Generative Models (Part II)

(2026)

Lecturer: Shaofeng Zhang

sherrylone.github.io



Contents

- Introduction
- Autoregressive Models
- Variational Autoencoder
- Generative Adversarial Net
- Diffusion Models



Contents

- Introduction
- Autoregressive Models
- Variational Autoencoder
- **Generative Adversarial Net**
- Diffusion Models



Generative Models So Far

Autoregressive Models directly maximize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta\left(x_i | x_1, \ldots, x_{i-1}\right)$$

Variational Autoencoders introduce a latent z, and maximize a lower bound:

$$p_\theta(x) = \int p_\theta(x|z)p(z)dz \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

Generative Models So Far

Autoregressive Models directly maximize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta\left(x_i | x_1, \ldots, x_{i-1}\right)$$

Variational Autoencoders introduce a latent z, and maximize a lower bound:

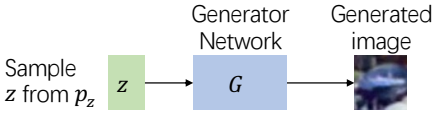
$$p_\theta(x) = \int p_\theta(x|z)p(z)dz \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x), p(z))$$

Generative Adversarial Networks give up on modeling $p(x)$, but allow us to draw samples from $p(x)$

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian) Sample $z \sim p(z)$ and pass to a Generator Network $x = G(z)$ Then x is a sample from the Generator distribution p_G. Want $p_G = p_{data}$!

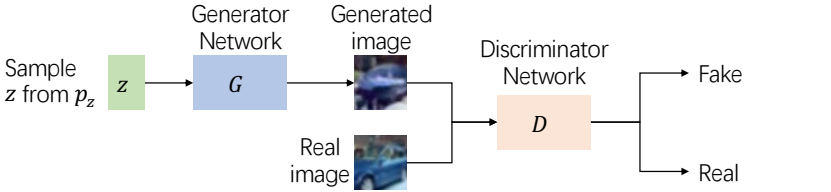


Train Generator Network G to convert z into fake data x sampled from p_G

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian) Sample $z \sim p(z)$ and pass to a Generator Network $x = G(z)$ Then x is a sample from the Generator distribution p_G. Want $p_G = p_{data}$!



Train Generator Network G to convert z into fake data x sampled from p_G By fooling the discriminator D.

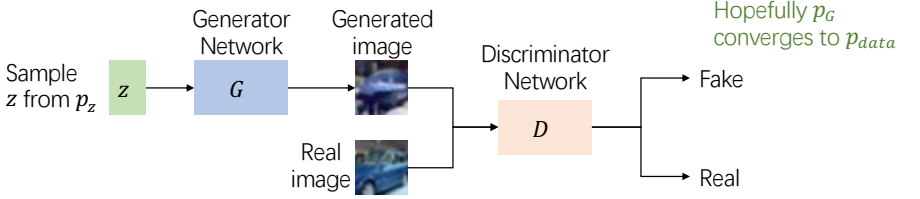
Train Discriminator Network D to classify data as real or fake

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior $p(z)$ (e.g. unit Gaussian) Sample $z \sim p(z)$ and pass to a Generator Network $x = G(z)$ Then x is a sample from the Generator distribution p_G. Want $p_G = p_{data}$!

Jointly train G and D.
Hopefully p_G
converges to p_{data}



Train Generator Network G to convert z into fake data x sampled from p_G
By fooling the discriminator D.

Train Discriminator Network D to classify data as real or fake

Generative Adversarial Networks: Training Objective

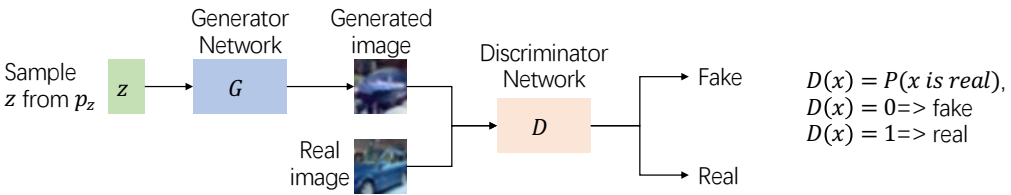
Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x \sim p_{data}}[\log D(x)] + E_{z \sim p(z)}[\log(1 - D(G(z)))])$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$



Generative Adversarial Networks: Training Objective

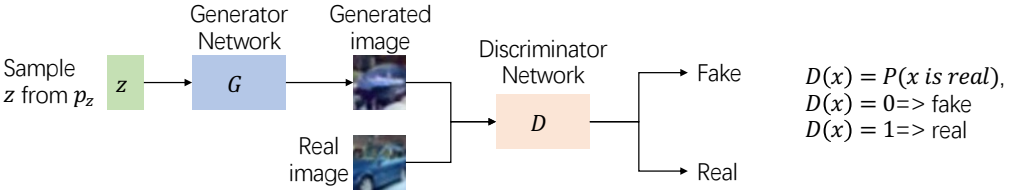
Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D \left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))]\right)$$

Imagine fixing G

Discriminator wants $D(x) = 1$ for real data

Discriminator wants $D(x) = 0$ for fake data



Generative Adversarial Networks: Training Objective

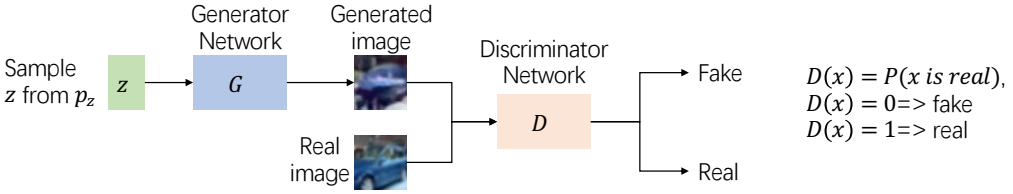
Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x \sim p_{data}}[\log D(x)] + E_{z \sim p(z)}[\log(1 - D(G(z)))])$$

Imagine fixing D

This term does not depend on G

Generator wants $D(x) = 1$ for fake data



Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$
$$= \min_G \max_D V(G, D),$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$
$$= \min_G \max_D V(G, D),$$

While True:

$$D = D + \alpha_D \frac{dV}{dD}$$

$$G = G + \alpha_G \frac{dV}{dG}$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$
$$= \min_G \max_D V(G, D),$$

We are not minimizing any overall loss! No training curves to look at!

While True:

$$D = D + \alpha_D \frac{dV}{dD}$$

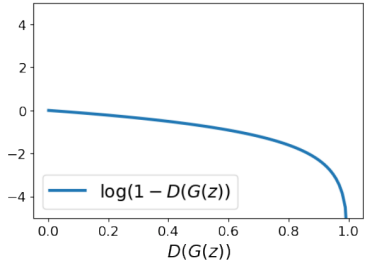
$$G = G + \alpha_G \frac{dV}{dG}$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so $D(G(z))$ close to 0



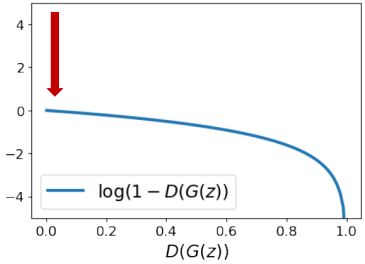
Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so $D(G(z))$ close to 0

Problem: Gradient for G are close to 0



Generative Adversarial Networks: Training Objective

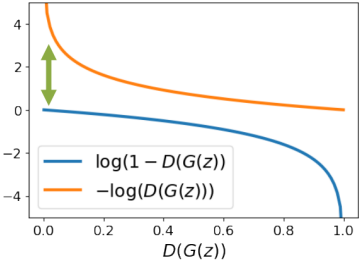
Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so $D(G(z))$ close to 0

Problem: Gradient for G are close to 0

Solution: Generator wants $D(G(z)) = 1$. Train generator to minimize $-\log(D(G(z)))$ and discriminator to maximize $\log(1 - D(G(z)))$ so generator gets strong gradients at start



Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

Why is this a good objective?

