had been extremely unstable due to bi-level training objective
- Will introduce recent simple yet effective techniques to solve this issue

**Early GANs suffer from several limitations:**

**Limitation 1**. Limited scalability with respect to data dimension

- GANs had been difficult to be trained on high-resolution data
- Will introduce recent crucial works to mitigate this problem (with controllability)

**Limitation 2**. Limited scalability with respect to dataset complexity

- Due to "mode collapse" problem, had been difficult to trained on complex data
- Will introduce recent works to scale-up GANs (even zero-shot text-to-image scale)

**Limitation 3**. Training instability

- GAN training had been extremely unstable due to bi-level training objective
- Will introduce recent simple yet effective techniques to solve this issue

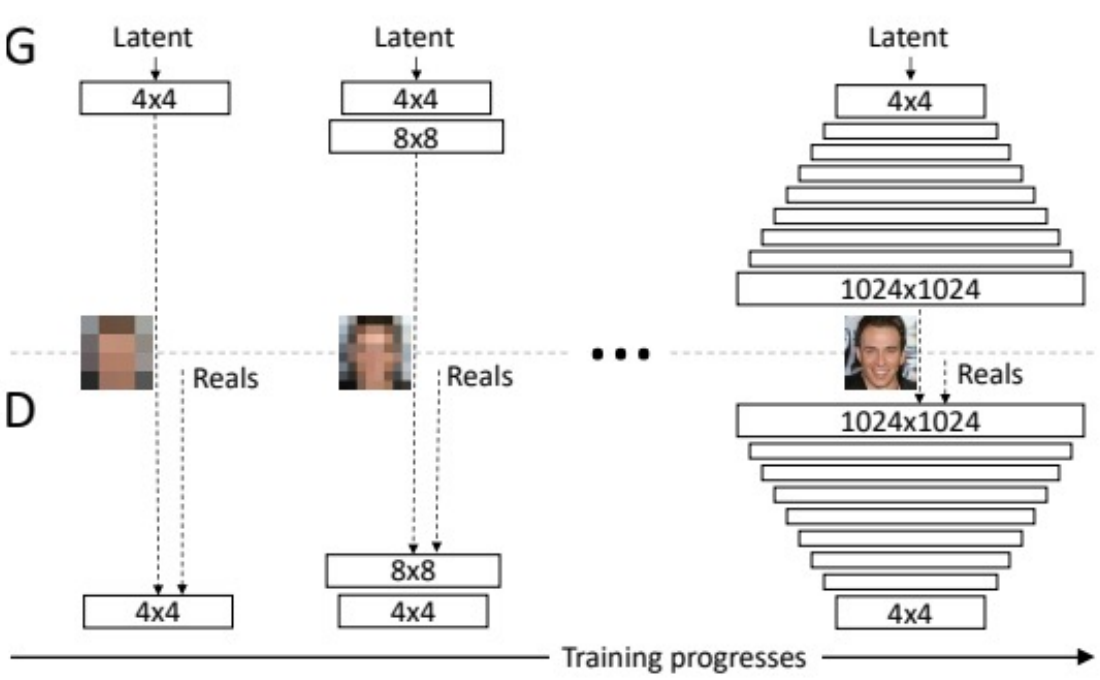
# Progressive GAN: High-Resolution Image Generation [Karras et al., 2018]

Early GANs could produce sharp images, but only at small resolutions

- It was still unstable on higher-resolution training despite some progress

Karras et al. (2018): **Progressive growing of G and D** (Progressive-GAN)

- Training GANs to directly generate high-res image might be too difficult!
- Progressive-GAN starts from learning low-resolution images
- It adds new layers to G and D during training for up-scaling into higher-resolution

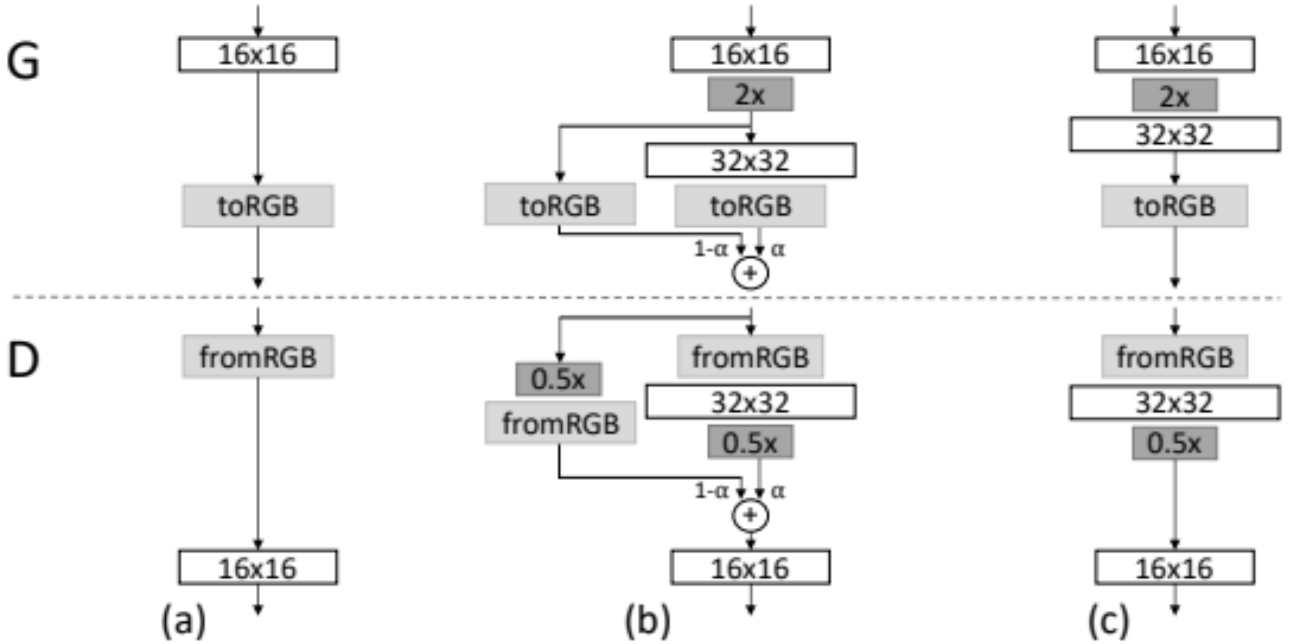





* source : Karras, et. al., Progressive growing of GANs for improved quality, stability, and variation, ICLR 2018

**Smooth fade-in** to the new layers during up-scale training

- To prevent "sudden shocks" to the pre-trained smaller-resolution layers
- **Example**: Upscaling transition **(b)** from 16 × 16 to 32 × 32 (**(a)** → **(c)**)




- Simply treat the higher resolution like a residual block
- The fade-in weight α increases linearly from 0 to 1 during training



* source : Karras, et. al., Progressive growing of GANs for improved quality, stability, and variation, ICLR 2018

# Progressive GAN: High-Resolution Image Generation [Karras et al., 2018]

1024x1024 images generated using the CELEBA-HQ dataset

https://www.youtube.com/watch?v=G06dEcZ-QTg&feature=youtu.be

Mao et al. (2016b) (128 × 128) Gulrajani et al. (2017) (128 × 128) Our (256 × 256)

Visual quality comparison: LSUN bedroom

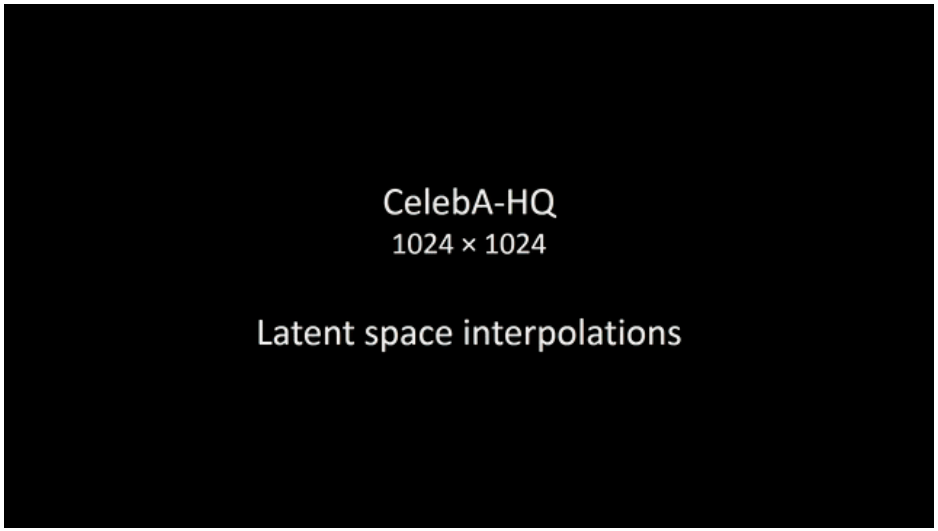
LSUN other categories generated image (256x256)



* source : Karras, et. al., Progressive growing of GANs for improved quality, stability, and variation, ICLR 2018

Interpolation on the latent space of GAN yields smooth, but non-linear changes

- Features not in both end-points appear along the interpolation path



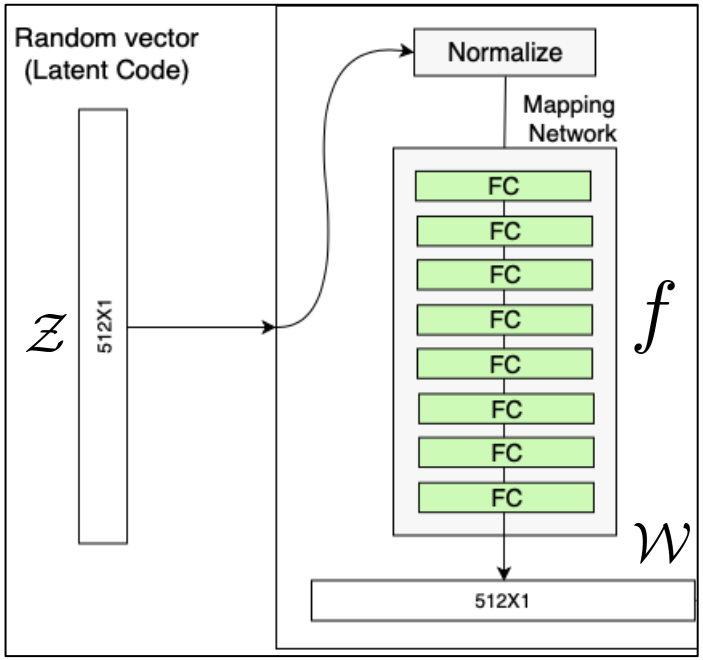

Latent space interpolations with Progressive GAN

**Unavoidable entanglement:** Input space must follow density of the training data

Karras et al. (2019): Consider intermediate latent space representing a "style"

StyleGAN proposes to use a **non-linear mapping network** $f : \mathcal{Z} \rightarrow \mathcal{W}$

- Implemented using an 8-layer fully-connected neural network



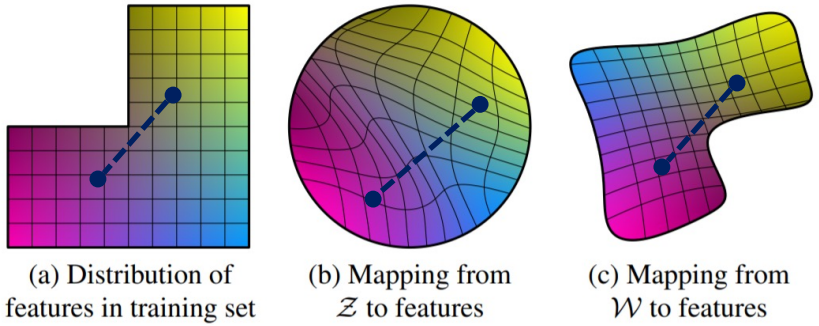




Illustration of disentanglement

Mapping from $\mathcal{Z}$ (input space) to meaningful features directly is too complex

Mapping from $\mathcal{W}$ (intermediate) to such features can be more simpler



* source : Rani Horev's blog

**Adaptive instance normalization** (AdaIN)

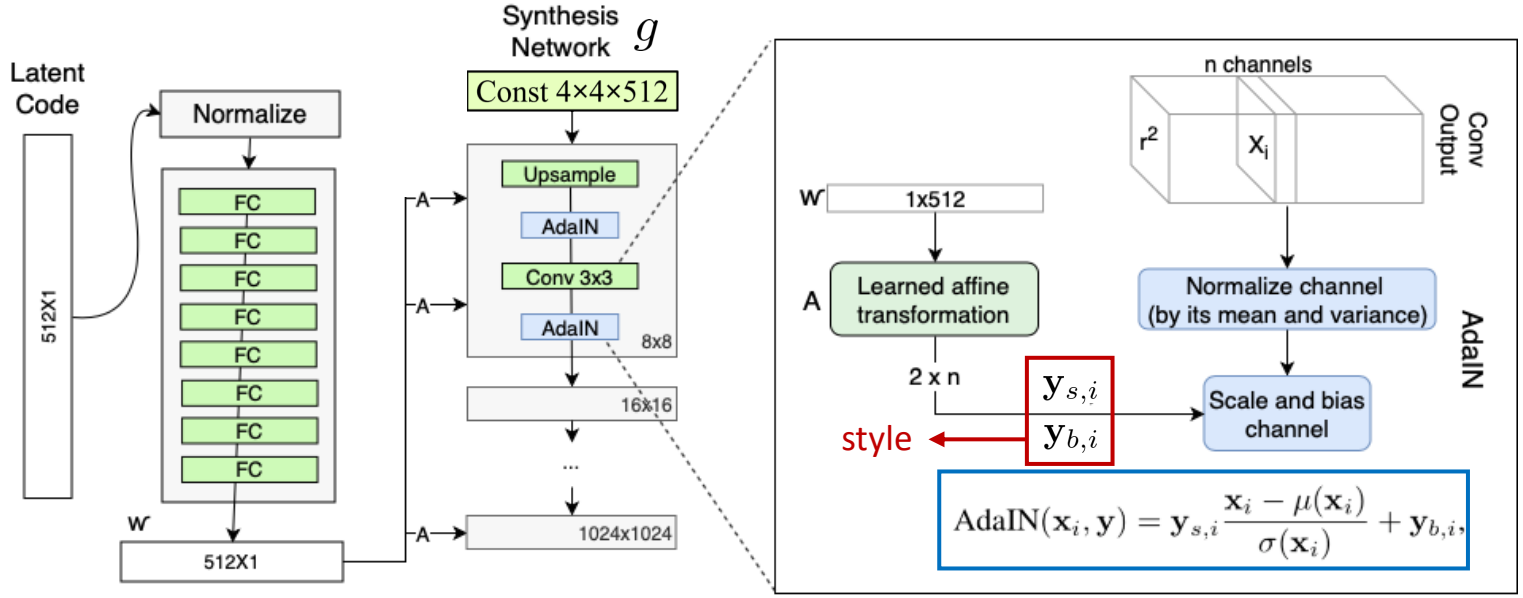
- Motivated by the instance normalization [Huang et al., 2017]

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

**$\mathbf{y} = (\mathbf{y_s}, \mathbf{y_b})$ is called by a "style"**

- A learned affine-transformation of $w \in \mathcal{W}$
- Can control high-level attributes (e.g., pose, identity of face images)

Applied after all the convolutional layer in the **synthesis network** $g$




* source : Rani Horev's blog

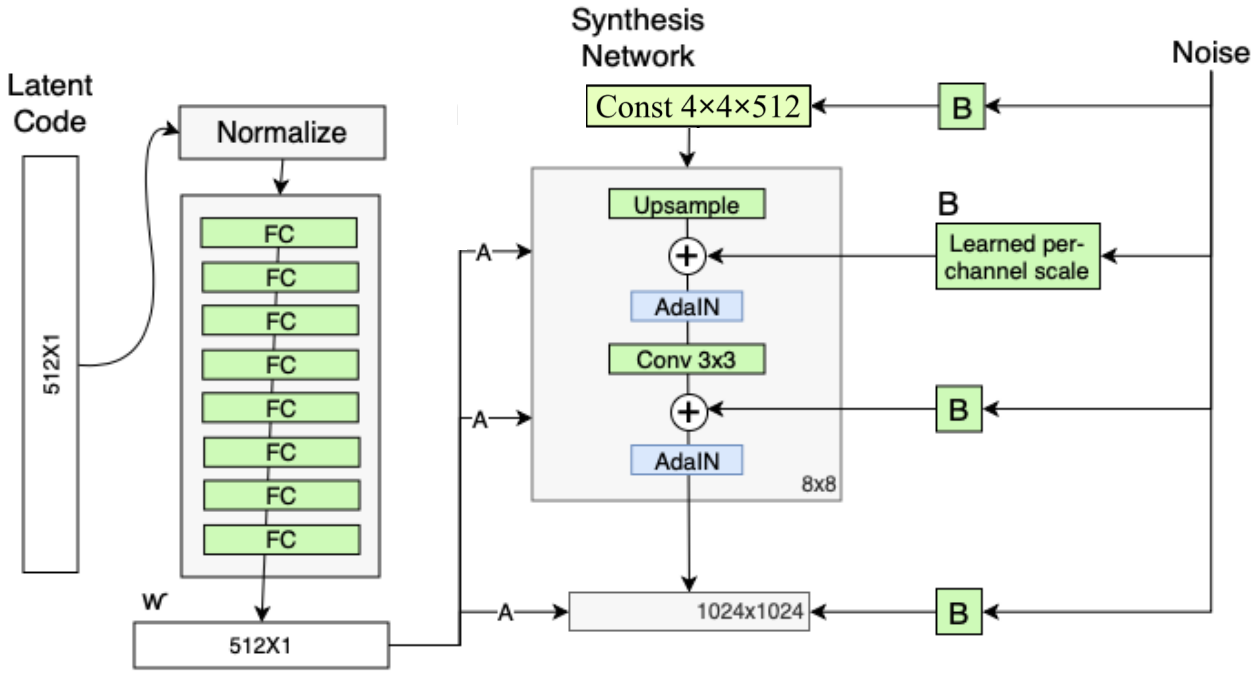
**Explicit noise inputs for stochastic variation**

- Single-channel images of Gaussian noise
- Aims to control the stochastic details, e.g., freckles, hair of face images

A noise channel $n$ is fed to every layer of the synthesis network $g$

- Broadcasted across features with learned per-feature scaling factors $B$

$$s(x_i, n) = x_i + B_i \cdot n$$



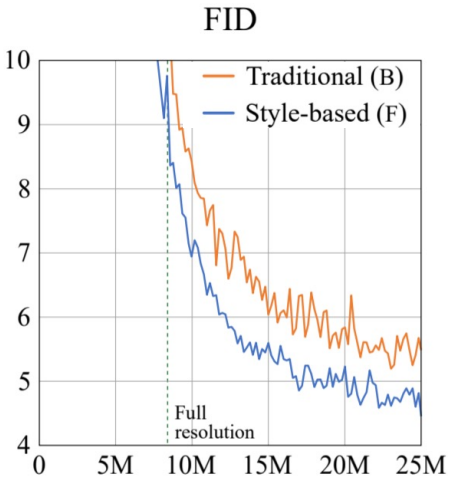



* source : Rani Horev's blog

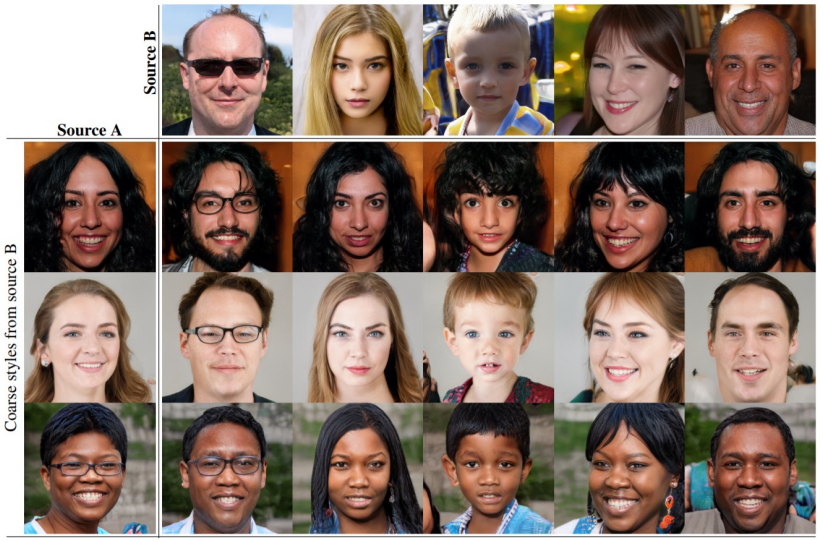
# StyleGAN: A Style-Based Generator Architecture [Karras et al., 2019]

StyleGAN improves state-of-the-art in terms of FID

| | Method | CelebA-HQ | FFHQ |
|---|---|---|---|
| A | Baseline Progressive GAN [30] | 7.79 | 8.04 |
| B | + Tuning (incl. bilinear up/down) | 6.11 | 5.25 |
| C | + Add mapping and styles | 5.34 | 4.85 |
| D | + Remove traditional input | 5.07 | 4.88 |
| E | + Add noise inputs | **5.06** | 4.42 |
| F | + Mixing regularization | 5.17 | **4.40** |




Better interpolation properties, and disentangles the latent factors of variation



Karras et al. (2020a): Some buggy-artifacts in StyleGAN samples

- **Blob-shaped artifacts** found in most of StyleGAN images (and hidden features)

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

**StyleGAN2** includes several design modifications on StyleGAN to address the issue

| Configuration | FFHQ, 1024×1024 | | | | LSUN Car, 512×384 | | | |
|---|---|---|---|---|---|---|---|---|
| | FID ↓ | Path length ↓ | Precision ↑ | Recall ↑ | FID ↓ | Path length ↓ | Precision ↑ | Recall ↑ |
| A Baseline StyleGAN [24] | 4.40 | 212.1 | **0.721** | 0.399 | 3.27 | 1484.5 | **0.701** | 0.435 |
| B + Weight demodulation | 4.39 | 175.4 | 0.702 | 0.425 | 3.04 | 862.4 | 0.685 | 0.488 |
| C + Lazy regularization | 4.38 | 158.0 | 0.719 | 0.427 | 2.83 | 981.6 | 0.688 | 0.493 |
| D + Path length regularization | 4.34 | **122.5** | 0.715 | 0.418 | 3.43 | 651.2 | 0.697 | 0.452 |
| E + No growing, new G & D arch. | 3.31 | 124.5 | 0.705 | 0.449 | 3.19 | 471.2 | 0.690 | 0.454 |
| F + Large networks (StyleGAN2) | **2.84** | 145.0 | 0.689 | **0.492** | **2.32** | **415.5** | 0.678 | **0.514** |
| Config A with large networks | 3.98 | 199.2 | 0.716 | 0.422 | – | – | – | – |

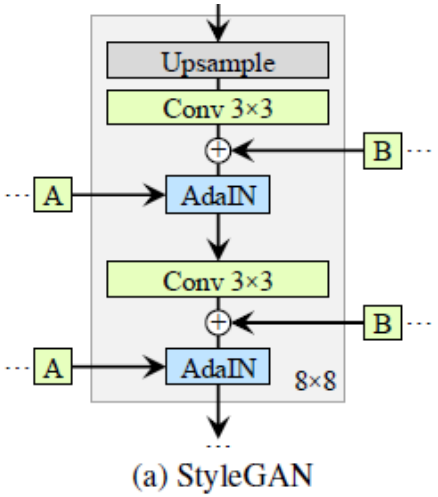
**Blob-shaped artifacts** found in most of StyleGAN images (and hidden features)

1. The anomaly starts to appear around 64×64 resolution
2. It becomes progressively stronger at higher resolutions

Figure 1. Instance normalization causes water droplet -like artifacts in StyleGAN images. These are not always obvious in the generated images, but if we look at the activations inside the generator network, the problem is always there, in all feature maps starting from the 64x64 resolution. It is a systemic problem that plagues all StyleGAN images.

If so, why the discriminator could not detect those artifacts?

- Karras et al. (2020a): **AdaIN operation** can be problematic
  - AdaIN normalizes each feature map separately
  - This can destroy any magnitude information in the features relative to each other
- **Hypothesis**: they "sneak" some information past AdaIN




(a) StyleGAN

Karras et al. (2020a): **AdaIN operation** can be problematic

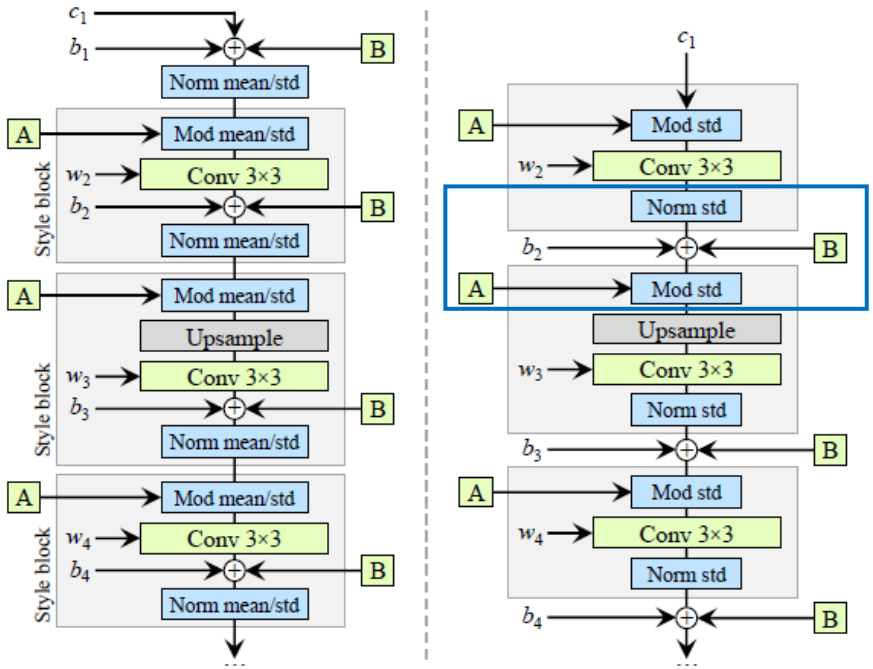
- AdaIN normalizes each feature map separately
- This can destroy any magnitude information in the features relative to each other

**Hypothesis**: they "sneak" some information past AdaIN

- **Observation**: the artifacts disappear when the normalization step is removed

**Generator architecture revisited ⇒ No artifacts anymore!**

1. **Bias outside the style block**
   - StyleGAN applies bias & noise "within" the style block
     - Inversely proportional impact to the current magnitude
   - This design is more predictable
2. **No norm/mod for means**
   - It was possible after (1) is made
   - Much simplifies the design




(b) StyleGAN (detailed) (c) Revised architecture

Karras et al. (2020a): **AdaIN operation** can be problematic

- AdaIN normalizes each feature map separately
- This can destroy any magnitude information in the features relative to each other

**Hypothesis**: they "sneak" some information past AdaIN

- **Observation**: the artifacts disappear when the normalization step is removed

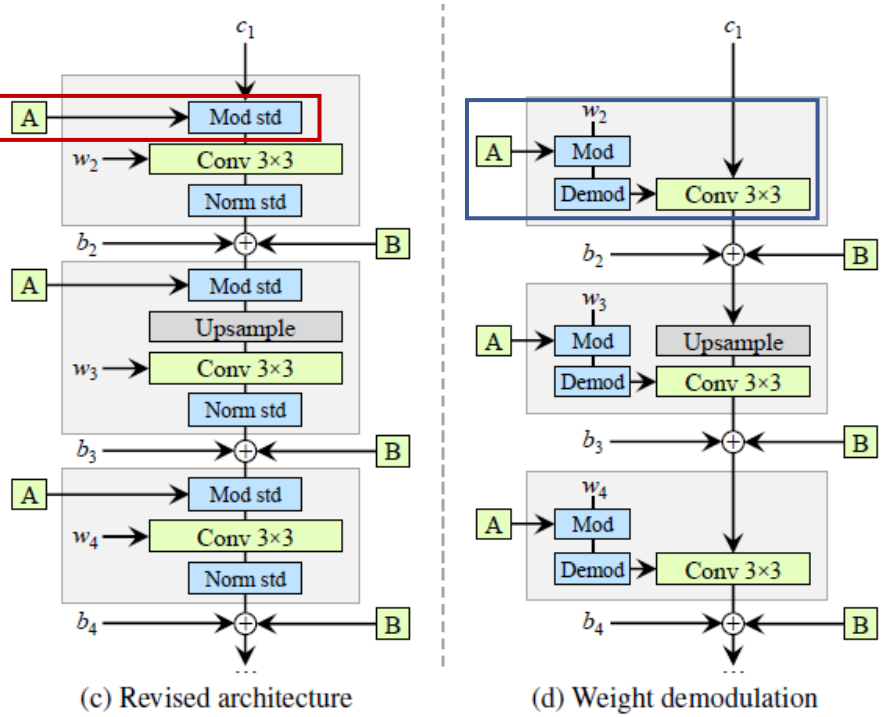
**Generator architecture revisited ⇒ No artifacts anymore!**

## 3. Weight de-modulation

- A "weaker notion" of AdaIN
- AdaIN is originally for removing the effect of input modulation
- StyleGAN2 instead implement these "Mod + AdaIN" by weight re-scaling

$$w'_{ijk} = s_i \cdot w_{ijk},$$

$$w''_{ijk} = w'_{ijk} \Big/ \sqrt{\sum_{i,k} {w'_{ijk}}^2 + \epsilon},$$

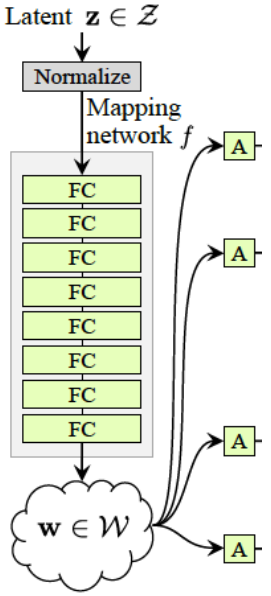


## Path length regularization

- Recall the mapping network $f : \mathcal{Z} \rightarrow \mathcal{W}$
- **Prior**: a fixed step in $\mathcal{W}$ results in a fixed-sized change in $g(w)$

$$\mathbb{E}_{\mathbf{w},\mathbf{y}\sim\mathcal{N}(0,\mathbf{I})} \left( \left\| \mathbf{J}_{\mathbf{w}}^{T}\mathbf{y} \right\|_2 - a \right)^2$$

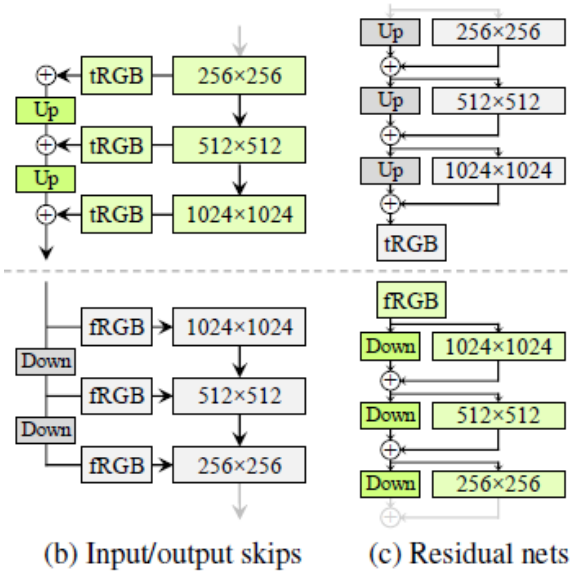
  - $\mathbf{w} \sim f(\mathbf{z})$ ; $\mathbf{y}$: random image
  - $\mathbf{J}_{\mathbf{w}} = \partial g(\mathbf{w})/\partial \mathbf{w}$: The Jacobian matrix




## Improved architectural design

- StyleGAN follows simple feedforward designs
- StyleGAN2 considers better architectural choices
  - **Skip connections for G**
  - **Residual network design for D**

| FFHQ | D original | | D input skips | | D residual | |
|---|---|---|---|---|---|---|
| | FID | PPL | FID | PPL | FID | PPL |
| G original | 4.32 | 265 | 4.18 | 235 | 3.58 | 269 |
| G output skips | 4.33 | 169 | 3.77 | 127 | **3.31** | **125** |
| G residual | 4.35 | 203 | 3.96 | 229 | 3.79 | 243 |




(b) Input/output skips (c) Residual nets

# StyleGAN2: Analyzing and Improving the Image Quality of StyleGAN [Karras et al., 2020a]

StyleGAN2 successfully removes the buggy-artifacts of StyleGAN

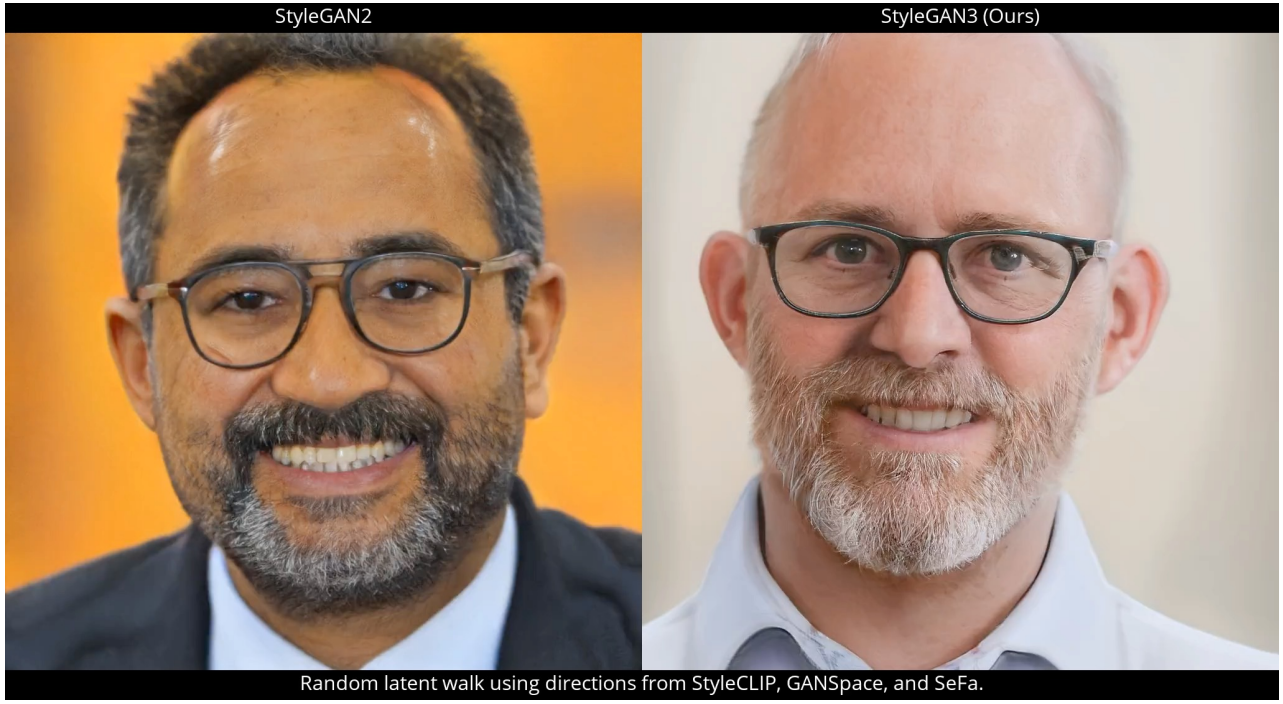
- **Weight de-modulation** significantly improves the recall of generations
- Simply using **larger StyleGAN** could not be comparable with StyleGAN2

| Configuration | FFHQ, 1024×1024 | | | | LSUN Car, 512×384 | | | |
|---|---|---|---|---|---|---|---|---|
| | FID ↓ | Path length ↓ | Precision ↑ | Recall ↑ | FID ↓ | Path length ↓ | Precision ↑ | Recall ↑ |
| A Baseline StyleGAN [24] | 4.40 | 212.1 | **0.721** | 0.399 | 3.27 | 1484.5 | **0.701** | 0.435 |
| B + Weight demodulation | 4.39 | 175.4 | 0.702 | 0.425 | 3.04 | 862.4 | 0.685 | 0.488 |
| C + Lazy regularization | 4.38 | 158.0 | 0.719 | 0.427 | 2.83 | 981.6 | 0.688 | 0.493 |
| D + Path length regularization | 4.34 | **122.5** | 0.715 | 0.418 | 3.43 | 651.2 | 0.697 | 0.452 |
| E + No growing, new G & D arch. | 3.31 | 124.5 | 0.705 | 0.449 | 3.19 | 471.2 | 0.690 | 0.454 |
| F + Large networks (StyleGAN2) | **2.84** | 145.0 | 0.689 | **0.492** | **2.32** | **415.5** | 0.678 | **0.514** |
| Config A with large networks | 3.98 | 199.2 | 0.716 | 0.422 | – | – | – | – |

# StyleGAN3: Alias-Free Generative Adversarial Networks [Karras et al., 2021]

Karras et al. (2021): Still, some buggy-artifacts in StyleGAN2 latent space

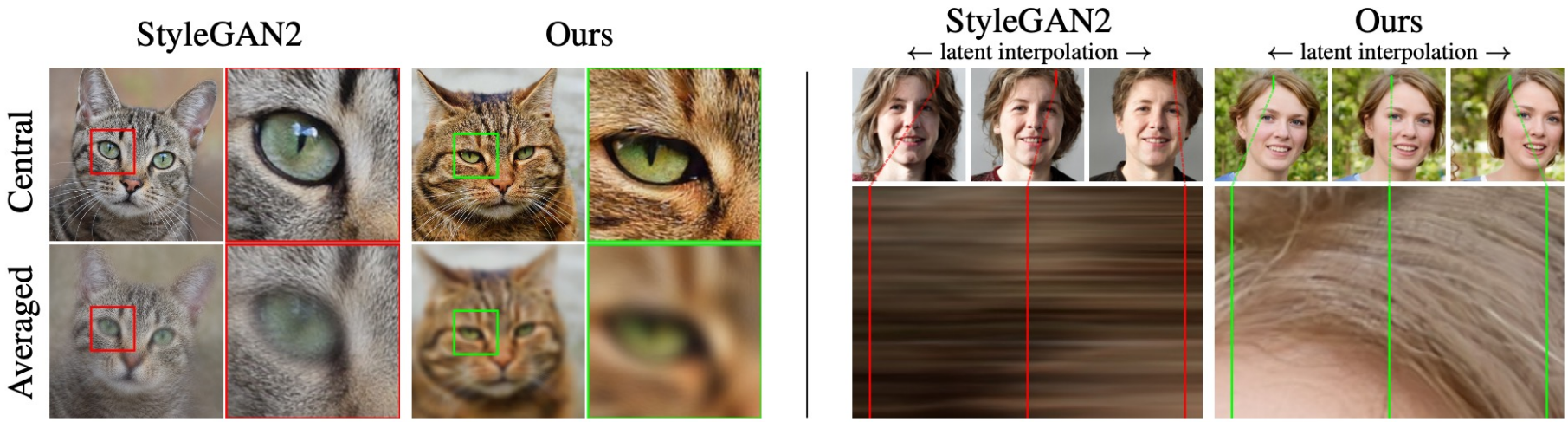
- **Texture striking** problem in most StyleGAN2 images
- Textures (or details) appear to be fixed in pixel coordinates




**StyleGAN3** includes several design modifications to address this issue

Current GANs do not synthesize images in a natural hierarchical manner

1. Previous coarse features control the "presence" of finer features
2. Yet, such details to be fixed in pixel coordinates




**Why?** Unintentional positional references drawn on the intermediate layers

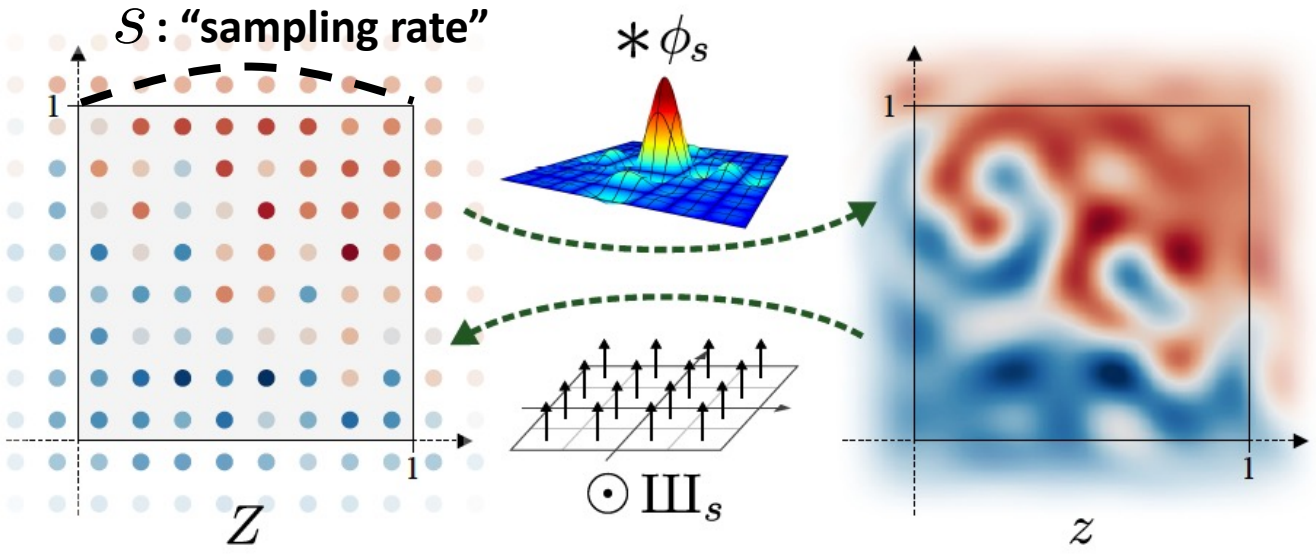
- Faint after-images of the pixel grid from non-ideal up-sampling (e.g., nearest)
- Pointwise application of non-linearities (e.g., ReLU)

**Goal:** Continuous *equivariance* to sub-pixel translation (& rotation) in all layers

Nyquist-Shannon sampling theorem [Shannon, 1949]

- Regular, discrete signal can represent any continuous signals of frequencies 0- $s/2$

Whittaker-Shannon interpolation formula [Shannon, 1949]




$$z(\boldsymbol{x}) = (\phi_s * Z)(\boldsymbol{x}), \text{ and } Z(\boldsymbol{x}) = \mathrm{Ш}_s \odot z(\boldsymbol{x})$$

$$\phi_s(\boldsymbol{x}) := \mathrm{sinc}(sx_0) \cdot \mathrm{sinc}(sx_1), \text{ where } \mathrm{sinc}(x) := \frac{\sin(\pi x)}{\pi x}$$

$$\mathrm{Ш}_s(\boldsymbol{x}) := \sum_{X \in \mathbb{Z}^2} \delta(\boldsymbol{x} - (X + \tfrac{1}{2})/s)$$

Continuous representation of network layers

- A neural network operation $\mathbf{F}$ works on a discrete feature map: $Z' = \mathbf{F}(Z)$
- Consider its continuous counterpart, $z' = \mathbf{f}(z)$, from the correspondence of $z \leftrightarrow Z$

$$\mathbf{f}(z) = \phi_{s'} * \mathbf{F}(\text{Ш}_s \odot z) \iff \mathbf{F}(Z) = \text{Ш}_{s'} \odot \mathbf{f}(\phi_s * Z)$$

**(a)** **(b)**

- … as long as **both (a) and (b)** are **band-limited** (to $s/2$ and $s'/2$, resp.)