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

Outer objective is then minimized by

$$p_G(x) = p_{data}(x)$$

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_G \max_D (E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))])$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

Outer objective is then minimized by

$$p_G(x) = p_{data}(x)$$

Caveats:

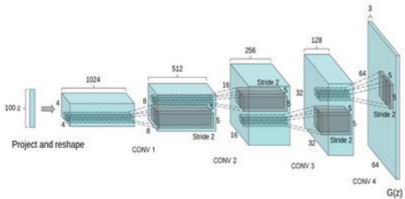
1. Neural nets with fixed capacity may not be able to represent optimal D and G
2. This tells us nothing about convergence to the solution with finite data

GAN Architectures: DC-GAN

Generator G and discriminator D are both neural networks

Usually CNNs ⋯ GANs fell out of favor before ViT became popular

DC-GAN was the first GAN architecture that worked on non-toy data





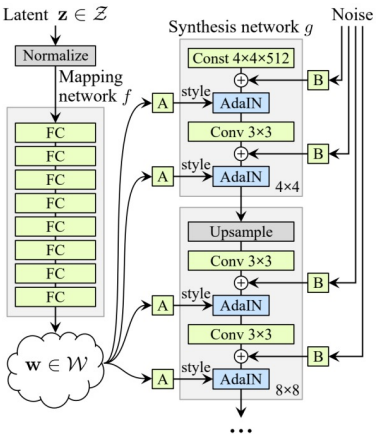
GAN Architectures: StyleGAN

Generator G and discriminator D are both neural networks.

StyleGAN uses a more complex architecture that injects noise via **adaptive normalization**.

At each layer predict a scale w and shift b the same shape as x:

$$AdaIN(x, w, b)_i = w_i \frac{x_i - u(x)}{\sigma(x)} + b_i$$



GANs: Latent Space Interpolation

Latent space is **smooth**.

Given latent vectors z_0 and z_1, we can **interpolate** between them:

$$z_t = tz_0 + (1-t)z_1$$
$$x_t = G(z_t)$$

The resulting image x_t smoothly interpolate between samples!



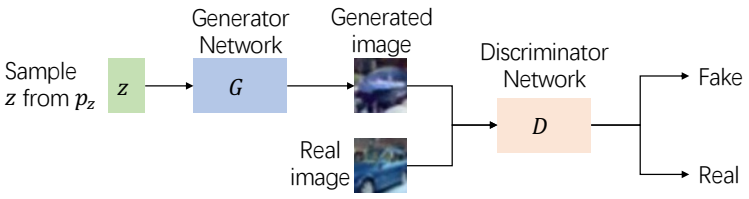
Karras et al, "Alias-Free Generative Adversarial Networks", NeurIPS 2021

Generative Adversarial Networks: Summary

Jointly train Generator and Discriminator with a minimax game

Pros:
- Simple formulation
- Very good image quality

Cons:
- No loss curve to look at
- Unstable Training
- Hard to scale to big models + data



These were the go-to generative models from ~2016 – 2021

Contents

- Introduction
- Autoregressive Models
- Variational Autoencoder
- Generative Adversarial Net
- **Diffusion Models**



Diffusion Models

Warning: Terminology and notation in this area is a mess!

There are many different mathematical formalisms; tons of variance in terminology and notation between papers.

We'll just cover the basics of a modern "clean" implementation (Rectified Flow)

Diffusion Models: Intuition

Pick a noise distribution $z \sim p_{noise}$
(Usually unit Gaussian)

Diffusion Models: Intuition

Pick a noise distribution $z \sim p_{noise}$
(Usually unit Gaussian)

Consider data x corrupted under varying **noise levels** t to give noisy data x_t



$t = 0$
No noise

$t = 1$
Full noise

Diffusion Models: Intuition

Pick a noise distribution $z \sim p_{noise}$
(Usually unit Gaussian)

Consider data x corrupted under varying **noise levels** t to give noisy data x_t

Train a neural network to remove a little bit of noise: $f_\theta(x_t, t)$

At inference time, sample $x_1 \sim p_{noise}$ and apply f_θ many times in sequence to generate a noiseless sample x_0

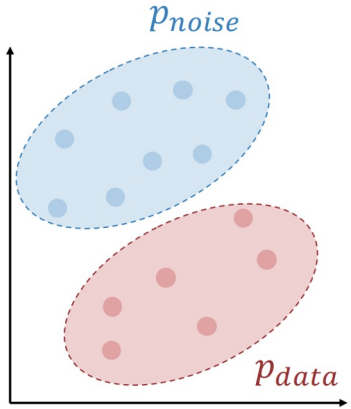


$t = 0$
No noise

$t = 1$
Full noise

Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise} (e.g. Gaussian) and samples from p_{data}



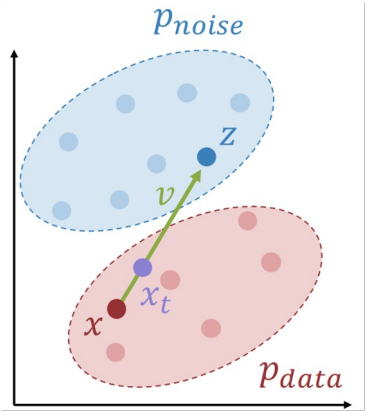
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise} (e.g. Gaussian) and samples from p_{data}

On each training iteration, sample:
$z \sim p_{noise}$, $x \sim p_{data}$, $t \sim Uniform[0,1]$

Set $x_t = (1-t)x + tz$, $v = z - x$



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

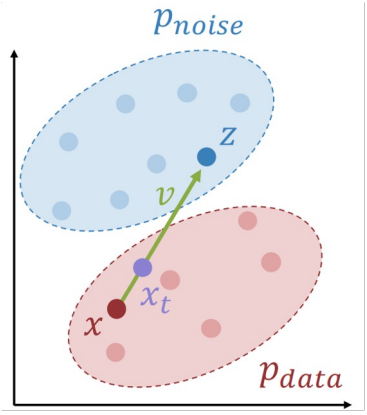
Diffusion Models: Rectified Flow

Suppose we have a simple p_{noise} (e.g. Gaussian) and samples from p_{data}

On each training iteration, sample:
$z \sim p_{noise}$, $x \sim p_{data}$, $t \sim Uniform[0,1]$

Set $x_t = (1-t)x + tz$, $v = z - x$

Train a neural network to predict v:
$$L = ||f_\theta(x_t, t) - v||_2^2$$

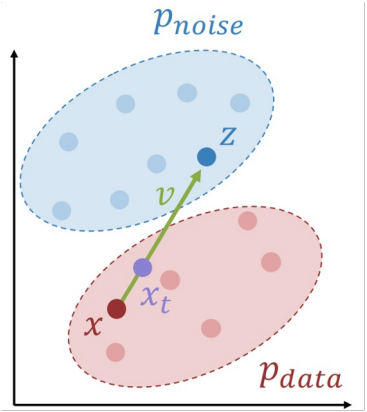


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Rectified Flow

Core training loop is just a few lines of code!