**"Equivariant" network layer** (w.r.t. a spatial transformation $\mathbf{t}$)**?**

1. $\mathbf{f} \circ \mathbf{t} = \mathbf{t} \circ \mathbf{f}$ (commute) in the continuous domain
2. $\mathbf{f}$ (and $\mathbf{t}$) must not generate frequency above the output bandlimit of $s'/2$

**Four operations in ConvNets**: Convolution, Up/down-sampling, and Nonlinearity

- Here, we primarily discuss on the case of translation equivariance
- See the full paper for additional tricks to handle the rotation equivariance

**Case 1: Convolution – $\mathbf{F}_{\text{conv}}(Z) = K * Z$**

- Consider a discrete kernel $K$ with sampling rate $s$

$$\mathbf{f}_{\text{conv}}(z) = \underbrace{\phi_s * \big(K}_{\text{Commutative}} * (\text{Ш}_s \odot z)\big) = K * \underbrace{\big(\phi_s * (\text{Ш}_s \odot z)\big)}_{z} = K * z$$

- Introduces no new frequencies – the bandlimit requirement fulfilled
- Convolution commutes with translation in the continuous domain

**Case 2: Up-sampling ($s' > s$) – $\mathbf{f}_{\text{up}}(z) = z$**

- Translation (and rotation) equivariance follow from being an identity in the continuous domain

$$\mathbf{F}_{\text{up}}(Z) = \text{Ш}_{s'} \odot (\phi_s * Z)$$

- The operation can be simpler to implement when $s' = ns$ ($n$: an integer)
- First interleave $Z$ with zeros, and then convolve it with $\text{Ш}_{s'} \odot \phi_s$

**Case 3: Down-sampling ($s' < s$) – $\mathbf{f}_{\text{down}}(z) = \psi_{s'} * z$, where $\psi_s := s^2 \cdot \phi_s$**

- Low-pass filter to $z$ to remove frequencies above the output bandlimit

$$\begin{aligned}\mathbf{F}_{\text{down}}(Z) &= \text{Ш}_{s'} \odot \left(\psi_{s'} * (\phi_s * Z)\right) = 1/s^2 \cdot \text{Ш}_{s'} \odot (\psi_{s'} * \psi_s * Z) \\ &= (s'/s)^2 \cdot \text{Ш}_{s'} \odot (\phi_{s'} * Z)\end{aligned}$$

- In case of $s = ns'$: a discrete convolution by dropping sample points
- Translation equivariance follows from the commutativity of $\mathbf{f}_{\text{down}}$

## Case 4: Non-linearities (e.g., ReLU)

- Pointwise nonlinearity $\sigma$ commutes with translation (and rotation) in the continuous domain
- However, The bandlimit constraint is problematic
  - ReLU in the continuous domain may introduce arbitrarily high frequencies
  - **Solution**: Low-pass filtering of $\sigma(z)$ via $\psi_s$

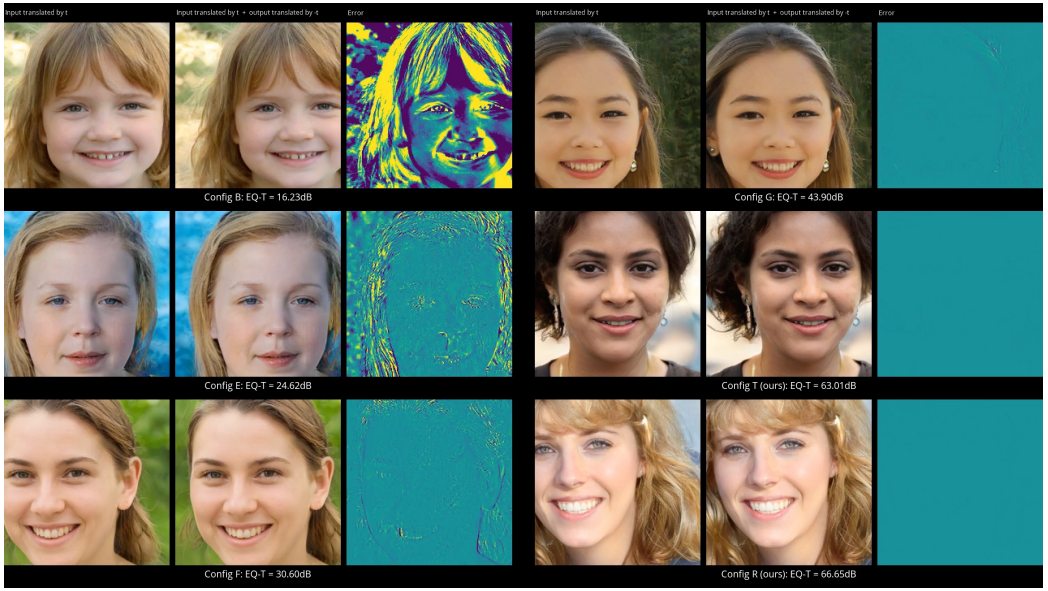
$$\mathbf{f}_\sigma(z) = \psi_s * \sigma(z) = s^2 \cdot \phi_s * \sigma(z)$$
$$\mathbf{F}_\sigma(Z) = s^2 \cdot \text{Ш}_s \odot (\phi_s * \sigma(\phi_s * Z))$$

- In practice, $\mathbf{F}_\sigma(Z)$ can't be realized without temporarily entering the continuous $z$
  - **Solution**: Upsampling – ReLU – Downsampling
  - Only a $2\times$ temporary upsampling was sufficient for high-quality equivariance

StyleGAN3 successfully achieves equivariance and avoid the texture-striking

- Several techniques (e.g., Fourier features) to compensate worse FID

| | Configuration | FID ↓ | EQ-T ↑ | EQ-R ↑ |
|---|---|---|---|---|
| A | StyleGAN2 | 5.14 | – | – |
| B | + Fourier features | 4.79 | 16.23 | 10.81 |
| C | + No noise inputs | 4.54 | 15.81 | 10.84 |
| D | + Simplified generator | 5.21 | 19.47 | 10.41 |
| E | + Boundaries & upsampling | 6.02 | 24.62 | 10.97 |
| F | + Filtered nonlinearities | 6.35 | 30.60 | 10.81 |
| G | + Non-critical sampling | 4.78 | 43.90 | 10.84 |
| H | + Transformed Fourier features | 4.64 | 45.20 | 10.61 |
| T | + Flexible layers (Alias-Free-T) | 4.62 | 63.01 | 13.12 |
| R | + Rotation equiv. (Alias-Free-R) | **4.50** | **66.65** | **40.48** |



**Early GANs suffer from several limitations:**

**Limitation 1**. Limited scalability with respect to data dimension

- GANs had been difficult to be trained on high-resolution data
- Will introduce recent crucial works to mitigate this problem (with controllability)

**Limitation 2**. Limited scalability with respect to dataset complexity

- Due to "mode collapse" problem, had been difficult to trained on complex data
- Will introduce recent works to scale-up GANs (even zero-shot text-to-image scale)

**Limitation 3**. Training instability

- GAN training had been extremely unstable due to bi-level training objective
- Will introduce recent simple yet effective techniques to solve this issue

## Data augmentations for GANs

**Collecting more data** is perhaps the best way to generalize better

**Data augmentation** (DA) makes artificial data instead of collecting more

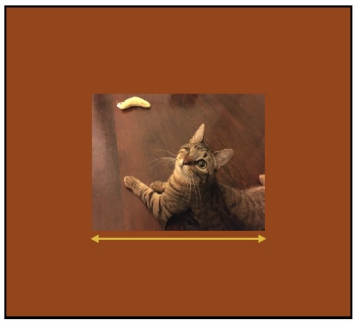
- Requires some knowledge on making "good" artificial data

Have been especially effective for **discriminative modeling**

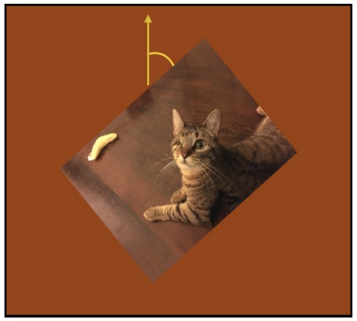
**Example**: Rigid transformation symmetries

- Translation, dilation, rotation, mirror symmetry, …
- Forms an affine group on pixels: $\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \mapsto \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$

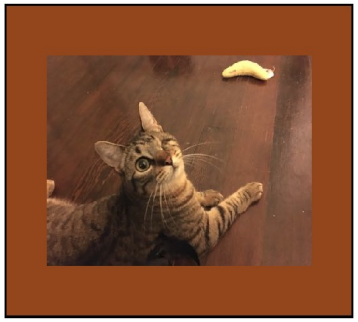

**Translation**



**Dilation**



**Rotation**



**Mirror symmetry**



*source : https://github.com/joanbruna/MathsDL-spring18/blob/master/lectures/lecture2.pdf

**Collecting more data** is perhaps the best way to generalize better

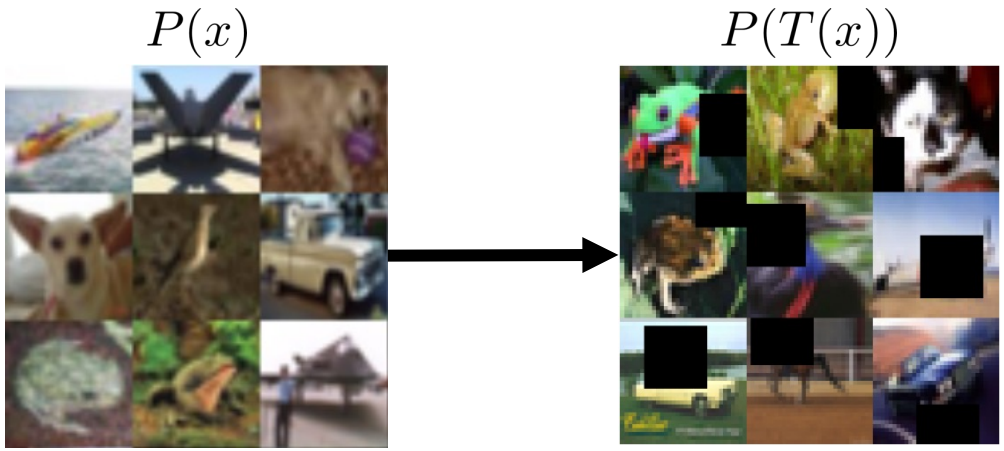
**Data augmentation** (DA) makes artificial data instead of collecting more

- Requires some knowledge on making "good" artificial data

Have been especially effective for **discriminative modeling**

**DA for GANs?** (or for generative modeling in general?)

- Not much explored until very recently [Zhang et al., 2019]
- **Why?** Current DA practices for discriminative modeling might by too strong
- How can we incorporate the distribution shifts $P(x) \rightarrow P(T(x))$?

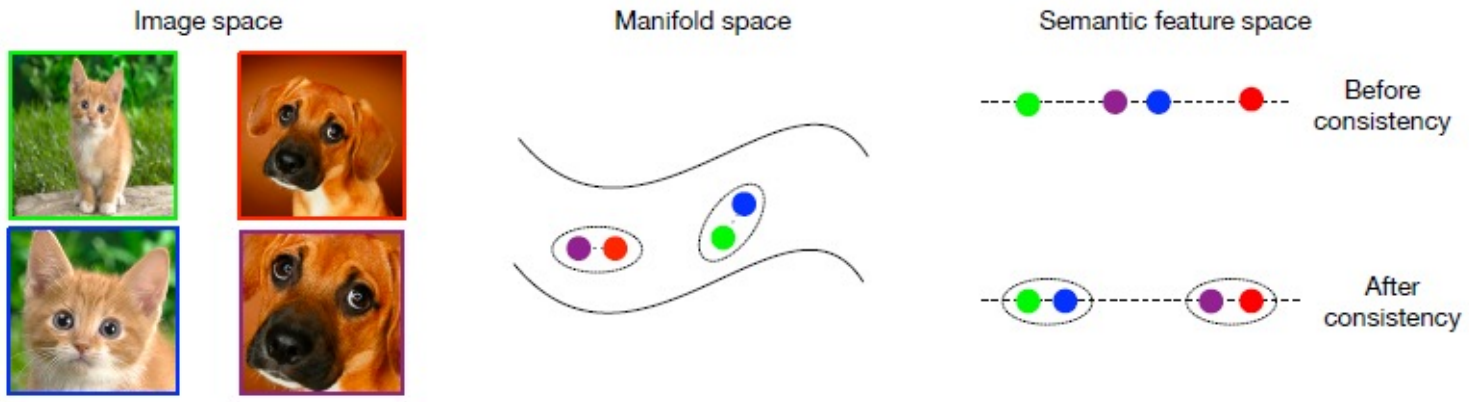


How can we incorporate the distribution shifts $P(x) \rightarrow P(T(x))$?

- Naïve augmentation of real images would shift the data distribution


Zhang et al. (2019): Consistency regularization for GANs (CRGAN)

- **Enforcing only "consistency"** can effectively incorporate $T(x)$



**Enforcing consistency** can effectively incorporate $P(x) \rightarrow P(T(x))$

- Training data is not directly augmented by $T$, but **only consider $\boldsymbol{D(x) \approx D(T(x))}$**
- D should learn representation that is invariant to $T$

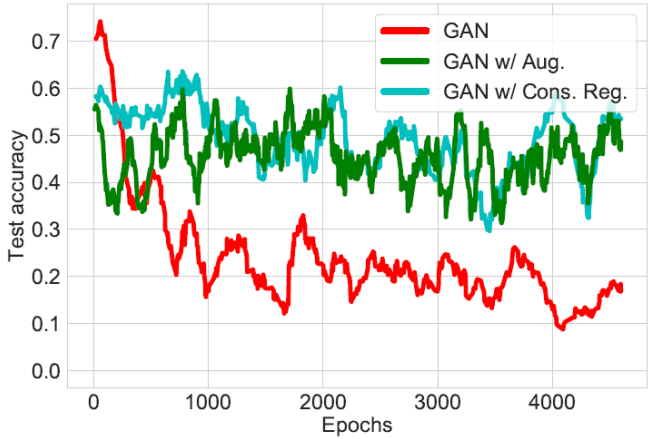
$$L_{cr} = \left\| D(x) - D(T(x)) \right\|^2,$$
$$L_D^{cr} = L_D + \lambda L_{cr}, \qquad L_G^{cr} = L_G.$$

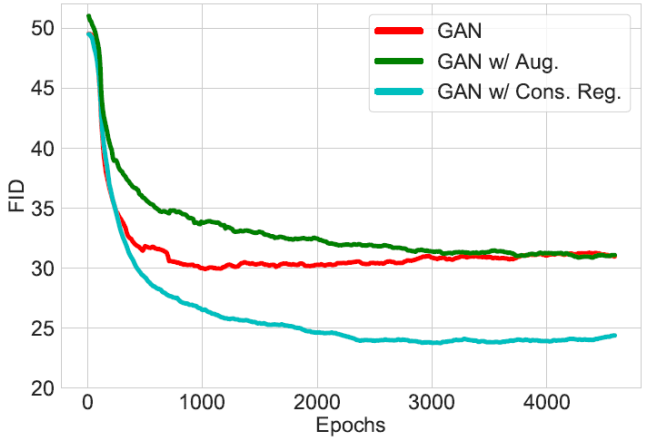
**Algorithm 1** Consistency Regularized GAN (CR-GAN). We use $\lambda = 10$ by default.

**Input:** generator and discriminator parameters $\theta_G, \theta_D$, consistency regularization coefficient $\lambda$, Adam hyperparameters $\alpha, \beta_1, \beta_2$, batch size $M$, number of discriminator iterations per generator iteration $N_D$

**for** number of training iterations **do**
  **for** $t = 1, ..., N_D$ **do**
    **for** $i = 1, ..., M$ **do**
      Sample $z \sim p(z)$, $x \sim p_{\text{data}}(x)$
      Augment $x$ to get $T(x)$
      $L_{cr}^{(i)} \leftarrow \left\| D(x) - D(T(x)) \right\|^2$ ← Only real images are augmented
      $L_D^{(i)} \leftarrow D(G(z)) - D(x)$
    **end for**
    $\theta_D \leftarrow \text{Adam}(\frac{1}{M} \sum_{i=1}^{M} (L_D^{(i)} + \lambda L_{cr}^{(i)}), \alpha, \beta_1, \beta_2)$
  **end for**
  Sample a batch of latent variables $\{z^{(i)}\}_{i=1}^{M} \sim p(z)$
  $\theta_G \leftarrow \text{Adam}(\frac{1}{M} \sum_{i=1}^{M} (-D(G(z))), \alpha, \beta_1, \beta_2)$ G is trained in the standard way
**end for**

## Does CR really learn differently than simple augmentation?




- Both CR and Aug. prevent overfitting of the discriminator
- However, CR is the one that could only meaningfully improve FIDs

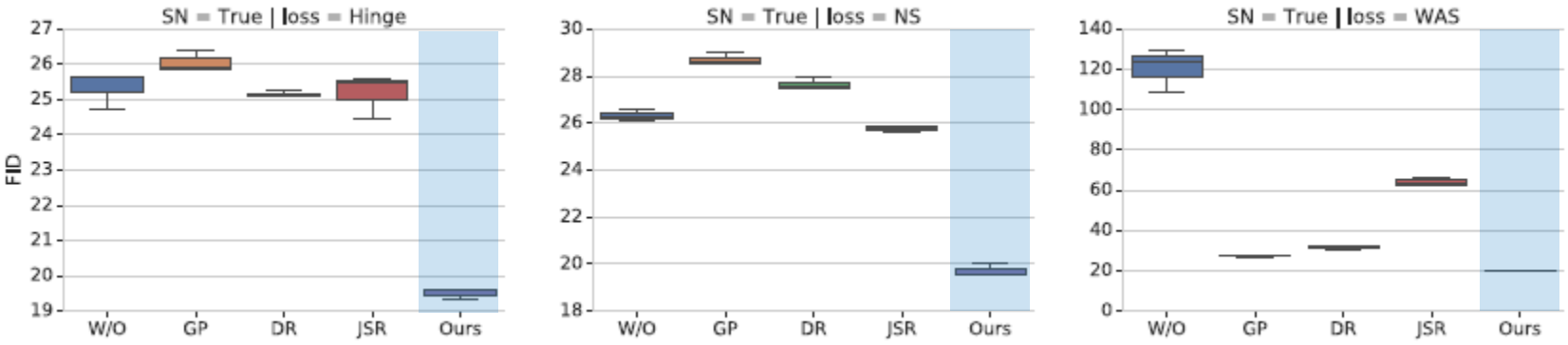
## Which augmentations should we use?

- The choice of augmentation does matter in GAN training
- For CR, a simple choice of "Random shift & flip" worked best

| Metric | Gaussian Noise | Random shift & flip | Cutout | Cutout w/ random shift & flip |
|---|---|---|---|---|
| FID | 21.91±0.32 | 16.04±0.17 | 17.10±0.29 | 19.46±0.26 |

Table 3: FID scores on CIFAR-10 for different types of image augmentation. Gaussian noise is the worst, and random shift and flip is the best, consistent with general consensus on the best way to perform image optimization on CIFAR-10 (Zagoruyko & Komodakis, 2016).

## CR surprisingly stabilizes GAN training on various existing practices




## CR further improves state-of-the-art BigGAN training

| Dataset | SNGAN | SAGAN | BigGAN | BigGAN* | CR-BigGAN* |
|---|---|---|---|---|---|
| CIFAR-10 | 17.5 | / | 14.73 | 20.42 | **11.48** |
| ImageNet | 27.62 | 18.65 | 8.73 | 7.75 | **6.66** |

Comparison of FIDs (lower is better)

**BigGAN**

**CR-BigGAN**

**Recall**: How can we incorporate the distribution shifts $P(x) \rightarrow P(T(x))$?

**Then would it be just enough with CR for GANs?**

- Still, CR does not perfectly prevent the shifting issue in GAN
- For certain augmentations, e.g., CutOut, CR often make "leakages"

(a) $8 \times 8$ cutout. (b) CR samples. (c) bCR samples.

Zhao et al. (2020): Balanced Consistency regularization (bCR)

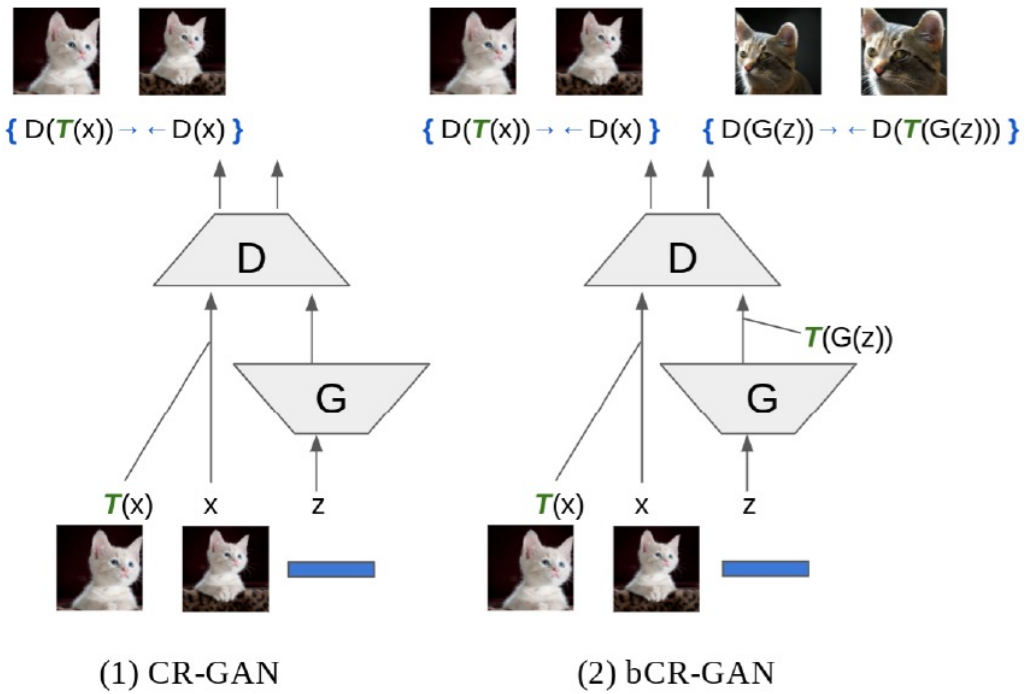
- bCR alleviates such leakages by also giving consistency to "fake" images

$$L_{\text{real}} \leftarrow \|D(x) - D(T(x))\|^2$$
$$L_{\text{fake}} \leftarrow \|D(G(z)) - D(T(G(z)))\|^2$$

Zhao et al. (2020): Balanced Consistency regularization (bCR)

- bCR alleviates such leakages by also giving consistency to "fake" images

$$L_{\text{real}} \leftarrow \|D(x) - D(T(x))\|^2$$
$$L_{\text{fake}} \leftarrow \|D(G(z)) - D(T(G(z)))\|^2$$




**Algorithm 1** Balanced Consistency Regularization (bCR)

**Input:** parameters of generator $\theta_G$ and discriminator $\theta_D$, consistency regularization coefficient for real images $\lambda_{\text{real}}$ and fake images $\lambda_{\text{fake}}$, number of discriminator iterations per generator iteration $N_D$, augmentation transform $T$ (for images, e.g. shift, flip, cutout, etc).

**for** number of training iterations **do**

  **for** $t = 1$ **to** $N_D$ **do**

    Sample batch $z \sim p(z)$, $x \sim p_{\text{real}}(x)$

    Augment both real $T(x)$ and fake $T(G(z))$ images

    $L_D \leftarrow D(G(z)) - D(x)$

    $L_{\text{real}} \leftarrow \|D(x) - D(T(x))\|^2$

    $L_{\text{fake}} \leftarrow \|D(G(z)) - D(T(G(z)))\|^2$

    $\theta_D \leftarrow \text{AdamOptimizer}(L_D + \lambda_{\text{real}} L_{\text{real}} + \lambda_{\text{fake}} L_{\text{fake}})$

  **end for**

  Sample batch $z \sim p(z)$

  $L_G \leftarrow -D(G(z))$

  $\theta_G \leftarrow \text{AdamOptimizer}(L_G)$

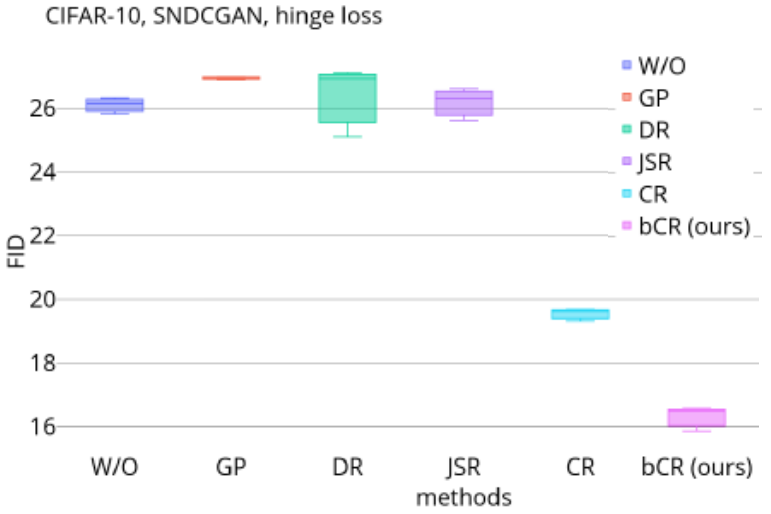
**end for**

**G is still trained in the standard way**

Zhao et al. (2020): Balanced Consistency regularization (bCR)

Despite its simplicity, bCR could achieve state-of-the-art BigGAN training

(a) Monarch Butterfly (our ICR vs baseline CR)



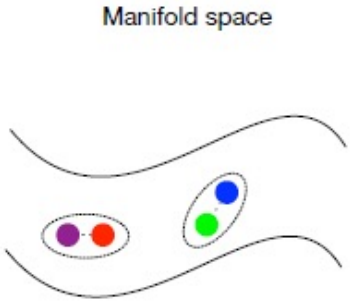

| Models | CIFAR-10 | ImageNet |
|---|---|---|
| SNGAN | 17.50 | 27.62 |
| BigGAN | 14.73 | 8.73 |
| CR-BigGAN | 11.48 | 6.66 |
| ICR-BigGAN (ours) | **9.21** | **5.38** |

**Is CR really necessary for GANs to incorporate data augmentations?**

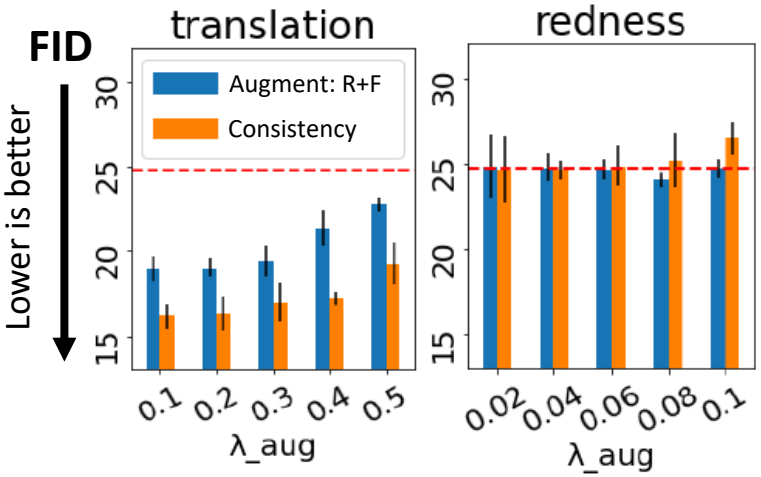
**Limitation of CR:** Fundamentally hard to incorporate stronger augmentations

**Example**: Color jittering

- The "redness" is not helpful to improve FID with CR
- Forcing CR for such a stronger augmentation might be too restrictive for D representation




How can we incorporate stronger augmentations for GANs?



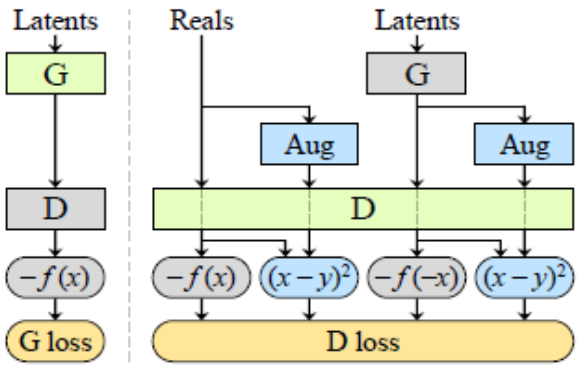

* source: [ZhaoZ et al., 2020b] Image Augmentations for GAN Training, 2020.

Is CR really necessary for GANs to incorporate data augmentations?

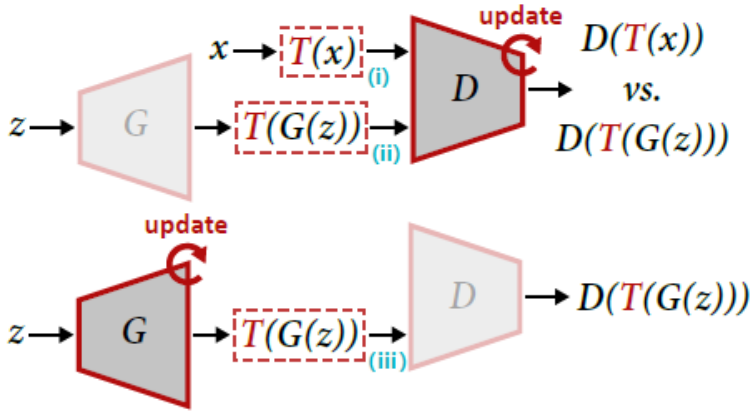
**How can we incorporate stronger augmentations for GANs?**

Two recent works propose a "even simpler" scheme for GANs

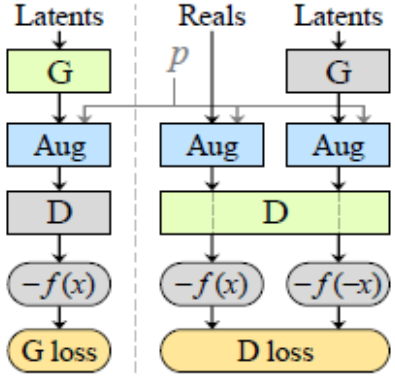
- [Zhao et al., 2020b] "Differentiable Augmentation for Data-Efficient GAN Training"
- [Karras et al., 2020b] "Training Generative Adversarial Networks with Limited Data"




**Balanced CR (bCR)**




**DiffAug**




**ADA**

## Beyond the Consistency Regularization [Zhao et al., 2020b; Karras et al., 2020b]

Is CR really necessary for GANs to incorporate data augmentations?

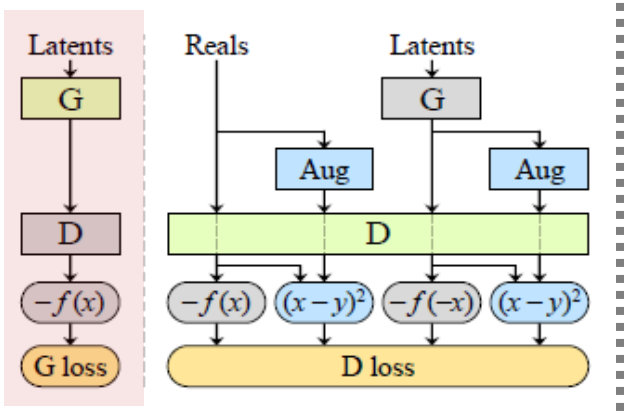
**How can we incorporate stronger augmentations for GANs?**

Two recent works propose a "even simpler" scheme for GANs

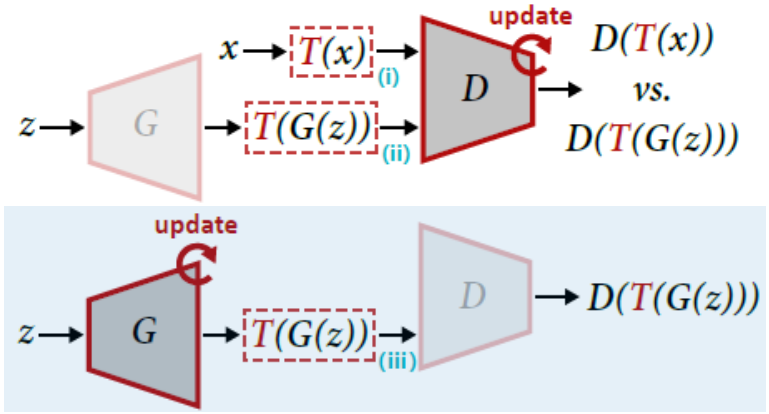
- [Zhao et al., 2020b] "Differentiable Augmentation for Data-Efficient GAN Training"
- [Karras et al., 2020b] "Training Generative Adversarial Networks with Limited Data"

**Idea**: Simply augment every input before D, even when G is trained

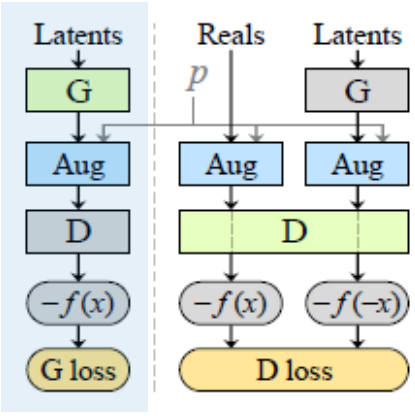
- No CR needed anymore, and accept stronger augmentations without leakages




**Balanced CR (bCR)**




**DiffAug**



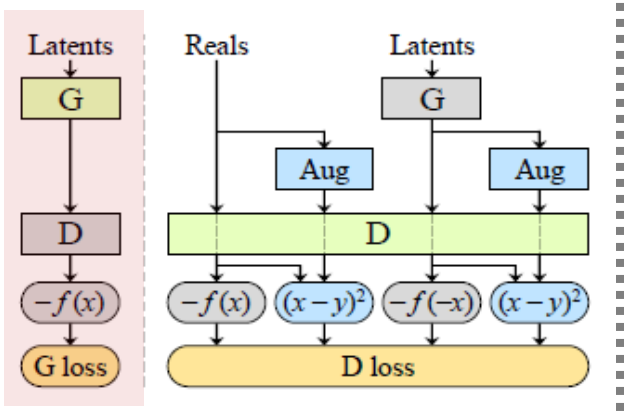

**ADA**

Two concurrent works propose a "even simpler" scheme for GANs

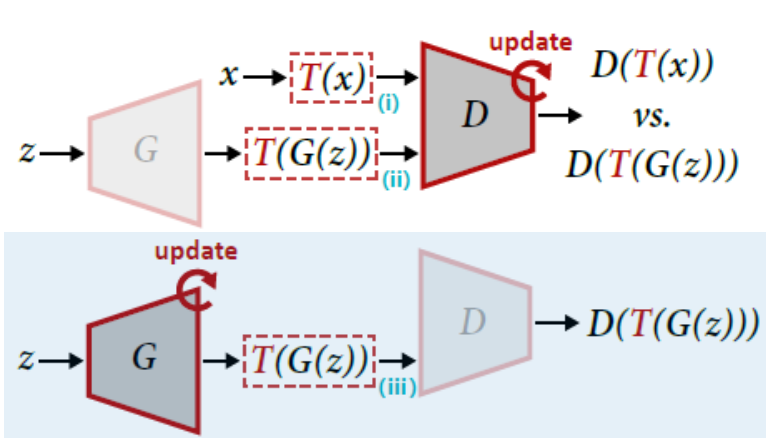
**Idea**: Simply augment every input before D, even when G is trained

Then, how could this approach have not been explored so far?

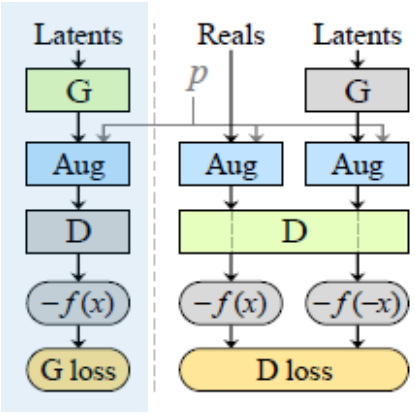
- This requires a differentiable implementation of $T(\cdot)$ for training G
  - **Example**: Non-saturating loss should minimize $\mathbb{E}_z[-\log(D(T(\boldsymbol{G}(z))))]$
- Nevertheless, most of the previous implementations of $T$ were non-differentiable
  - … as they were rather considered as pre-processing steps
- In this respect, the "differentiability" of $T$ is becoming increasingly important




**Balanced CR (bCR)**




**DiffAug**




**ADA**

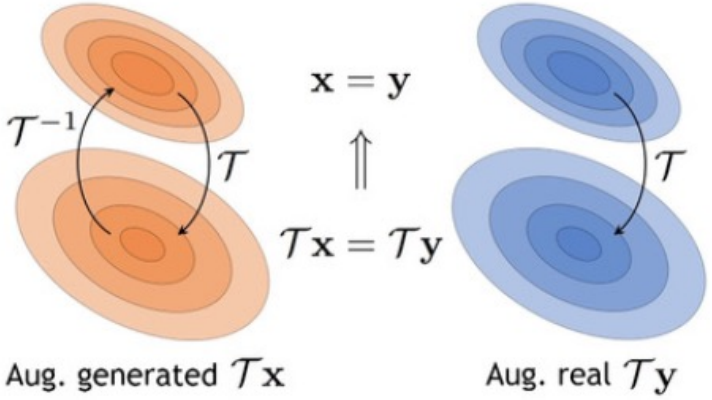
**Adaptive Discriminator Augmentation (ADA)** [Karras et al., 2020b]

Which augmentation should we use?

- **Key point**: There should be no leakage of augmentations

**Example**: Random $90°$ rotations as $\mathcal{T}$

- Assume $\mathbf{x}$: generated distribution and $\mathbf{y}$: target distribution
- **Q**: ADA matches $\mathcal{T}\mathbf{x} = \mathcal{T}\mathbf{y}$: then, does it always imply $\mathbf{x} = \mathbf{y}$?
- **A**: No, imagine when $\mathbf{x}$ goes like "E" below → augmentation leakage

**Adaptive Discriminator Augmentation (ADA)** [Karras et al., 2020b]

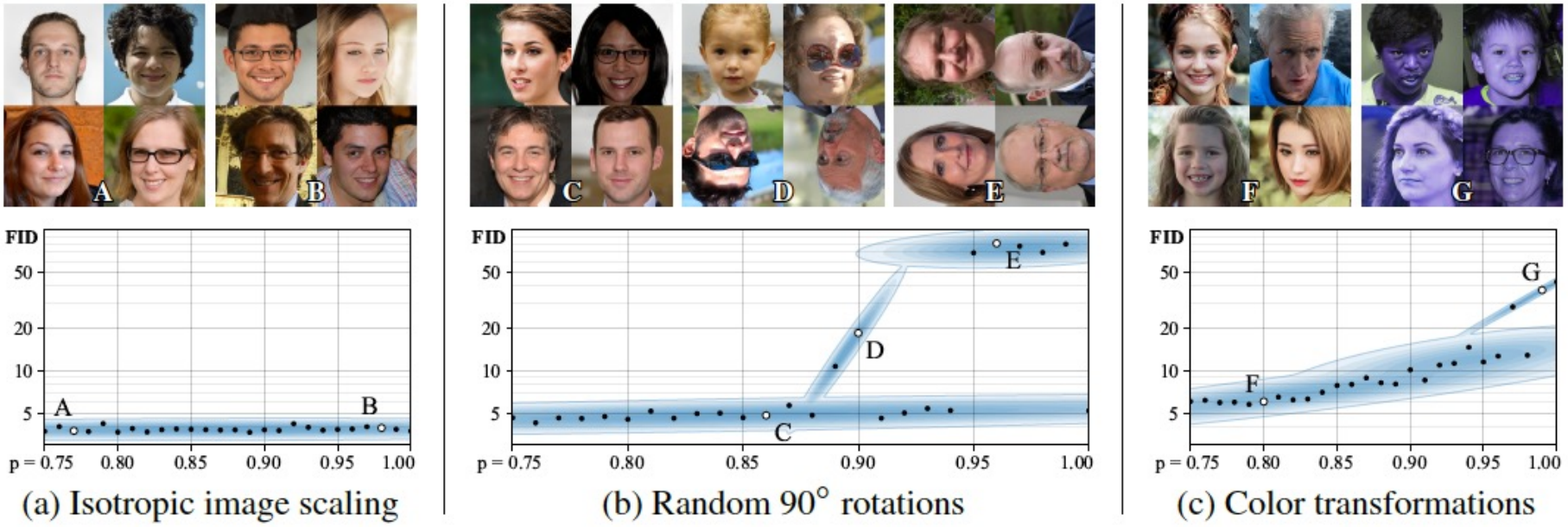
Which augmentation should we use?

- **Key point**: There should be no leakage of augmentations

**Example**: Random $90°$ rotations as $\mathcal{T}$

**Idea**: The leakage of any $\mathcal{T}$ can be controlled by setting $p \in [0, 1]$

The prob. of executing $\mathcal{T}$




(a) Isotropic image scaling (b) Random $90°$ rotations (c) Color transformations

**Adaptive Discriminator Augmentation (ADA)** [Karras et al., 2020b]

Which augmentation should we use?

- **Key point**: There should be no leakage of augmentations

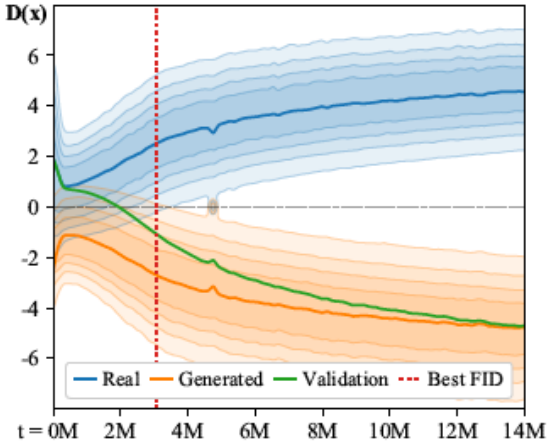
**Idea**: The leakage of any $\mathcal{T}$ can be controlled by setting $p \in [0, 1]$

The prob. of executing $\mathcal{T}$

ADA also proposes a heuristic to adaptively set $p$ in training by observing $r_v$

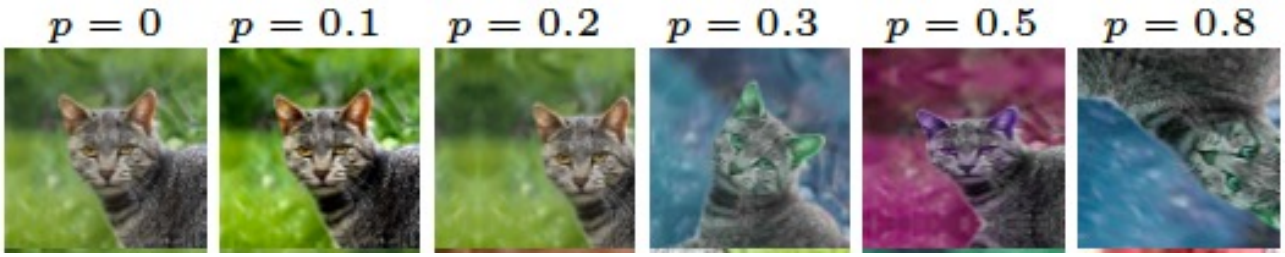
$$r_v = \frac{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{validation}}]}{\mathbb{E}[D_{\text{train}}] - \mathbb{E}[D_{\text{generated}}]}$$

- $r_v = 0$: No overfitting / $r_v = 1$: Complete overfitting
- $p$ of the augmentation is initially set to 0
- Increase/decrease $p$ when $r_v$ is low/high, resp.

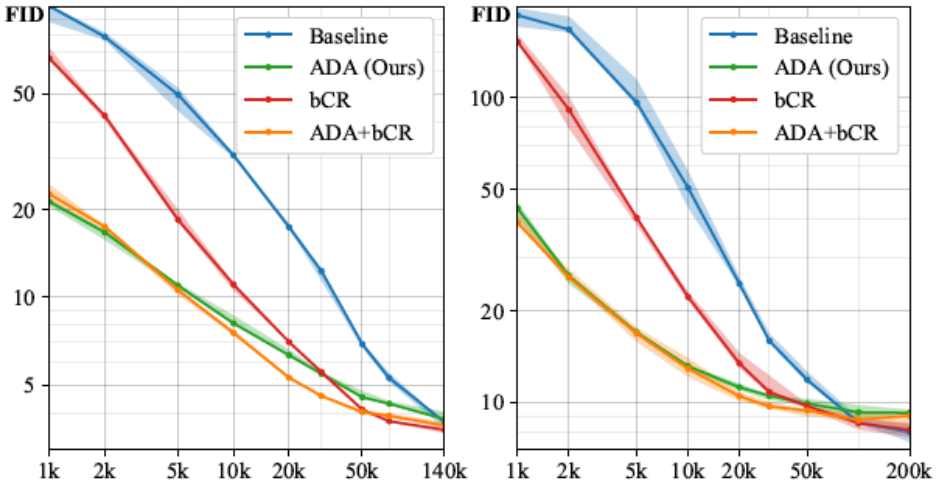


**Adaptive Discriminator Augmentation (ADA)** [Karras et al., 2020b]

- ADA successfully incorporate wider augmentations than bCR




- ADA works significantly better than bCR when # sample is small




(a) FFHQ (256 × 256) (b) LSUN CAT (256 × 256)

| Dataset | | Baseline | ADA | + bCR |
|---|---|---|---|---|
| FFHQ | 1k | 100.16 | **21.29** | 22.61 |
| | 5k | 49.68 | 10.96 | **10.58** |
| | 10k | 30.74 | 8.13 | **7.53** |
| | 30k | 12.31 | 5.46 | **4.57** |
| | 70k | 5.28 | 4.30 | **3.91** |
| | 140k | 3.71 | 3.81 | **3.62** |
| LSUN CAT | 1k | 186.91 | 43.25 | **38.82** |
| | 5k | 96.44 | 16.95 | **16.80** |
| | 10k | 50.66 | 13.13 | **12.90** |
| | 30k | 15.90 | 10.50 | **9.68** |
| | 100k | **8.56** | 9.26 | 8.73 |
| | 200k | **7.98** | 9.22 | 9.03 |

(c) Median FID

**Adaptive Discriminator Augmentation (ADA)** [Karras et al., 2020b]

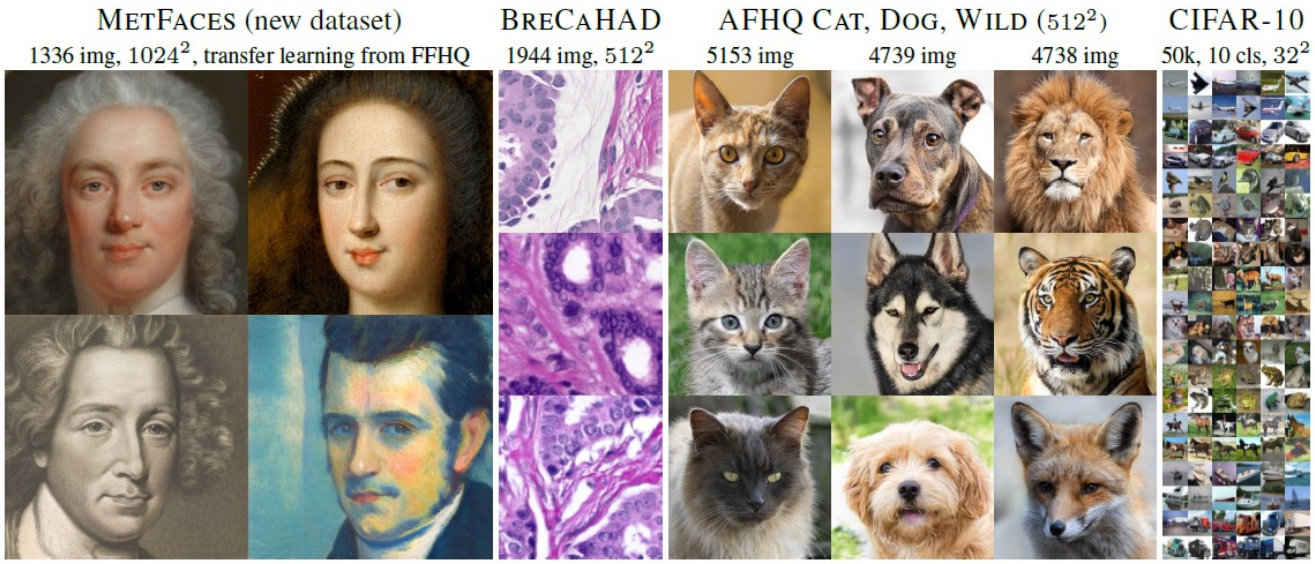
- ADA significantly improves GAN training especially on limited-sized datasets

| Dataset | Method | Scratch FID | Scratch KID $\times 10^3$ | Transfer KID $\times 10^3$ | + Freeze-D KID $\times 10^3$ |
|---|---|---|---|---|---|
| MetFaces | Baseline | 57.26 | 35.66 | 3.16 | 2.05 |
| | ADA | **18.22** | **2.41** | **0.81** | **1.33** |
| BreCaHAD | Baseline | 97.72 | 89.76 | 18.07 | 6.94 |
| | ADA | **15.71** | **2.88** | **3.36** | **1.91** |
| AFHQ Cat | Baseline | 5.13 | 1.54 | 1.09 | 1.00 |
| | ADA | **3.55** | **0.66** | **0.44** | **0.35** |
| AFHQ Dog | Baseline | 19.37 | 9.62 | 4.63 | 2.80 |
| | ADA | **7.40** | **1.16** | **1.40** | **1.12** |
| AFHQ Wild | Baseline | 3.48 | 0.77 | 0.31 | **0.12** |
| | ADA | **3.05** | **0.45** | **0.15** | 0.14 |

(a) Small datasets

| Method | Unconditional FID ↓ | Unconditional IS ↑ | Conditional FID ↓ | Conditional IS ↑ |
|---|---|---|---|---|
| ProGAN [19] | 15.52 | 8.56±0.06 | – | – |
| AutoGAN [13] | 12.42 | 8.55±0.10 | – | – |
| BigGAN [5] | – | – | 14.73 | 9.22 |
| + Tuning [22] | – | – | 8.47 | 9.07±0.13 |
| MultiHinge [22] | – | – | 6.40 | 9.58±0.09 |
| FQ-GAN [52] | – | – | 5.59±0.12 | 8.48 |
| Baseline | 8.32±0.09 | 9.21±0.09 | 6.96±0.41 | 9.53±0.06 |
| + ADA (Ours) | 5.33±0.35 | **10.02**±0.07 | 3.49±0.17 | **10.24**±0.07 |
| + Tuning (Ours) | **2.92**±0.05 | 9.83±0.04 | **2.42**±0.04 | 10.14±0.09 |

(b) CIFAR-10

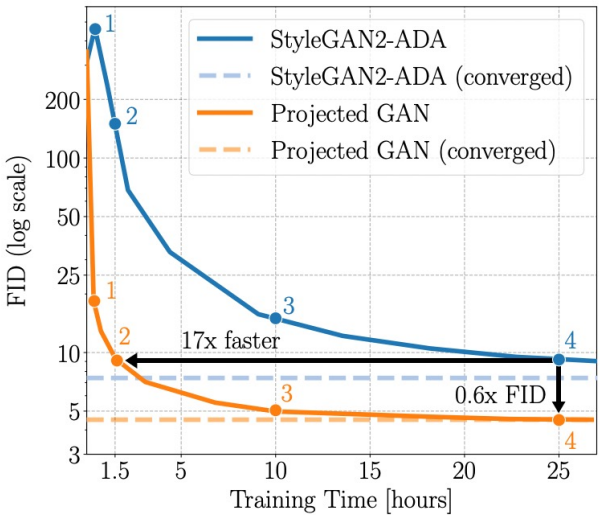


# Projected GANs Converge Faster [Sauer et al., 2021]

**Can we use pre-trained features for efficient GAN training?**

**Projected GAN:** Uses pre-trained features that dramatically boost GAN training

- Even 17x faster than ADA (up to 40x), and can reaches 0.6x lower FID




**Main idea:** Adapt multi-scale "feature projectors" $\{P_l\}$ into GAN training

- Here, projectors $\{P_l\}$ are based on pre-trained networks (will be described later)

$$\min_G \max_D \left( \mathbb{E}_{\mathbf{x}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))] \right)$$

Re-design classical GAN training

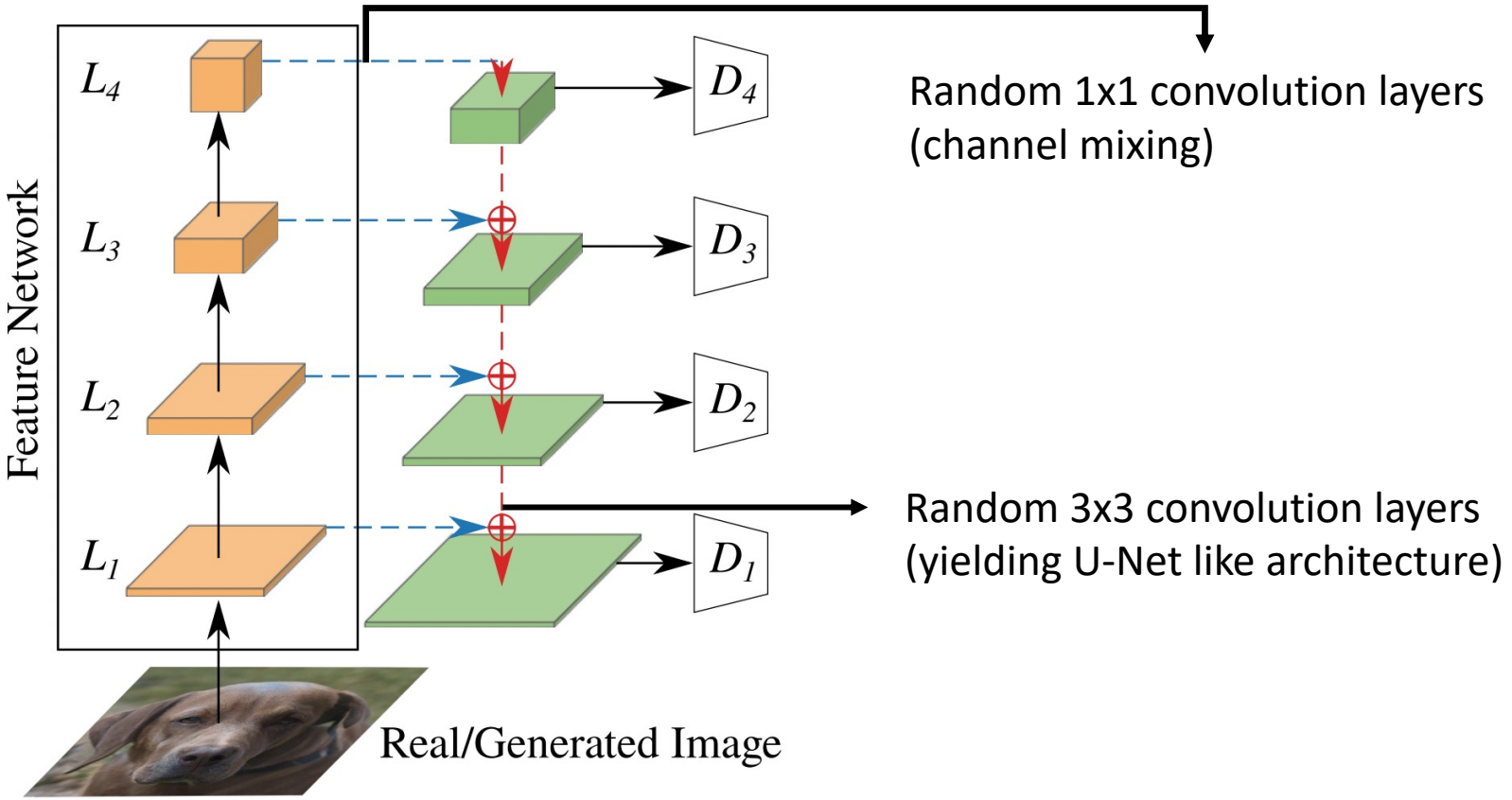
$$\min_G \max_{\{D_l\}} \sum_{l \in \mathcal{L}} \left( \mathbb{E}_{\mathbf{x}}[\log D_l(P_l(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}}[\log(1 - D_l(P_l(G(\mathbf{z}))))] \right)$$

"projectors"

"multi-scale"

**How to use pretrained-networks as feature projectors $\{P_l\}$?**

- Observed using intermediate features $L_i$ directly is not beneficial
- They might be too easy to discriminate → added some "random projections"



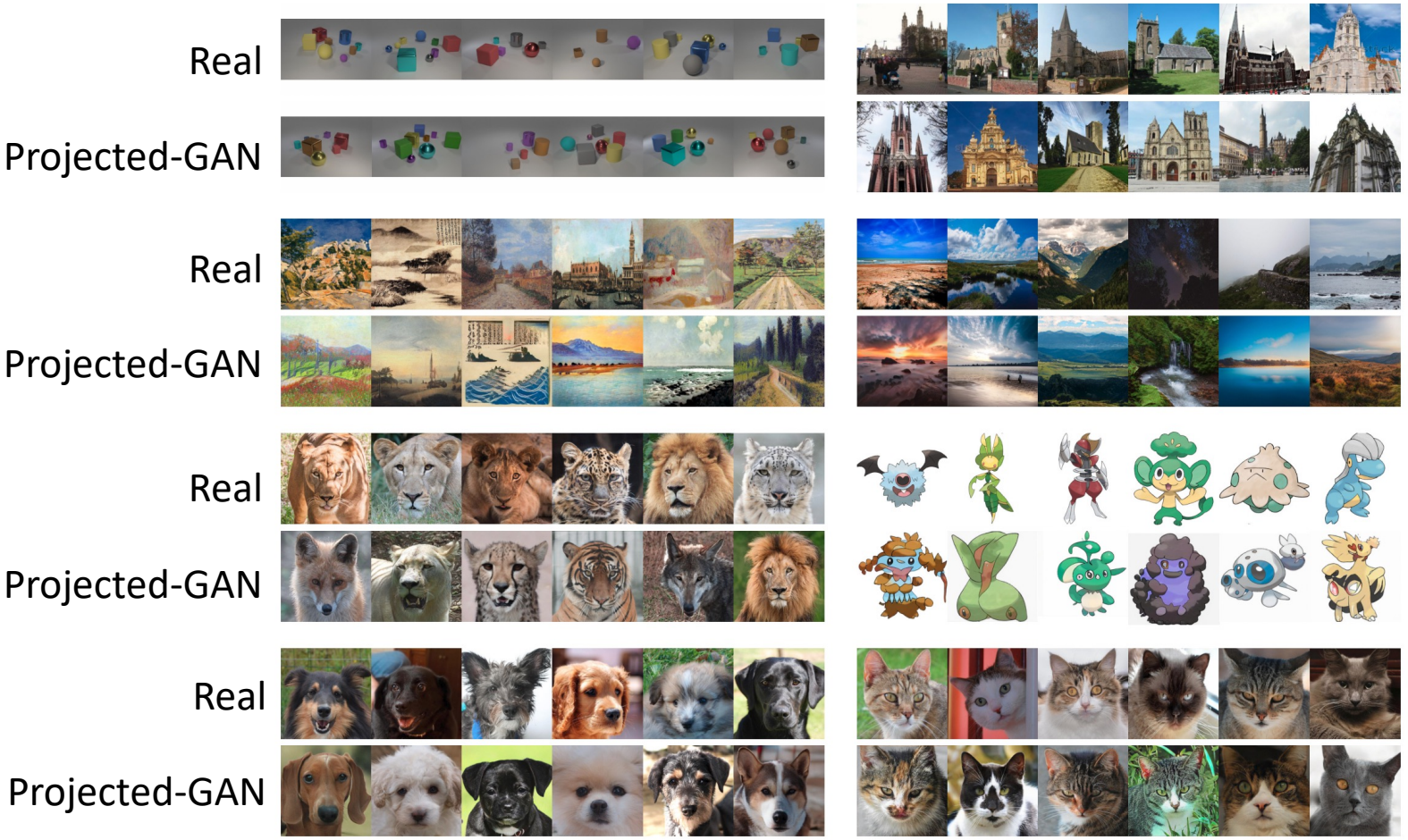
**Which pre-trained feature network is beneficial?**

- Tried various popular pretrained image models (EfficientNet, ResNet, ViT)
- Mostly relied on pretrained networks for image classification (or CLIP)
- Turns out "compact" EfficientNets outperform others

| | EfficientNet | | | | | ResNet | | | Transformer | |
|---|---|---|---|---|---|---|---|---|---|---|
| | lite0 | lite1 | lite2 | lite3 | lite4 | R18 | R50 | R50-CLIP | DeiT | ViT |
| Params (M) ↓ | 2.96 | 3.72 | 4.36 | 6.42 | 11.15 | 11.18 | 23.51 | 23.53 | 92.36 | 317.52 |
| IN top-1 ↑ | 75.48 | 76.64 | 77.47 | 79.82 | 81.54 | 69.75 | 79.04 | N/A | 85.42 | 85.16 |
| FID ↓ | 2.53 | 1.65 | 1.69 | 1.79 | 2.35 | 4.16 | 4.40 | 3.80 | 2.46 | 12.38 |

**Projected GANs show better/faster convergence, both on large-/small- datasets**

- The method is even scalable to high-resolution datasets (megapixels)



**Projected GANs show better/faster convergence, both on large-/small- datasets**

- The method is even scalable to high-resolution datasets (megapixels)

| | FID | *Imgs* | FID | *Imgs* | FID | *Imgs* | FID | *Imgs* | FID | *Imgs* |
|---|---|---|---|---|---|---|---|---|---|---|
| | Large Datasets ($256^2$) | | | | | | | | | |
| | **CLEVR** | | **FFHQ** | | **Cityscapes** | | **Bedroom** | | **Church** | |
| SAGAN [86] | 26.04 | 10 M | 16.21 | 10 M | 12.81 | 10 M | 14.06 | 10 M | 6.15 | 10 M |
| StyleGAN2-ADA [32] | 10.17 | 10 M | 7.32 | 10 M | 8.35 | 10 M | 11.53 | 10 M | 5.85 | 10 M |
| GANsformers [26] | 9.24 | 10 M | 7.42 | 10 M | 5.23 | 10 M | 6.15 | 10 M | 5.47 | 10 M |
| FastGAN [49] | 3.24 | 10 M | 12.69 | 10 M | 8.78 | 1.8 M | 8.24 | 4.8 M | 8.43 | 8.9 M |
| Projected GAN | **0.89** | 4.5 M | **3.39** | 7.1 M | **3.41** | 1.7 M | **1.52** | 5.2 M | **1.59** | 9.2 M |
| Projected GAN* | 3.39 | **0.5 M** | 3.56 | **7.0 M** | 4.60 | **1.1 M** | 2.58 | **1.5 M** | 3.18 | **1.1 M** |
| StyleGAN2* [32, 33, 85] | 5.05 | 25 M | 3.62 | 25 M | - | - | 2.65 | 70 M | 3.39 | 88 M |
| | Small Datasets ($256^2$) | | | | | | | | | |
| | **Art Painting** | | **Landscape** | | **AnimalFace** | | **Flowers** | | **Pokemon** | |
| StyleGAN2-ADA [32] | 43.07 | 3.2 M | 15.99 | 6.3 M | 60.90 | 2.2 M | 21.66 | 3.8 M | 40.38 | 3.4 M |
| FastGAN [49] | 44.02 | 0.7 M | 16.44 | 1.8 M | 62.11 | 0.2 M | 26.23 | 0.8 M | 81.86 | 2.5 M |
| Projected GAN | **27.96** | 0.8 M | **6.92** | 3.5 M | **17.88** | 10 M | **13.86** | 1.8 M | **26.36** | 0.8 M |
| Projected GAN* | 40.22 | **0.2 M** | 14.99 | **0.6 M** | 58.07 | **0.02 M** | 21.60 | **0.2 M** | 36.57 | **0.3 M** |
| | $1024^2$ | | | | $512^2$ | | | | | |
| | **Art Painting** | | **Pokemon** | | **AFHQ-Cat** | | **AFHQ-Dog** | | **AFHQ-Wild** | |
| StyleGAN2-ADA [32] | 41.69 | 1.0 M | 56.76 | 0.6 M | 3.55 | 10 M | 7.40 | 10 M | 3.05 | 10 M |
| FastGAN [49] | 46.71 | 0.8 M | 56.46 | 0.8 M | 4.69 | 1.1 M | 13.09 | 1.6 M | 3.14 | **1.6 M** |
| Projected GAN | **32.07** | 0.9 M | **33.96** | 1.3 M | **2.16** | 3.7 M | **4.52** | 3.8 M | **2.17** | 5.4 M |
| Projected GAN* | 40.33 | **0.2 M** | 53.74 | **0.2 M** | 3.53 | **1.0 M** | 7.10 | **0.9 M** | 3.03 | **1.6 M** |

**Early GANs suffer from several limitations:**

**Limitation 1**. Limited scalability with respect to data dimension

- GANs had been difficult to be trained on high-resolution data
- Will introduce recent crucial works to mitigate this problem (with controllability)

**Limitation 2**. Limited scalability with respect to dataset complexity

- Due to "mode collapse" problem, had been difficult to trained on complex data
- Will introduce recent works to scale-up GANs (even zero-shot text-to-image scale)

**Limitation 3**. Training instability

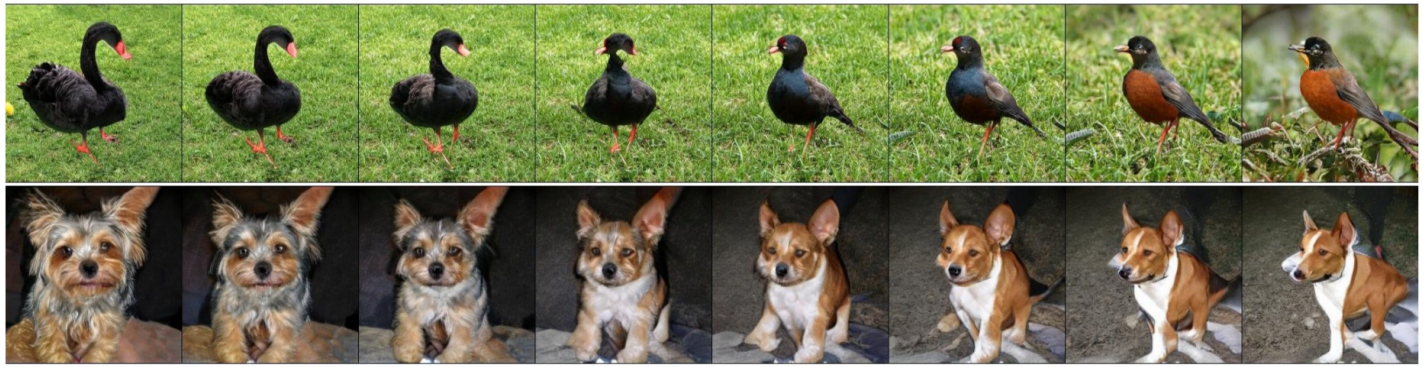
- GAN training had been extremely unstable due to bi-level training objective
- Will introduce recent simple yet effective techniques to solve this issue

## BigGAN: High-resolution, Diverse Image Generation [Brock et al., 2019]

BigGAN is a holistic approach of recent techniques for training GANs

Current cGAN techniques can be successfully scaled up to generate high-resolution, diverse samples from complex datasets such as ImageNet

Achieved first promising results in terms of IS and FID on 128×128 ImageNet





* source : Brock, et. al., Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019

## A holistic approach of previous GAN techniques

1. Based on prior popular GANs:

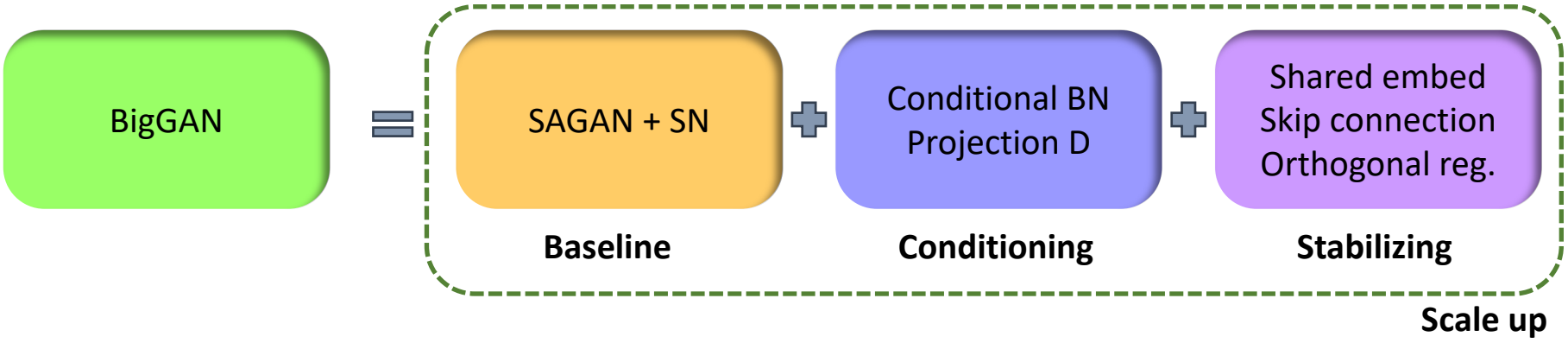
   SAGAN [Zhang et al., 2019] + Spectral normalization [Miyato et al., 2018]
2. Class-conditional modeling
   - G: **Class-conditional BatchNorm** [Dumoulin et al., 2017]
   - D: **Projection discriminator** [Miyato et al., 2018]

## Several further techniques needed to stabilize the large-scale training

1. Shared embedding of $\mathbf{y}$ across multiple layers
2. Skip connection (residual) of the latent variable
3. Orthogonal regularization

weight matrix

$$R_\beta(W) = \beta \|W^\top W \odot (\mathbf{1} - I)\|_{\mathrm{F}}^2$$





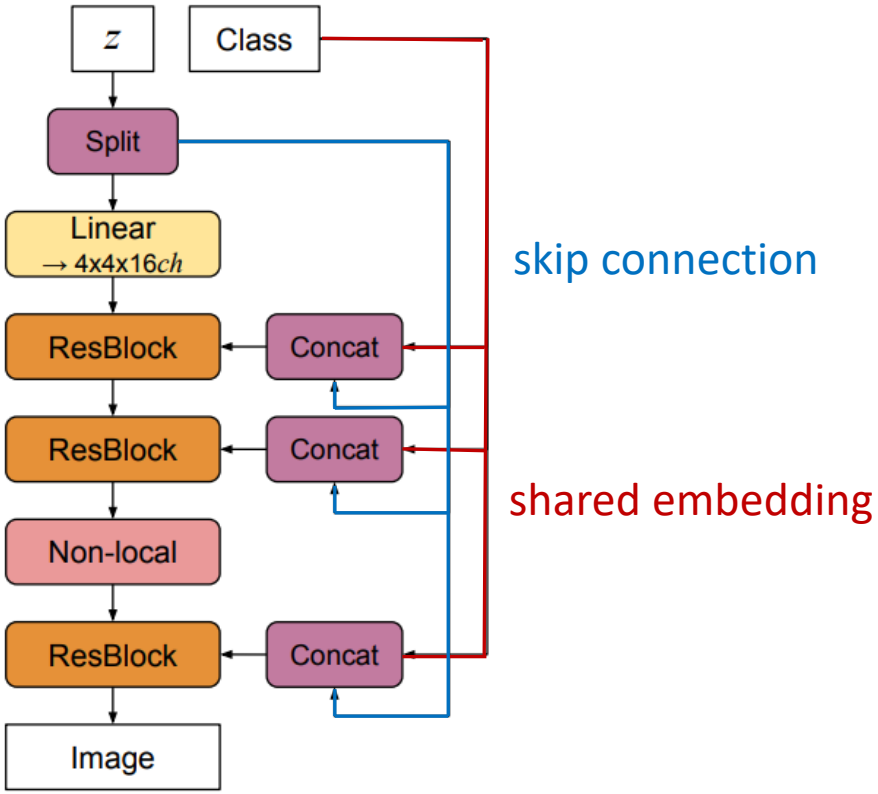

* source : Brock, et. al., Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019

Shared embedding of class information

- Instead of having a separate layer at the end for embedding [Miyato et al., 2018]
- Linearly projected to each layer's gains and biases [Perez et al., 2018]

Skip connections (skip-$z$) from $z$ across multiple layers of G

- Allows $z$ to directly influence the features at different resolutions




**The BigGAN architecture**

# BigGAN: High-resolution, Diverse Image Generation [Brock et al., 2019]

| Batch | Ch. | Param (M) | Shared | Skip-$z$ | Ortho. | Itr $\times 10^3$ | FID | IS |
|---|---|---|---|---|---|---|---|---|
| 256 | 64 | 81.5 | SA-GAN Baseline | | | 1000 | 18.65 | 52.52 |
| 512 | 64 | 81.5 | ✗ | ✗ | ✗ | 1000 | 15.30 | 58.77(±1.18) |
| 1024 | 64 | 81.5 | ✗ | ✗ | ✗ | 1000 | 14.88 | 63.03(±1.42) |
| 2048 | 64 | 81.5 | ✗ | ✗ | ✗ | 732 | 12.39 | 76.85(±3.83) |
| 2048 | 96 | 173.5 | ✗ | ✗ | ✗ | 295(±18) | 9.54(±0.62) | 92.98(±4.27) |
| 2048 | 96 | 160.6 | ✓ | ✗ | ✗ | 185(±11) | 9.18(±0.13) | 94.94(±1.32) |
| 2048 | 96 | 158.3 | ✓ | ✓ | ✗ | 152(±7) | 8.73(±0.45) | 98.76(±2.84) |
| 2048 | 96 | 158.3 | ✓ | ✓ | ✓ | 165(±13) | 8.51(±0.32) | 99.31(±2.10) |
| 2048 | 64 | 71.3 | ✓ | ✓ | ✓ | 371(±7) | 10.48(±0.10) | 86.90(±0.61) |

Scale up

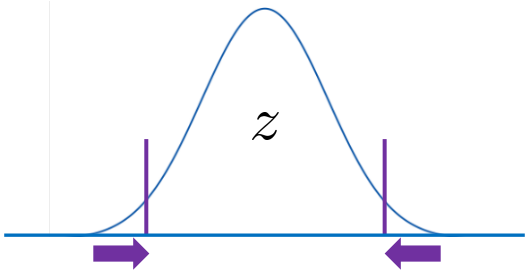
Stabilize

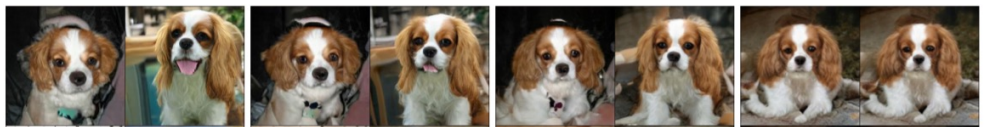
Increasing the **batch size by 8x** improves the state-of-the-art IS by 46%

Increasing the **width (# channels) by 1.5x** leads to a further improvement

**Truncation trick:** could further fine-control FID vs. IS

- Trade-off between variety vs. fidelity
- Simply truncate the variance of the latent variable



Variety

Fidelity



* source : Brock, et. al., Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019

# StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets [Sauer et al., 2021]

**Motivation:** Can we enjoy BigGAN-like scalability with StyleGAN-like architectures?

- BigGAN shows good scalability, but they lack of latent controllability
- StyleGANs show good controlability, but have poor scalability to large datasets

**StyleGAN-XL**: Careful architecture modification of StyleGAN can acheive the goal

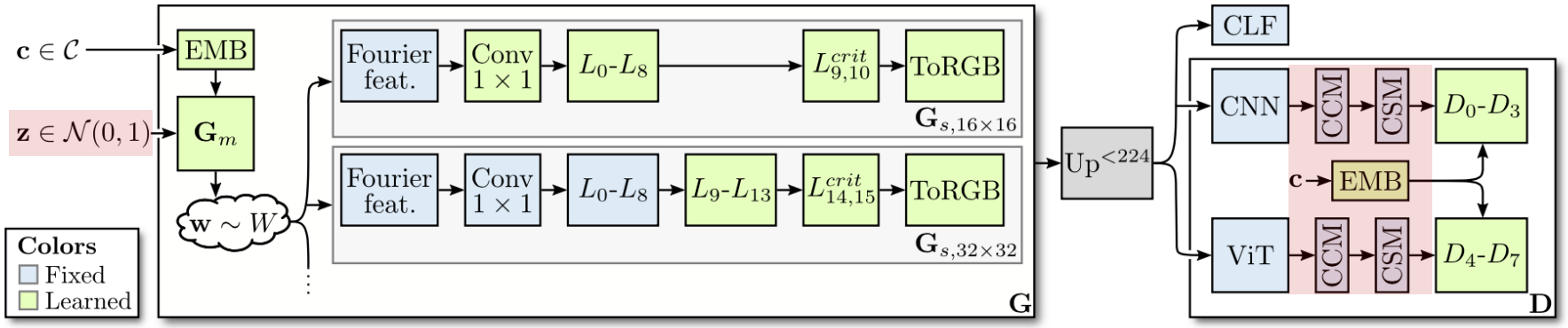
- Build upon the most recent StyleGAN3


Fig. 1. Class-conditional samples generated by StyleGAN3 (left) and StyleGAN-XL (right) trained on ImageNet at resolution $256^2$.

**Update 1:** Use projected GAN + smaller dimension for z

- **Projected GAN**: Exploit pre-trained feature from large image datasets
- **Small z:** Conventional $z \in \mathbb{R}^{512}$ is too high-dimensional and redundant
  - Reduce the dimension to 64, while maintaining the style code dimension $w \in \mathbb{R}^{512}$

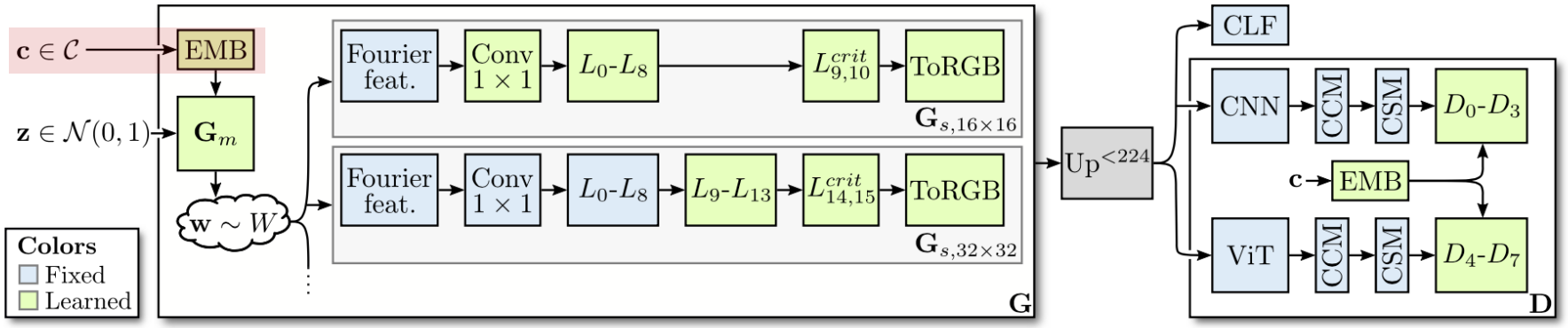
| | Configuration | FID ↓ | IS ↑ |
|---|---|---|---|
| **A** | StyleGAN3 | 53.57 | 15.30 |
| **B** | + Projected GAN & small **z** | 22.98 | 57.62 |
| **C** | + Pretrained embeddings | 20.91 | 35.79 |
| **D** | + Progressive growing | 19.51 | 35.74 |
| **E** | + ViT & CNN as $\mathbf{F}_{1,2}$ | 12.43 | 56.72 |
| **F** | + CLF guidance (StyleGAN-XL) | **12.24** | **86.21** |



**Update 2:** Use pretrained class embedding instead of fully learnable embedding

- **Prior work**: Use 512-dimensional embedding (learnable) from the one-hot class label
- **StyleGAN-XL:** Learn a mapping from pretrained class embedding (feature mean of each ImageNet class) to the 64-dimensional embedding

| | Configuration | FID ↓ | IS ↑ |
|---|---|---|---|
| **A** | StyleGAN3 | 53.57 | 15.30 |
| **B** | + Projected GAN & small **z** | 22.98 | 57.62 |
| **C** | + Pretrained embeddings | 20.91 | 35.79 |
| **D** | + Progressive growing | 19.51 | 35.74 |
| **E** | + ViT & CNN as $\mathbf{F}_{1,2}$ | 12.43 | 56.72 |
| **F** | + CLF guidance (StyleGAN-XL) | **12.24** | **86.21** |



**Update 3:** Reintroducing progressive growing

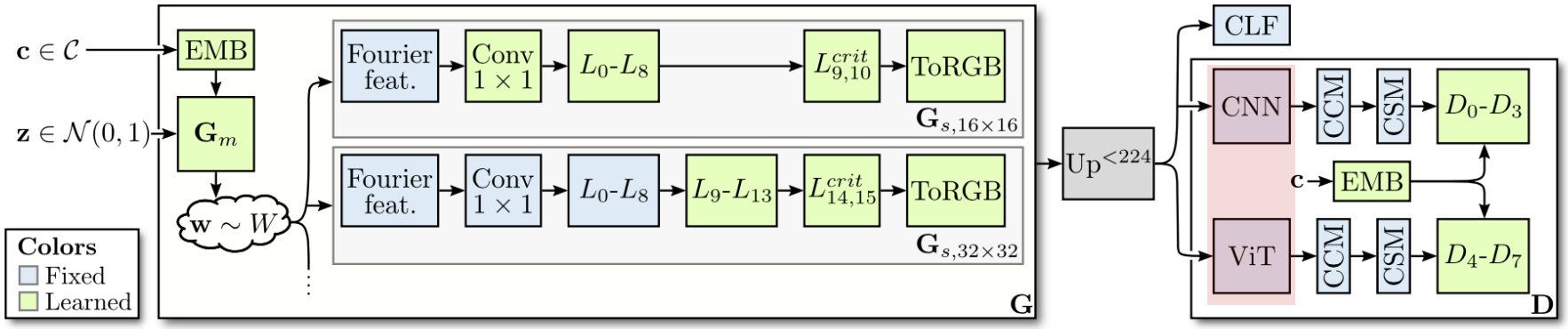
- **Progressive growing**: Discard in early StyleGANs as it is known to cause "texture striking" problem
- StyleGAN3 shows aliasing-preventing layer can prevent it; StyleGAN-XL reconsiders progressive growing

| | Configuration | FID ↓ | IS ↑ |
|---|---|---|---|
| **A** | StyleGAN3 | 53.57 | 15.30 |
| **B** | + Projected GAN & small **z** | 22.98 | 57.62 |
| **C** | + Pretrained embeddings | 20.91 | 35.79 |
| **D** | + Progressive growing | 19.51 | 35.74 |
| **E** | + ViT & CNN as $\mathbf{F}_{1,2}$ | 12.43 | 56.72 |
| **F** | + CLF guidance (StyleGAN-XL) | **12.24** | **86.21** |

**Update 4:** Use multiple pretrained networks for projected GAN

- **Projected GAN**: Investigates which pretrained backbone is the most beneficial, but not in their synergy
- In addition to EfficientNet (most beneficial), adopting ViT provides dramatic performance improvement

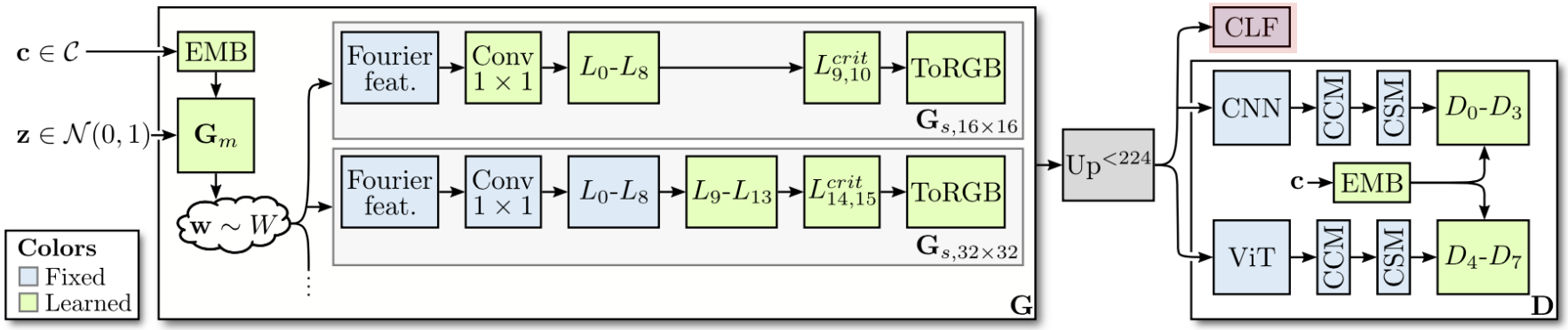
| | Configuration | FID ↓ | IS ↑ |
|---|---|---|---|
| **A** | StyleGAN3 | 53.57 | 15.30 |
| **B** | + Projected GAN & small **z** | 22.98 | 57.62 |
| **C** | + Pretrained embeddings | 20.91 | 35.79 |
| **D** | + Progressive growing | 19.51 | 35.74 |
| **E** | + ViT & CNN as $\mathbf{F}_{1,2}$ | 12.43 | 56.72 |
| **F** | + CLF guidance (StyleGAN-XL) | **12.24** | **86.21** |



**Update 5:** Classifier guidance during the training

- To further benefit from class label, use classifier guidance during training (NOT the guidance on sampling)
- Regularize the generator with cross-entropy loss on generated image and corresponding lables

| | Configuration | FID ↓ | IS ↑ |
|---|---|---|---|
| **A** | StyleGAN3 | 53.57 | 15.30 |
| **B** | + Projected GAN & small **z** | 22.98 | 57.62 |
| **C** | + Pretrained embeddings | 20.91 | 35.79 |
| **D** | + Progressive growing | 19.51 | 35.74 |
| **E** | + ViT & CNN as $\mathbf{F}_{1,2}$ | 12.43 | 56.72 |
| **F** | + CLF guidance (StyleGAN-XL) | **12.24** | **86.21** |



**Experiment:** Outperforms prior arts on class-conditional ImageNet generation

- Remarkably, it first succeeds ImageNet generation on $1024^2$ resolution

| Model | FID ↓ | sFID ↓ | rFID ↓ | IS ↑ | Pr ↑ | Rec ↑ |
|---|---|---|---|---|---|---|
| **Resolution** $128^2$ | | | | | | |
| BigGAN-deep ($\psi = 1.0$) [5] | 6.02 | 7.18 | 6.09 | 145.83 | 0.82 | 0.36 |
| BigGAN-deep ($\psi = 0.1$) [5] | 29.46 | 54.50 | 12.07 | 261.46 | **0.86** | 0.00 |
| CDM [22] | 3.52 | | | 128.80 | | |
| ADM [13] | 5.91 | 5.09 | 13.29 | 93.31 | 0.71 | **0.57** |
| ADM-G [13] | 2.97 | 5.09 | 3.80 | 141.37 | 0.75 | 0.56 |
| **StyleGAN-XL** ($\psi = 1.0$) | **2.54** | **4.02** | **1.31** | 180.58 | 0.75 | 0.46 |
| **StyleGAN-XL** ($\psi = 0.1$) | 28.29 | 47.44 | 64.85 | **369.98** | **0.86** | 0.00 |
| **Resolution** $512^2$ | | | | | | |
| BigGAN-deep ($\psi = 1.0$) [5] | 8.43 | 8.13 | 312.00 | 177.90 | 0.85 | 0.25 |
| BigGAN-deep ($\psi = 0.1$) [5] | 27.30 | 70.30 | 169.15 | 291.80 | **0.87** | 0.00 |
| ADM [13] | 23.24 | 10.19 | 561.32 | 58.06 | 0.70 | 0.39 |
| ADM-G-U [13] | 3.85 | 5.86 | 210.83 | 221.72 | 0.81 | **0.45** |
| **StyleGAN-XL** ($\psi = 1.0$) | **3.58** | **4.35** | **48.03** | 219.77 | 0.73 | 0.43 |
| **StyleGAN-XL** ($\psi = 0.1$) | 25.21 | 47.73 | 263.50 | **409.12** | 0.75 | 0.00 |

| Model | FID ↓ | sFID ↓ | rFID ↓ | IS ↑ | Pr ↑ | Rec ↑ |
|---|---|---|---|---|---|---|
| **Resolution** $256^2$ | | | | | | |
| StyleGAN2 [18] | 49.20 | | | | | |
| BigGAN-deep ($\psi = 1.0$) [5] | 6.95 | 7.36 | 75.24 | 202.65 | 0.86 | 0.24 |
| BigGAN-deep ($\psi = 0.1$) [5] | 27.00 | 69.88 | 19.66 | 313.67 | **0.88** | 0.00 |
| CDM [22] | 4.88 | | | 158.70 | | |
| ADM [13] | 10.94 | 6.02 | 125.78 | 100.98 | 0.72 | **0.50** |
| ADM-G-U [13] | 3.94 | 6.14 | 11.86 | 215.84 | 0.81 | 0.47 |
| **StyleGAN-XL** ($\psi = 1.0$) | **3.26** | **4.22** | **4.28** | 225.27 | 0.74 | 0.45 |
| **StyleGAN-XL** ($\psi = 0.1$) | 25.84 | 47.58 | 63.88 | **413.94** | 0.78 | 0.00 |
| **Resolution** $1024^2$ | | | | | | |
| **StyleGAN-XL** ($\psi = 1.0$) | **3.71** | **4.54** | **518.61** | 195.66 | 0.74 | **0.38** |
| **StyleGAN-XL** ($\psi = 0.1$) | 26.45 | 49.17 | 1522.00 | **367.97** | **0.80** | 0.00 |

**Experiment:** Also enjoys controllability of StyleGAN (interpolation/manipulation)


Fig. 5. **Interpolations**. StyleGAN-XL generates smooth interpolations between samples of different classes (Row 1 & Row 2). PTI [49] allows inverting to the latent space with low distortion (outermost image, Row 3 & Row 4), and consistently embeds out-of-domain inputs, such as the one on the bottom right.

**Experiment:** Also enjoys controllability of StyleGAN (interpolation/manipulation)

# Scaling GANs tor zero-shot Text-to-Image Synthesis [Sauer et al., 2023; Kang et al., 2023]

**Question:** Can we scale-up GANs even for zero-shot text-to-image syntehsis?

- There are two concurrent works that achieved remarkable progress
- In contrast to diffusion models, synthesis speed is really fast (<1 second)

**StyleGAN-T:** First achieves text-to-image generation using GANs

**GigaGAN:** Also achieves text-to-image generation, even on megapixel resolution

"a victorian house" → "a modern house"

"a cute puppy" → "a cute blue puppy, Madhubani painting"

"a landscape in winter" → "a landscape in fall"

A portrait of a human growing colorful flowers from her hair. Hyperrealistic oil painting. Intricate details.

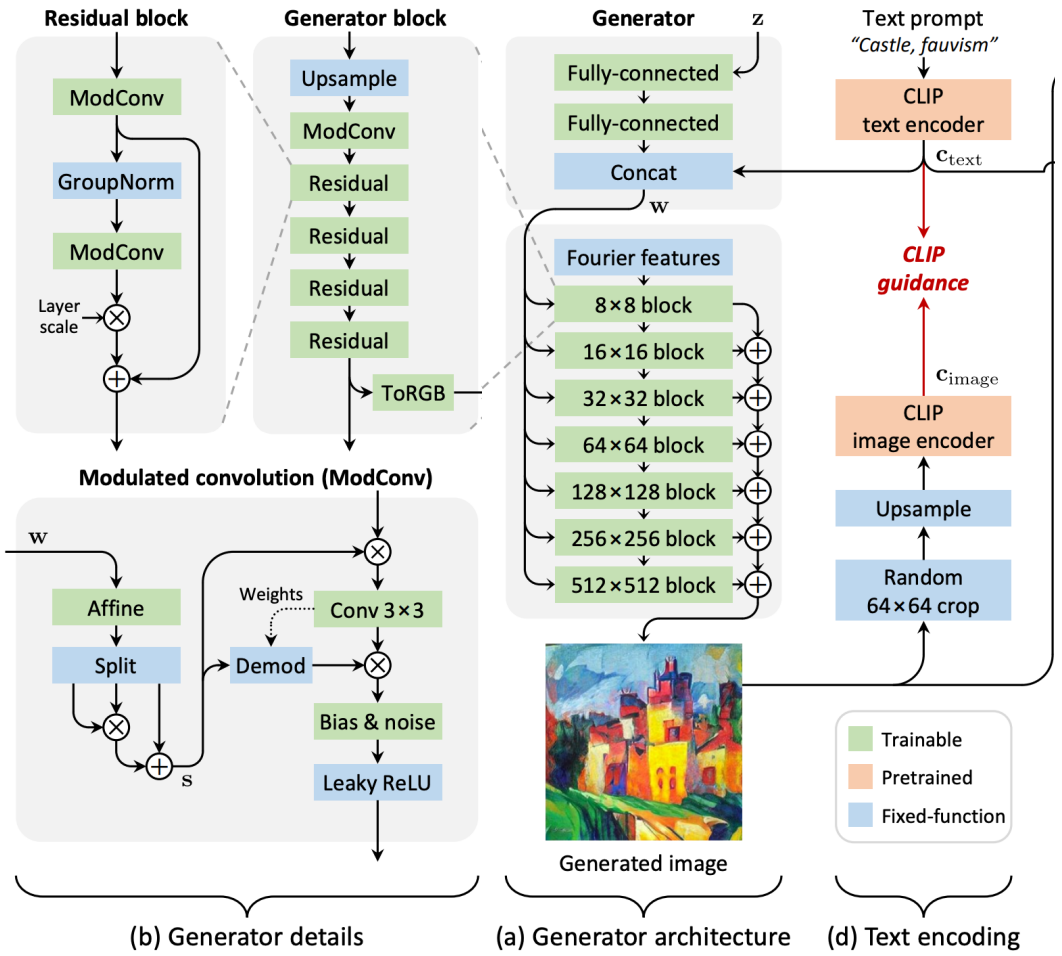
**StyleGAN-T:** Re-design StyleGAN-XL to achieve better scalability

**Generator:** Return-back to StyleGAN2 generator (instead of StyleGAN3)

- As equivariance may be too hard condition to achieve for T2I

**Some other details:**

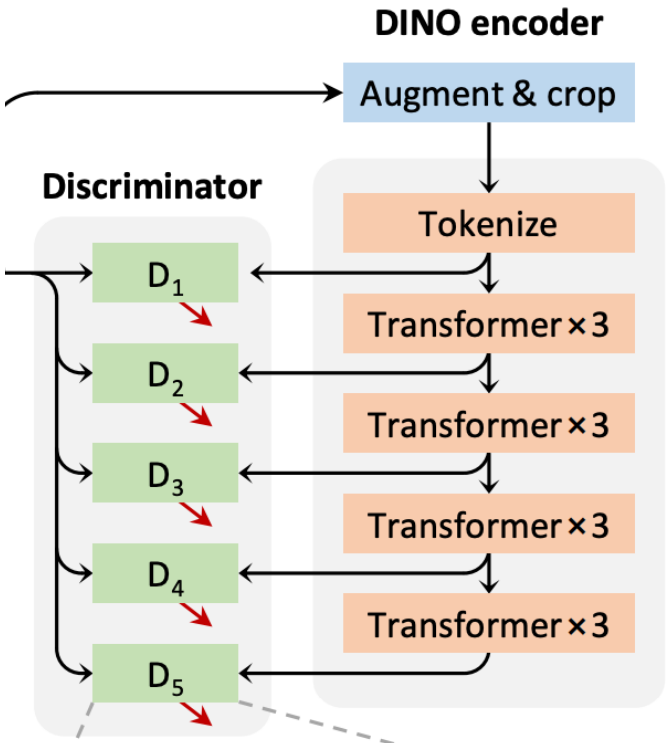
- Uses recent residual blocks for increasing the model capacity (normalize → block → scale)
- Text condition $c$ is often ignored → bypass it to mapping network and concatenate directly to $w$



**StyleGAN-T:** Re-design StyleGAN-XL to achieve better scalability

**Discriminator:** Redesign based on the principle of StyleGAN-XL

- **Pretrained feature network:** Switch to ViT trained with DINO
  - Lightweight, encodes semantic information at high-resolution
  - Leads to ~2.5x faster training compared with StyleGAN-XL



**Experiment:** Shows reasonable synthesis results with remarkably fast inference

| Model | Model type | Zero-shot $FID_{30k}$ | Speed [s] |
|---|---|---|---|
| Stable Diffusion * | Diffusion | 8.40 | – |
| eDiff-I | Diffusion | 7.60 | 26.0 |
| LDM * | Diffusion | 7.59 | – |
| GLIDE | Diffusion | 7.40 | 10.9 |
| LAFITE * | GAN | 14.80 | ∼**0.01** |
| StyleGAN-T | GAN | **7.30** | 0.06 |

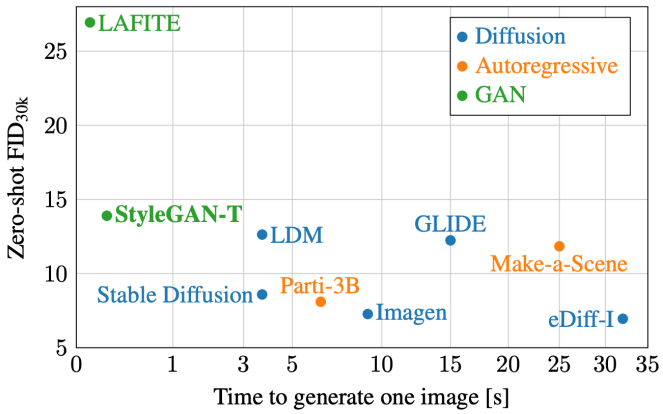
* downsampled to 64×64 pixels using Lanczos – not available

*Table 2.* **Comparison of FID on MS COCO 64×64.** Inference speeds are measured on an A100. For LAFITE we estimate what its speed would be at a native 64×64 resolution.

| Model | Model type | Zero-shot $FID_{30k}$ | Speed [s] |
|---|---|---|---|
| LDM | Diffusion | 12.63 | 3.7 |
| GLIDE | Diffusion | 12.24 | 15.0 |
| DALL·E 2 | Diffusion | 10.39 | – |
| Stable Diffusion * | Diffusion | 8.59 | 3.7 |
| Imagen | Diffusion | 7.27 | 9.1 |
| eDiff-I | Diffusion | **6.95** | 32.0 |
| DALL·E | Autoregressive | 27.50 | – |
| Ernie-ViLG | Autoregressive | 14.70 | – |
| Make-A-Scene * | Autoregressive | 11.84 | 25.0 |
| Parti-3B | Autoregressive | 8.10 | 6.4 |
| Parti-20B | Autoregressive | 7.23 | – |
| LAFITE | GAN | 26.94 | **0.02** |
| StyleGAN-T * | GAN | 13.90 | 0.10 |

* downsampled to 256×256 pixels using Lanczos – not available

*Table 3.* **Comparison of FID on MS COCO 256×** ıfer-




"a victorian house" → "a modern house"

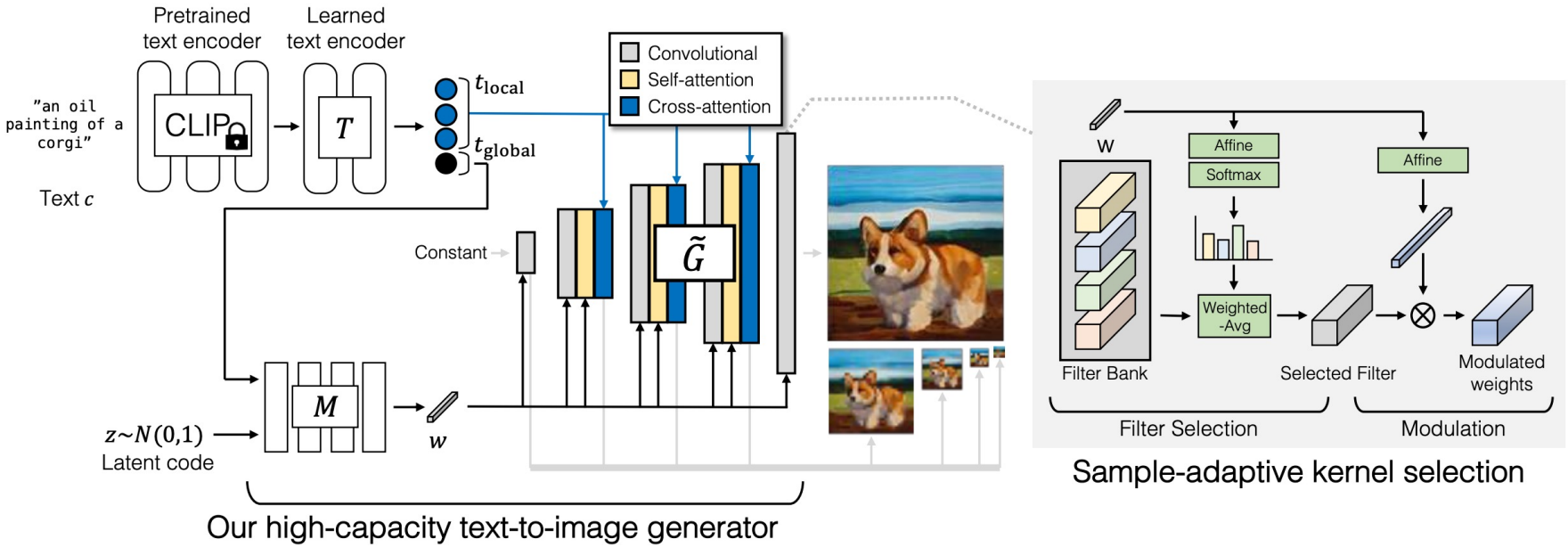
"a cute puppy" → "a cute blue puppy, Madhubani painting"