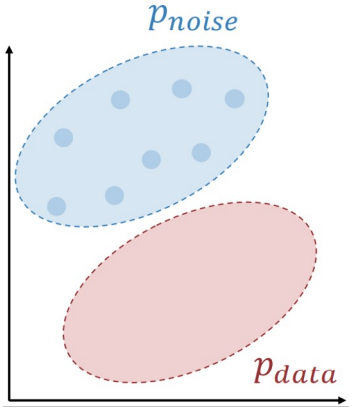
```
for x in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, t)
    loss = (z - x - v).square().sum()
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

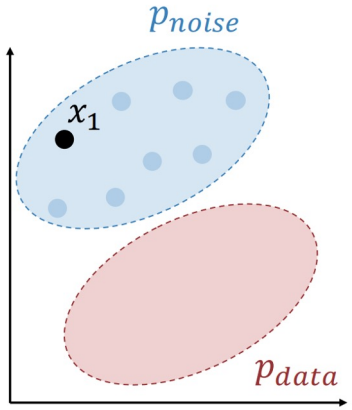


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$



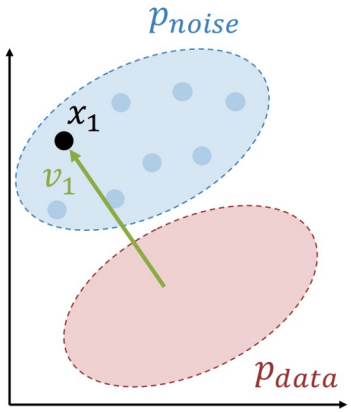
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$



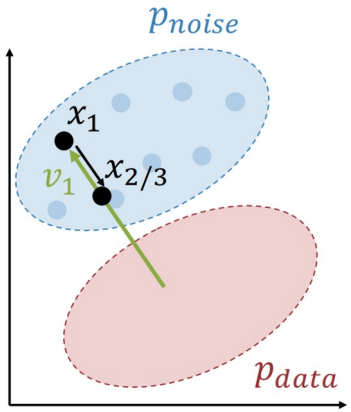
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$



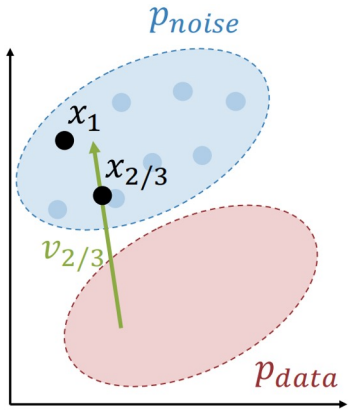
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$



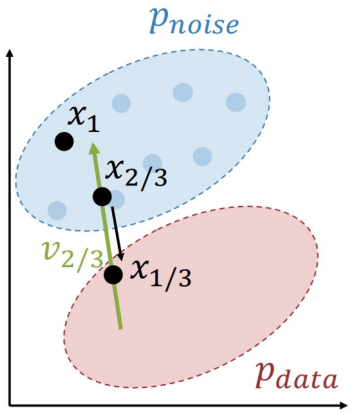
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$



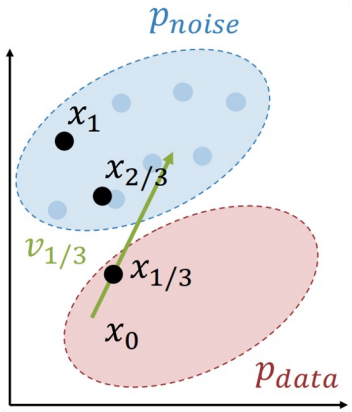
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$



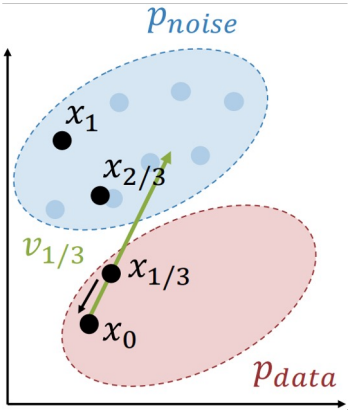
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$



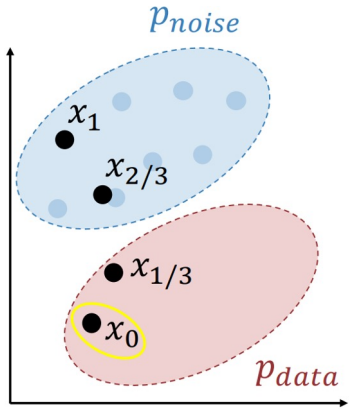
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$
Return x



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

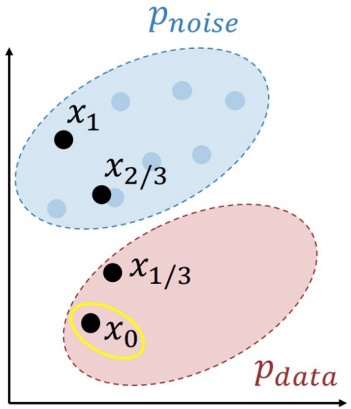
Diffusion Models: Sampling

Choose number of steps T (often $T = 50$)

Sample $x \sim p_{noise}$

For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
  Evaluate $v_t = f_\theta(x_t, t)$
  Step $x = x - \frac{v_t}{T}$
Return x

```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

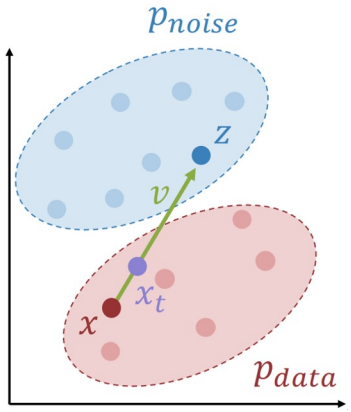
Rectified Flow: Summary

Training

```
for x in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    v = model(xt, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

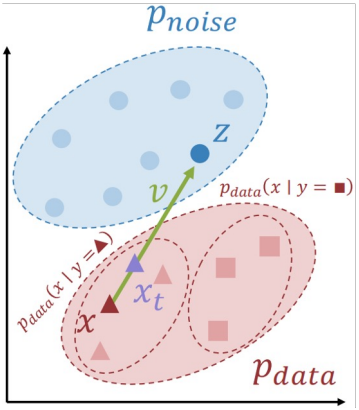
Conditional Rectified Flow

Training

```
for x in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    v = model(xt, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

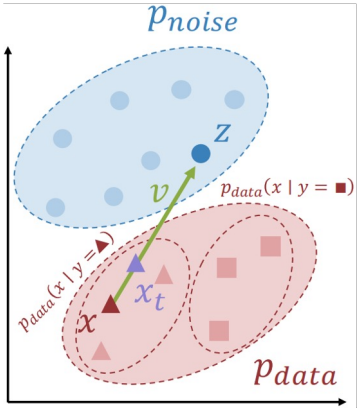
Conditional Rectified Flow

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

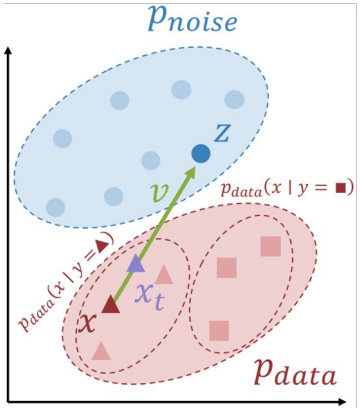
Conditional Rectified Flow

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, y, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

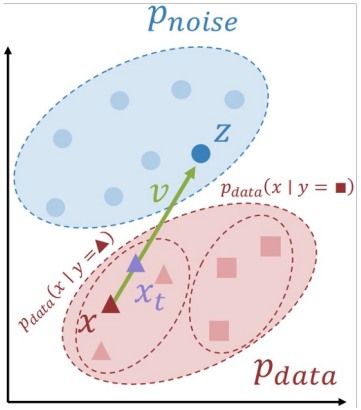
Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y

Randomly drop y during training.

Now the same model is conditional and unconditional!