"a landscape in winter" → "a landscape in fall"

**GigaGAN:** Shows more scalable text-to-image synthesis results

**Generator:** Also starts from StyleGAN2 generator architecture

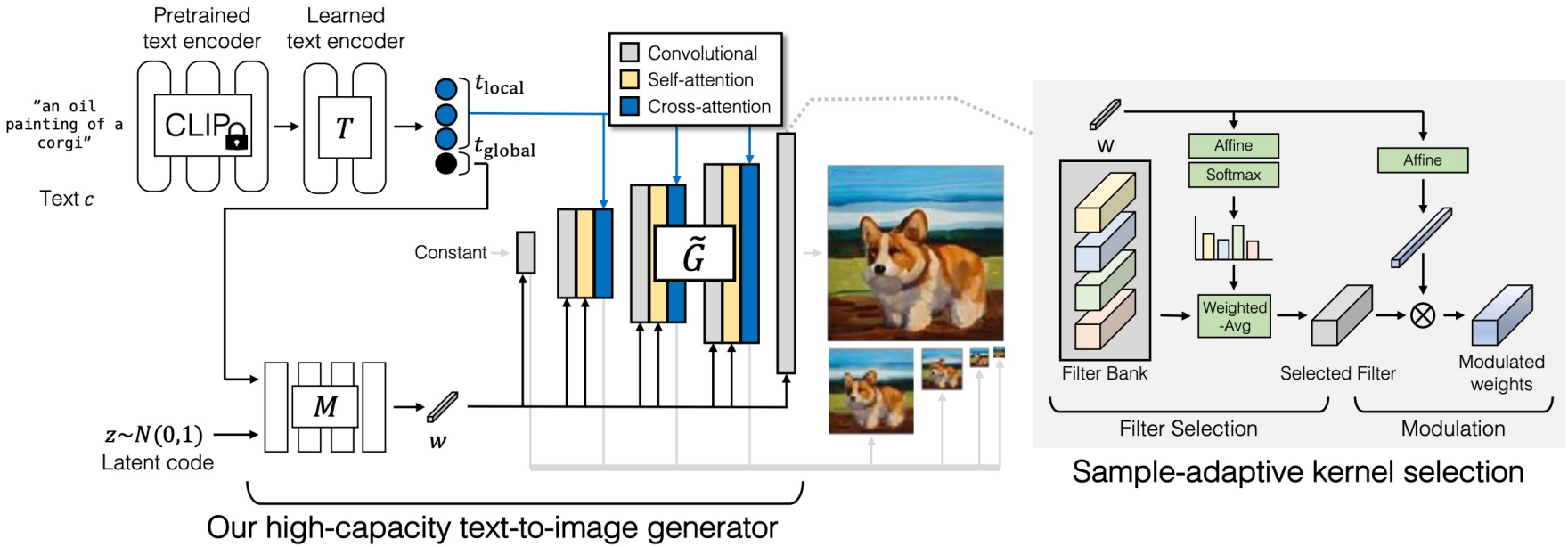
- Proposes several components to increase the model capacity in stable manner



**GigaGAN:** Shows more scalable text-to-image synthesis results

**Update 1.** Interleaving attention with convolution for long range relationship

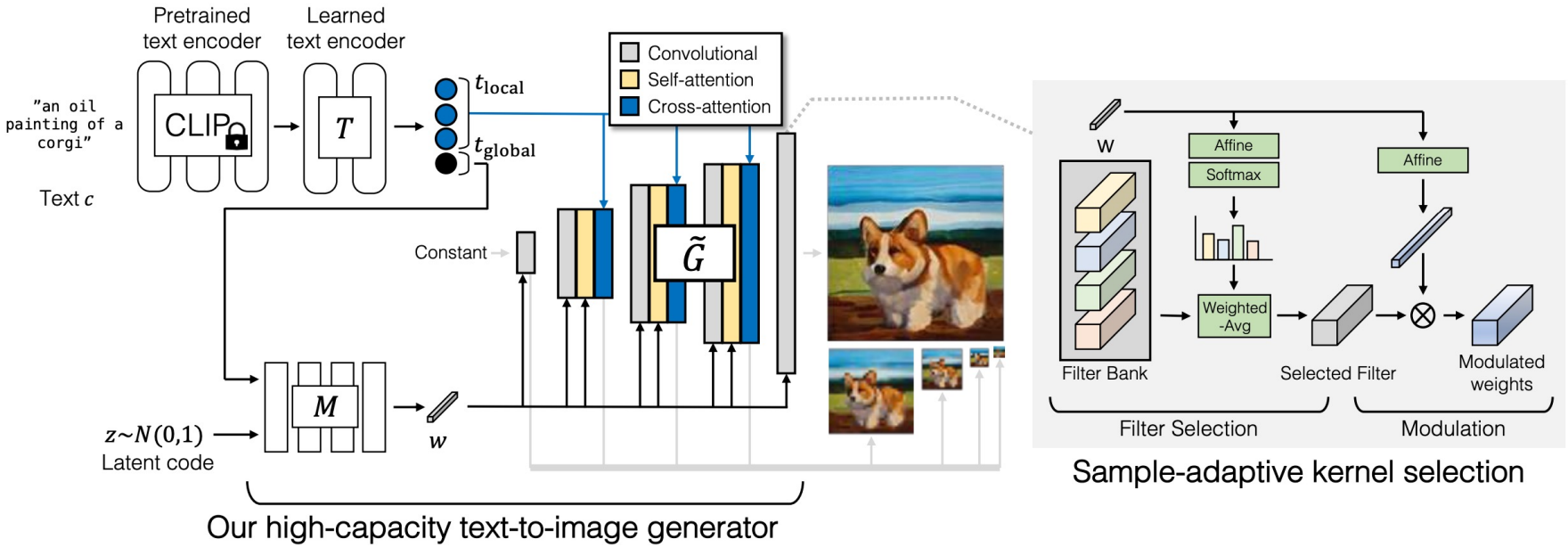
- To stabilize the training, forced Lipschitz continuity for the attention layers
- Used techniques in ViT-GAN that adopts attentions in GANs [Lee et al., 2022]



**GigaGAN:** Shows more scalable text-to-image synthesis results

**Update 2.** Improved text conditioning: use both of local/global text information

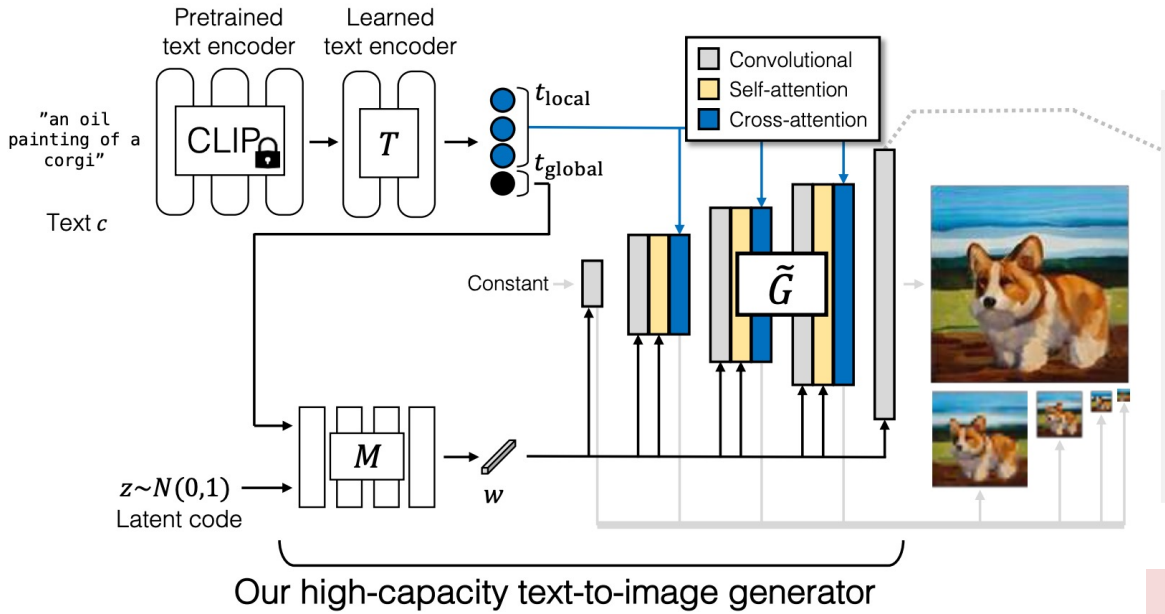
- **Local information** – feed into intermediate attention layer of the generator
- **Global information** – feed into mapping network to get the style code w

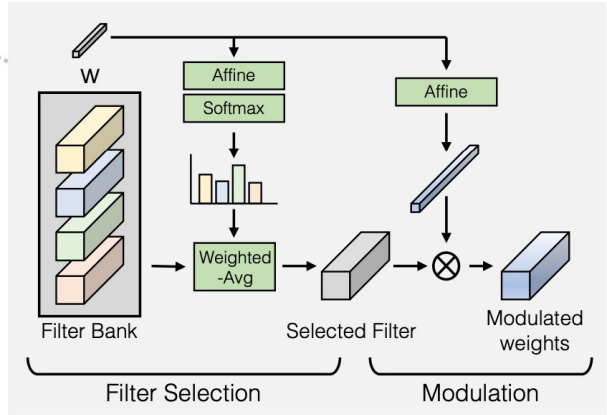


**GigaGAN:** Shows more scalable text-to-image synthesis results

**Update 3.** Sample adaptive kernel selection to increase the capacity of conv. layer

- Consider a "bank" of N convolution filters
- Style vector predict a set of weights to average across the filters



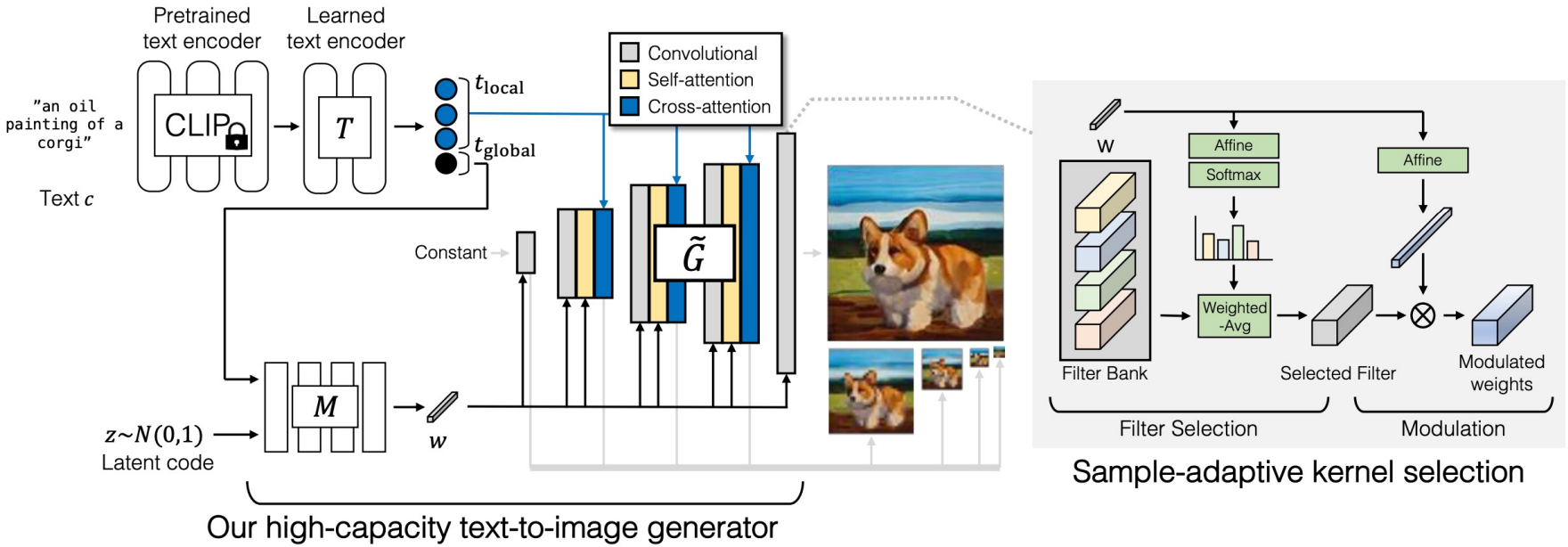

Sample-adaptive kernel selection

$$\mathbf{K} = \sum_{i=1}^{N} \mathbf{K}_i \cdot \mathrm{softmax}\left(W_{\text{filter}}^{\top}\mathbf{w} + b_{\text{filter}}\right)_i$$

**GigaGAN:** Shows more scalable text-to-image synthesis results

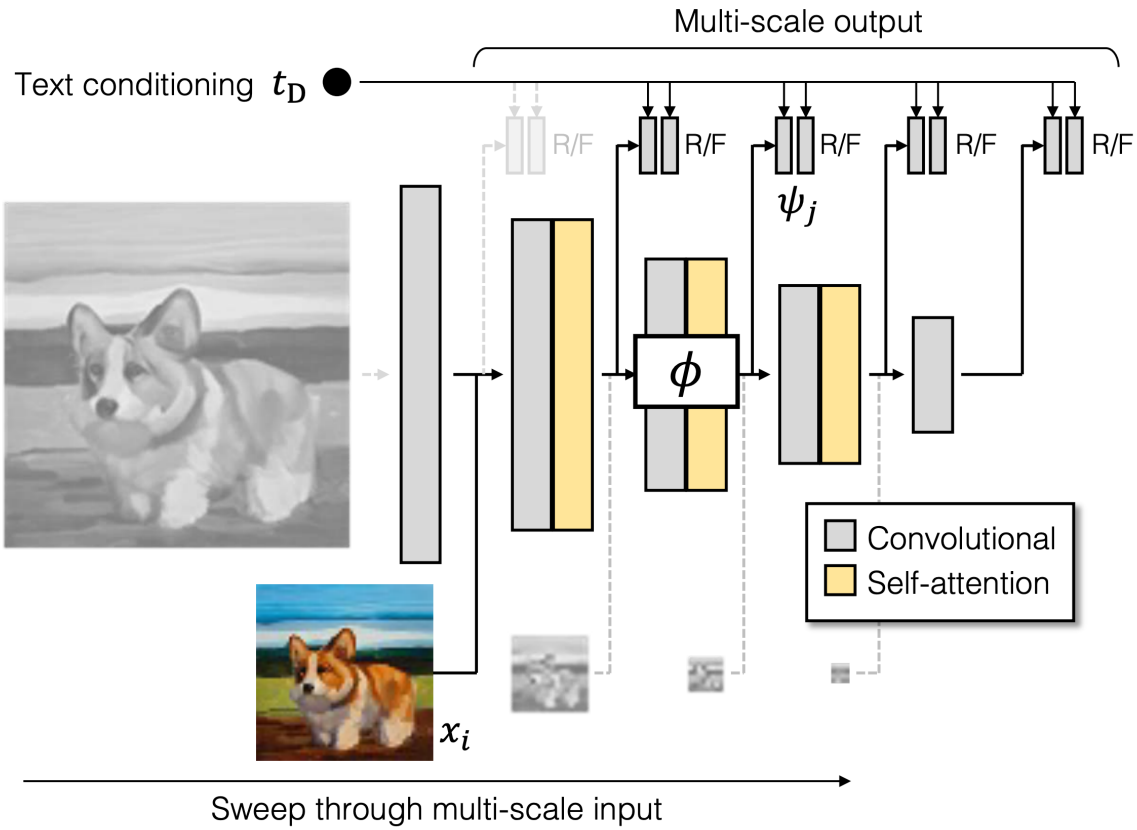
**Update 4.** Trains additional text-guided GAN-based upsampler

- Based on U-Net architectures
- Compared with diffusion model, can be efficient as it only requires a single step



**GigaGAN:** Shows more scalable text-to-image synthesis results

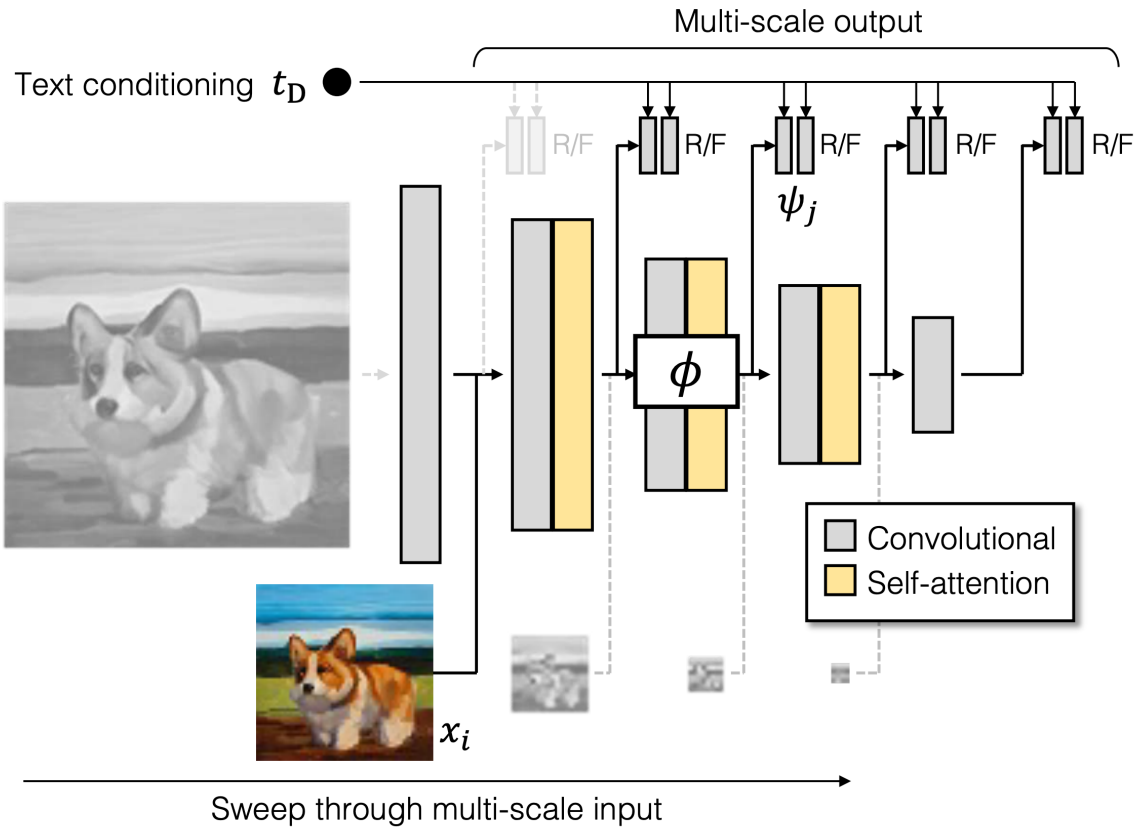
**Discriminator.** Proposes architecture/objectives specialized for text-condition



**GigaGAN:** Shows more scalable text-to-image synthesis results

**Update 1.** Design discriminator from a pretrained network weights (CLIP)

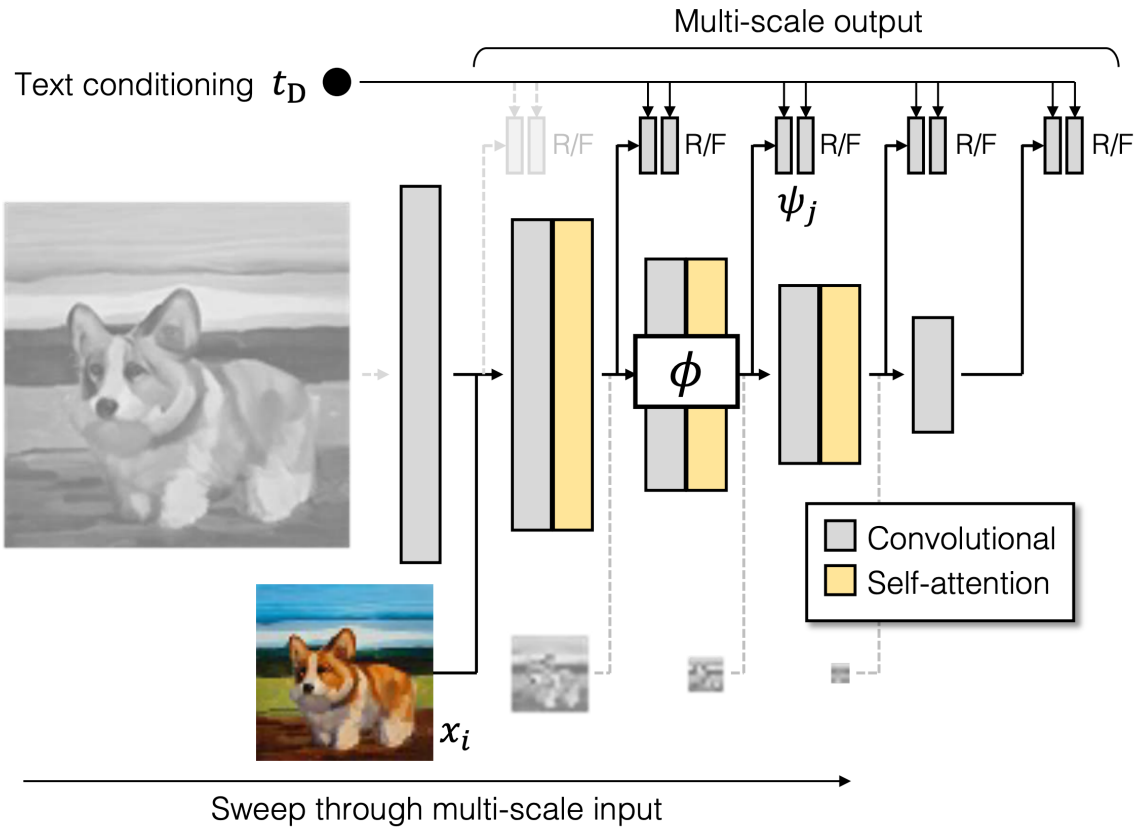
- A few learnable attention layers (yellow) are added



**GigaGAN:** Shows more scalable text-to-image synthesis results

**Update 2.** Multi-scale input, multi-scale output adversarial loss

- If not, early, low-resolution layers of the generator easily become inactive



**GigaGAN:** Shows more scalable text-to-image synthesis results

**Update 3.** Matching loss to enforce discriminators to incorporate conditioning

- Provide the sample and the random text condition as a fake pair as well

$$\mathcal{V}_{\text{match}} = \mathbb{E}_{\mathbf{x},\mathbf{c},\hat{\mathbf{c}}}\left[\log(1+\exp(D(\mathbf{x},\hat{\mathbf{c}}))) + \log(1+\exp(D(G(\mathbf{c}),\hat{\mathbf{c}}))\right]$$

**Update 4.** CLIP contrastive loss to enforce the generator

$$\mathcal{L}_{\text{CLIP}} = \mathbb{E}_{\{\mathbf{c}_n\}}\left[-\log\frac{\exp(\mathcal{E}_{\text{img}}(G(\mathbf{c}_0))^\top \mathcal{E}_{\text{txt}}(\mathbf{c}_0))}{\sum_n \exp(\mathcal{E}_{\text{img}}(G(\mathbf{c}_0))^\top \mathcal{E}_{\text{txt}}(\mathbf{c}_n)}\right]$$

**Experiment:** Shows scalable text-to-image synthesis results (up to megapixels)

| | Model | Type | # Param. | # Images | FID-30k ↓ | Inf. time |
|---|---|---|---|---|---|---|
| 256 | DALL·E [75] | Diff | 12.0B | 1.54B | 27.50 | - |
| | GLIDE [63] | Diff | 5.0B | 5.94B | 12.24 | 15.0s |
| | LDM [79] | Diff | 1.5B | 0.27B | 12.63 | 9.4s |
| | DALL·E 2 [74] | Diff | 5.5B | 5.63B | 10.39 | - |
| | Imagen [80] | Diff | 3.0B | 15.36B | 7.27 | 9.1s |
| | eDiff-I [5] | Diff | 9.1B | 11.47B | 6.95 | 32.0s |
| | Parti-750M [101] | AR | 750M | 3.69B | 10.71 | - |
| | Parti-3B [101] | AR | 3.0B | 3.69B | 8.10 | 6.4s |
| | Parti-20B [101] | AR | 20.0B | 3.69B | 7.23 | - |
| | LAFITE [108] | GAN | 75M | - | 26.94 | 0.02s |
| 512 | SD-v1.5* [78] | Diff | 0.9B | 3.16B | 9.62 | 2.9s |
| | Muse-3B [10] | AR | 3.0B | 0.51B | 7.88 | 1.3s |
| | **GigaGAN** | GAN | 1.0B | 0.98B | 9.09 | 0.13s |


Isometric underwater Atlantis city with a Greek temple in a bubble.

A hot air balloon in shape of a heart. Grand Canyon

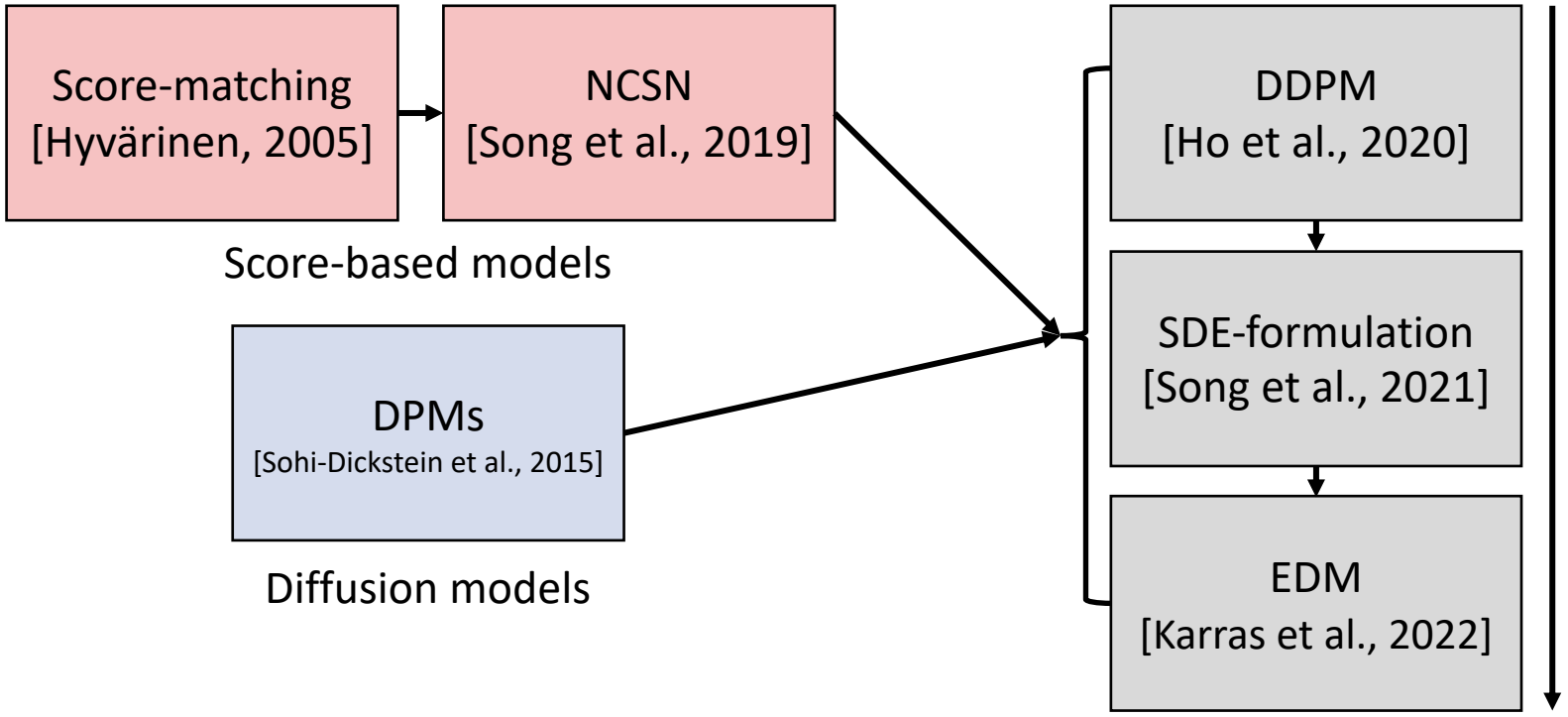
low poly bunny with cute eyes

A cube made of denim on a wooden table

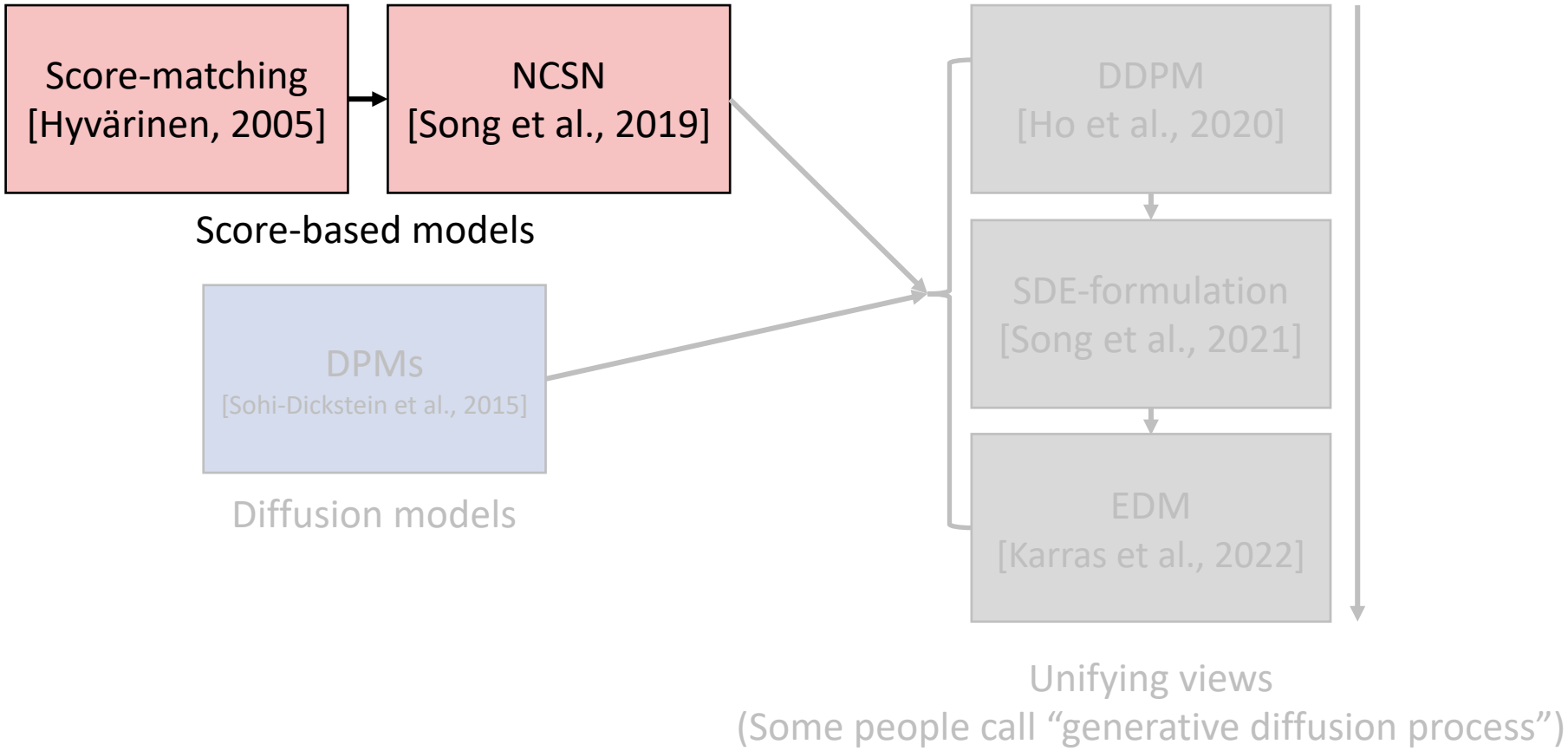
## Table of Contents

**"Diffusion models" and "score-based models" are now used interchangeably**

- But they share lots of similarities and differences simultaneously
- In this lecture, we first clarify how they are similar and different



# Overview: Score-based models and diffusion models



Score matching [Hyvärinen, 2005]

Score matching: Match the *scores* of data and model distribution $\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

- However, we **don't know** the scores of data distribution
- Instead, one can use the **equivalent form** (proof by integration of parts)

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\|\mathbf{s}_\theta(\mathbf{x}) - \mathbf{s}_{\text{data}}(\mathbf{x})\|_2^2] = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\text{tr}(\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x})) + \frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|_2^2\right] + \text{const.}$$

**Sampling:** Done with iterative procedure

- Starting from arbitrary prior distribution $\mathbf{x}_0 \sim \pi(\mathbf{x})$
- It is known as "Langevin dynamics"

$$\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\,\mathbf{z}_i, \quad i = 0, 1, \cdots, K$$

Step size — Gaussian noise

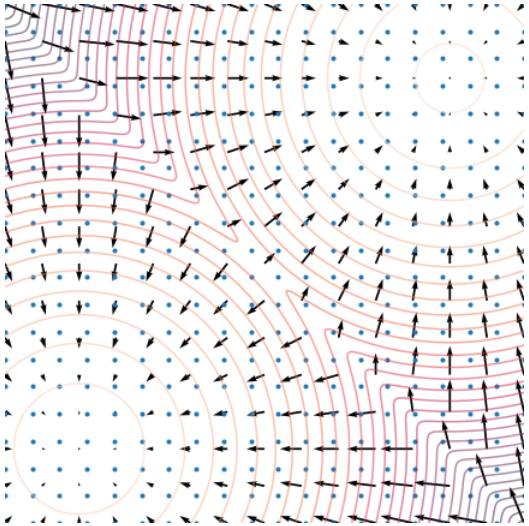
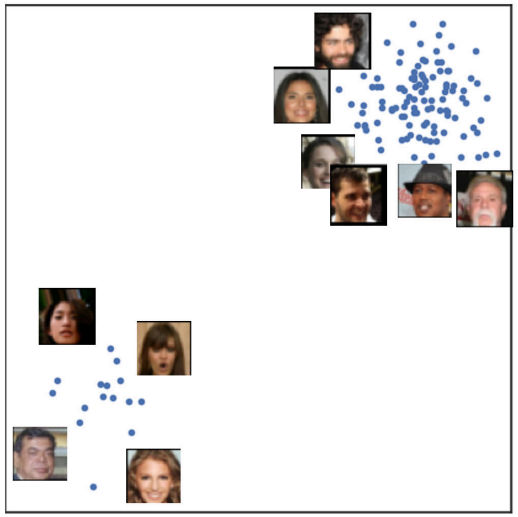


Image from https://yang-song.net/blog/2021/score/

Score matching [Hyvärinen, 2005]

Score matching: Match the *scores* of data and model distribution $\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

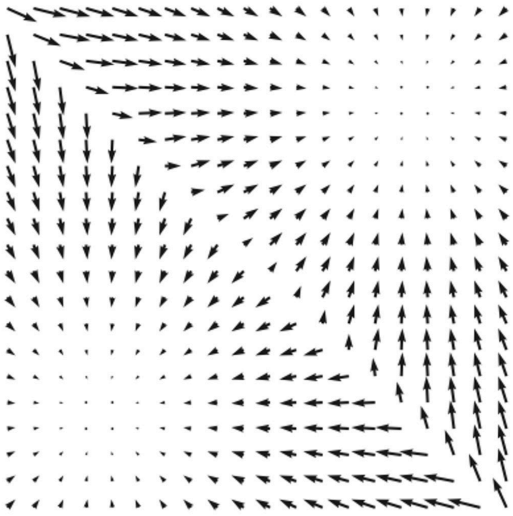
- However, we **don't know** the scores of data distribution
- Instead, one can use the **equivalent form** (proof by integration of parts)



Data samples

$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$

score matching



Scores

$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

Langevin dynamics

New samples

Image from https://yang-song.net/blog/2021/score/

Score matching [Hyvärinen, 2005]

Score matching: Match the *scores* of data and model distribution $\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$

- However, we **don't know** the scores of data distribution
- Instead, one can use the **equivalent form** (proof by integration of parts)

$$\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\|\mathbf{s}_\theta(\mathbf{x}) - \mathbf{s}_{\text{data}}(\mathbf{x})\|_2^2] = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\text{tr}(\nabla_{\mathbf{x}}\mathbf{s}_\theta(\mathbf{x})) + \frac{1}{2}\|\mathbf{s}_\theta(\mathbf{x})\|_2^2\right] + \text{const.}$$

**Weighted sum:** Leads inaccurate results in low-density region

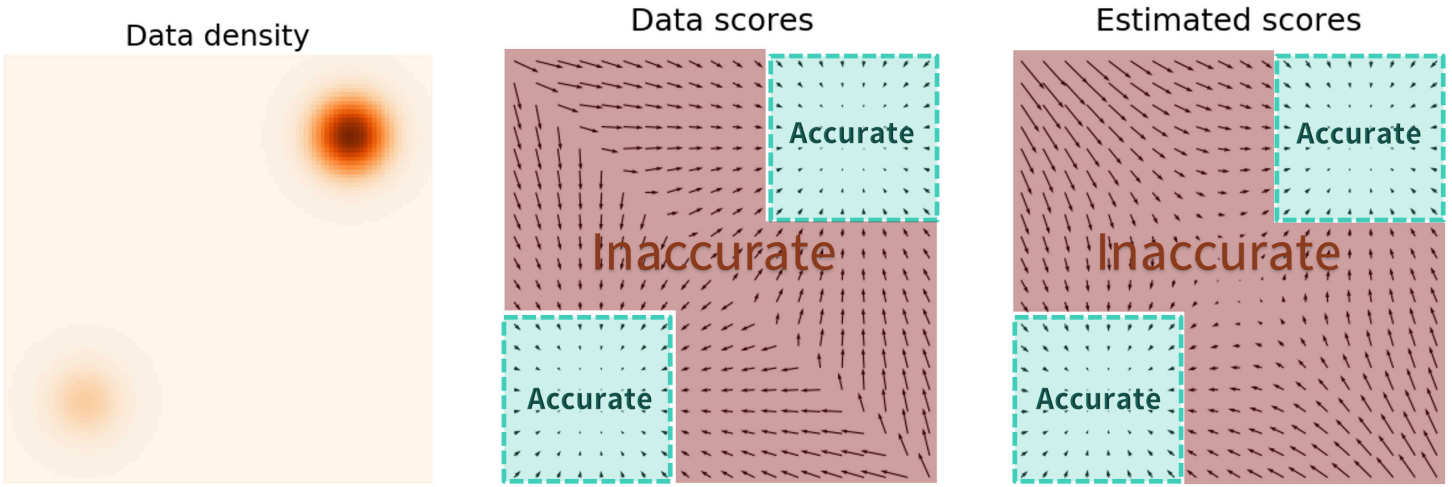
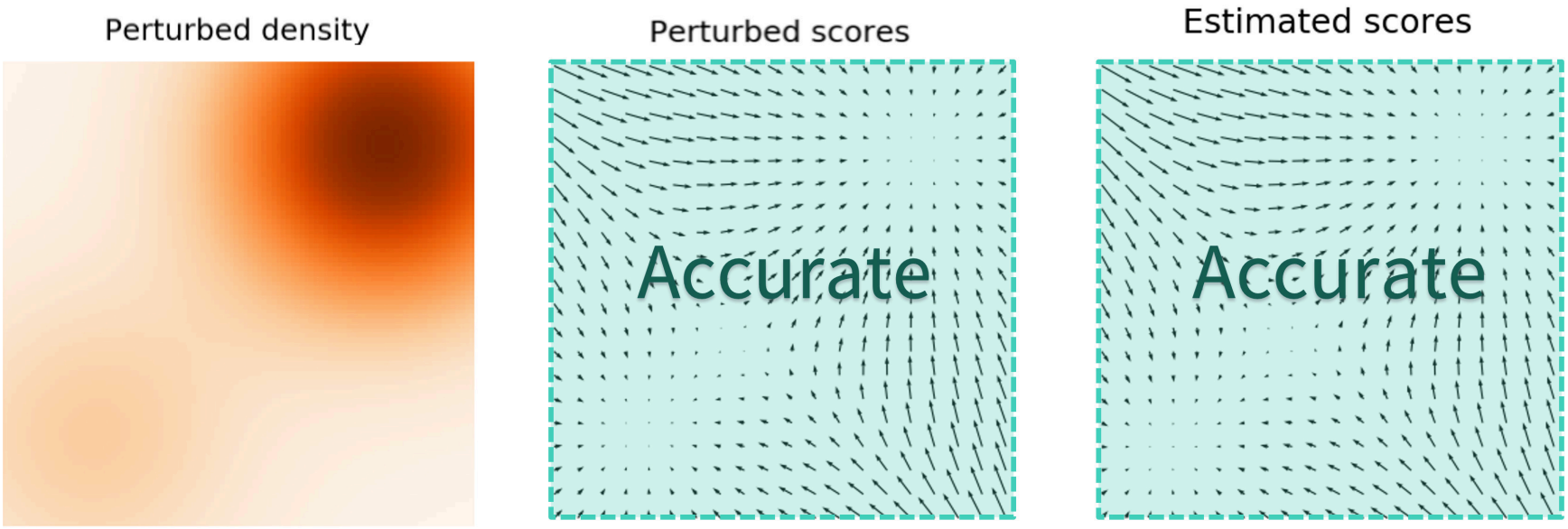



Image from https://yang-song.net/blog/2021/score/

How to mitigate this problem?

- **Perturb** data points with noise and train score-based models on the noisy data
- With large noise scale, can improve score estimation on low data density




**Tradeoff** caused by different noise scale:

- **Large noise:** Cover more low-density region but over-corrupts data from original one
- **Small noise:** Does not cover low-density region but preserves the original data

Image from https://yang-song.net/blog/2021/score/

**NCSN**: Use multiple noise scales simultaneously to achieve the best of both words

1. Consider noise-perturbed distribution for decreasing stdev. $\{\sigma_i\}_{i=1}^{L}$

$$p_{\sigma_i}(\mathbf{x}) = \int p(\mathbf{y})\mathcal{N}(\mathbf{x}; \mathbf{y}, \sigma_i^2 I)\mathrm{d}\mathbf{y}$$

Sampled easily: $\mathbf{x} + \sigma_i \mathbf{z}$, with $\mathbf{z} \sim \mathcal{N}(0, I)$

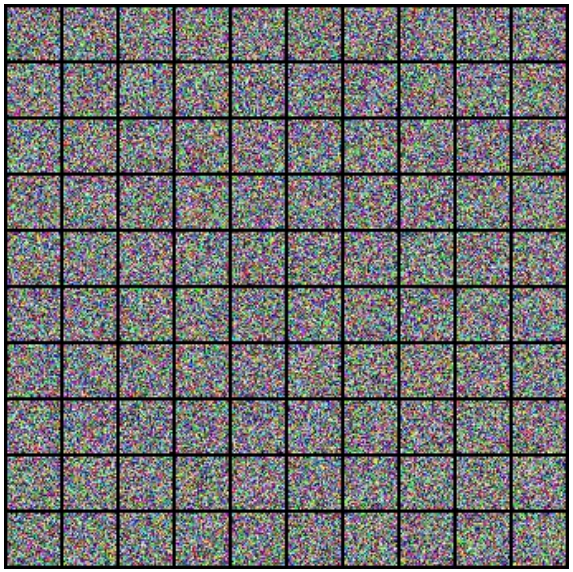
2. Train the score model $\mathbf{s}_\theta(\mathbf{x}, i) \approx \nabla_{\mathbf{x}} \log p_{\sigma_i}(\mathbf{x})$

**Sampling:** Done with "annealed" Langevin dynamics

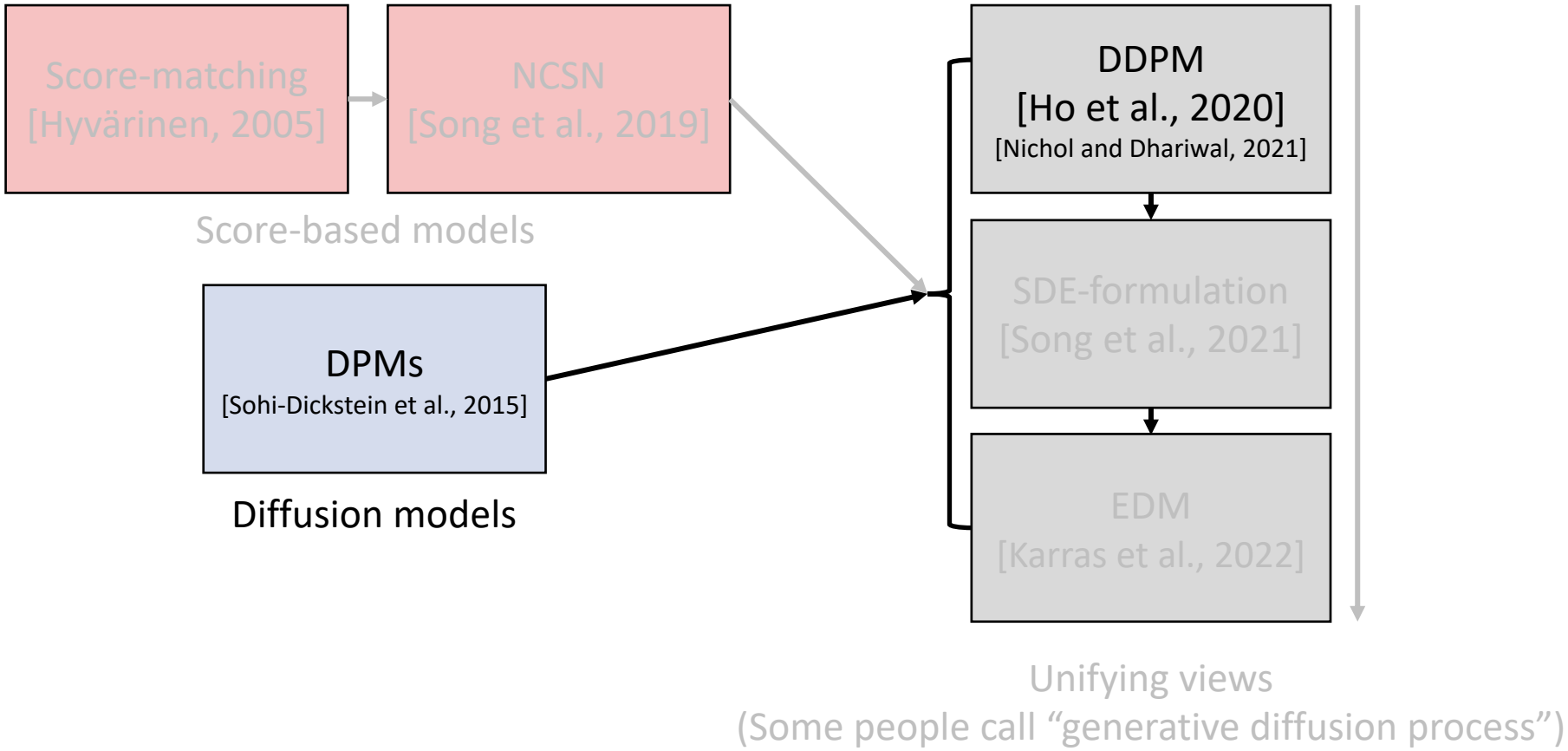
**Algorithm 1** Annealed Langevin dynamics.

**Require:** $\{\sigma_i\}_{i=1}^{L}, \epsilon, T$.

Initialize $\tilde{\mathbf{x}}_0$
**for** $i \leftarrow 1$ to $L$ **do**
  $\alpha_i \leftarrow \epsilon \cdot \sigma_i^2 / \sigma_L^2$ ▷ $\alpha_i$ is the step size.
  **for** $t \leftarrow 1$ to $T$ **do**
    Draw $\mathbf{z}_t \sim \mathcal{N}(0, I)$
    $\tilde{\mathbf{x}}_t \leftarrow \tilde{\mathbf{x}}_{t-1} + \frac{\alpha_i}{2}\mathbf{s}_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}_{t-1}, \sigma_i) + \sqrt{\alpha_i}\,\mathbf{z}_t$
  **end for**
  $\tilde{\mathbf{x}}_0 \leftarrow \tilde{\mathbf{x}}_T$
**end for**
**return** $\tilde{\mathbf{x}}_T$



# Overview: Score-based models and diffusion models



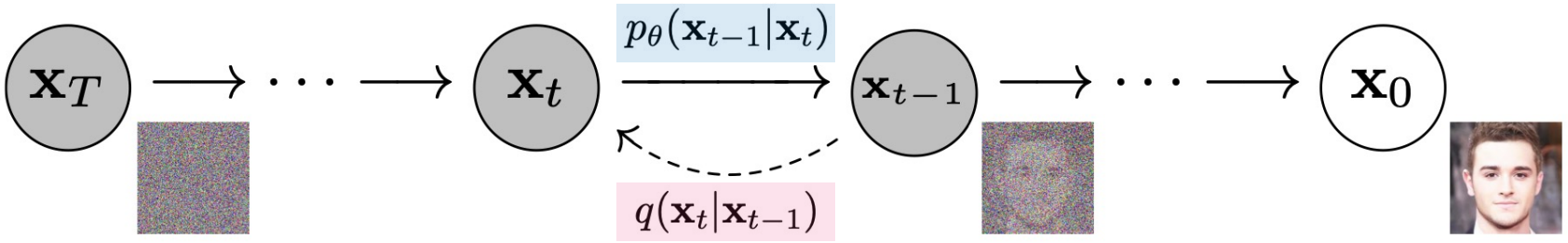
Diffusion probabilistic models [Sohl-Dickstein et al., 2015]

- Diffusion (forward) process: Markov chain that **gradually add noise** (of same dimension of data) to data until original the signal is destroyed

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

- Sampling (backward) process: Markov chain with learned Gaussian **denoising transition**, starting from standard Gaussian noise $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

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



Diffusion probabilistic models [Sohl-Dickstein et al., 2015]

- Here, the forward distribution $q(x_{t-1}|x_t, x_0)$ can be expressed as a closed form
- **Variational Lower Bound (VLB)** objective is given by the sum of local KL divergences (between Gaussians) – main difference with score-based models

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right]$$

$$= \mathbb{E}_q\Bigg[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Bigg]$$

Diffusion probabilistic models [Sohl-Dickstein et al., 2015]

- Here, the forward distribution $q(x_{t-1}|x_t, x_0)$ can be expressed as a closed form
- **Variational Lower Bound (VLB)** objective is given by the sum of local KL divergences (between Gaussians) – main difference with score-based models

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t\geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right]$$

$$= \mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \,\|\, p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,\|\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Big]$$

Closed form

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}),$$

$$\text{where} \quad \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

$$\alpha_t := 1 - \beta_t \text{ and } \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

DDPM [Ho et al., 2020] proposes a simple objective & model of DPMs:

(1) Use fixed constants for $\beta_t$ instead of letting them as learnable parameters

(2) For $\Sigma_\theta$, DDPM fix the variance $\Sigma_\theta(x_t, t) = \sigma_t^2 \boldsymbol{I}$, where $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$

(3) For $\mu_\theta$, DDPM uses the following parameterization to let model predict the noise $\epsilon$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right)$$

$$\mathbb{E}_q \Big[ \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \Big]$$

Ignored due to (1)
(no learnable parameters)

DDPM [Ho et al., 2020] proposes a simple objective & model of DPMs:

(1) Use fixed constants for $\beta_t$ instead of letting them as learnable parameters

(2) For $\Sigma_\theta$, DDPM fix the variance $\Sigma_\theta(x_t, t) = \sigma_t^2 \boldsymbol{I}$, where $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$

(3) For $\mu_\theta$, DDPM uses the following parameterization to let model predict the noise $\epsilon$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right)$$

$$\mathbb{E}_q \left[ \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]$$

Can be reduced to
"simple" denoising objective

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[ \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2 \right]$$

DDPM [Ho et al., 2020] proposes a simple objective & model of DPMs:

(1) Use fixed constants for $\beta_t$ instead of letting them as learnable parameters

(2) For $\Sigma_\theta$, DDPM fix the variance $\Sigma_\theta(x_t, t) = \sigma_t^2 \boldsymbol{I}$, where $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$

(3) For $\mu_\theta$, DDPM uses the following parameterization to let model predict the noise $\epsilon$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right)$$

$$\mathbb{E}_q \left[ \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]$$

$$\mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}} \left[ \frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2 \right]$$

Unweighted version (simple)
also works well

DDPM [Ho et al., 2020] proposes a simple objective & model of DPMs:

(1) Use fixed constants for $\beta_t$ instead of letting them as learnable parameters

(2) For $\Sigma_\theta$, DDPM fix the variance $\Sigma_\theta(x_t, t) = \sigma_t^2 \boldsymbol{I}$, where $\sigma_t^2 = \beta_t$ or $\tilde{\beta}_t$

(3) For $\mu_\theta$, DDPM uses the following parameterization to let model predict the noise $\epsilon$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right)$$

$$\mathbb{E}_q \Big[ \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \Big]$$

$$L_{\mathrm{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[ \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2 \right]$$

Final “simple” training objective

DDPM [Ho et al., 2020] provides connection between DPMs and score-matching:

- It resembles denoising score-matching over multiple noise scales

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2\right]$$

- Sampling also resembles Langevin dynamics with $\boldsymbol{\epsilon}_\theta$ (as the learned gradient)

**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

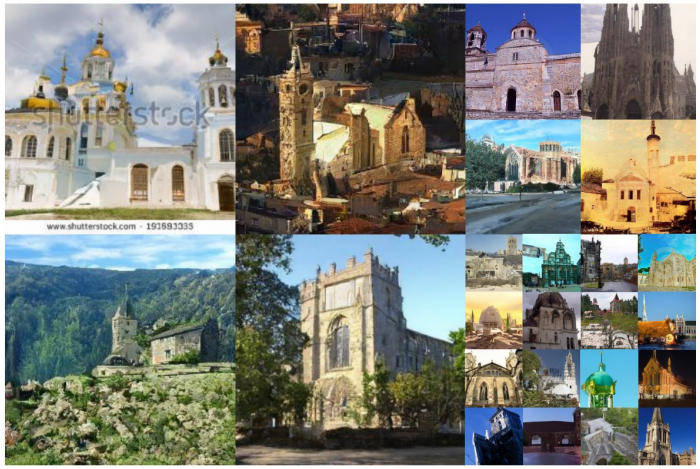
**Algorithm 2** Sampling

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

# Denoising Diffusion Probabilistic Models (DDPM) [Ho et al., 2020]

DDPM achieved the SOTA FID score (3.17) on CIFAR-10 generation

DDPM also generates high-resolution (256x256) images



## Improved Denoising Diffusion Probabilistic Models [Nichol and Dhariwal, 2021]

- This paper improves upon DDPM by introducing additional techniques:

1. Learned variance instead of fixed variance

$$\Sigma_\theta(x_t, t) = \exp(v \log \beta_t + (1 - v) \log \tilde{\beta}_t)$$

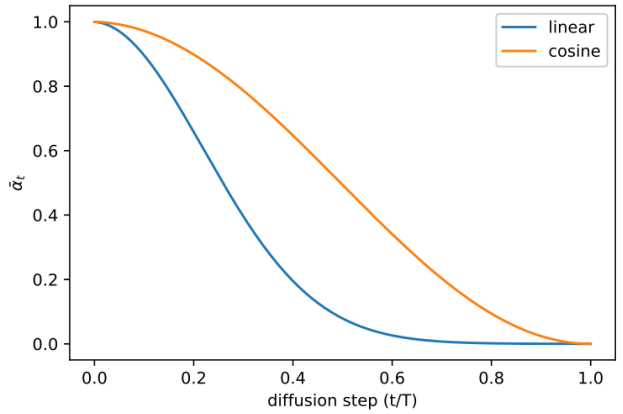
Learnable parameters

2. Hybrid objective of VLB and Simple objectives

$$L_{\text{hybrid}} = L_{\text{simple}} + \lambda L_{\text{vlb}}$$

$\Sigma_\theta$ can be learned through this loss

3. Different diffusion (cosine) schedule
   - Instead of linear schedule in DDPM
   - In particular, used different scheduling of hyperparameters $\bar{\alpha}_t$



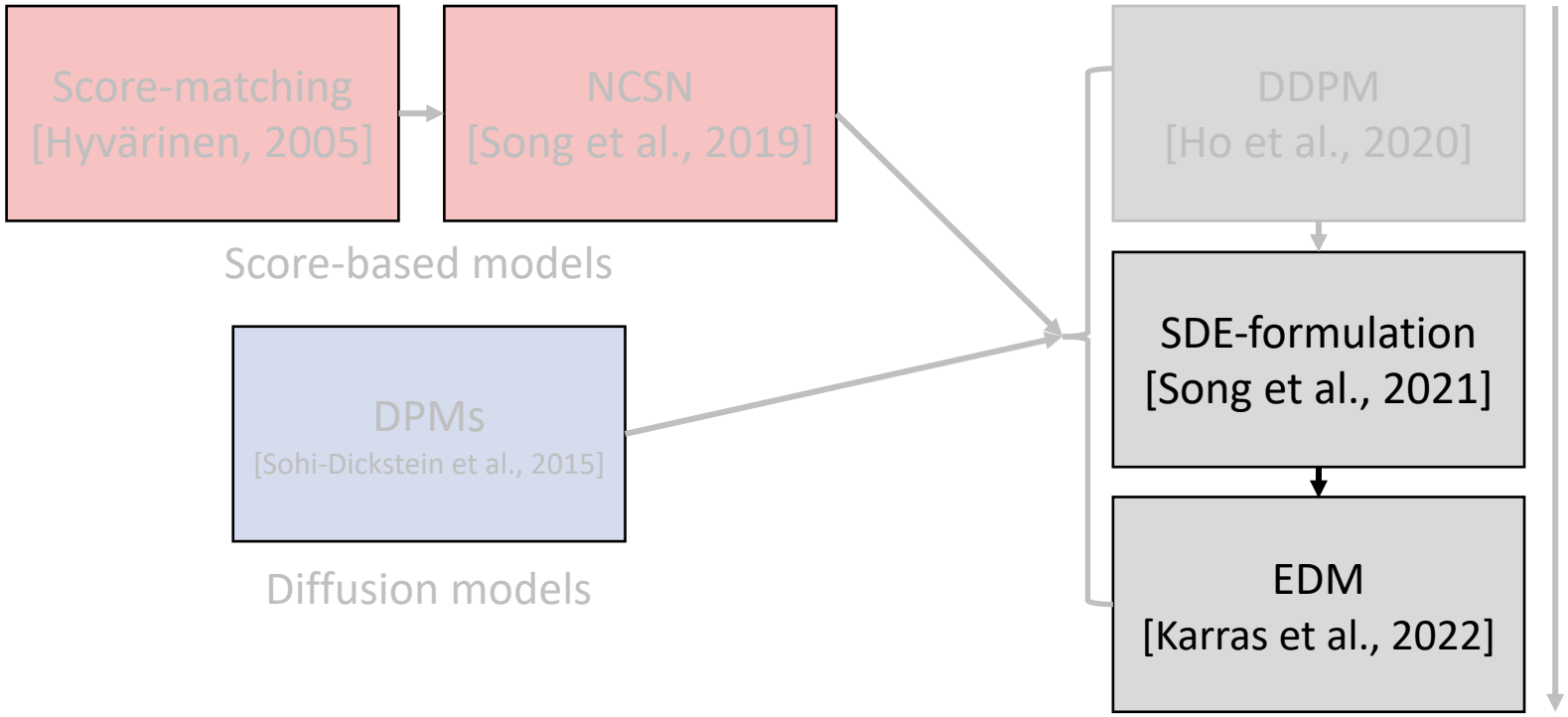
# Improved Denoising Diffusion Probabilistic Models [Nichol and Dhariwal, 2021]

- **Results:** Simple additional techniques in DDPM can improve performance of DDPM

*Table 3.* Comparison of DDPMs to other likelihood-based models on CIFAR-10 and Unconditional ImageNet $64 \times 64$. NLL is reported in bits/dim. On ImageNet $64 \times 64$, our model is competitive with the best convolutional models, but is worse than fully transformer-based architectures.

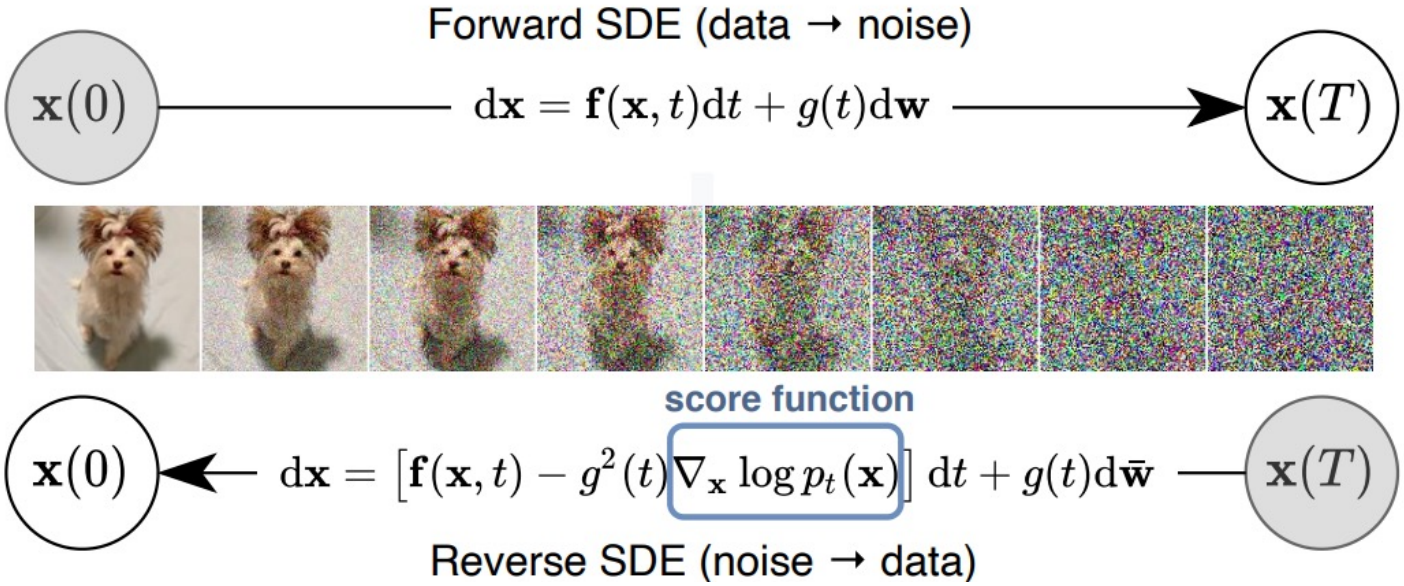
| Model | ImageNet | CIFAR |
|---|---|---|
| Glow (Kingma & Dhariwal, 2018) | 3.81 | 3.35 |
| Flow++ (Ho et al., 2019) | 3.69 | 3.08 |
| PixelCNN (van den Oord et al., 2016c) | 3.57 | 3.14 |
| SPN (Menick & Kalchbrenner, 2018) | 3.52 | - |
| NVAE (Vahdat & Kautz, 2020) | - | 2.91 |
| Very Deep VAE (Child, 2020) | 3.52 | 2.87 |
| PixelSNAIL (Chen et al., 2018) | 3.52 | 2.85 |
| Image Transformer (Parmar et al., 2018) | 3.48 | 2.90 |
| Sparse Transformer (Child et al., 2019) | 3.44 | **2.80** |
| Routing Transformer (Roy et al., 2020) | **3.43** | - |
| DDPM (Ho et al., 2020) | 3.77 | 3.70 |
| DDPM (cont flow) (Song et al., 2020b) | - | 2.99 |
| Improved DDPM (ours) | **3.53** | **2.94** |

# Overview: Score-based models and diffusion models




Unifying views
(Some people call “generative diffusion process”)

Score matching through SDE [Song et al., 2021]




Like DDPM, we consider forward diffusion but now "continuous" version (SDE):

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$$

Then, the reverse diffusion process also follows some SDE:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2\nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

Then, the reverse diffusion process also follows some SDE:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

One can sample the data (with reverse process) by learning the score function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_t \Big\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \big[ \big\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0)) \big\|_2^2 \big] \Big\}.$$

**Note:** under this formalization, NCSN/DDPM are discretization of different SDEs:

- NCSN: $\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}\left[\sigma^2(t)\right]}{\mathrm{d}t}}\mathrm{d}\mathbf{w} \rightarrow \mathbf{x}_i = \mathbf{x}_{i-1} + \sqrt{\sigma_i^2 - \sigma_{i-1}^2}\mathbf{z}_i$
- DDPM: $\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w} \rightarrow \mathbf{x}_i = \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\mathbf{z}_i$

Then, the reverse diffusion process also follows some SDE:

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

One can sample the data (with reverse process) by learning the score function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_t \Big\{ \lambda(t) \mathbb{E}_{\mathbf{x}(0)} \mathbb{E}_{\mathbf{x}(t)|\mathbf{x}(0)} \big[ \big\| \mathbf{s}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) - \nabla_{\mathbf{x}(t)} \log p_{0t}(\mathbf{x}(t) \mid \mathbf{x}(0)) \big\|_2^2 \big] \Big\}.$$

**Note:** under this formalization, NCSN/DDPM are discretization of different SDEs:

**Algorithm 2** PC sampling (VE SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \sigma_{\max}^2 \mathbf{I})$
**for** $i = N-1$ **to** $0$ **do**
  $\mathbf{x}'_i \leftarrow \mathbf{x}_{i+1} + (\sigma_{i+1}^2 - \sigma_i^2)\mathbf{s}_{\boldsymbol{\theta}*}(\mathbf{x}_{i+1}, \sigma_{i+1})$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}'_i + \sqrt{\sigma_{i+1}^2 - \sigma_i^2}\mathbf{z}$
  **for** $j = 1$ **to** $M$ **do**
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}*}(\mathbf{x}_i, \sigma_i) + \sqrt{2\epsilon_i}\mathbf{z}$
**return** $\mathbf{x}_0$

Continuous ver. of NCSN

**Algorithm 3** PC sampling (VP SDE)

$\mathbf{x}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
**for** $i = N-1$ **to** $0$ **do**
  $\mathbf{x}'_i \leftarrow (2 - \sqrt{1-\beta_{i+1}})\mathbf{x}_{i+1} + \beta_{i+1}\mathbf{s}_{\boldsymbol{\theta}*}(\mathbf{x}_{i+1}, i+1)$
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_i \leftarrow \mathbf{x}'_i + \sqrt{\beta_{i+1}}\mathbf{z}$ — Predictor
  **for** $j = 1$ **to** $M$ **do** — Corrector
    $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \epsilon_i \mathbf{s}_{\boldsymbol{\theta}*}(\mathbf{x}_i, i) + \sqrt{2\epsilon_i}\mathbf{z}$
**return** $\mathbf{x}_0$

Continuous ver. of DDPM

On CIFAR-10, the continuous version of DDPM/NCSN is SOTA for both:

- Likelihood estimation
- Sample generation

Table 2: NLLs and FIDs (ODE) on CIFAR-10.

| Model | NLL Test ↓ | FID ↓ |
|---|---|---|
| RealNVP (Dinh et al., 2016) | 3.49 | - |
| iResNet (Behrmann et al., 2019) | 3.45 | - |
| Glow (Kingma & Dhariwal, 2018) | 3.35 | - |
| MintNet (Song et al., 2019b) | 3.32 | - |
| Residual Flow (Chen et al., 2019) | 3.28 | 46.37 |
| FFJORD (Grathwohl et al., 2018) | 3.40 | - |
| Flow++ (Ho et al., 2019) | 3.29 | - |
| DDPM ($L$) (Ho et al., 2020) | $\leqslant 3.70^*$ | 13.51 |
| DDPM ($L_{\text{simple}}$) (Ho et al., 2020) | $\leqslant 3.75^*$ | 3.17 |
| DDPM | 3.28 | 3.37 |
| DDPM cont. (VP) | 3.21 | 3.69 |
| DDPM cont. (sub-VP) | 3.05 | 3.56 |
| DDPM++ cont. (VP) | 3.16 | 3.93 |
| DDPM++ cont. (sub-VP) | 3.02 | 3.16 |
| DDPM++ cont. (deep, VP) | 3.13 | 3.08 |
| DDPM++ cont. (deep, sub-VP) | **2.99** | **2.92** |

Table 3: CIFAR-10 sample quality.

| Model | FID↓ | IS↑ |
|---|---|---|
| **Conditional** | | |
| BigGAN (Brock et al., 2018) | 14.73 | 9.22 |
| StyleGAN2-ADA (Karras et al., 2020a) | **2.42** | **10.14** |
| **Unconditional** | | |
| StyleGAN2-ADA (Karras et al., 2020a) | 2.92 | 9.83 |
| NCSN (Song & Ermon, 2019) | 25.32 | $8.87 \pm .12$ |
| NCSNv2 (Song & Ermon, 2020) | 10.87 | $8.40 \pm .07$ |
| DDPM (Ho et al., 2020) | 3.17 | $9.46 \pm .11$ |
| DDPM++ | 2.78 | 9.64 |
| DDPM++ cont. (VP) | 2.55 | 9.58 |
| DDPM++ cont. (sub-VP) | 2.61 | 9.56 |
| DDPM++ cont. (deep, VP) | 2.41 | 9.68 |
| DDPM++ cont. (deep, sub-VP) | 2.41 | 9.57 |
| NCSN++ | 2.45 | 9.73 |
| NCSN++ cont. (VE) | 2.38 | 9.83 |
| NCSN++ cont. (deep, VE) | **2.20** | **9.89** |

Variants of different SDE solver (ignored in this lecture)

**Recall:** SDE-based formulation of score models

$$\mathrm{d}\mathbf{x} = [\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})]\mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}},$$

**Motivation:** There are "marginally equivalent" ODE (called probability flow ODE)

$$\mathrm{d}\boldsymbol{x} = \left[f(t)\,\boldsymbol{x} - \tfrac{1}{2}\,g(t)^2\,\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})\right]\,\mathrm{d}t$$

**Contribution:** Re-thinking practical design of diffusion models based on this ODE

| | | VP [48] | VE [48] | iDDPM [36] + DDIM [46] | Ours ("EDM") |
|---|---|---|---|---|---|
| **Sampling (Section 3)** | | | | | |
| ODE solver | | Euler | Euler | Euler | 2nd order Heun |
| Time steps | $t_{i<N}$ | $1 + \frac{i}{N-1}(\epsilon_s - 1)$ | $\sigma_{\max}^2 \left(\sigma_{\min}^2/\sigma_{\max}^2\right)^{\frac{i}{N-1}}$ | $u_{\lfloor j_0 + \frac{M-1-j_0}{N-1} i + \frac{1}{2} \rfloor}$, where $u_M = 0$, $u_{j-1} = \sqrt{\frac{u_j^2+1}{\max(\bar{\alpha}_{j-1}/\bar{\alpha}_j, C_1)} - 1}$ | $\left(\sigma_{\max}^{\frac{1}{\rho}} + \frac{i}{N-1}(\sigma_{\min}^{\frac{1}{\rho}} - \sigma_{\max}^{\frac{1}{\rho}})\right)^{\rho}$ |
| Schedule | $\sigma(t)$ | $\sqrt{e^{\frac{1}{2}\beta_d t^2 + \beta_{\min} t} - 1}$ | $\sqrt{t}$ | $t$ | $t$ |
| Scaling | $s(t)$ | $1/\sqrt{e^{\frac{1}{2}\beta_d t^2 + \beta_{\min} t}}$ | 1 | 1 | 1 |
| **Network and preconditioning (Section 5)** | | | | | |
| Architecture of $F_\theta$ | | DDPM++ | NCSN++ | DDPM | (any) |
| Skip scaling | $c_{\text{skip}}(\sigma)$ | 1 | 1 | 1 | $\sigma_{\text{data}}^2/\left(\sigma^2 + \sigma_{\text{data}}^2\right)$ |
| Output scaling | $c_{\text{out}}(\sigma)$ | $-\sigma$ | $\sigma$ | $-\sigma$ | $\sigma \cdot \sigma_{\text{data}}/\sqrt{\sigma_{\text{data}}^2 + \sigma^2}$ |
| Input scaling | $c_{\text{in}}(\sigma)$ | $1/\sqrt{\sigma^2 + 1}$ | 1 | $1/\sqrt{\sigma^2 + 1}$ | $1/\sqrt{\sigma^2 + \sigma_{\text{data}}^2}$ |
| Noise cond. | $c_{\text{noise}}(\sigma)$ | $(M-1)\,\sigma^{-1}(\sigma)$ | $\ln(\frac{1}{2}\sigma)$ | $M - 1 - \arg\min_j \lvert u_j - \sigma \rvert$ | $\frac{1}{4}\ln(\sigma)$ |
| **Training (Section 5)** | | | | | |
| Noise distribution | | $\sigma^{-1}(\sigma) \sim \mathcal{U}(\epsilon_t, 1)$ | $\ln(\sigma) \sim \mathcal{U}(\ln(\sigma_{\min}), \ln(\sigma_{\max}))$ | $\sigma = u_j,\ \ j \sim \mathcal{U}\{0, M-1\}$ | $\ln(\sigma) \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2)$ |
| Loss weighting | $\lambda(\sigma)$ | $1/\sigma^2$ | $1/\sigma^2$ | $1/\sigma^2$ (note: *) | $\left(\sigma^2 + \sigma_{\text{data}}^2\right)/(\sigma \cdot \sigma_{\text{data}})^2$ |

Although there are many new insights, we will cover only a few of them

**Sampling:** Conventional sampling can be viewed as "Euler's method" on PF ODE

- In small NFE regime, the error becomes large (inaccurate solver with large step size)
- **EDM:** Uses "Heun's 2nd order method" for sampling (accurate with smaller NFE)
- Euler ($O(h^2)$) / Heun's ($O(h^3)/2$) (local error / NFE)

**Algorithm 1** Deterministic sampling using Heun's 2nd order method with arbitrary $\sigma(t)$ and $s(t)$.

**procedure** HEUNSAMPLER($D_\theta(\boldsymbol{x};\sigma)$, $\sigma(t)$, $s(t)$, $t_{i\in\{0,\dots,N\}}$)

**sample** $\boldsymbol{x}_0 \sim \mathcal{N}\left(\mathbf{0},\ \sigma^2(t_0)\, s^2(t_0)\, \mathbf{I}\right)$ ▷ Generate initial sample at $t_0$

**for** $i \in \{0,\dots,N-1\}$ **do** ▷ Solve Eq. 4 over $N$ time steps

$$\boldsymbol{d}_i \leftarrow \left(\frac{\dot\sigma(t_i)}{\sigma(t_i)} + \frac{\dot s(t_i)}{s(t_i)}\right)\boldsymbol{x}_i - \frac{\dot\sigma(t_i)s(t_i)}{\sigma(t_i)} D_\theta\left(\frac{\boldsymbol{x}_i}{s(t_i)};\sigma(t_i)\right)$$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $t_i$

$\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + (t_{i+1}-t_i)\boldsymbol{d}_i$ ▷ Take Euler step from $t_i$ to $t_{i+1}$

**if** $\sigma(t_{i+1}) \neq 0$ **then** ▷ Apply 2nd order correction unless $\sigma$ goes to zero

$$\boldsymbol{d}'_i \leftarrow \left(\frac{\dot\sigma(t_{i+1})}{\sigma(t_{i+1})} + \frac{\dot s(t_{i+1})}{s(t_{i+1})}\right)\boldsymbol{x}_{i+1} - \frac{\dot\sigma(t_{i+1})s(t_{i+1})}{\sigma(t_{i+1})} D_\theta\left(\frac{\boldsymbol{x}_{i+1}}{s(t_{i+1})};\sigma(t_{i+1})\right)$$ ▷ Eval. $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $t_{i+1}$

$\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + (t_{i+1}-t_i)\left(\frac{1}{2}\boldsymbol{d}_i + \frac{1}{2}\boldsymbol{d}'_i\right)$ ▷ Explicit trapezoidal rule at $t_{i+1}$

**return** $\boldsymbol{x}_N$ ▷ Return noise-free sample at $t_N$

Although there are many new insights, we will cover only a few of them

**Sampling:** Add "stochasticity" in diffusion sampling benefits the sampling

- With several observations, proposes a "very heuristic" design of the stochastic sampler
- It improves the sampling quality even with smaller NFE regime

**Algorithm 2** Our stochastic sampler with $\sigma(t) = t$ and $s(t) = 1$.

**procedure** StochasticSampler($D_\theta(\boldsymbol{x};\sigma)$, $t_{i\in\{0,\dots,N\}}$, $\gamma_{i\in\{0,\dots,N-1\}}$, $S_{\text{noise}}$)
  **sample** $\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0},\ t_0^2\,\mathbf{I}\big)$
  **for** $i \in \{0,\dots,N-1\}$ **do**  ▷ $\gamma_i = \begin{cases} \min\left(\frac{S_{\text{churn}}}{N}, \sqrt{2}-1\right) & \text{if } t_i \in [S_{\text{tmin}}, S_{\text{tmax}}] \\ 0 & \text{otherwise} \end{cases}$
    **sample** $\boldsymbol{\epsilon}_i \sim \mathcal{N}\big(\mathbf{0},\ S_{\text{noise}}^2\,\mathbf{I}\big)$
    $\hat{t}_i \leftarrow t_i + \gamma_i t_i$  ▷ Select temporarily increased noise level $\hat{t}_i$
    $\hat{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i + \sqrt{\hat{t}_i^2 - t_i^2}\,\boldsymbol{\epsilon}_i$  ▷ Add new noise to move from $t_i$ to $\hat{t}_i$
    $\boldsymbol{d}_i \leftarrow \big(\hat{\boldsymbol{x}}_i - D_\theta(\hat{\boldsymbol{x}}_i;\hat{t}_i)\big)/\hat{t}_i$  ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $\hat{t}_i$
    $\boldsymbol{x}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i + (t_{i+1} - \hat{t}_i)\boldsymbol{d}_i$  ▷ Take Euler step from $\hat{t}_i$ to $t_{i+1}$
    **if** $t_{i+1} \neq 0$ **then**
      $\boldsymbol{d}'_i \leftarrow \big(\boldsymbol{x}_{i+1} - D_\theta(\boldsymbol{x}_{i+1};t_{i+1})\big)/t_{i+1}$  ▷ Apply 2nd order correction
      $\boldsymbol{x}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i + (t_{i+1} - \hat{t}_i)\big(\frac{1}{2}\boldsymbol{d}_i + \frac{1}{2}\boldsymbol{d}'_i\big)$
  **return** $\boldsymbol{x}_N$

Although there are many new insights, we will cover only a few of them

**Training:** Add some of pre-conditioning in the denoising objective

- Rather than a naïve noise-prediction, added some priors in the training objective

$$\mathbb{E}_{\sigma,\boldsymbol{y},\boldsymbol{n}}\Big[\underbrace{\lambda(\sigma)\,c_{\text{out}}(\sigma)^2}_{\text{effective weight}}\Big\|\underbrace{F_\theta\big(c_{\text{in}}(\sigma)\cdot(\boldsymbol{y}+\boldsymbol{n});c_{\text{noise}}(\sigma)\big)}_{\text{network output}}-\underbrace{\frac{1}{c_{\text{out}}(\sigma)}\big(\boldsymbol{y}-c_{\text{skip}}(\sigma)\cdot(\boldsymbol{y}+\boldsymbol{n})\big)}_{\text{effective training target}}\Big\|_2^2\Big]$$

| **Training (Section 5)** | | | | |
|---|---|---|---|---|
| Noise distribution | $\sigma^{-1}(\sigma)\sim\mathcal{U}(\epsilon_t,1)$ | $\ln(\sigma)\sim\mathcal{U}(\ln(\sigma_{\min}),\ln(\sigma_{\max}))$ | $\sigma=u_j,\ j\sim\mathcal{U}\{0,M-1\}$ | $\ln(\sigma)\sim\mathcal{N}(P_{\text{mean}},P_{\text{std}}^2)$ |
| Loss weighting $\lambda(\sigma)$ | $1/\sigma^2$ | $1/\sigma^2$ | $1/\sigma^2$ (note: *) | $(\sigma^2+\sigma_{\text{data}}^2)/(\sigma\cdot\sigma_{\text{data}})^2$ |

**Experiment:** Achieved SOTA FIDs on ImageNet-64 (1.36)

- Moreover, validated each component on various datasets and showed (near-)SOTA results

Table 2: Evaluation of our training improvements. The starting point (config A) is VP & VE using our **deterministic** sampler. At the end (configs E,F), VP & VE only differ in the architecture of $F_\theta$.

| | CIFAR-10 [28] at 32×32 | | | | FFHQ [26] 64×64 | | AFHQv2 [7] 64×64 | |
|---|---|---|---|---|---|---|---|---|
| | Conditional | | Unconditional | | Unconditional | | Unconditional | |
| **Training configuration** | VP | VE | VP | VE | VP | VE | VP | VE |
| A Baseline [48] (*pre-trained) | 2.48 | 3.11 | 3.01* | 3.77* | 3.39 | 25.95 | 2.58 | 18.52 |
| B + Adjust hyperparameters | 2.18 | 2.48 | 2.51 | 2.94 | 3.13 | 22.53 | 2.43 | 23.12 |
| C + Redistribute capacity | 2.08 | 2.52 | 2.31 | 2.83 | 2.78 | 41.62 | 2.54 | 15.04 |
| D + Our preconditioning | 2.09 | 2.64 | 2.29 | 3.10 | 2.94 | 3.39 | 2.79 | 3.81 |
| E + Our loss function | 1.88 | 1.86 | 2.05 | 1.99 | 2.60 | 2.81 | 2.29 | 2.28 |
| F + Non-leaky augmentation | **1.79** | **1.79** | **1.97** | **1.98** | **2.39** | **2.53** | **1.96** | **2.16** |
| NFE | 35 | 35 | 35 | 35 | 79 | 79 | 79 | 79 |

## Table of Contents

Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]

- Generalizes DDPM with much faster sampling process

**Main idea:** Introduce non-Markovian forward process

- Re-formulation of the forward process of diffusion models (DDPM):

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left( \frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{“ predicted } \mathbf{x}_0\text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\mathbf{x}_t)}_{\text{“direction pointing to } \mathbf{x}_t\text{”}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}}$$

- It only depends on the marginal $q(x_t|x_0)$!

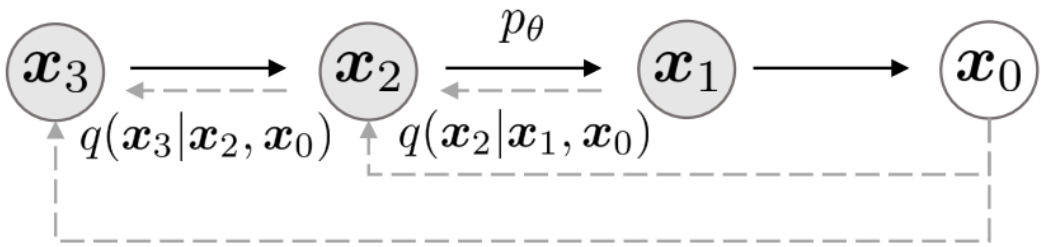
## Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]

- Generalizes DDPM with much faster sampling process

**Main idea:** Introduce non-Markovian forward process

- One may think alternative inference distribution that has the same marginal

$$q_\sigma(\boldsymbol{x}_t|\boldsymbol{x}_{t-1},\boldsymbol{x}_0) = \frac{q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0)q_\sigma(\boldsymbol{x}_t|\boldsymbol{x}_0)}{q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}$$



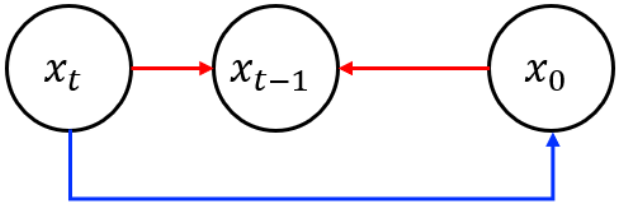

- Where $q_\sigma$ is set to have same marginal as in DDPM (**works with any $\boldsymbol{\sigma_t}$**)

$$q_\sigma(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t,\boldsymbol{x}_0) = \mathcal{N}\left(\sqrt{\alpha_{t-1}}\boldsymbol{x}_0 + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\frac{\boldsymbol{x}_t-\sqrt{\alpha_t}\boldsymbol{x}_0}{\sqrt{1-\alpha_t}}, \sigma_t^2\boldsymbol{I}\right)$$

Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]

Hence, generative process $p_\theta^{(t)}(x_{t-1}|x_t)$ can be defined with $q_\sigma(x_{t-1}|x_t, x_0)$:

1. From $x_t$, predict "denoised" observation $x_0$
2. Obtain sample $x_{t-1}$ from $q_\sigma(x_{t-1}|x_t, x_0)$ using predicted $x_0$ and $x_t$




How to predict $x_0$ from $x_t$?

- **Marginal**: $q(x_t|x_0) \coloneqq \int q(x_{1:t}|x_0)dx_{1:(t-1)} = N(x_t; \sqrt{\alpha_t}x_0, (1-\alpha_t)\boldsymbol{I})$
- From this, we can obtain $x_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon_t$
- By introducing a model $\epsilon_\theta^{(t)}(x_t)$ that predicts $\epsilon_t$, prediction of $x_0$ is given as:

$$f_\theta^{(t)}(x_t) \coloneqq (x_t - \sqrt{1-\alpha_t}\,\epsilon_\theta^{(t)}(x_t))/\sqrt{\alpha_t}$$

Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]

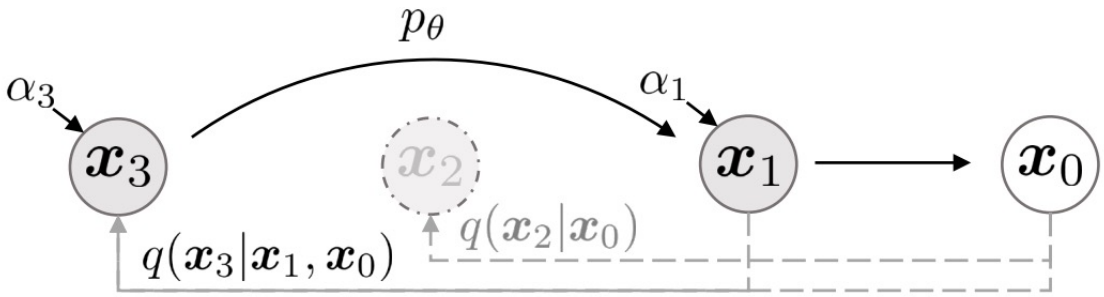
$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left( \frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{“ predicted } \mathbf{x}_0\text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\mathbf{x}_t)}_{\text{“direction pointing to } \mathbf{x}_t\text{”}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}}$$

One may also consider a deterministic process from $x_t$ to $x_0$ with $\sigma_t = 0$

- i.e., Denoising Diffusion Implicit Model (DDIM)

**Accelerated generation process**

- Objective does not depend on the specific forward process if $q_\sigma(x_t|x_0)$ is fixed
- Hence, we can consider “shorter” forward processes
- Consider a forward process over a subset $\{x_{\tau_1}, \dots, x_s\}$



Denoising Diffusion Implicit Models (DDIM) [Song et al., 2021]

**Experiments:** Accelerated sampling of DDIM still enables reasonable quality

- Number of sampling steps **S**
- Degree of stochasticity $\boldsymbol{\sigma}$
- Interpolation between DDPM ($\eta = 1$) and DDIM ($\eta = 0$)

Table 1: CIFAR10 and CelebA image generation measured in FID. $\eta = 1.0$ and $\hat{\sigma}$ are cases of DDPM (although Ho et al. (2020) only considered $T = 1000$ steps, and $S < T$ can be seen as simulating DDPMs trained with $S$ steps), and $\eta = 0.0$ indicates DDIM.

| | | CIFAR10 (32 × 32) | | | | | CelebA (64 × 64) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $S$ | 10 | 20 | 50 | 100 | 1000 | 10 | 20 | 50 | 100 | 1000 |
| $\eta$ | 0.0 | **13.36** | **6.84** | **4.67** | **4.16** | 4.04 | **17.33** | **13.73** | **9.17** | **6.53** | 3.51 |
| | 0.2 | 14.04 | 7.11 | 4.77 | 4.25 | 4.09 | 17.66 | 14.11 | 9.51 | 6.79 | 3.64 |
| | 0.5 | 16.66 | 8.35 | 5.25 | 4.46 | 4.29 | 19.86 | 16.06 | 11.01 | 8.09 | 4.28 |
| | 1.0 | 41.07 | 18.36 | 8.01 | 5.78 | 4.73 | 33.12 | 26.03 | 18.48 | 13.93 | 5.98 |

Deterministic generation process (DDIM) can generate good samples
with 10x~100x smaller sampling steps (=fast)

**Note:** Euler method is "linear" solver of the given ODE

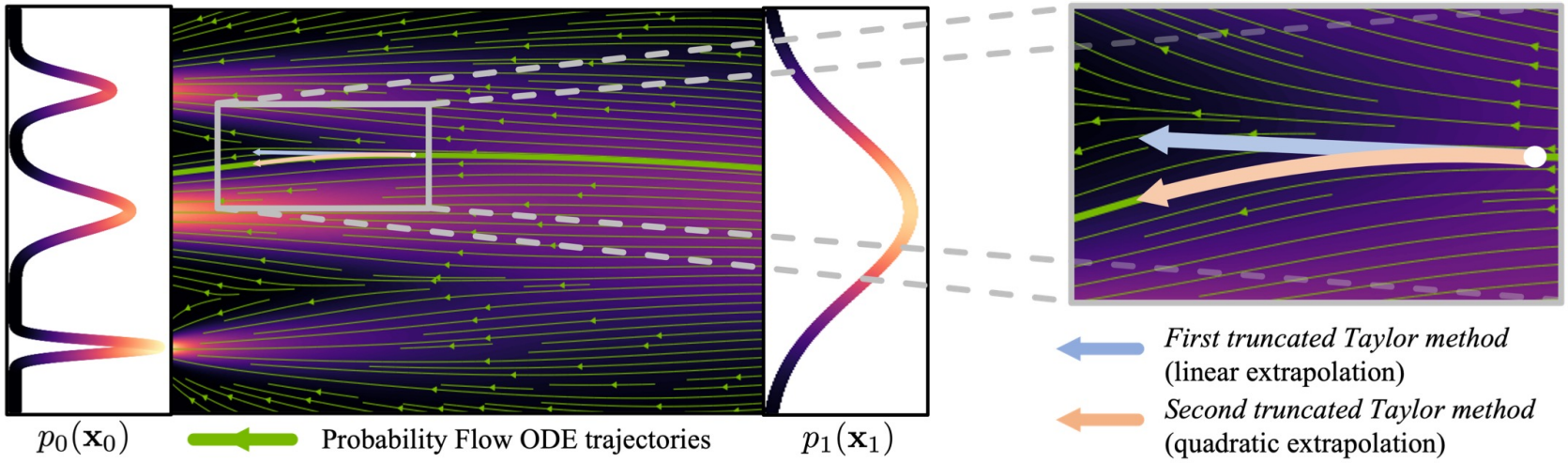
- **Smaller DDIM NFE:** Implies larger step size of the ODE solver (high error)

**GENIE:** Consider 2nd order ODE solver to reduce the error with smaller NFE

$$\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + h_n \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n) + \frac{1}{2} h_n^2 \frac{d\boldsymbol{\epsilon}_{\boldsymbol{\theta}}}{d\gamma_t}\Big|_{(\mathbf{x}_{t_n}, t_n)}$$

Original DDIM Sampler

2nd order (curvature)



**Motivation:** Alternative interpretation of DDIM sampling

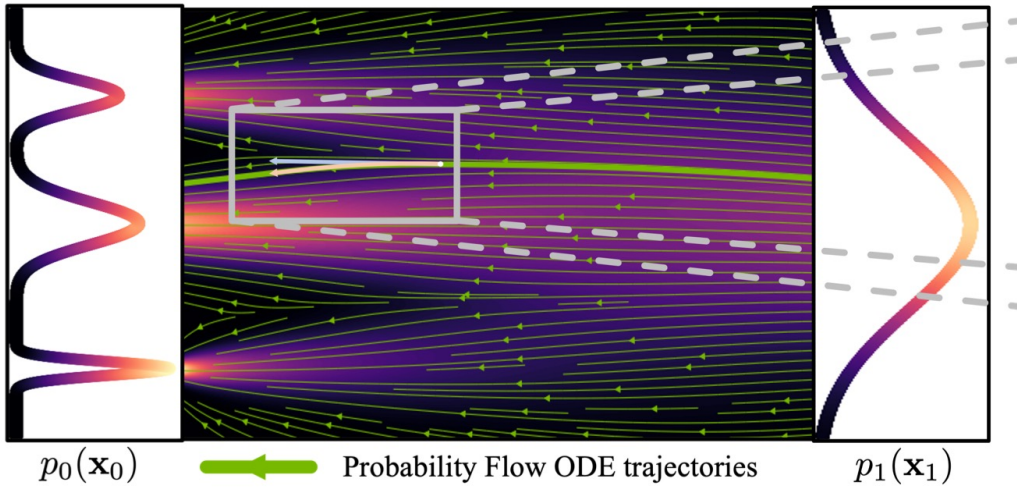
**Recall:** Diffusion model can be interpreted as the marginally equivalent ODE:

$$d\mathbf{x}_t = -\frac{1}{2}\beta_t \left[\mathbf{x}_t + \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] dt$$

- With $\gamma_t = \sqrt{\frac{1-\alpha_t^2}{\alpha_t^2}}$ and $\bar{\mathbf{x}}_t = \mathbf{x}_t\sqrt{1+\gamma_t^2}$ :

$$d\bar{\mathbf{x}}_t = \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\mathbf{x}_t, t\right) d\gamma_t$$

Hence, DDIM = Euler's method(ODE solver) on the following ODE above



**Note:** Euler method is "linear" solver of the given ODE

- **Smaller DDIM NFE:** Implies larger step size of the ODE solver (high error)

**GENIE:** Consider 2nd order ODE solver to reduce the error with smaller NFE

$$\bar{\mathbf{x}}_{t_{n+1}} = \bar{\mathbf{x}}_{t_n} + h_n \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_{t_n}, t_n) + \frac{1}{2} h_n^2 \frac{d\boldsymbol{\epsilon}_{\boldsymbol{\theta}}}{d\gamma_t}\Big|_{(\mathbf{x}_{t_n}, t_n)}$$

Original DDIM Sampler

2nd order (curvature)