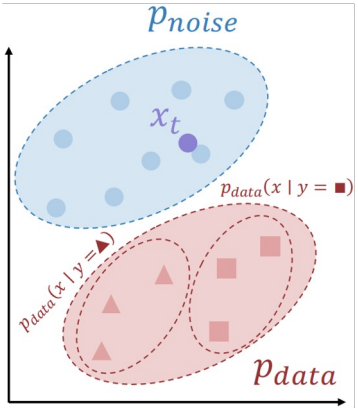
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y



Randomly drop y during training.

Now the same model is conditional and unconditional!
Consider a noisy x_t:

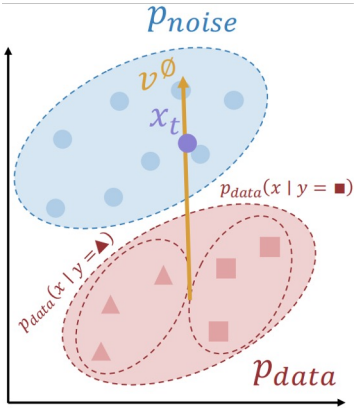
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y



Randomly drop y during training.

Now the same model is conditional and unconditional!
Consider a noisy x_t:

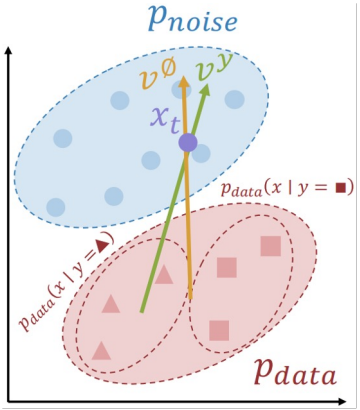
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y



Randomly drop y during training.

Now the same model is conditional and unconditional!
Consider a noisy x_t:
$v^{\phi} = f_\theta(x, y_\phi, t)$ points toward $p(x)$.
$v^{y} = f_\theta(x_t, y, t)$ points toward $p(x|y)$.

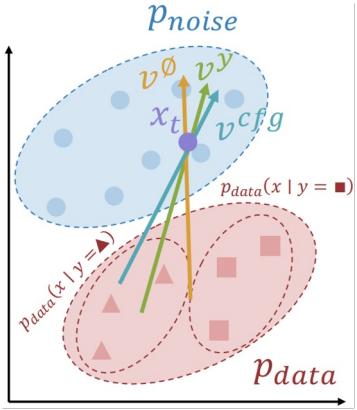
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y



Randomly drop y during training.

Now the same model is conditional and unconditional!
Consider a noisy x_t:
$v^{\phi} = f_\theta(x, y_\phi, t)$ points toward $p(x)$.
$v^{y} = f_\theta(x_t, y, t)$ points toward $p(x|y)$.
$v^{cfg} = (1+w)v^{y} - wv^{\phi}$ points more toward $p(x|y)$

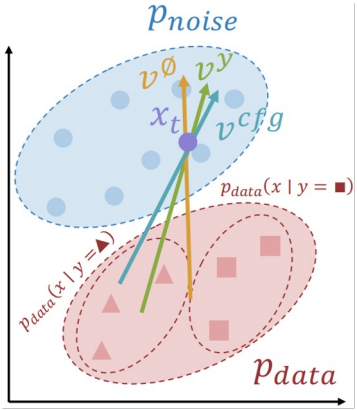
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y



Randomly drop y during training.

Now the same model is conditional and unconditional!
Consider a noisy x_t:
$v^{\phi} = f_\theta(x, y_\phi, t)$ points toward $p(x)$.
$v^{y} = f_\theta(x_t, y, t)$ points toward $p(x|y)$.
$v^{cfg} = (1+w)v^{y} - wv^{\phi}$ points more toward $p(x|y)$
During sampling, step according to v^{cfg}

Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

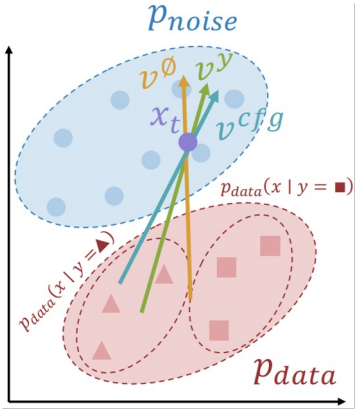
Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Can we control how much we "emphasize" the conditioning y



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

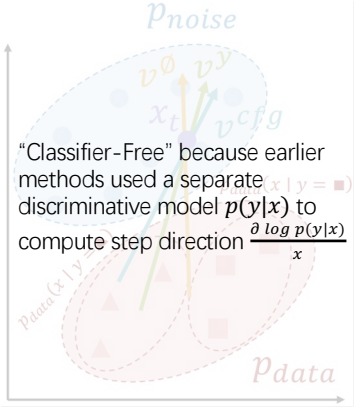
Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Can we control how much we "emphasize" the conditioning y



"Classifier-Free" because earlier methods used a separate discriminative model $p(y|x)$ to compute step direction $\frac{\partial \log p(y|x)}{x}$

Dhariwal and Nichol, "Diffusion Models beat GANs on Image Synthesis", arXiv 2021
Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Classifier-Free Guidance (CFG)

Training

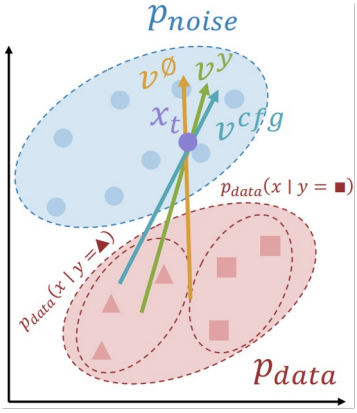
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Can we control how much we "emphasize" the conditioning y

Used everywhere in practice! Very important for high-quality outputs



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Classifier-Free Guidance (CFG)

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

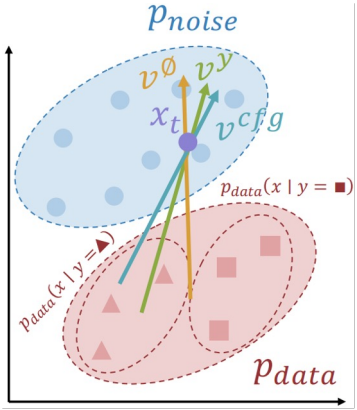
Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Can we control how much we "emphasize" the conditioning y

Used everywhere in practice! Very important for high-quality outputs

Doubles the cost of sampling…



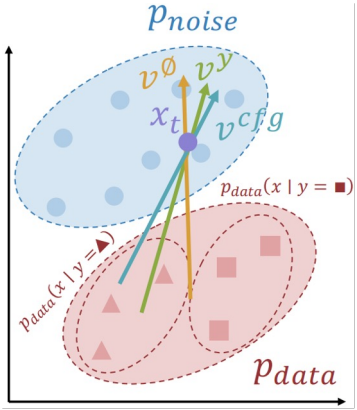
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



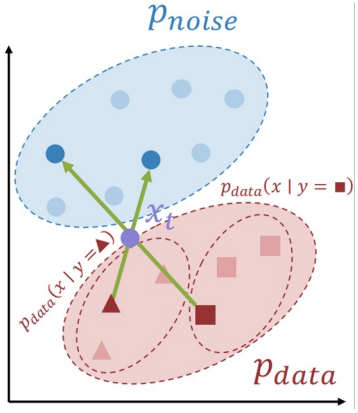
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same x_t; network must average over them

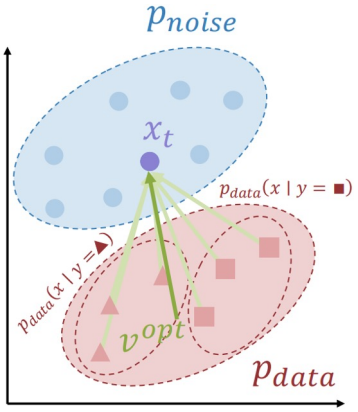
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}

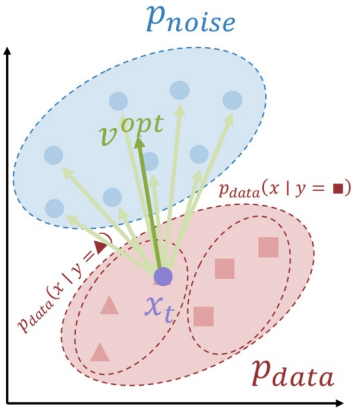
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}

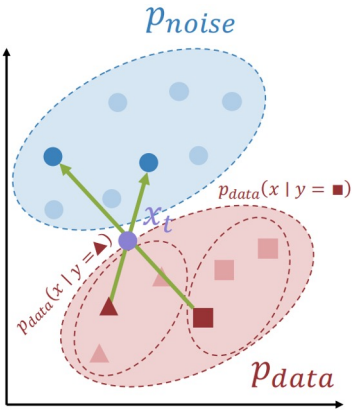
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous

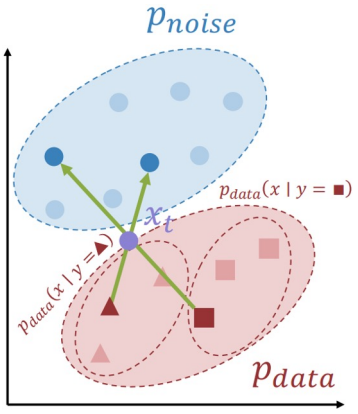
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!

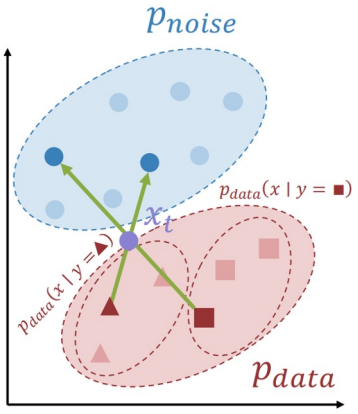
Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Optimal Prediction

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?



There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule

Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

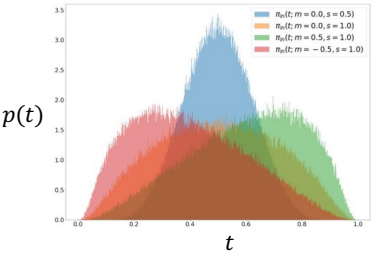
Noise Schedules

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Noise Schedules

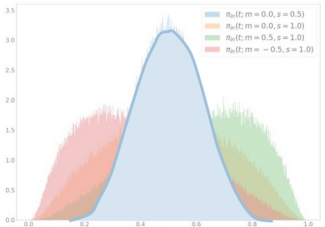
Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.randn(()).sigmoid()
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule

$p(t)$



t

Put more emphasis on middle noise
Common choice: **logit-normal sampling**

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

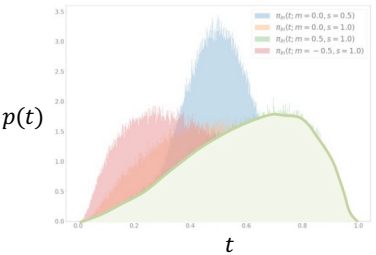
Noise Schedules

Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.randn(()).sigmoid()
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise ($t = 1$) is easy: optimal v is mean of p_{data}
Full noise ($t = 0$) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise
Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Diffusion: Rectified Flow

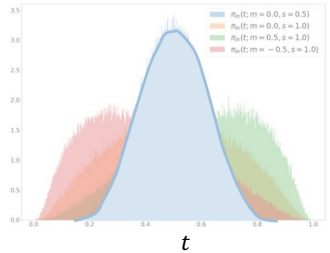
Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.randn(()).sigmoid()
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Simple and scalable setup for many generative modeling problems!

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```



Put more emphasis on middle noise
Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Diffusion: Rectified Flow

Training

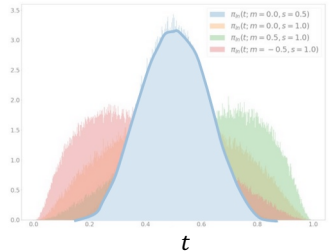
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.randn(()).sigmoid()
    xt = (1-t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Simple and scalable setup for many generative modeling problems!

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Problem: Does not work naively on high-resolution data



Put more emphasis on middle noise
Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = (1-t)x + tz$
Set $v_{gt} = z - x$
Compute $v_{pred} = f_\theta(x_t, t)$
Compute loss $||v_{gt} - v_{pred}||_2^2$

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = (1-t)x + tz$
Set $v_{gt} = z - x$
Compute $v_{pred} = f_\theta(x_t, t)$
Compute loss $||v_{gt} - v_{pred}||_2^2$



Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = a(t)x + b(t)z$
Set $y_{gt} = c(t)x - d(t)z$
Compute $y_{pred} = f_\theta(x_t, t)$
Compute loss $||y_{gt} - y_{pred}||_2^2$

Generalized Diffusion

Variance Preserving (VP)

$a(t) = \sqrt{\sigma(t)}$
$b(t) = \sqrt{1 - \sigma(t)}$

If x and z are independent and variance=1, then x_t also has variance=1

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = a(t)x + b(t)z$
Set $y_{gt} = c(t)x - d(t)z$
Compute $y_{pred} = f_\theta(x_t, t)$
Compute loss $||y_{gt} - y_{pred}||_2^2$

Generalized Diffusion

Variance Exploding (VE)

$a(t) = 1$
$b(t) = \sigma(t)$

σ_t needs to be big enough to drown out all signal in x.

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = a(t)x + b(t)z$
Set $y_{gt} = c(t)x - d(t)z$
Compute $y_{pred} = f_\theta(x_t, t)$
Compute loss $||y_{gt} - y_{pred}||_2^2$

Generalized Diffusion

x-prediction

$y_{gt} = x \quad [c(t) = 1, d(t) = 0]$

ϵ-Prediction

$y_{gt} = z \quad [c(t) = 0, d(t) = 1]$

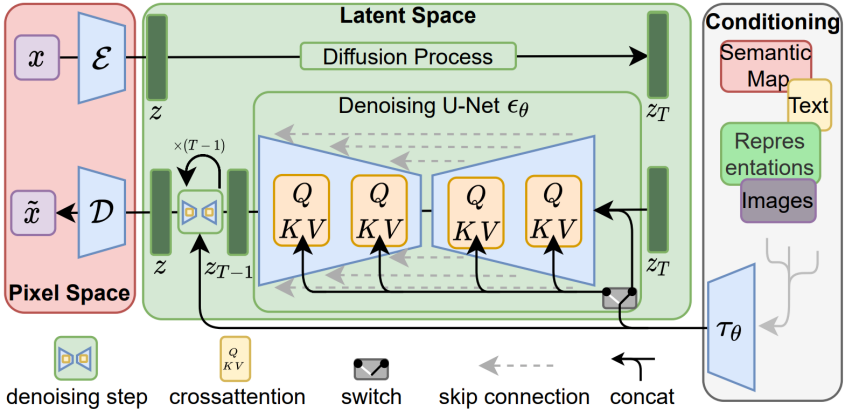
v-Prediction

$y_{gt} = b(t)z - a(t)x \quad [c(t) = b(t), d(t) = -a(t)]$

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = a(t)x + b(t)z$
Set $y_{gt} = c(t)x - d(t)z$
Compute $y_{pred} = f_\theta(x_t, t)$
Compute loss $||y_{gt} - y_{pred}||_2^2$