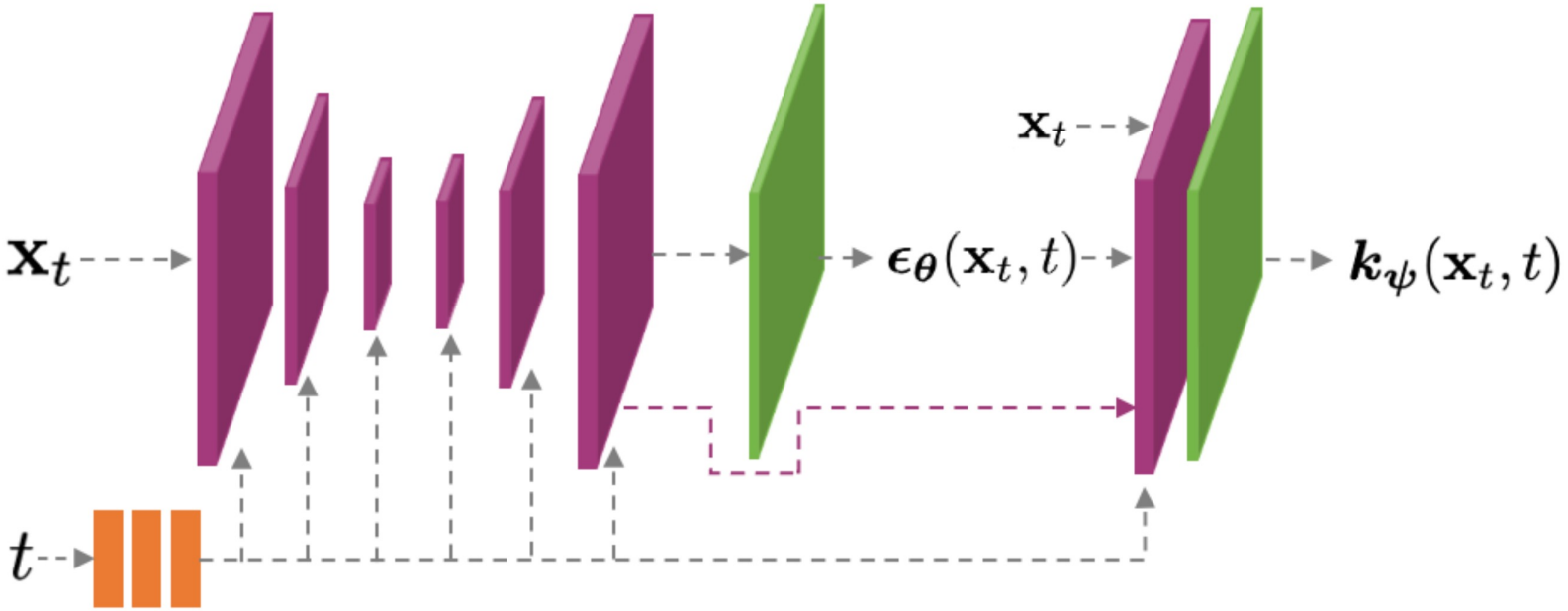
$$d_{\gamma_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \frac{1}{\sqrt{\gamma_t^2+1}} \underbrace{\frac{\partial \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}_{\text{JVP}_1} - \frac{\gamma_t}{1+\gamma_t^2} \underbrace{\frac{\partial \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial \mathbf{x}_t}\mathbf{x}_t}_{\text{JVP}_2} + \frac{\partial \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)}{\partial t}\frac{dt}{d\gamma_t}$$

However, the blue part requires the gradient of the score function

- Which is prohibitive as it leads to >2x computation for each single step

**Solution:** Gradient distillation build upon the pretrained diffusion models

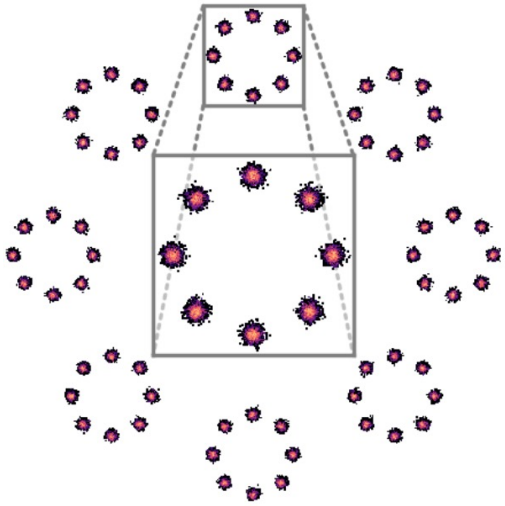
- Namely, add a small prediction head to predict Jacobian vector products (JVPs)
- During sampling, use prediction head (+2% overhead) without computing gradients



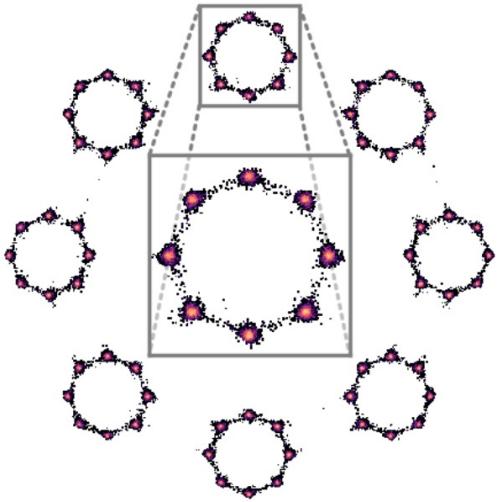

$$\min_{\psi} \mathbb{E}_{t\sim\mathcal{U}[t_{\text{cutoff}},1],\mathbf{x}_0\sim p(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})} \left[g_{\text{d}}(t)\|\boldsymbol{k}_{\psi}(\alpha_t\mathbf{x}_0+\sigma_t\boldsymbol{\epsilon},t)-d_{\gamma_t}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\alpha_t\mathbf{x}_0+\sigma_t\boldsymbol{\epsilon},t)\|_2^2\right]$$

**Experiment:** GENIE accurately samples the toy distribution with multiple nodes

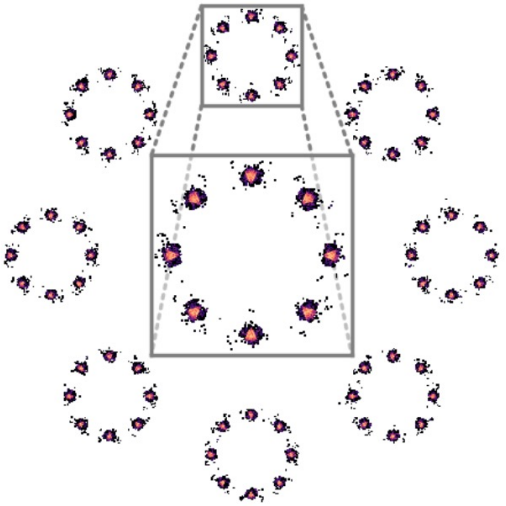
- While DDIM (linear solver) fails under the same NFE



(a) Ground truth



(b) DDIM



(c) GENIE

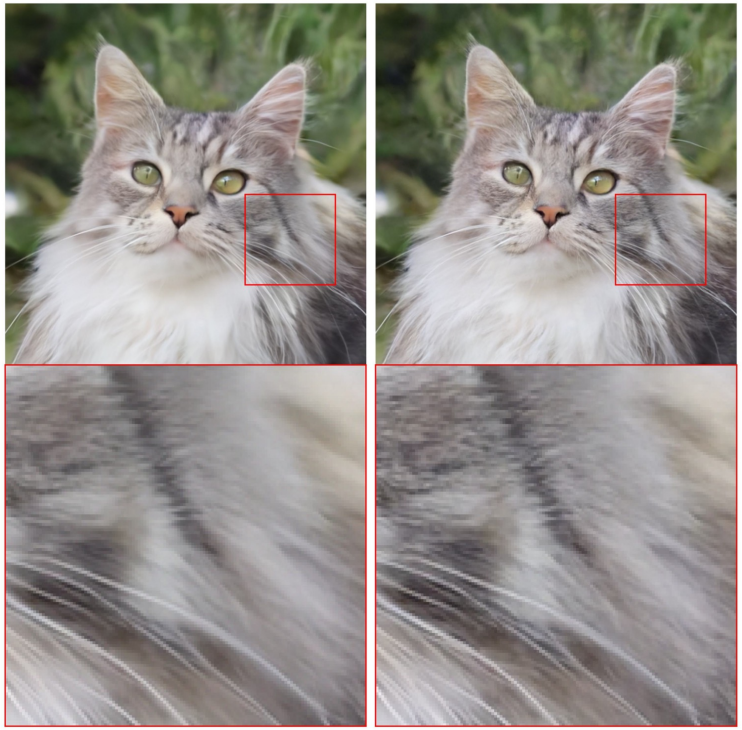
**Experiment:** GENIE shows good sampling (with efficiency) on image generation

| Method | NFEs=5 | NFEs=10 | NFEs=15 | NFEs=20 | NFEs=25 |
|---|---|---|---|---|---|
| GENIE (ours) (∗) | **11.2** | **5.28** | **4.49** | **3.94** | **3.64** |
| GENIE (ours) | 13.9 | 5.97 | **4.49** | **3.94** | 3.67 |
| DDIM [58] (∗) | 27.6 | 11.2 | 7.35 | 5.87 | 5.16 |
| DDIM [58] | 29.7 | 11.2 | 7.35 | 5.87 | 5.16 |
| S-PNDM [63] | 35.9 | 10.3 | 6.61 | 5.20 | 4.51 |
| F-PNDM [63] | N/A | N/A | 10.3 | 5.96 | 4.73 |
| Euler–Maruyama | 325 | 230 | 164 | 112 | 80.3 |
| FastDDIM [64] (†) | - | 9.90 | - | 5.05 | - |
| Learned Sampler [66] (†/ ∗) | 12.4 | 7.86 | 5.90 | 4.72 | 4.25 |
| Learned Sampler [66] (†) | 14.3 | 8.15 | 5.94 | 4.89 | 4.47 |
| Analytic DDIM [65] (†) | - | 14.0 | - | - | 5.71 |
| CLD-SGM [60] | 334 | 306 | 236 | 162 | 106 |
| VESDE-PC [57] | 461 | 461 | 461 | 461 | 462 |

FID on CIFAR-10 (unconditional)

**Experiment:** GENIE shows good sampling (with efficiency) on image generation

| Method | NFEs=5 | NFEs=10 | NFEs=15 |
|---|---|---|---|
| GENIE (ours) | **5.53** | **4.90** | **4.83** |
| DDIM [58] | 9.47 | 6.64 | 5.85 |
| S-PNDM [63] | 14.6 | 11.0 | 8.83 |
| F-PNDM [63] | N/A | N/A | 11.7 |



DDIM GENIE

There are several more concurrent works in this direction:

- [Zhang and Chen, 2022] applies advanced ODE solvers, e.g., Runge-Kutta
- [Lu et al., 2022] introduces a new schedulers for accelerating the sampling
- [Karras et al., 2022] proposes a new design/solver for diffusion model ODEs.

We omit the details in this lecture, but you can take a look if you are interested


Figure 1: Samples by DDIM [19] with 10, 15, 20, 100 number of function evaluations (NFE), and DPM-Solver (ours) with only 10 NFE, using the pre-trained DPMs on ImageNet 256×256 with classifier guidance [4].

[Zhang and Chen., 2022] Fast Sampling of Diffusion Models with Exponential Integrator
[Lu et al., 2022] DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps
[Karras et al., 2022] Elucidating the Design Space of Diffusion-Based Generative Models

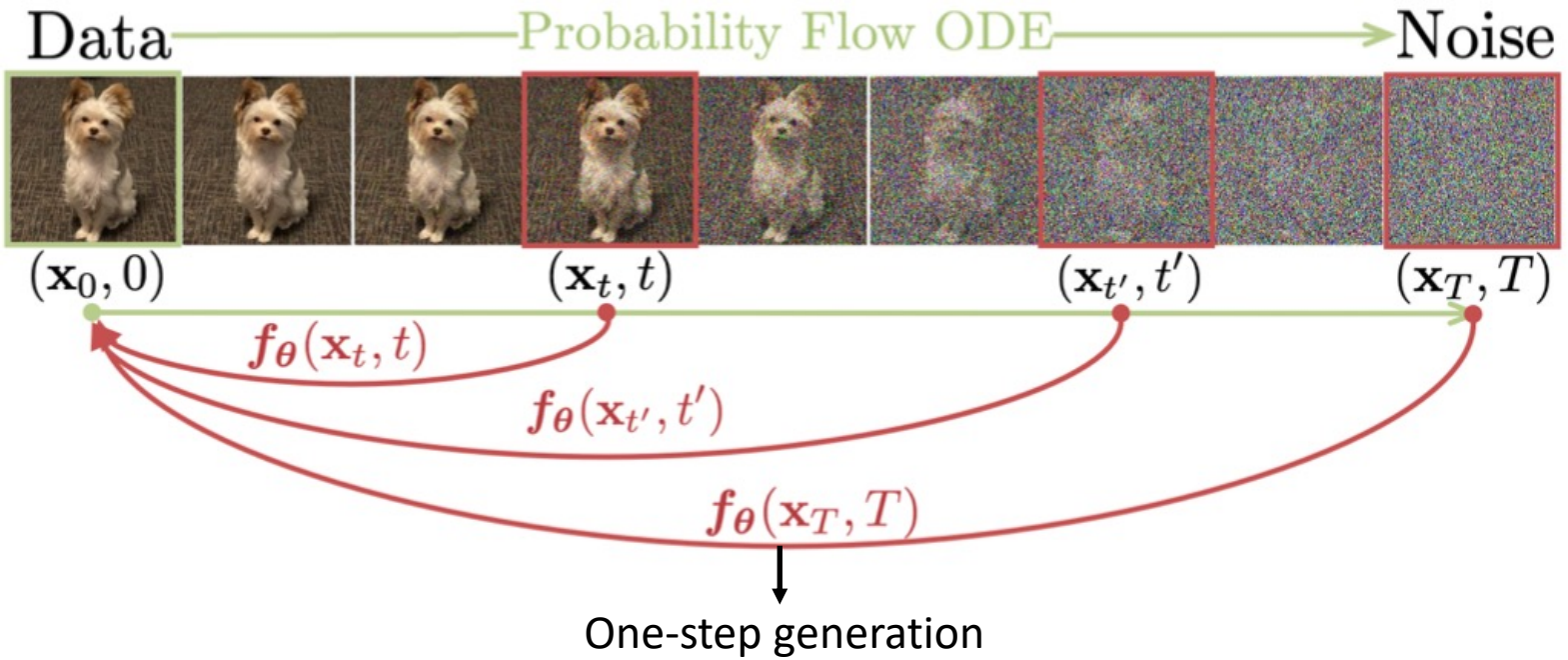
**Consistency models** [Song et al., 2023]

**Motivation:** Score models still require multiple steps;

- Unlike other generative models (e.g., GANs)

**Summary:** Pre-trained score-models can be "distilled" efficiently

- i.e.) They can be turned into single-step (or double-step) generative model



**Recall:** Score-models can be re-interpreted as marginally equivalent ODE:

- i.e.) probability flow (PF) ODE

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = -t\boldsymbol{s}_{\boldsymbol{\phi}}(\mathbf{x}_t, t)$$

**Idea:** Consider a "consistency" function for a given PF ODE trajectory

- We want $\boldsymbol{f} : (\mathbf{x}_t, t) \mapsto \mathbf{x}_\epsilon$ with $\boldsymbol{f}(\mathbf{x}_t, t) = \boldsymbol{f}(\mathbf{x}_{t'}, t')$ if they are in the same ODE trajectory

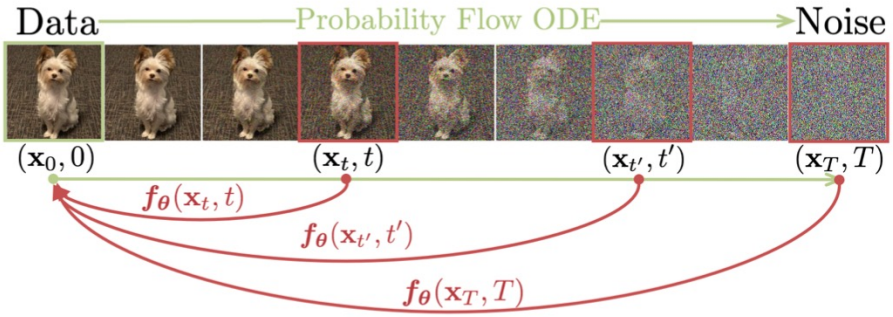



Figure 1: Given a Probability Flow (PF) ODE that smoothly converts data to noise, we learn to map any point (*e.g.*, $\mathbf{x}_t$, $\mathbf{x}_{t'}$, and $\mathbf{x}_T$) on the ODE trajectory to its origin (*e.g.*, $\mathbf{x}_0$) for generative modeling. Models of these mappings are called consistency models, as their outputs are trained to be consistent for points on the same trajectory.

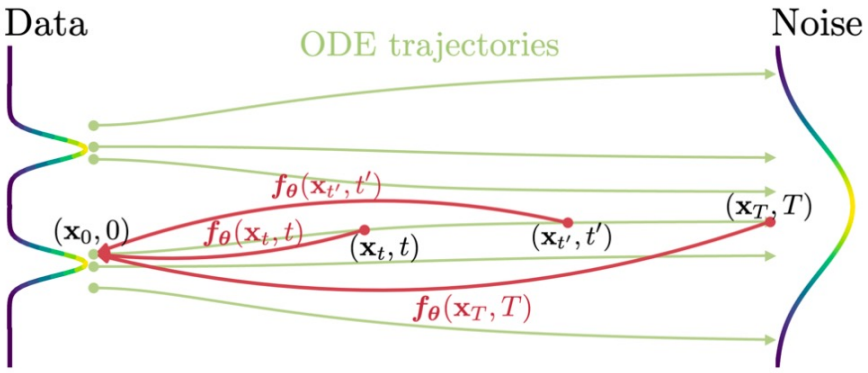



Figure 2: Consistency models are trained to map points on any trajectory of the PF ODE to the trajectory's origin.

**Sampling:** If we have a ideal consistency function, sampling is easy (single-step)

- Optionally, one can consider multi-step sampling to improve the quality further

**Algorithm 1** Multistep Consistency Sampling

**Input:** Consistency model $\boldsymbol{f}_{\boldsymbol{\theta}}(\cdot,\cdot)$, sequence of time points $\tau_1 > \tau_2 > \cdots > \tau_{N-1}$, initial noise $\hat{\mathbf{x}}_T$

$\mathbf{x} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_T, T)$ → **Single-step:** One forward operation from $\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, T^2\boldsymbol{I})$

**for** $n = 1$ **to** $N-1$ **do**
  Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
  $\hat{\mathbf{x}}_{\tau_n} \leftarrow \mathbf{x} + \sqrt{\tau_n^2 - \epsilon^2}\mathbf{z}$
  $\mathbf{x} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_{\tau_n}, \tau_n)$
**end for** → **Multi-step:** Optional to improve the sample quality

**Output:** $\mathbf{x}$

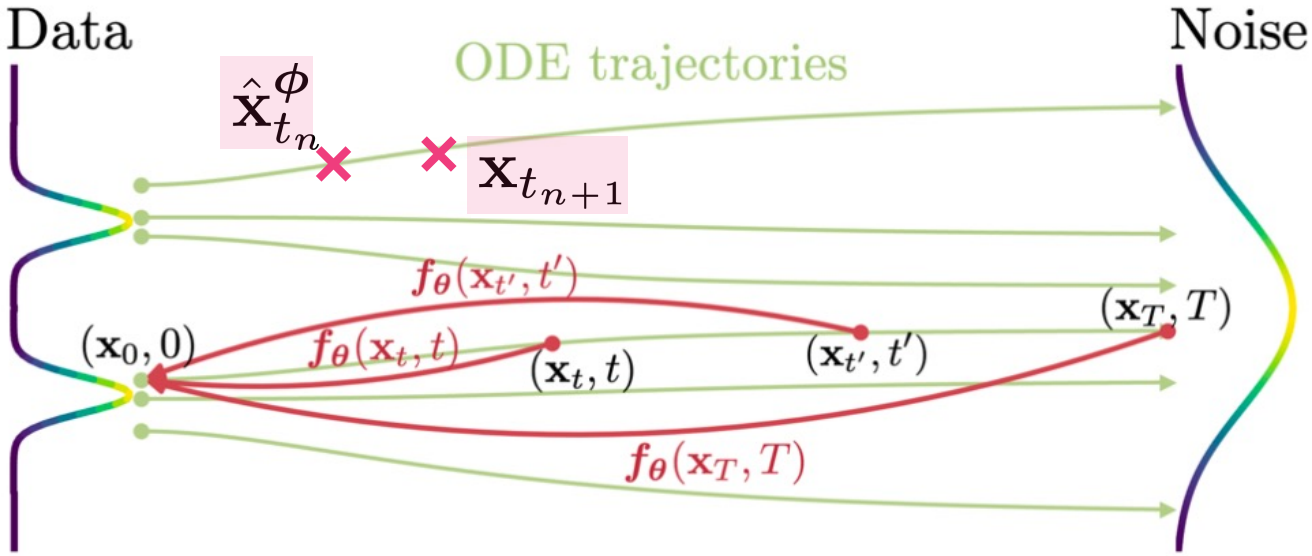
**Training:** Minimize the distance between a adjacency pair on the same trajectory

- Optionally, one can consider multi-step sampling to improve the quality further

$$\mathcal{L}_{CD}^{N}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}) := \mathbb{E}[\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}^{\boldsymbol{\phi}}_{t_n}, t_n))]$$

EMA param.

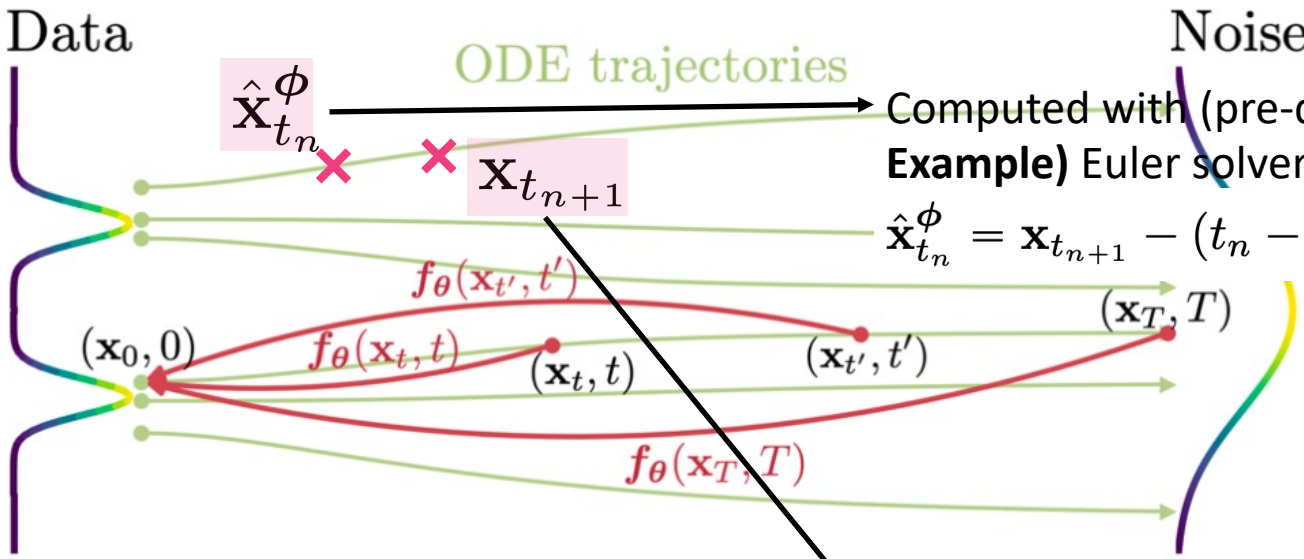
**Pre-defined distance functions:** L2, L1, or LPIPS



**Training:** Minimize the distance between a adjacency pair on the same trajectory

- Optionally, one can consider multi-step sampling to improve the quality further

$$\mathcal{L}_{CD}^{N}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}) := \mathbb{E}[\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}}, t_n))]$$




Computed with (pre-defined) ODE solver $\Phi$

**Example)** Euler solver

$$\hat{\mathbf{x}}_{t_n}^{\boldsymbol{\phi}} = \mathbf{x}_{t_{n+1}} - (t_n - t_{n+1}) t_{n+1} \boldsymbol{s}_{\boldsymbol{\phi}}(\mathbf{x}_{t_{n+1}}, t_{n+1})$$

Sampled from transition density $\mathcal{N}(\mathbf{x}, t_{n+1}^2 \boldsymbol{I})$

**Training:** Minimize the distance between a adjacency pair on the same trajectory

- Optionally, one can consider multi-step sampling to improve the quality further

$$\mathcal{L}_{CD}^{N}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}) := \mathbb{E}[\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^{-}}(\hat{\mathbf{x}}^{\phi}_{t_n}, t_n))]$$

**Algorithm 2** Consistency Distillation (CD)

**Input:** dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate $\eta$, ODE solver $\Phi(\cdot, \cdot; \boldsymbol{\phi})$, $d(\cdot, \cdot)$, $\lambda(\cdot)$, and $\mu$

$\boldsymbol{\theta}^{-} \leftarrow \boldsymbol{\theta}$

**repeat**

Sample $\mathbf{x} \sim \mathcal{D}$ and $n \sim \mathcal{U}[\![1, N-1]\!]$

Sample $\mathbf{x}_{t_{n+1}} \sim \mathcal{N}(\mathbf{x}; t_{n+1}^2 \boldsymbol{I})$

$\hat{\mathbf{x}}^{\phi}_{t_n} \leftarrow \mathbf{x}_{t_{n+1}} + (t_n - t_{n+1}) \Phi(\mathbf{x}_{t_{n+1}}, t_{n+1}; \boldsymbol{\phi})$

1. Sampling "adjacent" pair on the same trajectory

$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi}) \leftarrow$

$\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^{-}}(\hat{\mathbf{x}}^{\phi}_{t_n}, t_n))$

2. Compute the distance

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^{-}; \boldsymbol{\phi})$

$\boldsymbol{\theta}^{-} \leftarrow \text{stopgrad}(\mu \boldsymbol{\theta}^{-} + (1 - \mu) \boldsymbol{\theta})$

3. Optimization

**until** convergence

The paper shows consistency training can be done even without pre-trained model

- They show adjacency pair can be ”approximated” well without having pre-trained models

**Algorithm 2** Consistency Distillation (CD)

**Input:** dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate $\eta$, ODE solver $\Phi(\cdot,\cdot;\boldsymbol{\phi})$, $d(\cdot,\cdot)$, $\lambda(\cdot)$, and $\mu$

$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$

**repeat**

Sample $\mathbf{x} \sim \mathcal{D}$ and $n \sim \mathcal{U}[\![1, N-1]\!]$

Sample $\mathbf{x}_{t_{n+1}} \sim \mathcal{N}(\mathbf{x}; t_{n+1}^2 \boldsymbol{I})$

$\hat{\mathbf{x}}^{\boldsymbol{\phi}}_{t_n} \leftarrow \mathbf{x}_{t_{n+1}} + (t_n - t_{n+1})\Phi(\mathbf{x}_{t_{n+1}}, t_{n+1}; \boldsymbol{\phi})$

$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi}) \leftarrow$

$\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x}_{t_{n+1}}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\hat{\mathbf{x}}^{\boldsymbol{\phi}}_{t_n}, t_n))$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-; \boldsymbol{\phi})$

$\boldsymbol{\theta}^- \leftarrow \mathrm{stopgrad}(\mu \boldsymbol{\theta}^- + (1-\mu)\boldsymbol{\theta})$

**until** convergence

Problematic part;
we don’t have the score function

The paper shows consistency training can be done even without pre-trained model

- They show adjacency pair can be ”approximated” well without having pre-trained models

**Algorithm 3** Consistency Training (CT)

**Input:** dataset $\mathcal{D}$, initial model parameter $\boldsymbol{\theta}$, learning rate $\eta$, step schedule $N(\cdot)$, EMA decay rate schedule $\mu(\cdot)$, $d(\cdot,\cdot)$, and $\lambda(\cdot)$

$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$ and $k \leftarrow 0$

**repeat**

  Sample $\mathbf{x} \sim \mathcal{D}$, and $n \sim \mathcal{U}[\![1, N(k)-1]\!]$

  Sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$

  $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-) \leftarrow$

    $\lambda(t_n) d(\boldsymbol{f}_{\boldsymbol{\theta}}(\mathbf{x} + t_{n+1}\mathbf{z}, t_{n+1}), \boldsymbol{f}_{\boldsymbol{\theta}^-}(\mathbf{x} + t_n\mathbf{z}, t_n))$

  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\theta}^-)$

  $\boldsymbol{\theta}^- \leftarrow \mathrm{stopgrad}(\mu(k)\boldsymbol{\theta}^- + (1-\mu(k))\boldsymbol{\theta})$

  $k \leftarrow k+1$

**until** convergence

Approximation with proofs
(omit the detail in this lecture)

**Experiment:** Achieve reasonable results even with single-step forward operation

- For distillation, it outperforms prior distillation/efficient sampling methods

| | METHOD | NFE (↓) | FID (↓) | IS (↑) |
|---|---|---|---|---|
| | **Diffusion + Samplers** | | | |
| Effi. Sampling via better ODE solver | DDIM (Song et al., 2020) | 50 | 4.67 | |
| | DDIM (Song et al., 2020) | 20 | 6.84 | |
| | DDIM (Song et al., 2020) | 10 | 8.23 | |
| | DPM-solver-2 (Lu et al., 2022) | 12 | 5.28 | |
| | DPM-solver-3 (Lu et al., 2022) | 12 | 6.03 | |
| | 3-DEIS (Zhang & Chen, 2022) | 10 | **4.17** | |
| | **Diffusion + Distillation** | | | |
| Distillation | Knowledge Distillation* (Luhman & Luhman, 2021) | 1 | 9.36 | |
| | DFNO* (Zheng et al., 2022) | 1 | 4.12 | |
| | 1-Rectified Flow (+distill)* (Liu et al., 2022) | 1 | 6.18 | 9.08 |
| | 2-Rectified Flow (+distill)* (Liu et al., 2022) | 1 | 4.85 | 9.01 |
| | 3-Rectified Flow (+distill)* (Liu et al., 2022) | 1 | 5.21 | 8.79 |
| | PD (Salimans & Ho, 2022) | 1 | 8.34 | 8.69 |
| | **CD** | 1 | **3.55** | **9.48** |
| | PD (Salimans & Ho, 2022) | 2 | 5.58 | 9.05 |
| | **CD** | 2 | **2.93** | **9.75** |

**Experiment:** Achieve reasonable results even with single-step forward operation

- For from-scratch training, not state-of-the-arts but comparable results

| METHOD | NFE (↓) | FID (↓) | IS (↑) |
|---|---|---|---|
| **Direct Generation** | | | |
| BigGAN (Brock et al., 2019) | 1 | 14.7 | 9.22 |
| CR-GAN (Zhang et al., 2019) | 1 | 14.6 | 8.40 |
| AutoGAN (Gong et al., 2019) | 1 | 12.4 | 8.55 |
| E2GAN (Tian et al., 2020) | 1 | 11.3 | 8.51 |
| ViTGAN (Lee et al., 2021) | 1 | 6.66 | 9.30 |
| TransGAN (Jiang et al., 2021) | 1 | 9.26 | 9.05 |
| StyleGAN2-ADA (Karras et al., 2020) | 1 | 2.92 | **9.83** |
| StyleGAN-XL (Sauer et al., 2022) | 1 | **1.85** | |
| Score SDE (Song et al., 2021) | 2000 | 2.20 | **9.89** |
| DDPM (Ho et al., 2020) | 1000 | 3.17 | 9.46 |
| LSGM (Vahdat et al., 2021) | 147 | 2.10 | |
| PFGM (Xu et al., 2022) | 110 | 2.35 | 9.68 |
| EDM (Karras et al., 2022) | 36 | **2.04** | 9.84 |
| 1-Rectified Flow (Liu et al., 2022) | 1 | 378 | 1.13 |
| Glow (Kingma & Dhariwal, 2018) | 1 | 48.9 | 3.92 |
| Residual Flow (Chen et al., 2019a) | 1 | 46.4 | |
| GLFlow (Xiao et al., 2019) | 1 | 44.6 | |
| DenseFlow (Grcić et al., 2021) | 1 | 34.9 | |
| DC-VAE (Parmar et al., 2021) | 1 | 17.9 | 8.20 |
| **CT** | 1 | **8.70** | **8.49** |
| **CT** | 2 | **5.83** | **8.85** |

## Table of Contents

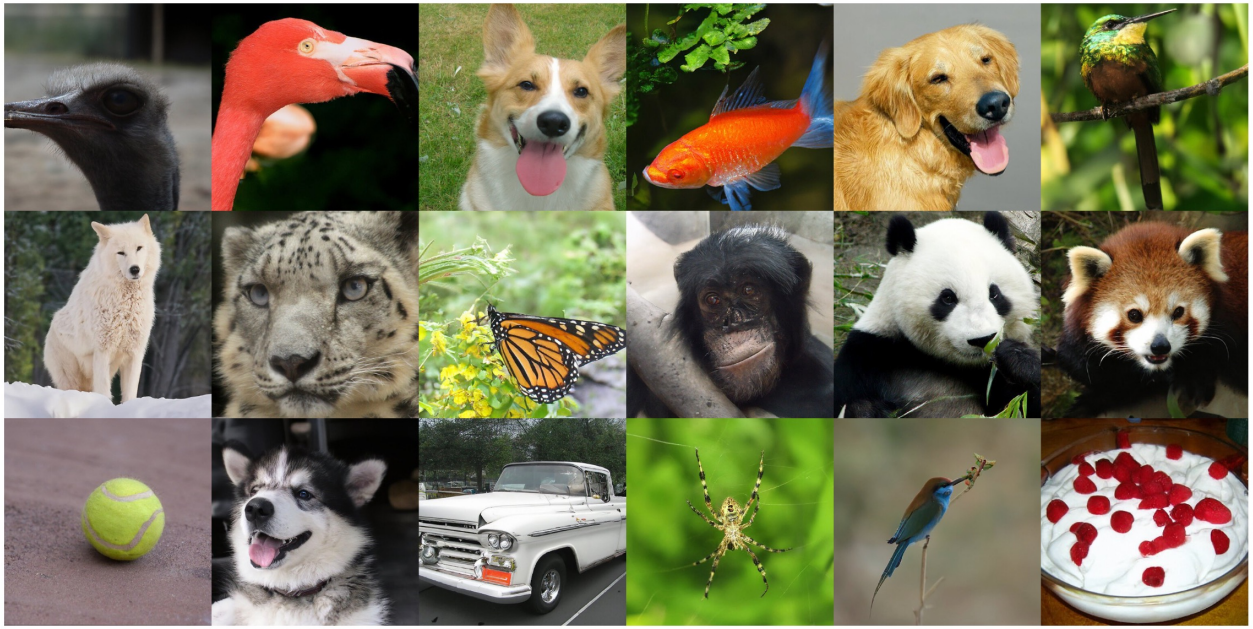
Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]

**Motivation**: Improve the image fidelity of diffusion model using class information

- Class-conditional GANs already used class information heavily [Brock et al., 2019]

**Contribution:** Proposed classifier guidance to use class-information extensively

- Truncation-like effect (in GANs): tradeoff exists between fidelity and diversity
- Strong class guidance - the fidelity of images improves but the diversity decreases.



Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]

**Main idea of classifier guidance:**

- 1. Train a classifier $p_\phi(y|x_t, t)$ on noisy images $x_t$
- 2. Use gradients $\nabla_{x_t} \log p_\phi(y|x_t, t)$ to guide the diffusion sampling process

**Classifier-guidance:**

- **Recall**: Score is derived from the noise prediction model $\epsilon_\theta(x_t)$:

$$\nabla_{x_t} \log p_\theta(x_t) = -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t)$$

- Score for "conditional generation":

$$\begin{aligned} \nabla_{x_t} \log(p_\theta(x_t) p_\phi(y|x_t)) &= \nabla_{x_t} \log p_\theta(x_t) + \nabla_{x_t} \log p_\phi(y|x_t) \\ &= -\frac{1}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t) + \nabla_{x_t} \log p_\phi(y|x_t) \end{aligned}$$

- Hence, classifier-guidance uses the following modified score:

$$\hat{\epsilon}(x_t) := \epsilon_\theta(x_t) - \sqrt{1-\bar{\alpha}_t}\, \nabla_{x_t} \log p_\phi(y|x_t)$$

Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]

**Main idea of classifier guidance:**

- 1. Train a classifier $p_\phi(y|x_t, t)$ on noisy images $x_t$
- 2. Use gradients $\nabla_{x_t} \log p_\phi(y|x_t, t)$ to guide the diffusion sampling process

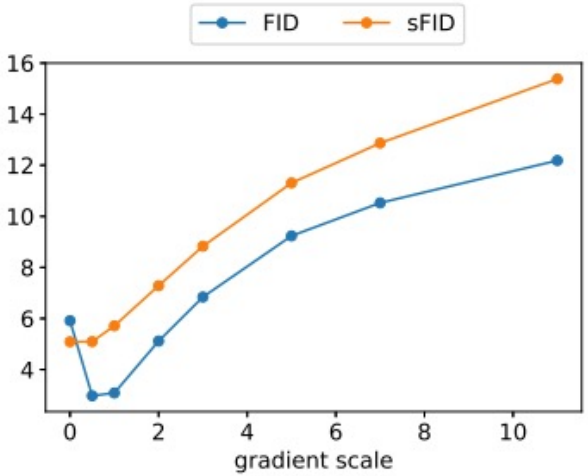
**Classifier-guidance:**

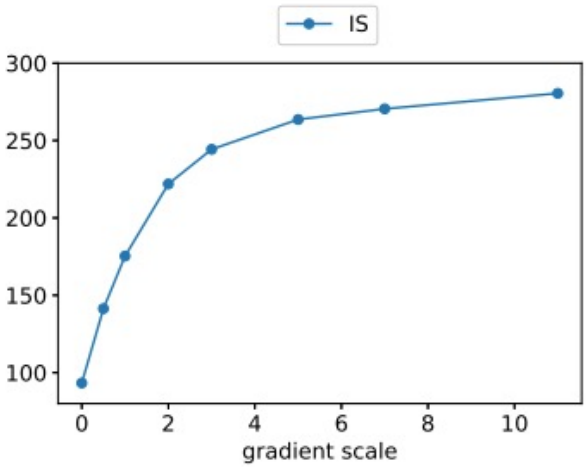
**Algorithm 1** Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale $s$.

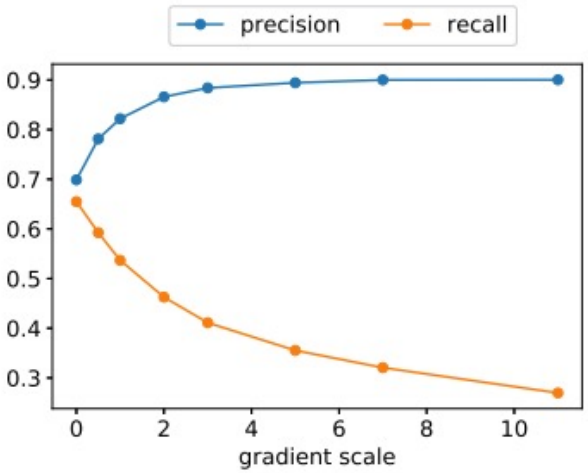
Input: class label $y$, gradient scale $s$
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
**for all** $t$ from $T$ to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
**end for**
**return** $x_0$

Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]

- Classifier guidance indeed improves the image quality (higher IS & precision)
- However, it also shows limited diversity (higer FID & recall)




# Diffusion Models Beat GANs on Image Synthesis [Dhariwal and Nichol, 2021]

- Classifier-guidance enables to outperform state of the art generative models

| Model | FID | sFID | Prec | Rec |
|---|---|---|---|---|
| **LSUN Bedrooms 256×256** | | | | |
| DCTransformer† [42] | 6.40 | 6.66 | 0.44 | **0.56** |
| DDPM [25] | 4.89 | 9.07 | 0.60 | 0.45 |
| IDDPM [43] | 4.24 | 8.21 | 0.62 | 0.46 |
| StyleGAN [27] | 2.35 | 6.62 | 0.59 | 0.48 |
| **ADM (dropout)** | **1.90** | **5.59** | **0.66** | 0.51 |
| **LSUN Horses 256×256** | | | | |
| StyleGAN2 [28] | 3.84 | 6.46 | 0.63 | 0.48 |
| **ADM** | 2.95 | **5.94** | 0.69 | **0.55** |
| **ADM (dropout)** | **2.57** | 6.81 | **0.71** | **0.55** |
| **LSUN Cats 256×256** | | | | |
| DDPM [25] | 17.1 | 12.4 | 0.53 | 0.48 |
| StyleGAN2 [28] | 7.25 | **6.33** | 0.58 | 0.43 |
| **ADM (dropout)** | **5.57** | 6.69 | **0.63** | **0.52** |
| **ImageNet 64×64** | | | | |
| BigGAN-deep* [5] | 4.06 | 3.96 | **0.79** | 0.48 |
| IDDPM [43] | 2.92 | **3.79** | 0.74 | 0.62 |
| **ADM** | 2.61 | **3.77** | 0.73 | 0.63 |
| **ADM (dropout)** | **2.07** | 4.29 | 0.74 | **0.63** |

| Model | FID | sFID | Prec | Rec |
|---|---|---|---|---|
| **ImageNet 128×128** | | | | |
| BigGAN-deep [5] | 6.02 | 7.18 | **0.86** | 0.35 |
| LOGAN† [68] | 3.36 | | | |
| **ADM** | 5.91 | **5.09** | 0.70 | **0.65** |
| **ADM-G (25 steps)** | 5.98 | 7.04 | 0.78 | 0.51 |
| **ADM-G** | **2.97** | **5.09** | 0.78 | 0.59 |
| **ImageNet 256×256** | | | | |
| DCTransformer† [42] | 36.51 | 8.24 | 0.36 | **0.67** |
| VQ-VAE-2†‡ [51] | 31.11 | 17.38 | 0.36 | 0.57 |
| IDDPM‡ [43] | 12.26 | 5.42 | 0.70 | 0.62 |
| SR3†‡ [53] | 11.30 | | | |
| BigGAN-deep [5] | 6.95 | 7.36 | **0.87** | 0.28 |
| **ADM** | 10.94 | 6.02 | 0.69 | 0.63 |
| **ADM-G (25 steps)** | 5.44 | 5.32 | 0.81 | 0.49 |
| **ADM-G** | **4.59** | **5.25** | 0.82 | 0.52 |
| **ImageNet 512×512** | | | | |
| BigGAN-deep [5] | 8.43 | 8.13 | **0.88** | 0.29 |
| **ADM** | 23.24 | 10.19 | 0.73 | **0.60** |
| **ADM-G (25 steps)** | 8.41 | 9.67 | 0.83 | 0.47 |
| **ADM-G** | **7.72** | **6.57** | 0.87 | 0.42 |

**Classifier-Free Diffusion Guidance** [Ho and Salimans, 2021]

**Motivation**: Training classifier on noisy sample might be problematic

- Even more problematic with extreme setup (e.g., training CLIP on noisy samples)

**Contribution**: Introduce class-guidance without training/requiring any classifiers

- Used in most popular foundation diffusion models (e.g., Stable-diffusion)

**Classifier-Free Diffusion Guidance** [Ho and Salimans, 2021]

**Motivation**: Training classifier on noisy sample might be problematic

- Even more problematic with extreme setup (e.g., training CLIP on noisy samples)

**Main idea**: Train **conditional/unconditional distribution** as a single diffusion model

- **Conditional model:** $p_\theta(\boldsymbol{z}|\boldsymbol{c})$ with score $\epsilon_\theta(\boldsymbol{z}_\lambda, \boldsymbol{c})$
- **Unconditional diffusion model:** $p_\theta(\boldsymbol{z})$ with score $\epsilon_\theta(\boldsymbol{z}_\lambda) = \epsilon_\theta(\boldsymbol{z}_\lambda, \boldsymbol{c} = \mathbf{0})$

**Algorithm 1** Joint training a diffusion model with classifier-free guidance

**Require:** $p_{\text{uncond}}$: probability of unconditional training

1: **repeat**
2: $(\mathbf{x}, \mathbf{c}) \sim p(\mathbf{x}, \mathbf{c})$ ▷ Sample data with conditioning from the dataset
3: $\mathbf{c} \leftarrow \varnothing$ with probability $p_{\text{uncond}}$ ▷ Randomly discard conditioning to train unconditionally
4: $\lambda \sim p(\lambda)$ ▷ Sample log SNR value
5: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
6: $\mathbf{z}_\lambda = \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon}$ ▷ Corrupt data to the sampled log SNR value
7: Take gradient step on $\nabla_\theta \left\| \boldsymbol{\epsilon}_\theta(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon} \right\|^2$ ▷ Optimization of denoising model
8: **until** converged

**Classifier-Free Diffusion Guidance** [Ho and Salimans, 2021]

**Motivation**: Training classifier on noisy sample might be problematic

- Even more problematic with extreme setup (e.g., training CLIP on noisy samples)

**Main idea**: Train **conditional/unconditional distribution** as a single diffusion model

- **Sampling:** Use refined score; (linear combination of conditional/unconditional score)

**Algorithm 2** Conditional sampling with classifier-free guidance

**Require:** $w$: guidance strength
**Require:** $\mathbf{c}$: conditioning information for conditional sampling
**Require:** $\lambda_1, \ldots, \lambda_T$: increasing log SNR sequence with $\lambda_1 = \lambda_{\min}, \lambda_T = \lambda_{\max}$

$\mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
**for** $t = 1, \ldots, T$ **do**
  ▷ Form the classifier-free guided score at log SNR $\lambda_t$
  $\tilde{\boldsymbol{\epsilon}}_t = (1 + w)\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\theta(\mathbf{z}_t)$
  ▷ Sampling step (could be replaced by another sampler, e.g. DDIM)
  $\tilde{\mathbf{x}}_t = (\mathbf{z}_t - \sigma_{\lambda_t}\tilde{\boldsymbol{\epsilon}}_t)/\alpha_{\lambda_t}$
  $\mathbf{z}_{t+1} \sim \mathcal{N}(\tilde{\boldsymbol{\mu}}_{\lambda_{t+1}|\lambda_t}(\mathbf{z}_t, \tilde{\mathbf{x}}_t), (\tilde{\sigma}^2_{\lambda_{t+1}|\lambda_t})^{1-v}(\sigma^2_{\lambda_t|\lambda_{t+1}})^v)$ if $t < T$ else $\mathbf{z}_{t+1} = \tilde{\mathbf{x}}_t$
**end for**
**return** $\mathbf{z}_{T+1}$

**Classifier-Free Diffusion Guidance** [Ho and Salimans, 2021]

**Motivation**: Training classifier on noisy sample might be problematic

- Even more problematic with extreme setup (e.g., training CLIP on noisy samples)

**Main idea**: Train **conditional/unconditional distribution** as a single diffusion model

- **Sampling:** Use refined score; (linear combination of conditional/unconditional score)

Why it works? It has a role of an generative classifier $p^i(\boldsymbol{c}|\boldsymbol{z_\lambda}) \propto p(\boldsymbol{z_\lambda}|\boldsymbol{c})/p(\boldsymbol{z_\lambda})$.