Advanced Diffusion Models: LDM



Advanced Diffusion Models: LDM

CelebA-HQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
DC-VAE [63]	15.8	-	-
VQGAN+T. [23] (k=400)	10.2	-	-
PGGAN [39]	8.0	-	-
LSGM [93]	7.22	-	-
UDM [43]	7.16	-	-
LDM-4 (ours, 500-s†)	**5.11**	0.72	0.49

FFHQ 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
ImageBART [21]	9.57	-	-
U-Net GAN (+aug) [77]	10.9 (7.6)	-	-
UDM [43]	5.54	-	-
StyleGAN [41]	4.16	0.71	0.46
ProjectedGAN [76]	**3.08**	0.65	0.46
LDM-4 (ours, 200-s)	4.98	**0.73**	**0.50**

LSUN-Churches 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
DDPM [30]	7.89	-	-
ImageBART [21]	7.32	-	-
PGGAN [39]	6.42	-	-
StyleGAN [41]	4.21	-	-
StyleGAN2 [42]	3.86	-	-
ProjectedGAN [76]	**1.59**	0.61	0.44
*LDM-8** (ours, 200-s)	4.02	**0.64**	**0.52**

LSUN-Bedrooms 256 × 256			
Method	FID ↓	Prec. ↑	Recall ↑
ImageBART [21]	5.51	-	-
DDPM [30]	4.9	-	-
UDM [43]	4.57	-	-
StyleGAN [41]	2.35	0.59	0.48
ADM [15]	1.90	**0.66**	**0.51**
ProjectedGAN [76]	**1.52**	0.61	0.34
LDM-4 (ours, 200-s)	2.95	**0.66**	0.48

Advanced Diffusion Models: LDM

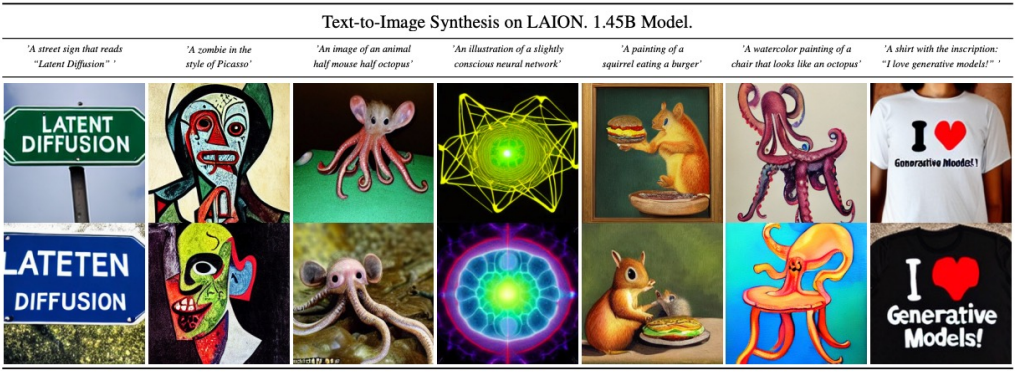


Figure 5. Samples for user-defined text prompts from our model for text-to-image synthesis, *LDM-8 (KL)*, which was trained on the LAION [78] database. Samples generated with 200 DDIM steps and $\eta = 1.0$. We use unconditional guidance [32] with $s = 10.0$.

Advanced Diffusion Models: DiT

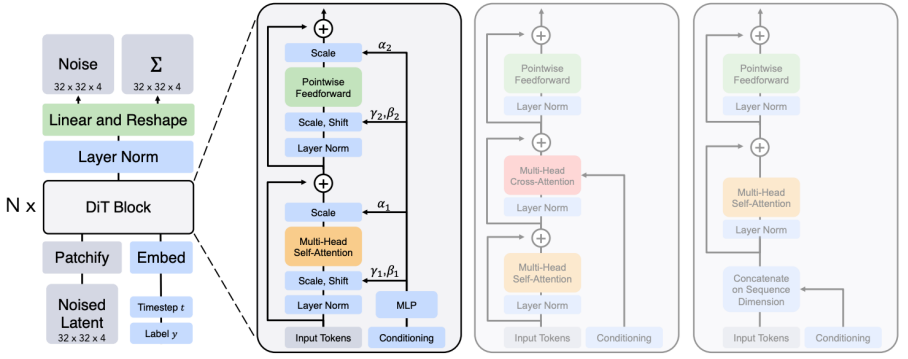
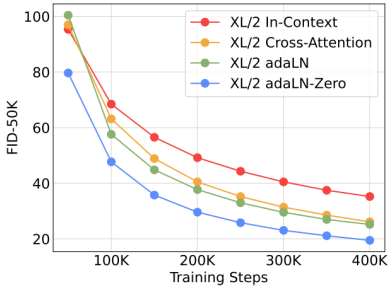


Figure 3. **The Diffusion Transformer (DiT) architecture.** *Left:* We train conditional latent DiT models. The input latent is decomposed into patches and processed by several DiT blocks. *Right:* Details of our DiT blocks. We experiment with variants of standard transformer blocks that incorporate conditioning via adaptive layer norm, cross-attention and extra input tokens. Adaptive layer norm works best.

Advanced Diffusion Models: DiT



Class-Conditional ImageNet 256×256					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [2]	6.95	7.36	171.4	0.87	0.28
StyleGAN-XL [53]	2.30	4.02	265.12	0.78	0.53
ADM [9]	10.94	6.02	100.98	0.69	0.63
ADM-U	7.49	5.13	127.49	0.72	0.63
ADM-G	4.59	5.25	186.70	0.82	0.52
ADM-G, ADM-U	3.94	6.14	215.84	0.83	0.53
CDM [20]	4.88	-	158.71	-	-
LDM-8 [48]	15.51	-	79.03	0.65	0.63
LDM-8-G	7.76	-	209.52	0.84	0.35
LDM-4	10.56	-	103.49	0.71	0.62
LDM-4-G (cfg=1.25)	3.95	-	178.22	0.81	0.55
LDM-4-G (cfg=1.50)	3.60	-	247.67	**0.87**	0.48
DiT-XL/2	9.62	6.85	121.50	0.67	**0.67**
DiT-XL/2-G (cfg=1.25)	3.22	5.28	201.77	0.76	0.62
DiT-XL/2-G (cfg=1.50)	**2.27**	**4.60**	**278.24**	0.83	0.57

Table 2. **Benchmarking class-conditional image generation on ImageNet 256×256.** DiT-XL/2 achieves state-of-the-art FID.

Advanced Diffusion Models: REPA

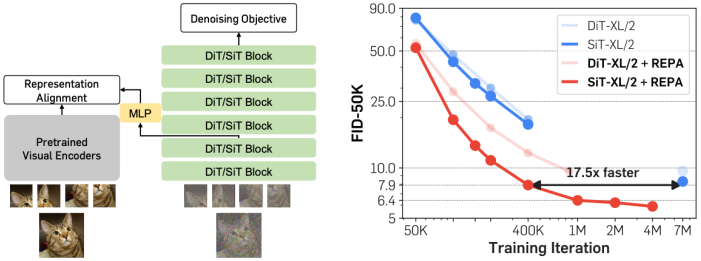
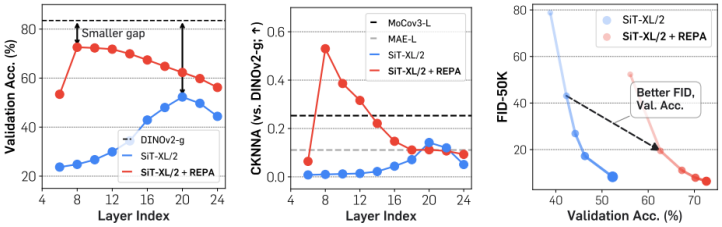


Figure 1: **Representation alignment makes diffusion transformer training significantly easier.** Our framework, REPA, explicitly aligns the diffusion model representation with powerful pretrained visual representation through a simple regularization. Notably, model training becomes significantly more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

Advanced Diffusion Models: REPA



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

Figure 3: **Bridging the representation gap:** (a) Our method, REPA significantly reduces the "semantic gap" between diffusion transformers and DINOv2, as demonstrated by the linear probing results on ImageNet classification. (b) With REPA, the alignment between diffusion transformers and DINOv2 improves substantially, even after just a few (*e.g.*, 8) layers. (c) Notably, with improved alignment, we can push the SiT model's generation-representation envelope: within the same number of training iterations, it delivers both better generation quality and stronger linear probing results. We use a single network trained with REPA at layer 8 and perform the evaluation at different layers.

Advanced Diffusion Models: REPA

Table 2: **Component-wise analysis** on ImageNet 256×256. All models are SiT-L/2 trained for 400K iterations. All metrics except accuracy (Acc.) are measured with the SDE Euler-Maruyama sampler with NFE=250 and without classifier-free guidance. For Acc., we report linear probing results on the ImageNet validation set using the latent features aligned with the target representation. We fix $\lambda = 0.5$ here. ↓ and ↑ indicate whether lower or higher values are better, respectively.

Iter.	Target Repr.	Depth	Objective	FID↓	sFID↓	IS↑	Pre.↑	Rec.↑	Acc.↑
400K	Vanilla SiT-L/2 (Ma et al., 2024a)			18.8	5.29	72.0	0.64	0.64	N/A
400K	MAE-L	8	NT-Xent	12.5	4.89	90.7	0.68	0.63	57.3
400K	DINO-B	8	NT-Xent	11.9	5.00	92.9	0.68	0.63	59.3
400K	MoCov3-L	8	NT-Xent	11.9	5.06	92.2	0.68	0.64	63.0
400K	I-JEPA-H	8	NT-Xent	11.6	5.21	98.0	0.68	0.64	62.1
400K	CLIP-L	8	NT-Xent	11.0	5.25	100.4	0.67	0.66	67.2
400K	SigLIP-L	8	NT-Xent	10.2	5.15	107.0	0.69	0.64	68.8
400K	DINOv2-L	8	NT-Xent	10.0	5.09	106.6	0.68	0.65	68.1
400K	DINOv2-B	8	NT-Xent	9.7	5.13	107.5	0.69	0.64	65.7
400K	DINOv2-L	8	NT-Xent	10.0	5.09	106.6	0.68	0.65	68.1
400K	DINOv2-g	8	NT-Xent	9.8	5.22	108.9	0.69	0.64	65.7
400K	DINOv2-L	6	NT-Xent	10.3	5.23	106.5	0.69	0.65	66.2
400K	DINOv2-L	8	NT-Xent	10.0	5.09	106.6	0.68	0.65	68.1
400K	DINOv2-L	10	NT-Xent	10.5	5.50	105.0	0.68	0.65	68.6
400K	DINOv2-L	12	NT-Xent	11.2	5.14	100.2	0.68	0.64	69.4
400K	DINOv2-L	14	NT-Xent	11.6	5.61	99.5	0.67	0.65	70.0
400K	DINOv2-L	16	NT-Xent	12.1	5.34	96.1	0.67	0.64	71.1
400K	DINOv2-L	8	NT-Xent	10.0	5.09	106.6	0.68	0.65	68.1
400K	DINOv2-L	8	Cos. sim.	9.9	5.34	111.9	0.68	0.65	68.2

Advanced Diffusion Models: CogVideoX

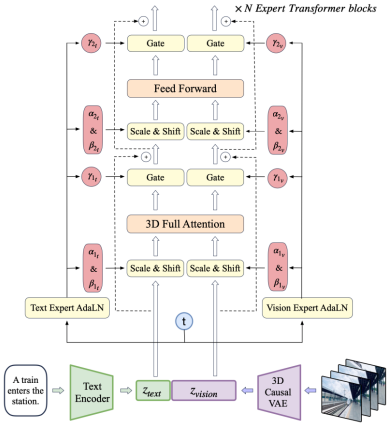


Figure 3: **The overall architecture of CogVideoX.**

Advanced Diffusion Models: CogVideoX

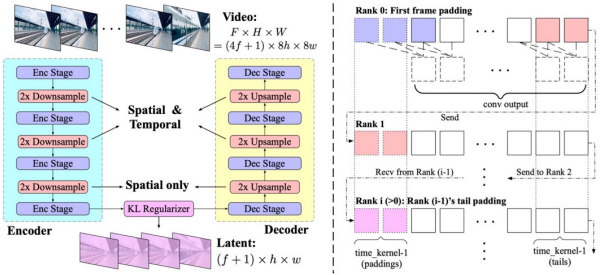


Figure 4: (a) The structure of the 3D VAE in CogVideoX. It comprises an encoder, a decoder and a latent space regularizer, achieving a 8×8×4 compression from pixels to the latents. (b) The context parallel implementation on the temporally causal convolution.

Advanced Diffusion Models: CogVideoX

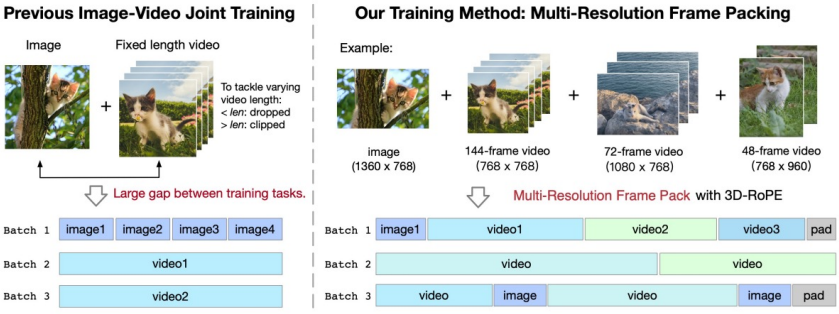


Figure 6: The diagram of mixed-duration training and Frame Pack. To fully utilize the data and enhance the model's generalization capability, we train on videos of different duration within the same batch.

Advanced Diffusion Models: CogVideoX

Table 3: Evaluation results of CogVideoX-5B and CogVideoX-2B.

Models	Human Action	Scene	Dynamic Degree	Multiple Objects	Appear. Style	Dynamic Quality	GPT4o-MT Score
T2V-Turbo(Li et al., 2024)	95.2	**55.58**	49.17	54.65	24.42	–	–
AnimateDiffGuo et al. (2023)	92.6	50.19	40.83	36.88	22.42	–	2.62
VideoCrafter-2.0(Chen et al., 2024a)	95.0	55.29	42.50	40.66	**25.13**	43.6	2.68
OpenSora V1.2(Zheng et al., 2024b)	85.8	42.47	47.22	58.41	23.89	63.7	2.52
Show-1(Zhang et al., 2023a)	95.6	47.03	44.44	45.47	23.06	57.7	–
Gen-2(runway, 2023)	89.2	48.91	18.89	55.47	19.34	43.6	2.62
Pika(pik, 2023)	88.0	44.80	37.22	46.69	21.89	52.1	2.48
LaVie-2(Wang et al., 2023b)	96.4	49.59	31.11	64.88	25.09	–	2.46
CogVideoX-2B	96.6	55.35	**66.39**	57.68	24.37	57.7	3.09
CogVideoX-5B	**96.8**	55.44	62.22	**70.95**	24.44	**69.5**	**3.36**