- **Note:** gradient of the generative classifier (generative classifier):

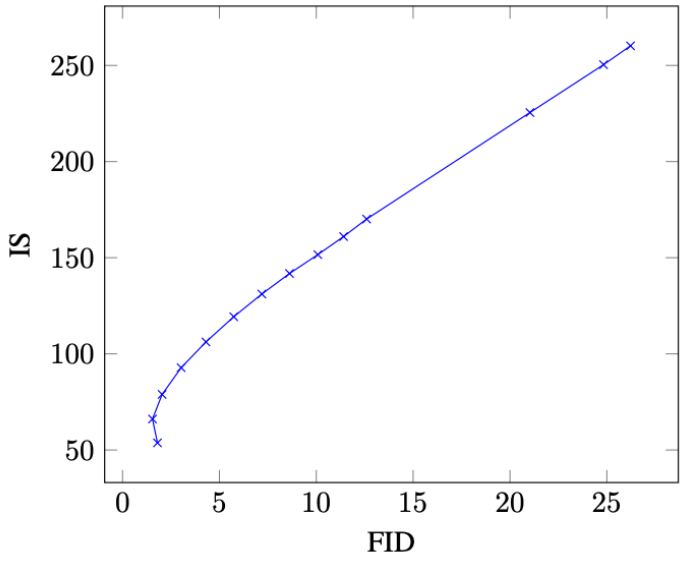
$$\nabla_{\mathbf{z}_\lambda} \log p^i(\mathbf{c}|\mathbf{z}_\lambda) = -\frac{1}{\sigma_\lambda}[\boldsymbol{\epsilon}^*(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon}^*(\mathbf{z}_\lambda)]$$

## **Classifier-Free Diffusion Guidance** [Ho and Salimans, 2021]

- Guidance balances diversity vs. quality of the sample (even without classifiers) well

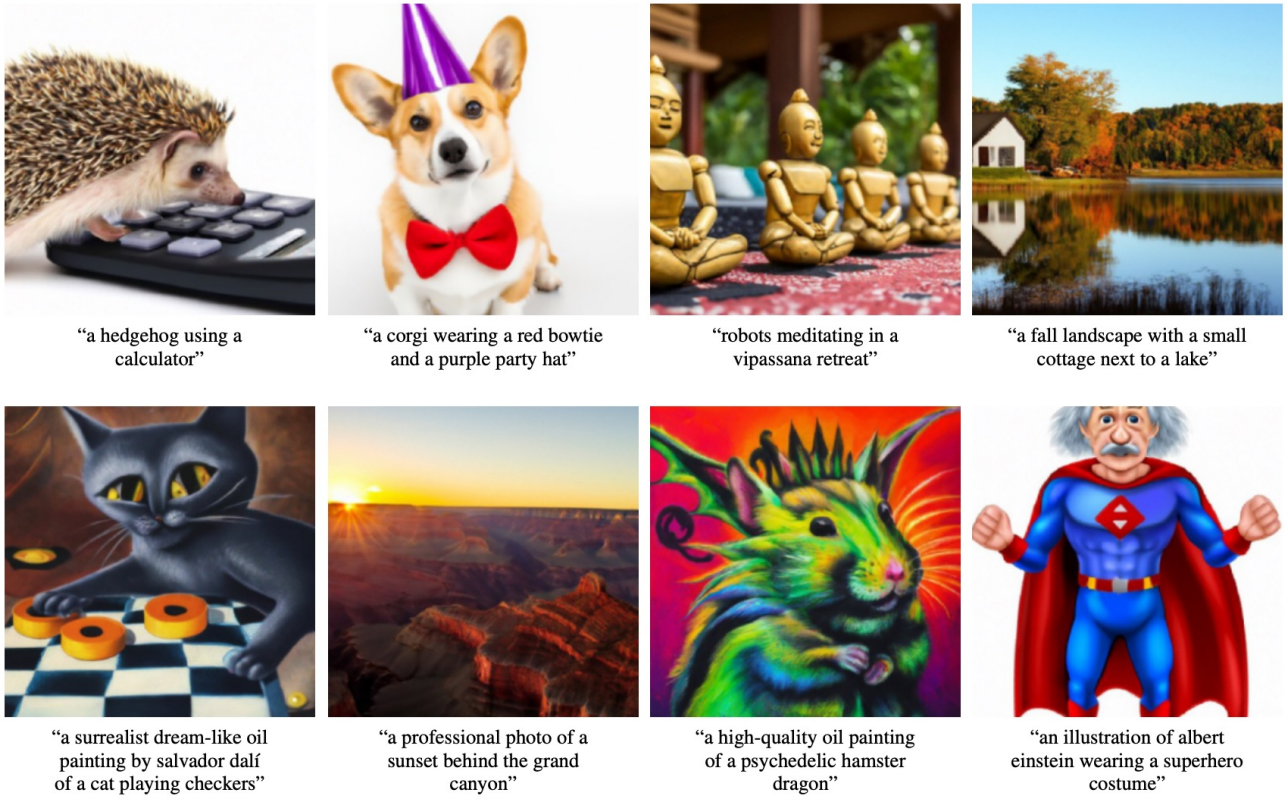
| Method | FID (↓) | IS (↑) |
|---|---|---|
| ADM [3] | 2.07 | - |
| CDM [6] | **1.48** | 67.95 |
| Ours, no guidance | 1.80 | 53.71 |
| Ours, with guidance | | |
| $w = 0.1$ | 1.55 | 66.11 |
| $w = 0.2$ | 2.04 | 78.91 |
| $w = 0.3$ | 3.03 | 92.8 |
| $w = 0.4$ | 4.30 | 106.2 |
| $w = 0.5$ | 5.74 | 119.3 |
| $w = 0.6$ | 7.19 | 131.1 |
| $w = 0.7$ | 8.62 | 141.8 |
| $w = 0.8$ | 10.08 | 151.6 |
| $w = 0.9$ | 11.41 | 161 |
| $w = 1.0$ | 12.6 | 170.1 |
| $w = 2.0$ | 21.03 | 225.5 |
| $w = 3.0$ | 24.83 | 250.4 |
| $w = 4.0$ | 26.22 | **260.2** |

Figure 1: ImageNet 64x64 results



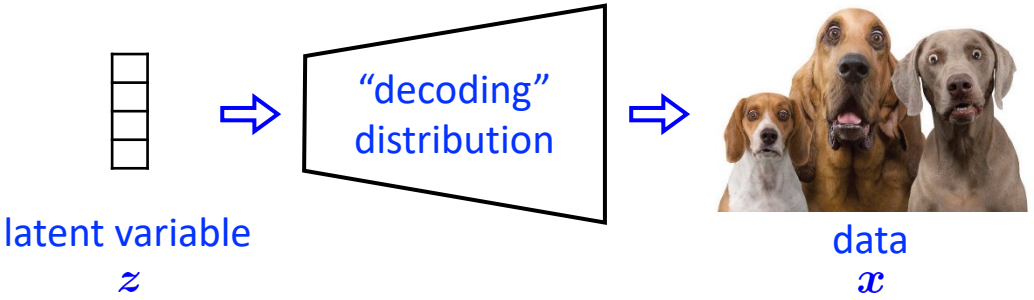
**GLIDE: Zero-shot** Text-to-image diffusion model with classifier-free guidance

- **Some details (for text condition):**
  - Encode text into a sequence of $K$ tokens and feed it into Transformer model
  - Use final token embedding as a class embedding.



## Table of Contents

Consider the following generative model:




- Fixed prior on random latent variable
  - e.g., standard Normal distribution

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbb{I})$$

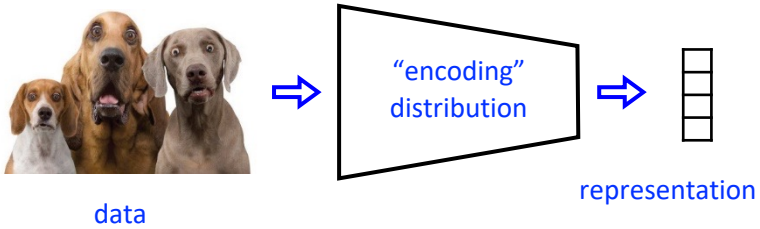
- Parameterized likelihood (decoder) for generation:
  - e.g., Normal distribution parameterized by neural network

$$p_\theta(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}; f_{\text{dec}}(\boldsymbol{z}), \mathbb{I})$$

- Resulting generative distribution (to optimize):

$$\log p_\theta(\boldsymbol{x}) = \log \int_{\boldsymbol{z}} p_\theta(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} = \log \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[p(\boldsymbol{x}|\boldsymbol{z})]$$

Variational autoencoder (VAE) introduce an auxiliary distribution (encoder) [Kingma et al., 2013]




$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}; f_{\text{enc},\mu}(\boldsymbol{x}), f_{\text{enc},\sigma}(\boldsymbol{x}))$$

Each $\log p_\theta(\boldsymbol{x})$ term is replaced by its lower bound:

$$\begin{aligned}
\log p_\theta(\boldsymbol{x}) &\geq \log p_\theta(\boldsymbol{x}) - \min_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) \\
&= \log p_\theta(\boldsymbol{x}) + \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{z}|\boldsymbol{x}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\
&= \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}) + \log p_\theta(\boldsymbol{z}|\boldsymbol{x}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})] \\
&= \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))
\end{aligned}$$

Bound becomes equality when $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$

The training objective becomes:

tractable between two Gaussian distributions

$$\max_\theta \sum_{n=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}) \geq \max_\theta \max_\phi \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$$

$$\approx \max_\theta \max_\phi \sum_{n=1}^{N}\sum_{k=1}^{N} \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))$$

where latent variables are sampled by $\boldsymbol{z}^{(n,k)} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})$

However, non-trivial to train with back propagation due to sampling procedure:

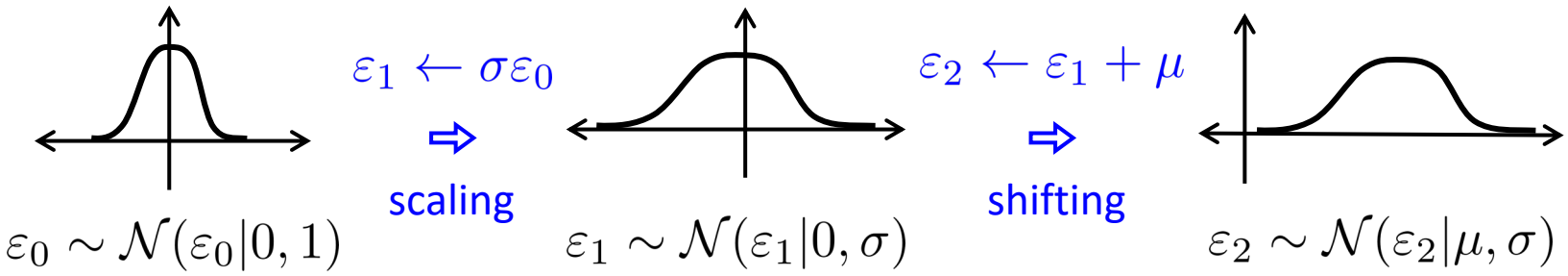
$$\nabla_\phi \mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{N} - \nabla_\phi \log p_\theta(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) + \nabla_\phi \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(n)})||p(\boldsymbol{z}))$$

⇩

Since $\boldsymbol{z}^{(n,k)}$ is fixed after being sampled, $\nabla_\phi \log p(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n,k)}) = 0$ ?

Reparameterization trick is based on the change-of-variables formula:

$$\varepsilon_2 \sim \mathcal{N}(\varepsilon_2|\mu,\sigma) \;\Leftrightarrow\; \varepsilon_2 = \mu + \sigma\varepsilon_0, \qquad \varepsilon_0 \sim \mathcal{N}(\varepsilon_0|0,1)$$




Latent variable $\boldsymbol{z}^{(n,k)}$ can be similarly parameterized by encoder network:

$$\boldsymbol{z}^{(n,k)} \sim \mathcal{N}(\boldsymbol{z}; f_{\text{enc},\mu}(\boldsymbol{x}^{(n)}), f_{\text{enc},\sigma}(\boldsymbol{x}^{(n)}))$$

$$\Updownarrow$$

$$\boldsymbol{z}^{(n,k)} = f_{\text{enc},\mu}(\boldsymbol{x}^{(n)}) + f_{\text{enc},\sigma}(\boldsymbol{x}^{(n)}) \odot \boldsymbol{\varepsilon}^{(n,k)}, \qquad \boldsymbol{\varepsilon}^{(n,k)} \sim \mathcal{N}(\boldsymbol{\varepsilon}|\mathbf{0},\mathbf{1})$$

## Improving VAEs

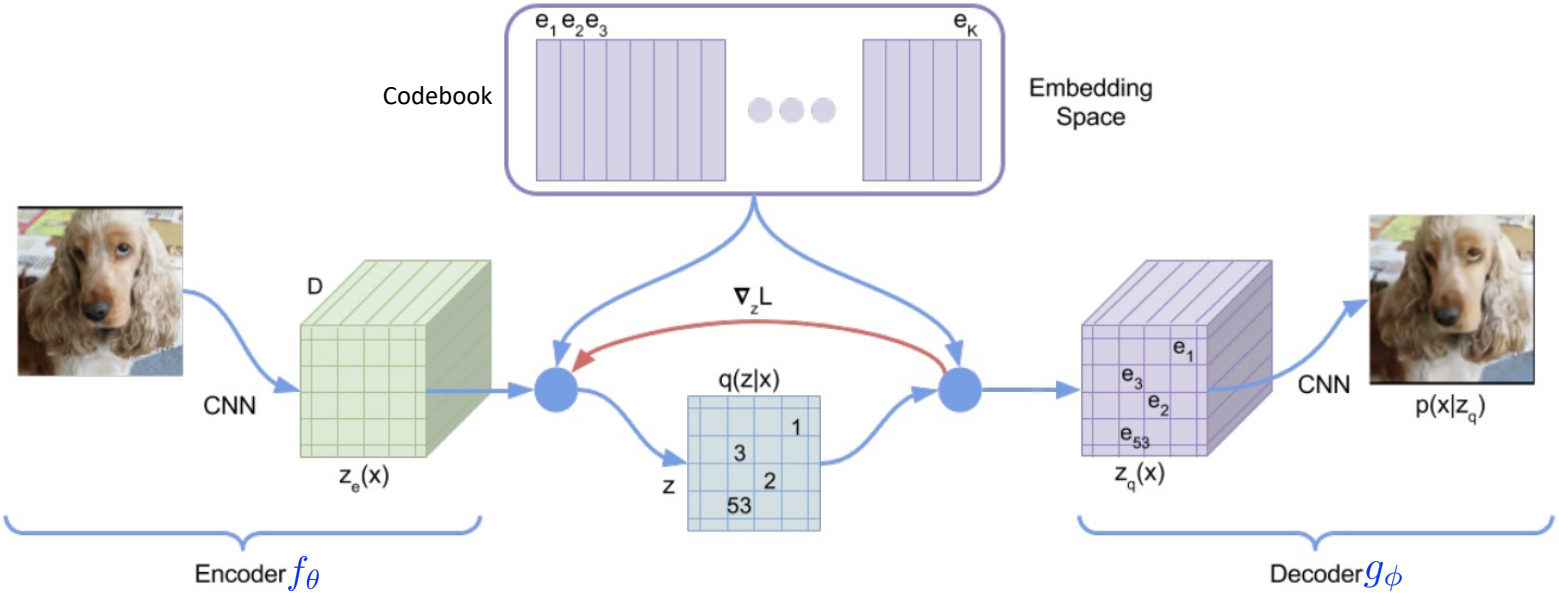
Although VAE has many advantages (e.g., fast sampling, full mode covering, latent embedding), there are issues that lead to **poor generation quality**

- Posterior collapse (latents are ignored when paired with powerful decoder)
  - **Careful optimization:** various techniques for continuous latent-space VAEs
  - **Use discrete latent space:** Vector-quantized VAE (VQ-VAE, VQ-GAN)

- Improve model expressivity
  - **Use expressive prior distribution:** Gaussian mixtures, normalizing flow
  - **Use hierarchical architectures:** Hierarchical VAE, Diffusion Models

VQ-VAE [Oord et al., 2017]

- Each data is embedded into combination of 'discrete' latent vectors: $\{e_1, \cdots, e_K\}$
- **i.e.)** each encoder output is quantized to the nearest vector among $K$ codebook vectors




- Restriction of latent space achieves high generation quality including:
  - Images, videos, audios, etc.

VQ-VAE [Oord et al., 2017]

- The objective of VQ-VAE composed of three terms:
  - Reconstruction loss (1)
  - VQ loss (2):
    - Optimization of codebook vectors
  - Commitment loss (3):
    - Regularization to get encoder outputs and codebook close

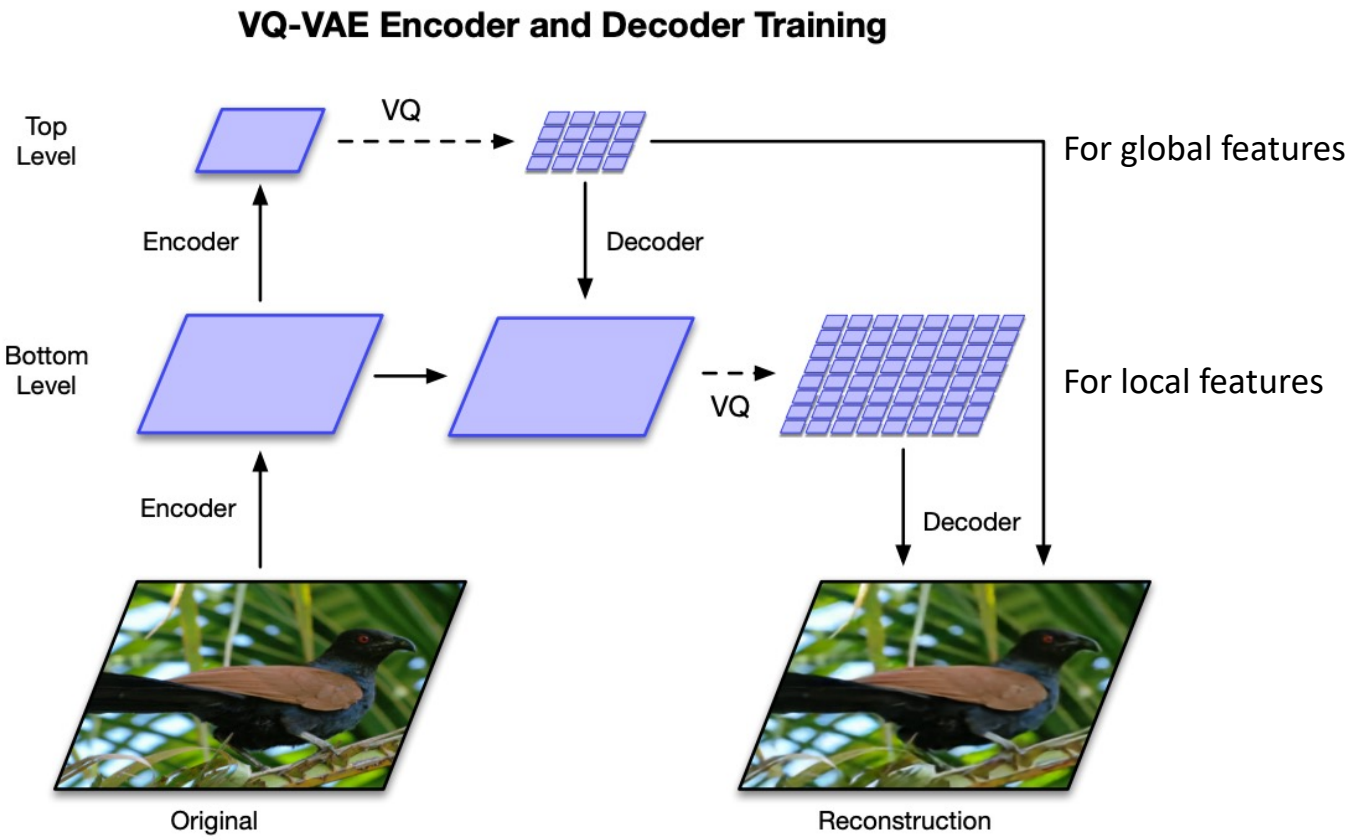
$$\mathcal{L} = \underbrace{||g_\phi(e) - x||_2^2}_{(1)} + \underbrace{||\mathtt{sg}(f_\theta(x)) - e||_2^2}_{(2)} + \underbrace{\beta||f_\theta(x) - \mathtt{sg}(e)||_2^2}_{(3)}$$

VQ-VAE like methods (i.e. discrete prior) recently shows remarkable success on:

- DALL-E (text-image generative model) – image is encoded via VQ-VAE
- Many audio self-supervised learning method

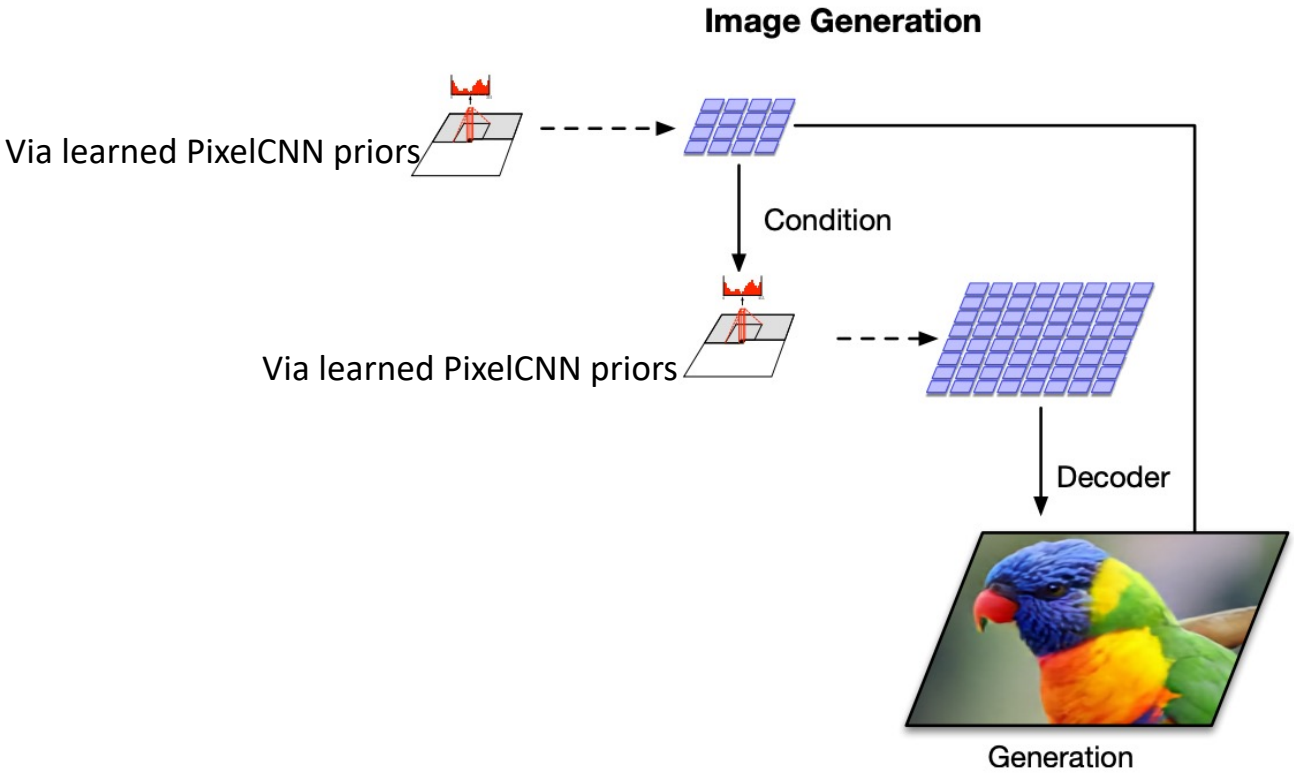
VQ-VAE-2 [Razavi et al., 2019b]

- Different from VQ-VAE, vector quantization occurs twice (top, bottom level)
- For both consideration of local/global features for high-fidelity image



VQ-VAE-2 [Razavi et al., 2019b]

- After VQ-VAE-2 training, train two pixelCNN priors for new image generation
- They autoregressively fill out each quantized latent vector space




- Generated images are comparable to state-of-the-art GAN model (e.g. BigGAN)

## Improving VAEs

Although VAE has many advantages (e.g., fast sampling, full mode covering, latent embedding), there are issues that lead to **poor generation quality**

- Posterior collapse (latents are ignored when paired with powerful decoder)
  - **Careful optimization:** various techniques for continuous latent-space VAEs
  - **Use discrete latent space:** Vector-quantized VAE (VQ-VAE, VQ-GAN)

- Improve model expressivity
  - **Use expressive prior distribution:** Gaussian mixtures, normalizing flow
  - **Use hierarchical architectures:** Hierarchical VAE, Diffusion Models

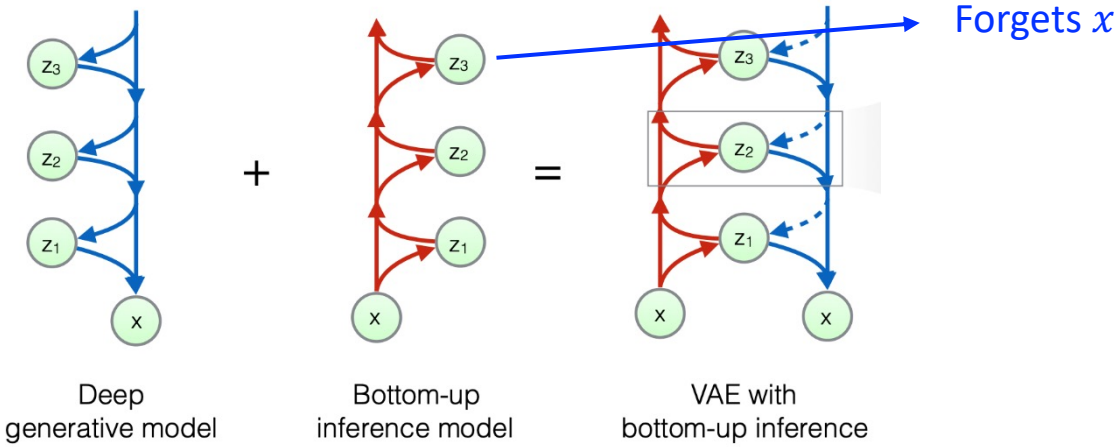
NVAE [Vahdat et al., 2020]

- Hierarchical VAEs use the factorized latent space $p_\theta(z) = \prod_l p_\theta(z_l|z_{<l})$
- Here, the ELBO objective is given by

$$\mathcal{L}_{\text{VAE}}(\boldsymbol{x}) := \mathbb{E}_{q(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x}|\boldsymbol{z})\right] - \mathrm{KL}(q(\boldsymbol{z}_1|\boldsymbol{x})||p(\boldsymbol{z}_1)) - \sum_{l=2}^{L} \mathbb{E}_{q(\boldsymbol{z}_{<l}|\boldsymbol{x})}\left[\mathrm{KL}(q(\boldsymbol{z}_l|\boldsymbol{x}, \boldsymbol{z}_{<l})||p(\boldsymbol{z}_l|\boldsymbol{z}_{<l}))\right],$$

However, **prior attempts** on hierarchical VAE were not so successful due to:

1. Long-range correlation: upper latents often **forget** the data information

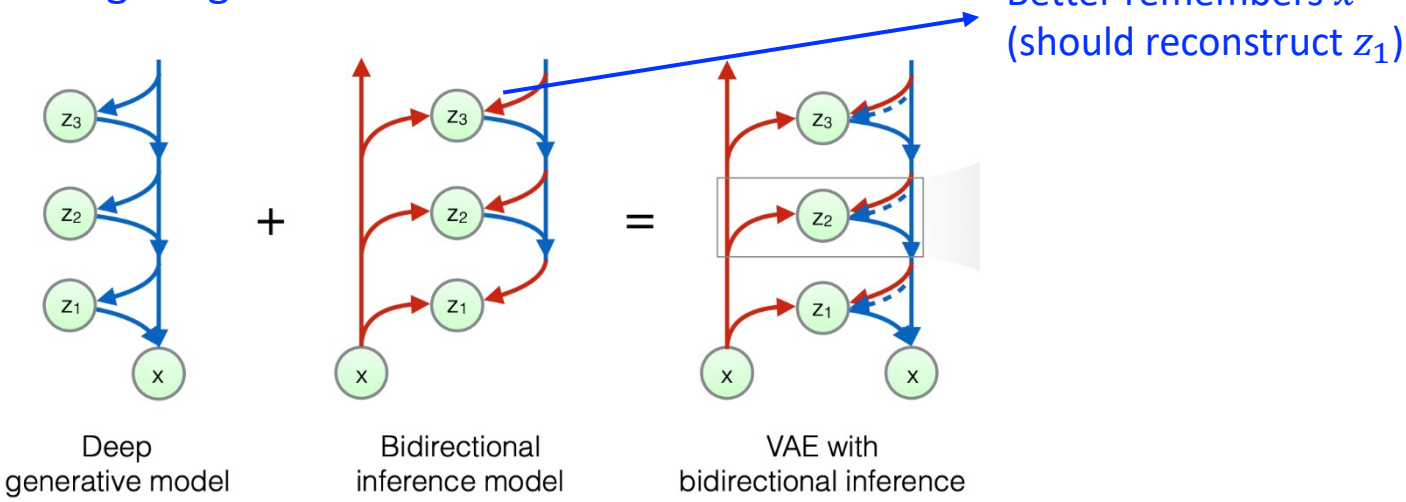



2. Unstable (unbounded) KL term: even more severe for hierarchical VAEs since they **jointly learn** the prior distribution $p_\theta(z)$

NVAE [Vahdat et al., 2020]

**Idea 1.** Bidirectional encoder (originally from [Kingma et al., 2016])

- Enforce upper latents (e.g., $z_3$) to predict the lower latents (e.g., $z_1$)
  → Improve the long-range correlation issue




- **Training:** **posterior** $q_\phi(z|x)$ is inferred by both encoder and decoder (aggregate them) and **prior** $p_\theta(z)$ is jointly inferred by decoder
  - Recall that the KL term is a function of $q_\phi(z|x)$ and $p_\theta(z)$

- **Inference:** Sample **prior** $p_\theta(z)$ from decoder and generate sample $x$

NVAE [Vahdat et al., 2020]

**Idea 2.** Taming the unstable KL term

1. Residual normal distribution
   - For each factorized **prior** distribution

$$p(z_l^i|\boldsymbol{z}_{<l}) := \mathcal{N}(\mu_i(\boldsymbol{z}_{<l}), \sigma_i(\boldsymbol{z}_{<l})),$$

     define **approximate posterior** as (instead of directly predict $\mu_i$, $\sigma_i$)

$$q(z_l^i|\boldsymbol{z}_{<l}, \boldsymbol{x}) := \mathcal{N}(\mu_i(\boldsymbol{z}_{<l}) + \Delta\mu_i(\boldsymbol{z}_{<l}, \boldsymbol{x}), \sigma_i(\boldsymbol{z}_{<l}) \cdot \Delta\sigma_i(\boldsymbol{z}_{<l}, \boldsymbol{x})),$$

   - Then, the **KL term** of ELBO is given by

$$\mathrm{KL}(q(z^i|\boldsymbol{x})||p(z^i)) = \frac{1}{2}\left(\frac{\Delta\mu_i^2}{\sigma_i^2} + \Delta\sigma_i^2 - \log\Delta\sigma_i^2 - 1\right)$$

2. Spectral regularization
   - Enforce *Lipschitz smoothness* of encoder to bound KL divergence
   - Regularize the *largest singular value* of convolutional layers (estimated by power iteratio n [Yoshida & Miyato, 2017])

NVAE [Vahdat et al., 2020]

**Experiments:**

- Generate high-resolution (256x256) images

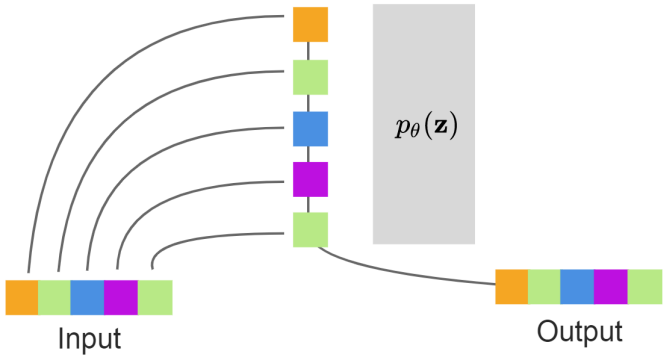
- SOTA test negative log-likelihood (NLL) on non-autoregressive models

| Method | MNIST 28×28 | CIFAR-10 32×32 | ImageNet 32×32 | CelebA 64×64 | CelebA HQ 256×256 | FFHQ 256×256 |
|---|---|---|---|---|---|---|
| NVAE w/o flow | **78.01** | 2.93 | - | 2.04 | - | 0.71 |
| NVAE w/ flow | 78.19 | **2.91** | 3.92 | **2.03** | **0.70** | **0.69** |
| **VAE Models with an Unconditional Decoder** | | | | | | |
| BIVA [36] | 78.41 | 3.08 | 3.96 | 2.48 | - | - |
| IAF-VAE [4] | 79.10 | 3.11 | - | - | - | - |
| DVAE++ [20] | 78.49 | 3.38 | - | - | - | - |
| Conv Draw [42] | - | 3.58 | 4.40 | - | - | - |
| **Flow Models without any Autoregressive Components in the Generative Model** | | | | | | |
| VFlow [59] | - | 2.98 | - | - | - | - |
| ANF [60] | - | 3.05 | 3.92 | - | 0.72 | - |
| Flow++ [61] | - | 3.08 | **3.86** | - | - | - |
| Residual flow [50] | - | 3.28 | 4.01 | - | 0.99 | - |
| GLOW [62] | - | 3.35 | 4.09 | - | 1.03 | - |
| Real NVP [63] | - | 3.49 | 4.28 | 3.02 | - | - |

VD-VAE [Child, 2021]

- Autoregressive models have outperformed VAEs (will be covered later)
- **Main idea:** However, very deep VAEs generalize autoregressive models

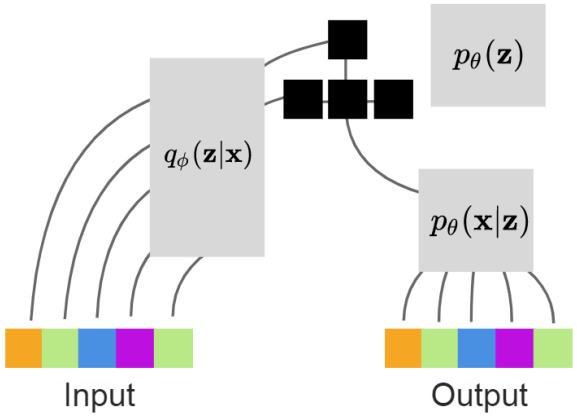
Latent variables are identical to observed variables




**Observation 1**: Hierarchical VAEs with N layers (N = dimension of data D) generalizes autoregressive models

- e.g.) learns deterministic identity function

Latent variables allow for parallel generation




**Observation 2:** VAEs with fewer layers (N < D) can still model data by learning efficient hierarchies of latent variables

- e.g.) learns conditional independence

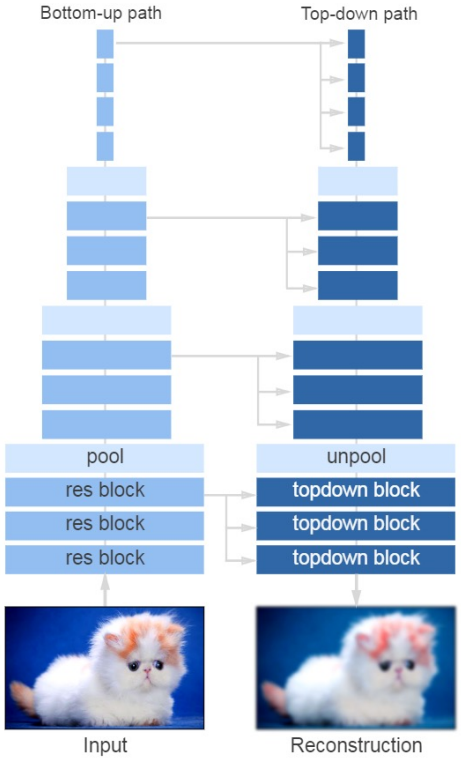
VD-VAE [Child, 2021]

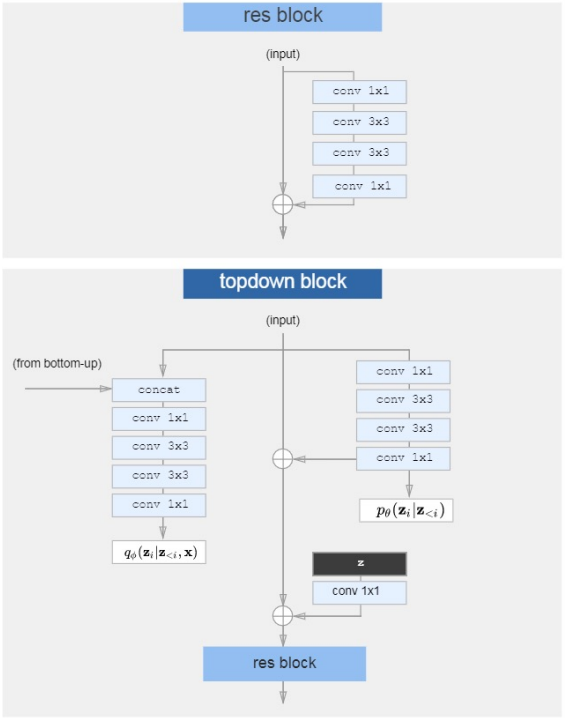
Empirically, deep VAEs often suffer from unstable training

- **Recap:** NVAE requires complex techniques to stabilize KL

**Q:** How to make VAE deeper?

**Idea 1:** Top-down architecture with bottleneck residual blocks



## Very Deep VAE (VD-VAE)

VD-VAE [Child, 2021]

Empirically, deep VAEs often suffer from unstable training

- **Recap:** NVAE requires complex techniques to stabilize KL

**Q:** How to make VAE deeper?

**Idea 2:** Additional simple techniques

- Transposed CNNs **=> Nearest-neighbor upsampling**
- **Scale down weight initialization** of final layer in residual block
- **Gradient skipping**: skip updates when gradient norm is above threshold

VD-VAE [Child, 2021]

**Experiment:** Very deep VAEs (>50 layers) can outperform autoregressive models with fewer parameters while maintaining fast sampling

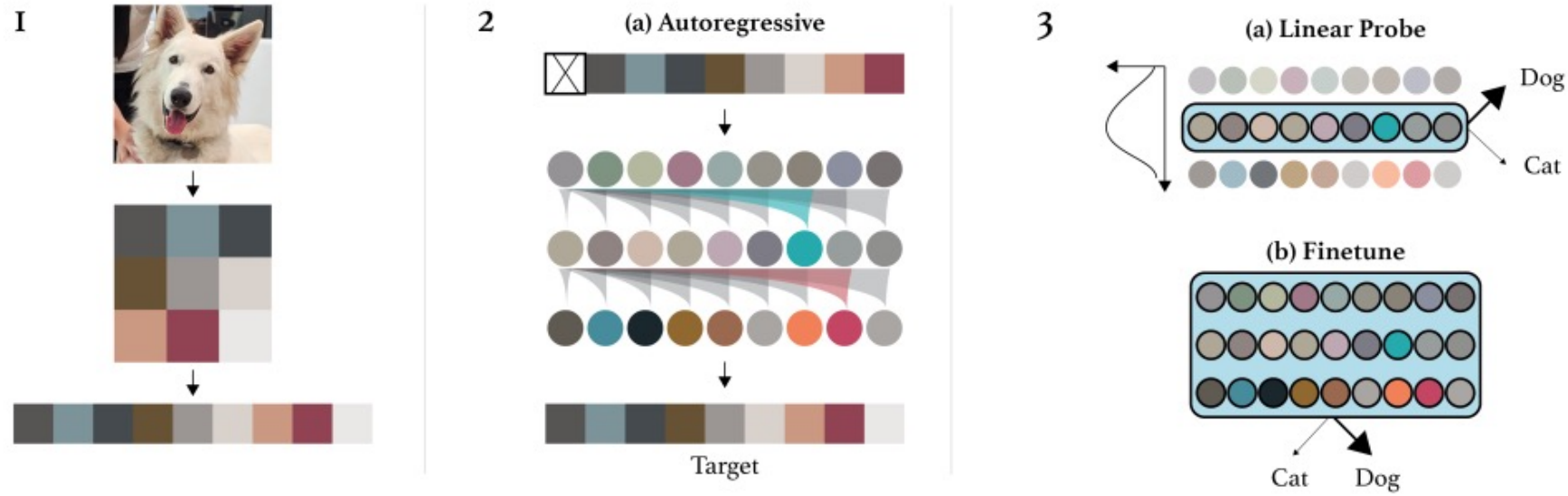
| | Model type | Params | Depth | Sampling | NLL |
|---|---|---|---|---|---|
| **CIFAR-10** | | | | | |
| PixelCNN++ (Salimans et al., 2017) | AR | 53M* | | $D$ | 2.92 |
| PixelSNAIL (Chen et al., 2017) | AR | | | $D$ | 2.85 |
| Sparse Transformer (Child et al., 2019) | AR | 59M | | $D$ | **2.80** |
| VLAE (Chen et al., 2016) | VAE | | | $D$ | $\leq$ 2.95 |
| IAF-VAE (Kingma et al., 2016) | VAE | | 12 | 1 | $\leq$ 3.11 |
| Flow++ (Ho et al., 2019) | Flow | 31M | | 1 | $\leq$ 3.08 |
| BIVA (Maaløe et al., 2019) | VAE | 103M | 15 | 1 | $\leq$ 3.08 |
| NVAE (Vahdat & Kautz, 2020) | VAE | 131M | 30 | 1 | $\leq$ 2.91 |
| Very Deep VAE **(ours)** | VAE | 39M | 45 | 1 | $\leq$ **2.87** |
| **ImageNet-32** | | | | | |
| Gated PixelCNN | AR | 177M* | 10 | $D$ | 3.83 |
| Image Transformer (Parmar et al., 2018) | AR | | | $D$ | **3.77** |
| BIVA | VAE | 103M* | 15 | 1 | $\leq$ 3.96 |
| NVAE | VAE | 268M | 28 | 1 | $\leq$ 3.92 |
| Flow++ | Flow | 169M | | 1 | $\leq$ 3.86 |
| Very Deep VAE **(ours)** | VAE | 119M | 78 | 1 | $\leq$ **3.80** |
| **ImageNet-64** | | | | | |
| Gated PixelCNN | AR | 177M* | | $D$ | 3.57 |
| SPN (Menick & Kalchbrenner, 2018) | AR | 150M | | $D$ | 3.52 |
| Sparse Transformer | AR | 152M | | $D$ | **3.44** |
| Glow (Kingma & Dhariwal, 2018) | Flow | | | 1 | 3.81 |
| Flow++ | Flow | 73M | | 1 | $\leq$ 3.69 |
| Very Deep VAE **(ours)** | VAE | 125M | 75 | 1 | $\leq$ **3.52** |
| **FFHQ-256 (5 bit)** | | | | | |
| NVAE | VAE | | 36 | 1 | $\leq$ 0.68 |
| Very Deep VAE **(ours)** | VAE | 115M | 62 | 1 | $\leq$ **0.61** |
| **FFHQ-1024 (8 bit)** | | | | | |
| Very Deep VAE **(ours)** | VAE | 115M | 72 | 1 | $\leq$ **2.42** |

## Table of Contents

**Generative Pretraining from Pixels** [Chen et al., 2020]

- Apply GPT [Brown et al., 2020] to image domain by flattening image to 1D.
- Train autoregressive transformer which predicts the pixels without knowledge of 2D input structure.

$$L_{AR} = \mathop{\mathbb{E}}_{x \sim X}[-\log p(x)] \quad \text{where} \quad p(x) = \prod_{i=1}^{n} p(x_{\pi_i} | x_{\pi_1}, \ldots, x_{\pi_{i-1}}, \theta)$$



## **Generative Pretraining from Pixels** [Chen et al., 2020]

- It even outperforms supervised representation with ImageNet in transfer learning.

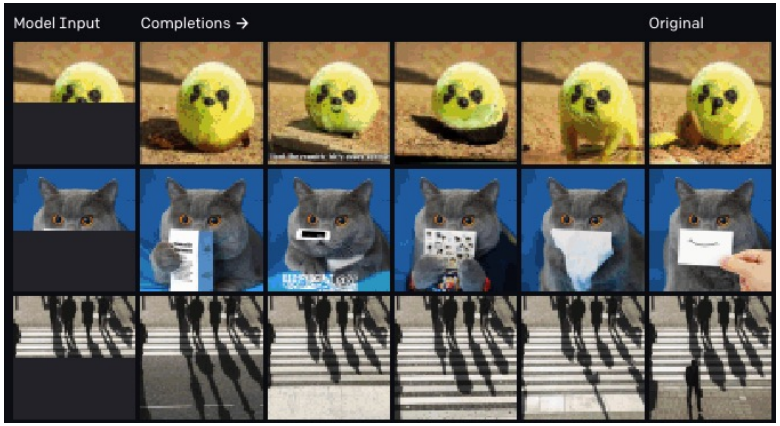
| Model | Acc | Unsup Transfer | Sup Transfer |
|---|---|---|---|
| **CIFAR-10** | | | |
| ResNet-152 | 94 | | √ |
| SimCLR | 95.3 | √ | |
| iGPT-L | 96.3 | √ | |
| **CIFAR-100** | | | |
| ResNet-152 | 78.0 | | √ |
| SimCLR | 80.2 | √ | |
| iGPT-L | 82.8 | √ | |
| **STL-10** | | | |
| AMDIM-L | 94.2 | √ | |
| iGPT-L | 95.5 | √ | |

Linear probing

| Model | Acc | Unsup Transfer | Sup Transfer |
|---|---|---|---|
| **CIFAR-10** | | | |
| AutoAugment | 98.5 | | |
| SimCLR | 98.6 | √ | |
| GPipe | 99.0 | | √ |
| iGPT-L | 99.0 | √ | |
| **CIFAR-100** | | | |
| iGPT-L | 88.5 | √ | |
| SimCLR | 89.0 | √ | |
| AutoAugment | 89.3 | | |
| EfficientNet | 91.7 | | √ |

Full finetuning

- It also shows inpainting ability



**Scaling Autoregressive Video Models** [Weissenborn et al., 2020]

- Apply GPT to video domain by flattening video to 1D.
- However, using all pixels from a video is computationally infeasible
  - e.g.) 32x32 video of length 16 has 16 * 32 * 32 * 3 = 49,152 pixels
    - Much longer than the input length of GPT3 (=2048), ImageGPT (=3072)

**Main idea:** Reduce the complexity of autoregressive video generation by

- 1) Designing an efficient self-attention layer for videos
- 2) Operating on sub-sampled videos instead of pixels

**Scaling Autoregressive Video Models** [Weissenborn et al., 2020]

- Apply GPT to video domain by flattening video to 1D.
- However, using all pixels from a video is computationally infeasible

**Idea 1:** Video Transformer with multiple stacked block-local self-attention

- Reduces the computation cost of self-attention over videos, by
  1. Decompose a video of $(T, H, W)$ into $n_p = t \cdot h \cdot w$ blocks of $(t, h, w)$
  2. Separately apply self attentions over $n_p$ blocks
     - Attention complexity $(T \cdot H \cdot W)^2 \Rightarrow n_p \cdot (t \cdot h \cdot w)^2$
  3. Concatenate the outputs and process through a fully connected layer

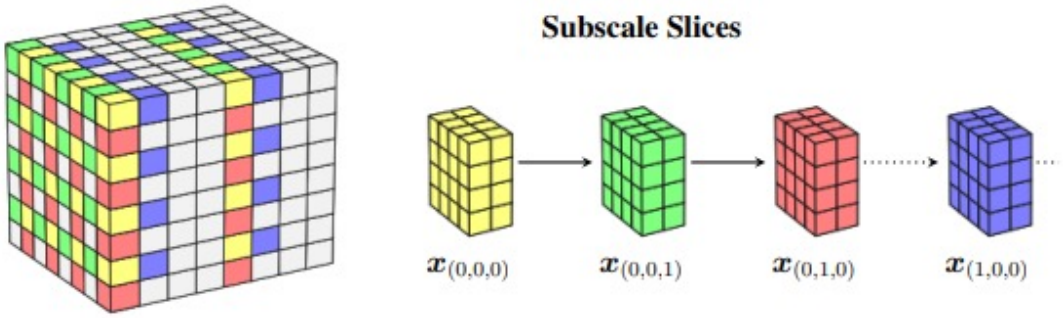
- For the connectivity between all pixels, use different block sizes at every layer

**Scaling Autoregressive Video Models** [Weissenborn et al., 2020]

- Apply GPT to video domain by flattening video to 1D.
- However, using all pixels from a video is computationally infeasible

**Idea 2:** Divide the video into non-overlapping 3D blocks

- Further reduces the complexity by decomposing the video itself
- Introduce a subscale factor $\boldsymbol{s} = (s_t, s_h, s_w)$ that divides a video into $s = (s_t \cdot s_h \cdot s_w)$ sub-sampled videos (slices)
  - Then, each slice is processed through the block-local self-attention layers



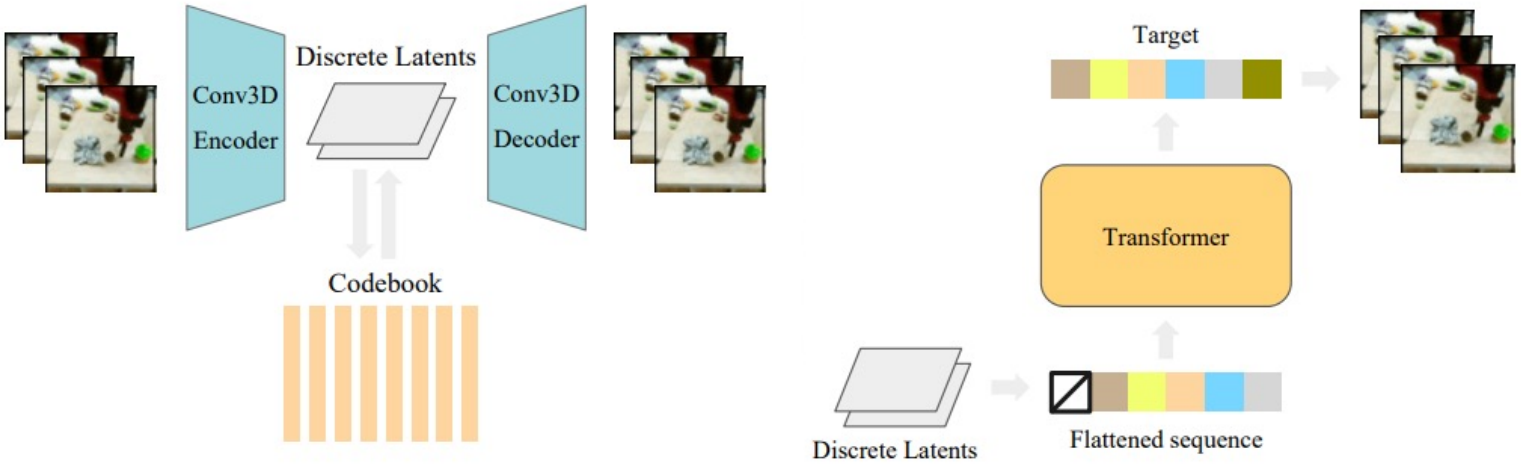

- And sequentially generate $x_{(0,0,0)}, x_{(0,0,1)}, \ldots$
  - e.g.) If we use $\boldsymbol{s} = (4, 2, 2)$, each slice consists of 4 * 16 * 16 * 3 = 3072 pixels
    - Attention complexity: $49152^2$ => $3072^2$ (256 times lower)

**VideoGPT** [Yan et al., 2021]

- Other approach for autoregressive video generation
- Learns downsampled discrete representations over space-time

## Main idea of VideoGPT

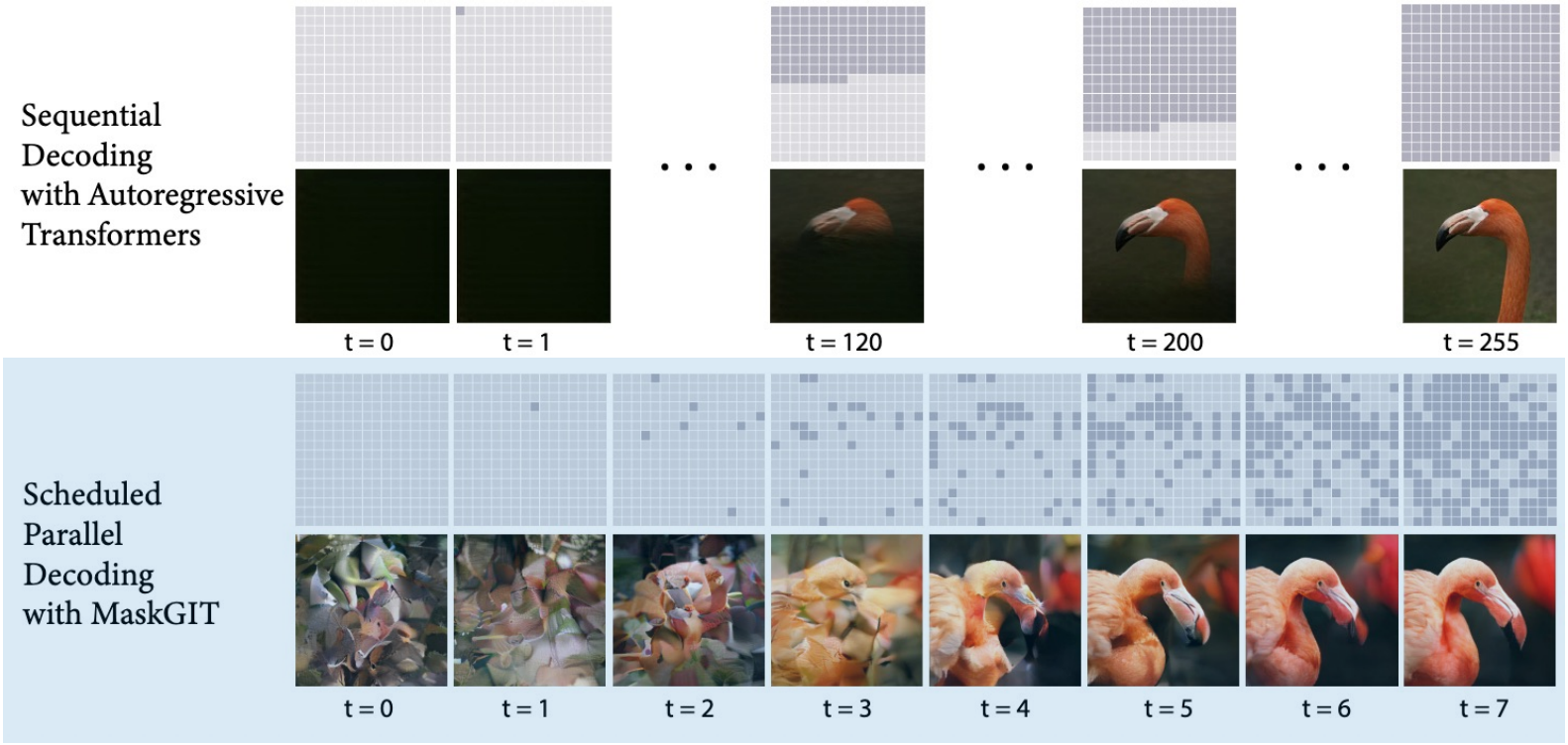
1. Train a VQ-VAE with 3D CNNs on the video data to learn discrete latent representations downsampled over space-time
2. Train autoregressive transformer (Image-GPT architecture) in the latent space for learning a prior
3. Decode the predicted discrete latents using the VQ-VAE decoder



## Table of Contents

**MaskGIT** [Chang et al., 2022]

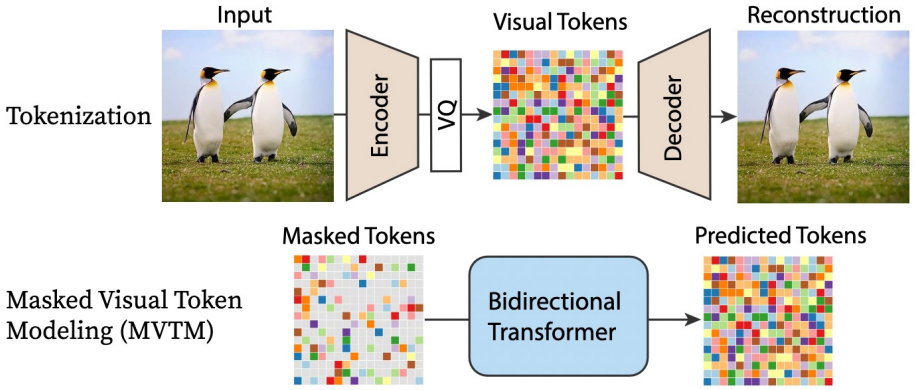
- Non-autoregressive generative modeling based on VQ-VAE + Transformer
- Enables parallel decoding and thus much faster sampling



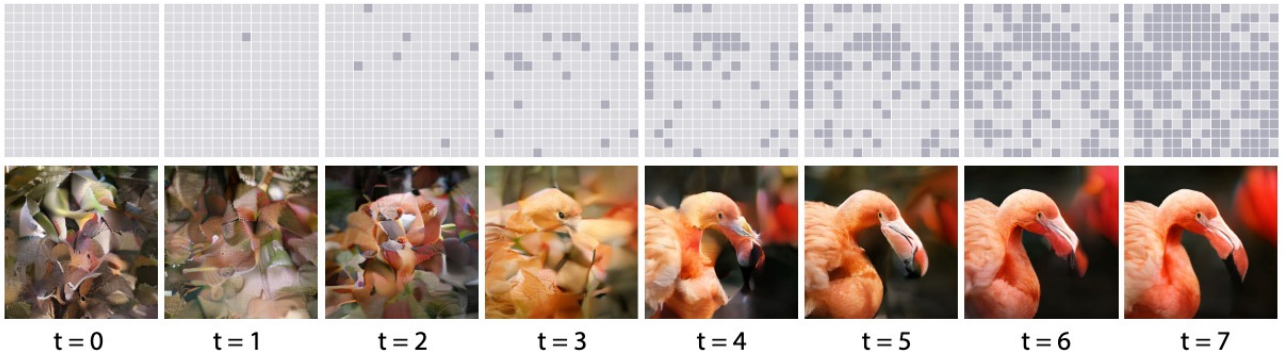
**MaskGIT** [Chang et al., 2022]

- **Stage 1.** Tokenization of images to discrete visual tokens
- **Stage 2.** Masked modeling via bidirectional Transformer

For sampling, MaskGiT starts from a blank canvas withall the tokens masked out




Scheduled Parallel Decoding with MaskGIT



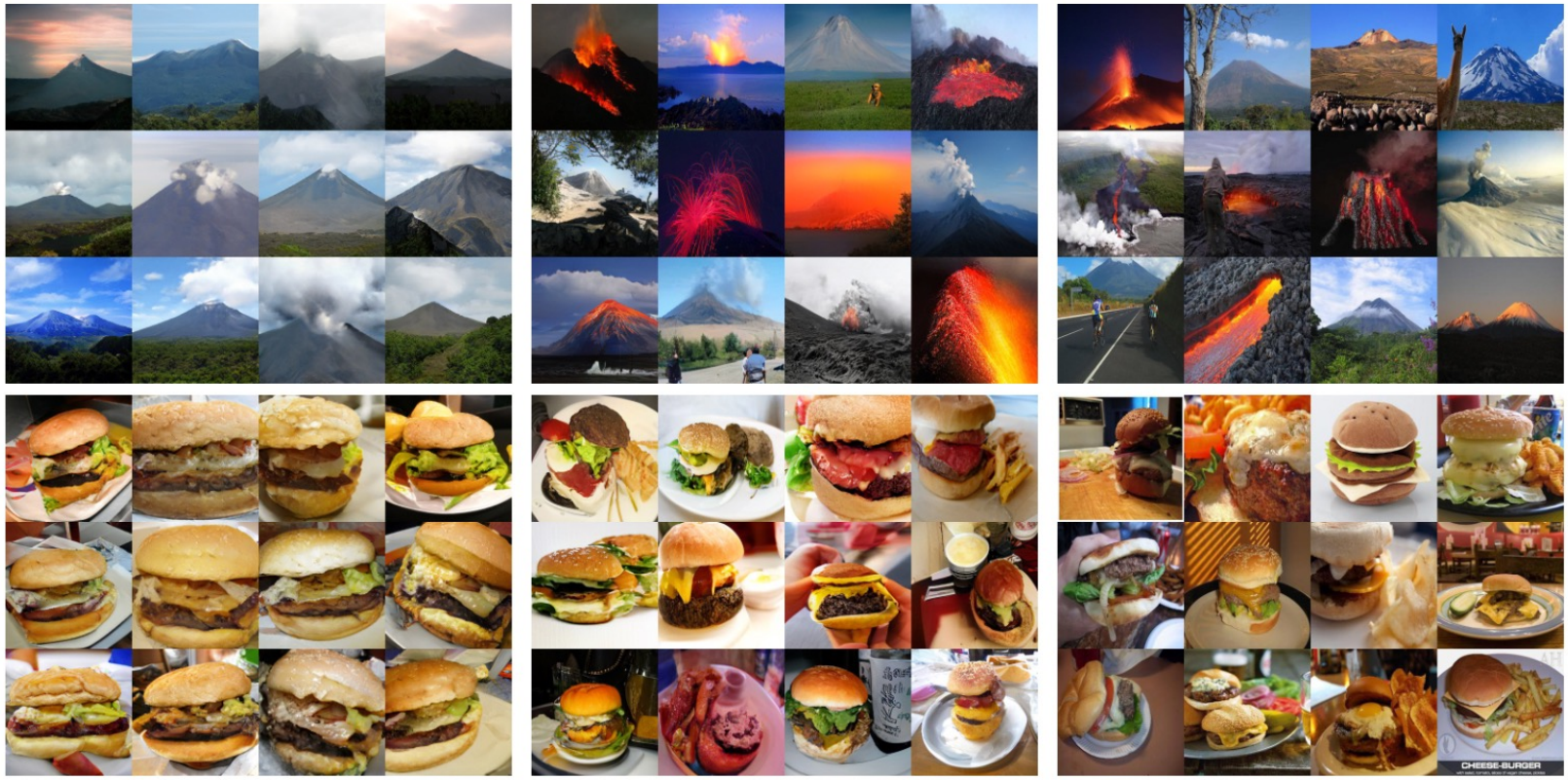
**Experiment:** It achieves state-of-the-art results with faster inference time

- In particular, on classifier accuracy score (CAS) score
- Compared with autoregressive modeling (e.g., VQGAN), much faster (=8 step)

| Model | FID ↓ | IS ↑ | Prec ↑ | Rec ↑ | # params | # steps | CAS ×100 ↑ | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | Top-1 (76.6) | Top-5 (93.1) |
| **ImageNet 256×256** | | | | | | | | |
| DCTransformer [32] □ | 36.51 | n/a | 0.36 | **0.67** | 738M | >1024 | | |
| BigGAN-deep [4] | 6.95 | **198.2** | **0.87** | 0.28 | 160M | 1 | 43.99 | 67.89 |
| Improved DDPM [33]□ | 12.26 | n/a | 0.70 | 0.62 | 280M | 250 | | |
| ADM [12]□ | 10.94 | 101.0 | 0.69 | 0.63 | 554M | 250 | | |
| VQVAE-2 [37]□ | 31.11 | ∼45 | 0.36 | 0.57 | 13.5B† | 5120 | 54.83 | 77.59 |
| VQGAN [15]□ | 15.78 | 78.3 | n/a | n/a | 1.4B | 256 | | |
| VQGAN* | 18.65 | 80.4 | 0.78 | 0.26 | 227M | 256 | 53.10 | 76.18 |
| **MaskGIT (Ours)** | **6.18** | 182.1 | 0.80 | 0.51 | 227M | *8* | **63.14** | **84.45** |
| **ImageNet 512×512** | | | | | | | | |
| BigGAN-deep [4] | 8.43 | **232.5** | **0.88** | 0.29 | 160M | 1 | 44.02 | 68.22 |
| ADM [12]□ | 23.24 | 58.06 | 0.73 | **0.60** | 559M | 250 | | |
| VQGAN* | 26.52 | 66.8 | 0.73 | 0.31 | 227M | 1024 | 51.29 | 74.24 |
| **MaskGIT (Ours)** | **7.32** | 156.0 | 0.78 | 0.50 | 227M | *12* | **63.43** | **84.79** |

**Experiment:** It achieves state-of-the-art results with faster inference time

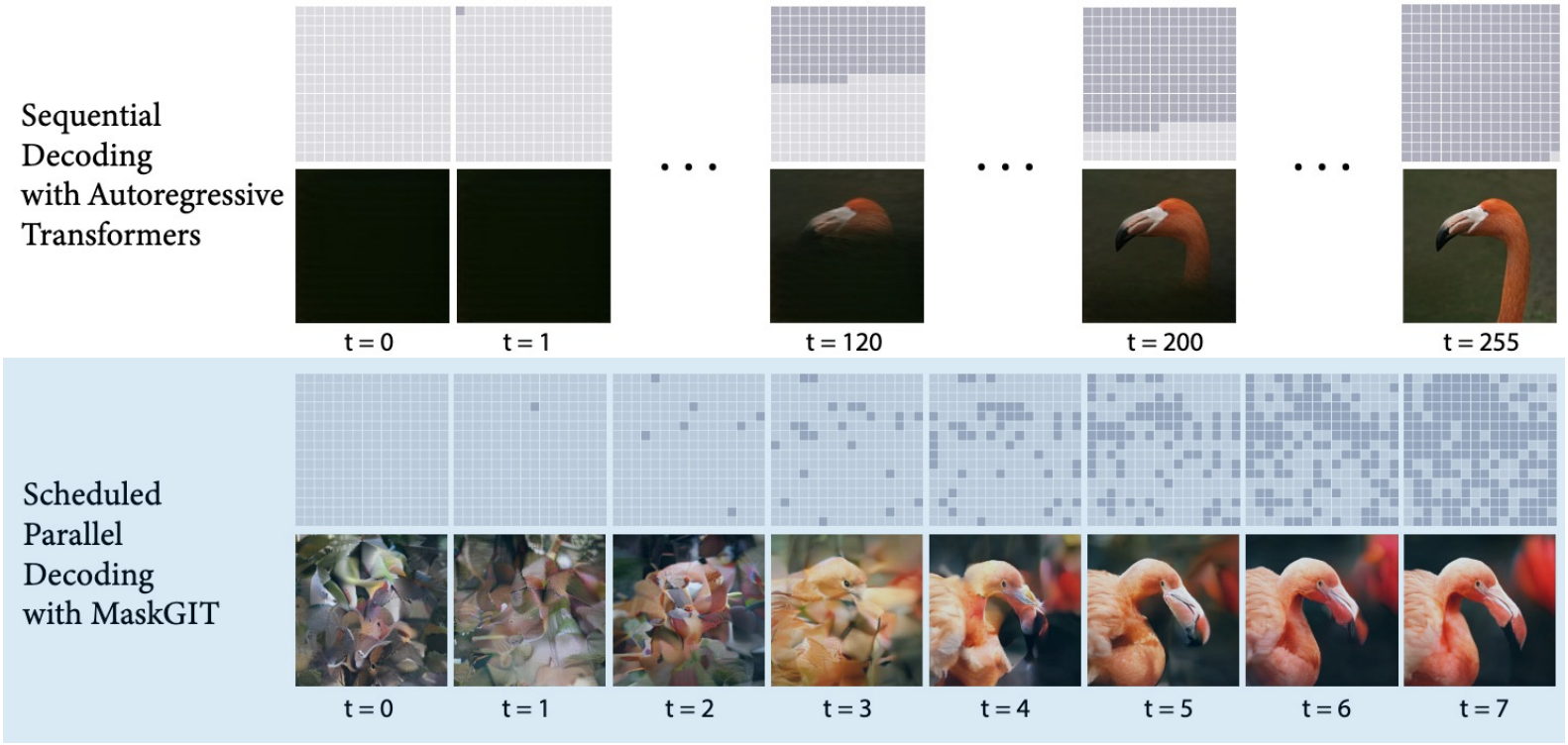
- In particular, on classifier accuracy score (CAS) score
- Compared with autoregressive modeling (e.g., VQGAN), much faster (=8 step)




BigGAN-deep (FID=6.95) MaskGIT (FID=6.18) Training Set

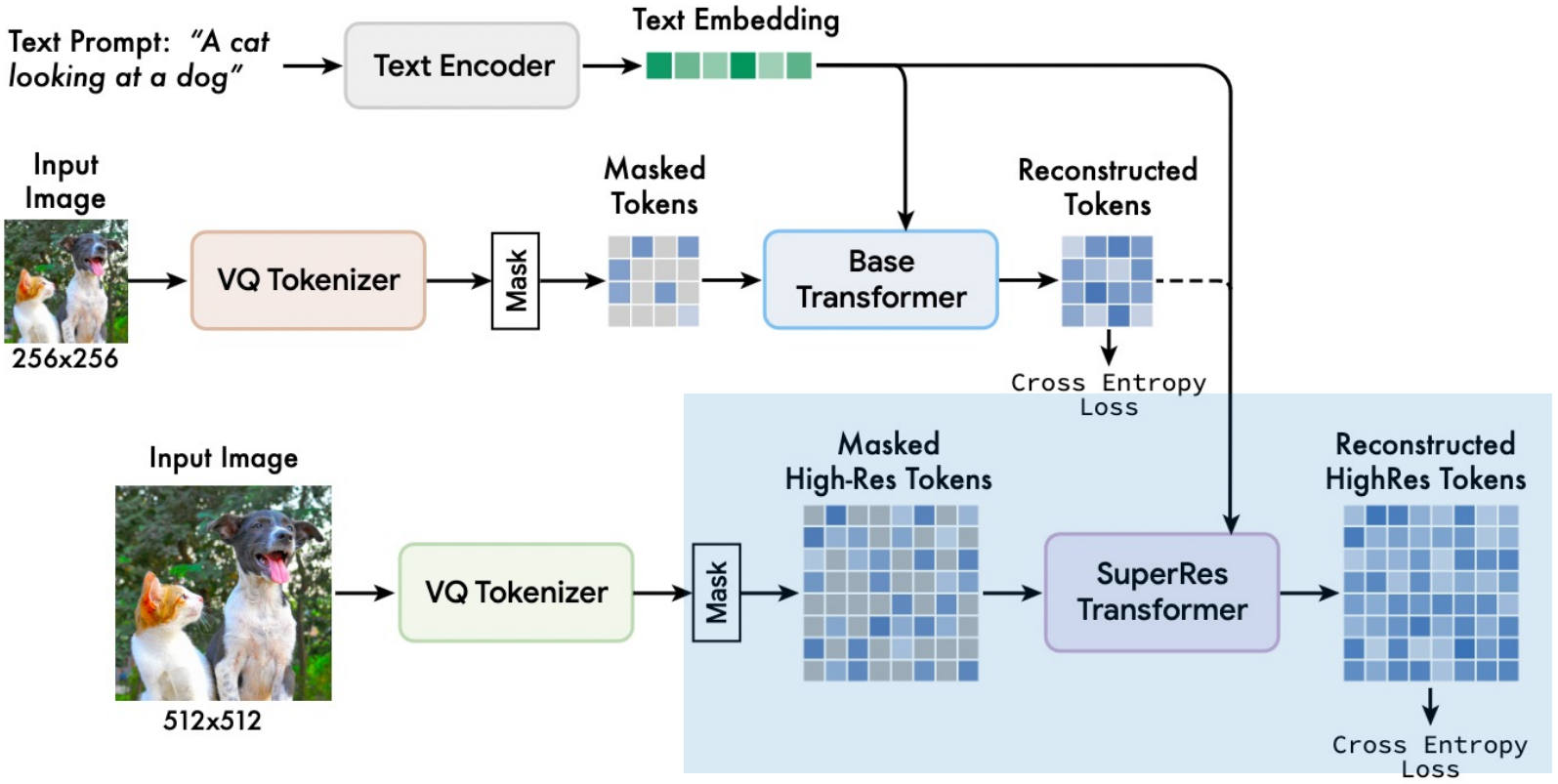
**MaskGIT** [Chang et al., 2022]

- Non-autoregressive generative modeling based on VQ-VAE + Transformer
- Enables parallel decoding and thus much faster sampling



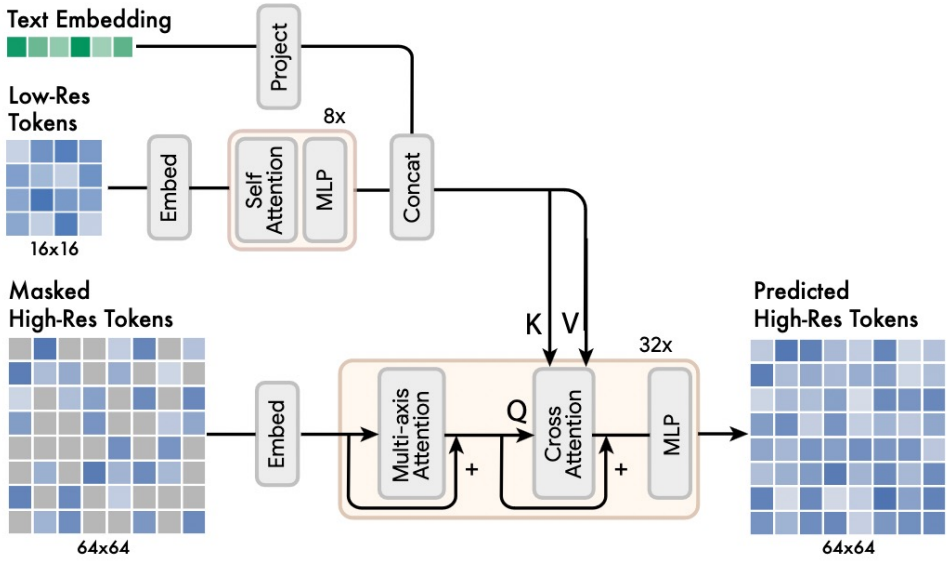
**Muse:** Scaling up MaskGiT for text-to-image generation task

- To scale-up the generation to high-resolution images, use hierarchical architecture
- Hence, it generates resolution beyond 256x256

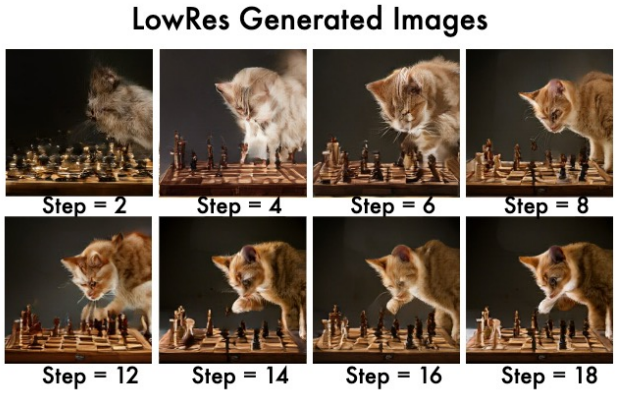


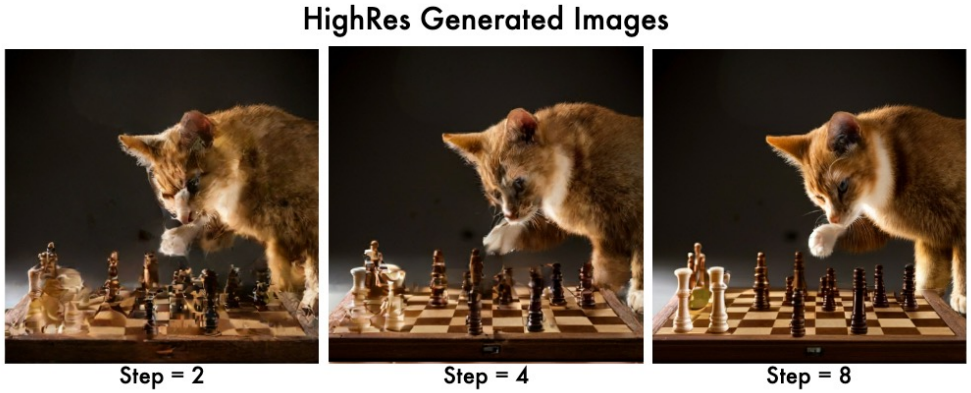
**Muse:** Scaling up MaskGiT for text-to-image generation task

- To scale-up the generation to high-resolution images, use hierarchical architecture
- Hence, it generates resolution beyond 256x256







**Muse:** Scaling up MaskGiT for text-to-image generation task

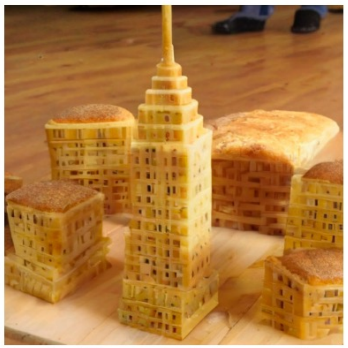
- In particular, on classifier accuracy score (CAS) score
- Compared with autoregressive modeling (e.g., VQGAN), much faster (=8 step)

A fluffy baby sloth with a knitted hat trying to figure out a laptop, close up.

A sheep in a wine glass.

A futuristic city with flying cars.

A large array of colorful cupcakes, arranged on a maple table to spell MUSE.



Manhattan skyline made of bread.

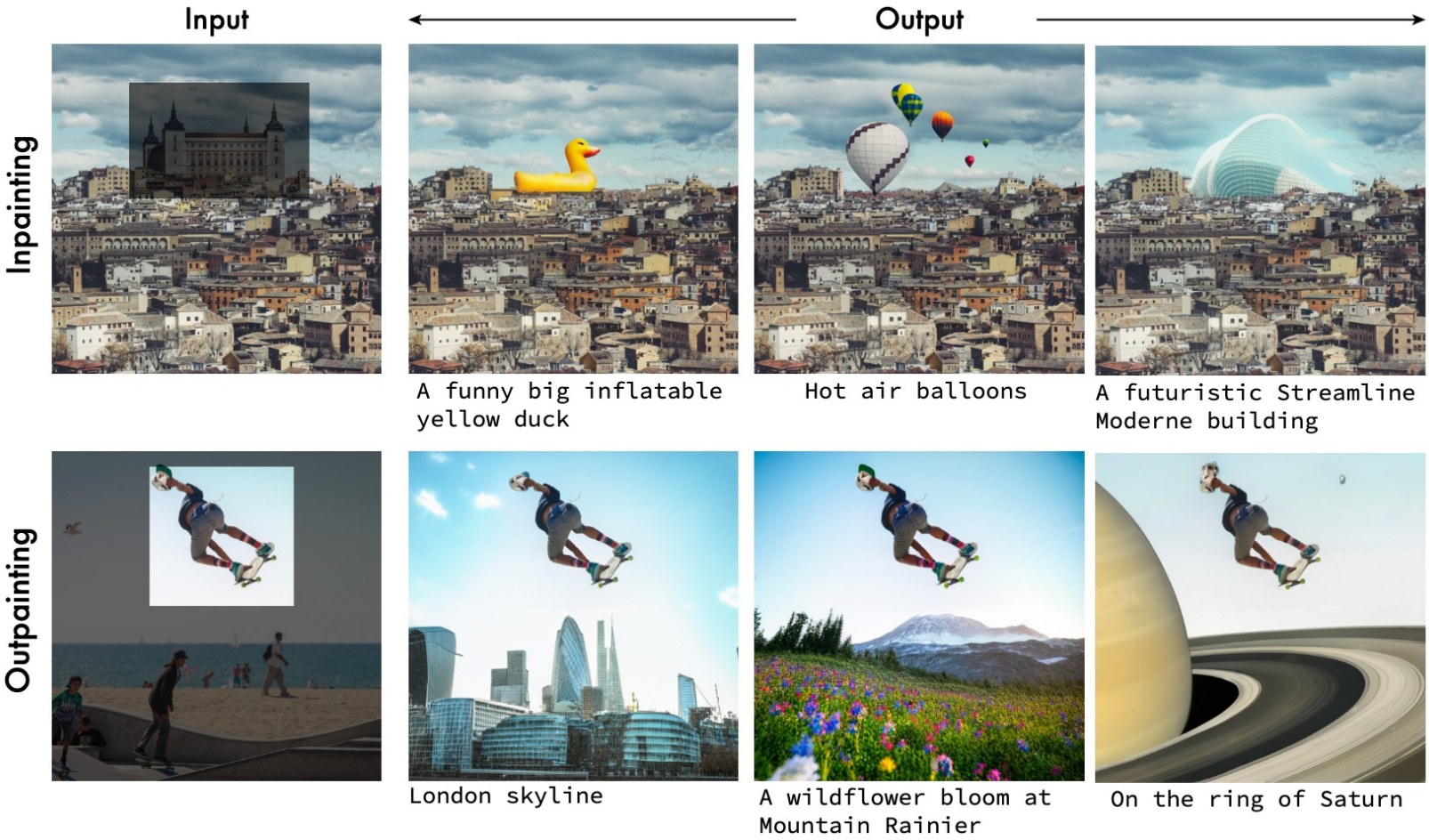
Astronauts kicking a football in front of Eiffel tower.

Two cats doing research.

3D mesh of Titanic floating on a water lily pond in the style of Monet.

**Muse:** Scaling up MaskGiT for text-to-image generation task

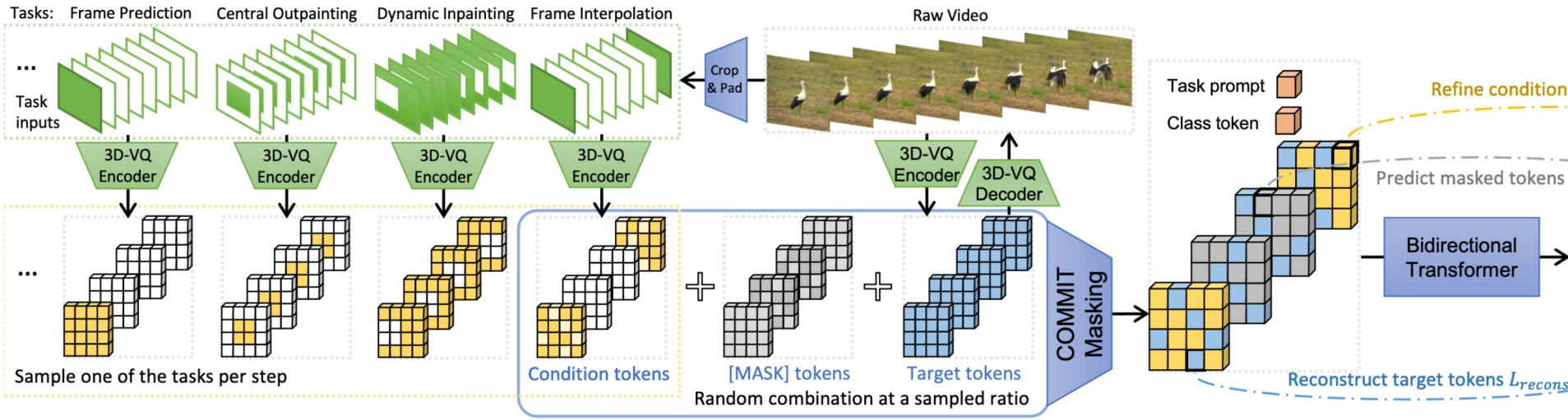
- In particular, on classifier accuracy score (CAS) score
- Compared with autoregressive modeling (e.g., VQGAN), much faster (=8 step)



**MAGVIT:** MaskGiT for video generation

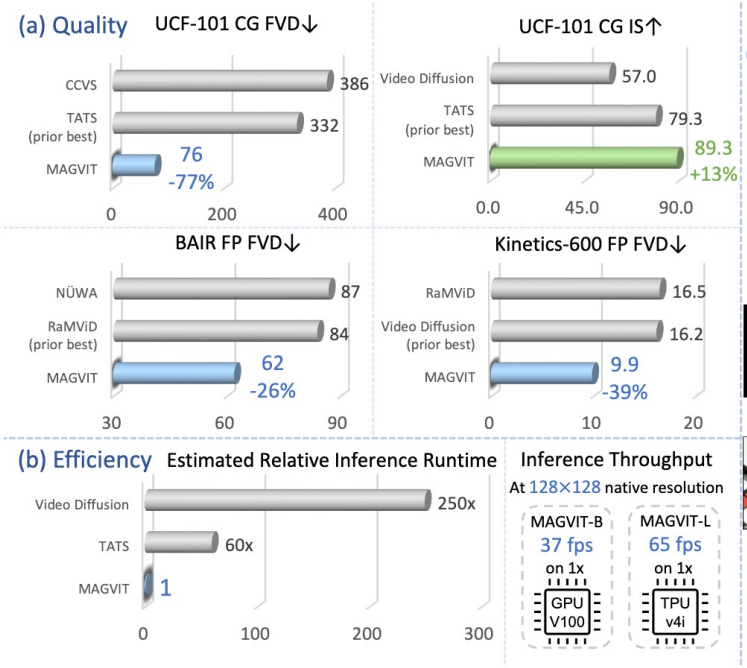
- Demonstrates diverse applications as well as generation, e.g., prediction
- All applications can be achieved with a single model

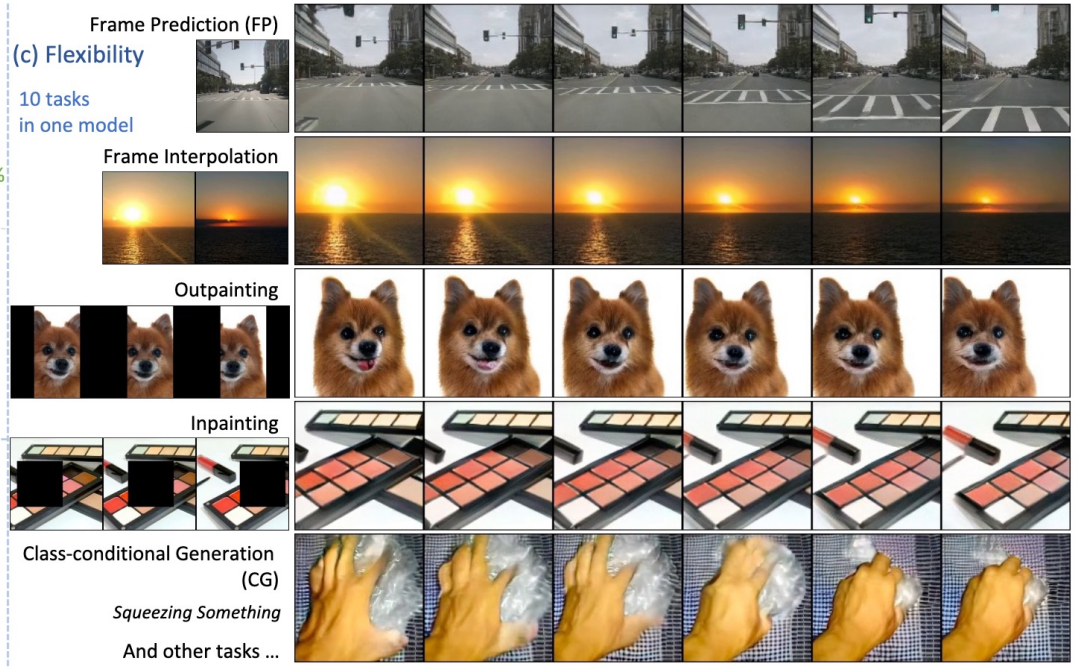
**Main idea:** Considers diverse masking suitable for each task during training



**MAGVIT:** MaskGiT for video generation

- Demonstrates diverse applications as well as generation, e.g., prediction
- All applications can be achieved with a single model



**MAGVIT:** MaskGiT for video generation

- Demonstrates diverse applications as well as generation, e.g., prediction
- All applications can be achieved with a single model

Panorama generation

Video prediction

Video interpolation

## Table of Contents

## Summary

- We discussed **3 different categories** of generative models
  1. **GANs:** Implicit generative modeling
  2. **Generative diffusion process:** Related to score-matching or diffusion process
  3. **Others:** VAEs, autoregressive models, and masked modeling

- Each of them has its own drawback, but they have been remarkably mitigated:
  - **GANs**: Recently showed strong scalability
  - **Generative diffusion process:** Faster synthesis with efficient solver and distillation

- Currently, **generative diffusion process** is the dominant generative model due to:
  - **Open-sourced foundation models**: Such as Stable-diffusion models
  - **Strong generalization power:** Much less suffer from mode collapse problem
  - But still, other generative models can be a game changer in the future!

# References


[Goodfellow, et. al., 2014] Generative adversarial nets, NIPS 2014
link: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[Theis, et. al., 2016] A note on the evaluation of generative models, ICLR 2016
link: http://bethgelab.org/media/publications/1511.01844v1.pdf

[Radford, et. al., 2015] Unsupervised representation learning with deep convolutional generative adversarial networks.
link: https://arxiv.org/pdf/1511.06434.pdf

[Ledig, et. al., 2017] Photo-realistic single image super-resolution using a generative adversarial networks, CVPR 2017
link: http://openaccess.thecvf.com/content_cvpr_2017/papers/Ledig_Photo-Realistic_Single_Image_CVPR_2017_paper.pdf

[Zhu, et. al., 2017] Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV 2017
link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8237506

[Karras, et. al., 2018] Progressive growing of GANs for improved quality, stability, and variation, ICLR 2018
link: https://arxiv.org/abs/1710.10196

[Salimans, et. al., 2016] Improved techniques for training GANS, NIPS 2016
link: https://arxiv.org/abs/1606.03498

[Huszar 2015] How (not) to train your generative model: scheduled sampling, likelihood, adversary?
link: https://arxiv.org/pdf/1511.05101.pdf

[Mirza et al., 2014] Conditional Generative Adversarial Nets
link: https://arxiv.org/pdf/1411.1784.pdf

# References


[Odena, et. al., 2017] Conditional Image Synthesis with Auxiliary Classifier GANs, ICML 2017
link: https://arxiv.org/pdf/1610.09585.pdf

[Basart et. al., 2017] Analysis of Generative Adversarial Models
link: https://newtraell.cs.uchicago.edu/files/ms_paper/xksteven.pdf

[Dumoulin, et. al., 2017] A Learned Representation for Artistic Style, ICLR 2017
link: https://arxiv.org/pdf/1610.07629.pdf

[DeVries, et. al., 2017] Modulating early visual processing by language, NIPS 2017
link: https://arxiv.org/pdf/1707.00683.pdf

[Miyato, et. al., 2018] cGANs with Projection Discriminator, ICLR 2018
link: https://arxiv.org/pdf/1802.05637.pdf

[Arjovsky, et. al., 2017] Wasserstein GAN, ICML 2017
link: https://arxiv.org/pdf/1701.07875.pdf

[Arjovsdky and Bottou, 2017] Towards principled methods for training generative adversarial networks, ICLR 2017
link: https://arxiv.org/pdf/1701.04862.pdf

[Villani, 2009] Optimal transport: old and new, Grundlehren der mathematischen wissenschaften 2009
link: http://cedricvillani.org/wp-content/uploads/2012/08/preprint-1.pdf

[Reed, et. al., 2016] Generative adversarial text to image synthesis, ICML 2016
link: https://arxiv.org/pdf/1605.05396.pdf

[Wang, et. al., 2004] Image quality assessment: from error visibility to structural similarity, IEEE transactions on image processing 2004
link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1284395

# References


[Radford, et. al., 2015] Unsupervised representation learning with deep convolutional generative adversarial networks, 2015
link: https://arxiv.org/pdf/1511.06434.pdf

[Gulrajani, et. al., 2017] Improved training of Wasserstein GANs, NIPS 2017
link: https://arxiv.org/pdf/1704.00028.pdf

[Miyato, et. al., 2018] Spectral normalization for generative adversarial networks, ICLR 2018

link: https://arxiv.org/pdf/1802.05957.pdf

[Zhang, et. al., 2019] Self-Attention Generative Adversarial Networks, ICML 2019
link: https://arxiv.org/pdf/1805.08318.pdf

[Wang, et. al., 2018] Non-local neural networks, CVPR 2018
link: https://arxiv.org/pdf/1711.07971.pdf

[Brock, et. al., 2019] Large Scale GAN Training for High Fidelity Natural Image Synthesis, ICLR 2019
link: https://arxiv.org/pdf/1809.11096.pdf

[Perez, et. al., 2019] FiLM: Visual Reasoning with a General Conditioning Layer, AAAI 2018
link: https://arxiv.org/pdf/1709.07871.pdf

[Karras, et. al., 2019] Large Scale GAN Training for High Fidelity Natural Image Synthesis, CVPR 2019
link: https://arxiv.org/pdf/1809.11096.pdf

[Huang, et. al., 2017] Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization, ICCV 2017
link: https://arxiv.org/pdf/1703.06868.pdf

# References


[Lucic et al., 2018] Are GANs Created Equal? A Large-Scale Study, NIPS 2018
link: https://arxiv.org/abs/1711.10337

[Kurach et al., 2019] A Large-Scale Study on Regularization and Normalization in GANs, ICML 2019
link: http://proceedings.mlr.press/v97/kurach19a.html

[Kodali et al., 2017] On Convergence and Stability of GANs, 2018
link: https://arxiv.org/abs/1705.07215v5

[Ba et al., 2016] Layer Normalizatiton, 2016
link: https://arxiv.org/abs/1607.06450

[Ioffe & Szegedy, 2015] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015
link: https://arxiv.org/abs/1502.03167

[Mao et al., 2017] Least Squares Generative Adversarial Networks, 2016
link: https://arxiv.org/abs/1611.04076

[Zhang et al., 2019] Consistency Regularization for Generative Adversarial Networks, ICLR 2020
link: https://arxiv.org/abs/1910.12027

[Karras et al., 2020a] Analyzing and Improving the Image Quality of StyleGAN, CVPR 2020
link: https://arxiv.org/abs/1912.04958

[Karras et al., 2020b] Training Generative Adversarial Networks with Limited Data, NeurIPS 2020
link: https://arxiv.org/abs/2006.06676

[Zhao et al., 2020a] Improved Consistency Regularization for GANs, 2020
link: https://arxiv.org/abs/2002.04724

[Zhao et al., 2020b] Differentiable Augmentation for Data-Efficient GAN Training, NeurIPS 2020
link: https://arxiv.org/abs/2006.10738

# References


[Kingma et al., 2013] Auto-Encoding Variational Bayes, ICLR 2013
link: https://arxiv.org/abs/1802.06455

[Burda et al., 2016] Importance Weighted Autoencoders, ICLR 2016
link: https://arxiv.org/abs/1509.00519

[Kim et al., 2018] Semi-Amortized Variational Autoencoders, ICML 2018
link: https://arxiv.org/abs/1802.02550

[Bowman et al., 2016] Generating Sentences from a Continuous Space, CONLL 2016
link: https://arxiv.org/abs/1511.06349

[Razavi et al., 2019a] Preventing Posterior Collapse with delta-VAEs, ICLR 2019
link: https://arxiv.org/abs/1901.03416

[Tolstikhin et al., 2018] Wasserstein Auto-Encoders, ICLR 2018
link: https://arxiv.org/abs/1711.01558

[He et al., 2019] Lagging Inference Networks and Posterior Collapse in Variational Autoencoders, ICLR 2019
link: https://arxiv.org/abs/1901.05534

[Oord et al., 2017] Neural Discrete Representation Learning, NeurIPS 2017
link: https://arxiv.org/abs/1711.00937

[Razavi et al., 2017b] Generating Diverse High-Fidelity Images with VQ-VAE-2, NeurIPS 2019
link: https://arxiv.org/abs/1906.00446

[Vahdat et al., 2020] NVAE: A Deep Hierarchical Variational Autoencoder, NeurIPS 2020
link: https://arxiv.org/abs/2007.03898

[Child et al., 2021] Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images, ICLR 2021
link: https://arxiv.org/abs/2011.10650

# References


[Sohl-Dickstein et al., 2015] Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
link: https://arxiv.org/abs/1503.03585

[Ho et al., 2020] Denoising Diffusion Probabilistic Models, NeurIPS 2020
link: https://arxiv.org/abs/2006.11239

[Nichol and Dhariwal, 2021] Improved Denoising Diffusion Probabilistic Models, ICML 2021
link: https://arxiv.org/abs/2102.09672

[Song et al., 2021] Denoising Diffusion Implicit Models, ICLR 2021
link: https://arxiv.org/abs/2010.02502

[Dhariwal and Nichol, 2021] Diffusion Models Beat GANs on Image Synthesis, NeurIPS 2021
link: https://arxiv.org/abs/2105.05233

[Ho et al., 2021] Classifier-Free Diffusion Guidance, NeurIPS Workshop on Deep Generative Models and Downstream Applications 2021
link: https://openreview.net/pdf?id=qw8AKxfYbI

[Nichol et al., 2021] GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models, arXiv 2021
link: https://arxiv.org/abs/2112.10741

# References


[LeCun et al., 2006] A Tutorial on Energy-Based Learning, Technical report 2006
link: http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf

[Du & Mordatch, 2019] Implicit Generation and Generalization in Energy-Based Models, NeurIPS 2019
link: https://arxiv.org/abs/1903.08689

[Welling & Teh, 2011] Bayesian Learning via Stochastic Gradient Langevin Dynamics, ICML 2011
link: https://dl.acm.org/doi/10.5555/3104482.3104568

[Zhao et al., 2017] Energy-based Generative Adversarial Network, ICLR 2017
link: https://arxiv.org/abs/1609.03126

[Grathwohl et al., 2020] Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One, ICLR 2020
link: https://arxiv.org/abs/1912.03263

[Song & Kingma, 2021] How to Train Your Energy-Based Models, arXiv 2021
link: https://arxiv.org/abs/2101.03288

[Hyvärinen, 2005] Estimation of Non-Normalized Statistical Models by Score Matching, JMLR 2005
link: https://jmlr.org/papers/v6/hyvarinen05a.html

[Vincent, 2011] A Connection Between Score Matching and Denoising Autoencoders, Neural Computation 2011
link: https://ieeexplore.ieee.org/document/6795935

[Song et al., 2019] Generative Modeling by Estimating Gradients of the Data Distribution, NeurIPS 2019
link: https://arxiv.org/abs/1907.05600

[Song et al., 2021] Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021
link: https://arxiv.org/abs/2011.13456