Advanced Diffusion Models: VideoREPA

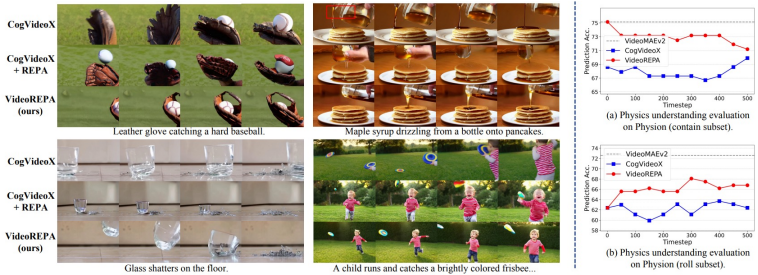


Figure 1: *Left:* Visual comparison of video generation results from CogVideoX [55] (baseline), CogVideoX finetuned with REPA [57], and our proposed VideoREPA. Red rectangles denote phenomena that violate physical commonsense for easier distinguish. **Our VideoREPA generates videos that most closely adhere to real-world physical laws.** *Right:* Evaluation of physics understanding on the Object Contact Prediction (OCP) task within the Physion benchmark [5]. The plots illustrate a significant gap in physics understanding between the SSL video encoder VideoMAEv2 and the T2V model CogVideoX. **The proposed VideoREPA substantially narrows this understanding gap.**

Advanced Diffusion Models: VideoREPA

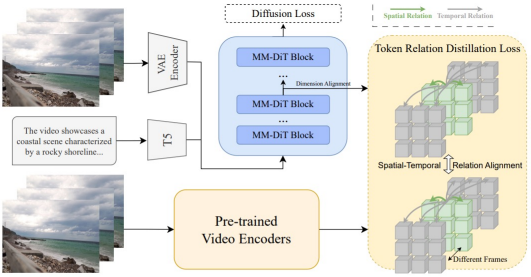


Figure 2: Overview of VideoREPA. Our VideoREPA enhances physics in T2V models by distilling physics knowledge from pre-trained SSL video encoders. We apply Token Relation Distillation (TRD) loss to align pairwise token similarities between video SSL representations and intermediate features in diffusion transformer blocks. Within each representation, tokens form spatial relations with other tokens in the same latent frame and temporal relations with tokens in other latent frames.

Advanced Diffusion Models: VideoREPA

Table 1: Results of Videophy. † denotes the results reported from WISA [45] and ∗ denotes detailed prompt input, see Section 4.2. Semantic Adherence (SA) measures the video-text alignment and fidelity. **Importantly**, Physical Commensense (PC) measures whether generated videos follow the physics laws in the real-world.

Methods	Solid-Solid		Solid-Fluid		Fluid-Fluid		Overall	
	SA	PC	SA	PC	SA	PC	SA	PC
VideoCrafter2	50.4	32.2	50.7	27.4	48.1	29.1	50.3	29.7
DreamMachine	55.1	21.7	59.6	23.3	58.2	18.2	57.5	21.8
LaVIE	40.8	18.3	48.6	37.0	69.1	50.9	48.7	31.5
Cosmos-Diffusion-7B†	-	-	-	-	-	-	57	18
HunyuanVideo*	55.2	16.1	67.1	30.1	54.5	54.5	60.2	28.2
PhyT2V†	-	-	-	-	-	-	61	37
WISA (Koala dataset)†	-	-	-	-	-	-	62	33
WISA (WISA dataset)†	-	-	-	-	-	-	67	38
CogVideoX-2B	37.8	12.6	67.1	30.1	45.5	50.9	51.6	26.2
CogVideoX-2B*	49.6	13.3	71.2	28.1	60.0	50.9	60.5	25.6
VideoREPA-2B*	52.4	18.2	77.4	32.2	60.0	52.7	64.2	29.7
CogVideoX-5B†	-	-	-	-	-	-	60	33
CogVideoX-5B	53.1	18.2	75.3	32.9	56.4	61.8	63.1	31.4
CogVideoX-5B*	**62.9**	19.6	76.0	33.6	72.7	61.8	70.0	32.3
VideoREPA-5B*	58.0	**28.0**	**82.9**	**39.0**	**80.0**	**74.5**	**72.1**	**40.1**

Advanced Diffusion Models: WorldForge

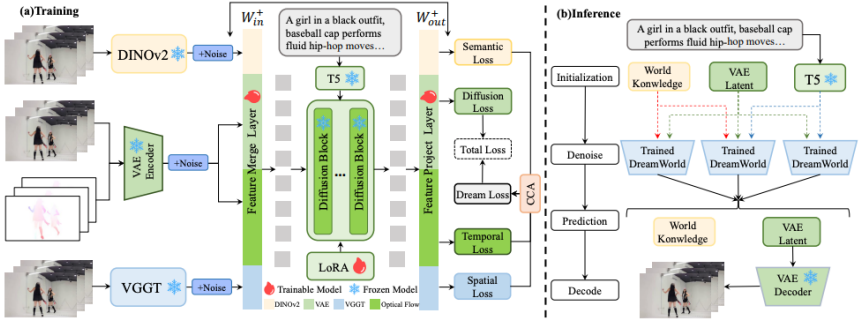


Figure 2. **Overview of DreamWorld. (a) Training:** Expert Models are first employed to extract multimodal features, which are then noise-added and concatenated before being fused through a linear layer $\mathbf{W}_{\mathbf{in}}^{+}$. The resulting prediction is mapped via another linear layer $\mathbf{W}_{\mathbf{out}}^{+}$ to jointly predict appearance and world knowledge, with a Dream Loss constrained by *Consistency Constraint Annealing (CCA)* to ensure generation fidelity. **(b) Inference:** We introduce *Multi-Source Inner-Guidance*, a mechanism that leverages inherent noise features to direct the final video generation process.

Advanced Diffusion Models: WorldForge

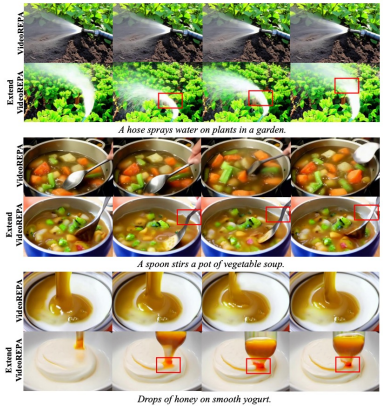


Figure 1. Limitations of extending VideoREPA to multi-source knowledge, leading to structural implausibility and unnatural distortion. The physics score(PC) on the Videophy benchmark dropped from **29.7** to **24.1**.

Advanced Diffusion Models: WorldForge

Table 1. **Quantitative comparison on VBench.** DreamWorld demonstrates significant improvements over baselines, particularly in temporal dynamics, semantic understanding and spatial relationships, achieving the best performance across all summary scores. **FT** denotes the fine-tuned version, and **Reimpl.** indicates our re-implementation of the method.

Method	Temporal					Semantic					Spatial	Summary		Overall Score
	Subject Consistency	Background Consistency	Temporal Flickering	Motion Smoothness	Dynamic Degree	Object Class	Color	Human Action	Multiple Objects	Scene	Relationship	Quality Score	Semantic Score	
Wan2.1-T2V-1.3B	91.83	94.71	99.17	96.51	65.00	76.09	89.93	74.60	53.66	20.03	62.37	79.81	65.43	76.93
Wan2.1-T2V-1.3B(FT)	93.59	95.81	**99.36**	97.21	54.08	79.90	88.57	78.98	58.20	28.55	63.31	81.26	68.47	78.71
VideoJAM(Reimpl.)	91.51	**96.01**	99.13	96.05	73.88	79.22	88.92	79.20	59.66	28.34	66.17	81.18	69.08	78.76
DreamWorld (Ours)	93.62	94.95	98.81	**98.07**	**79.16**	81.32	**92.61**	**81.20**	**65.03**	**29.71**	70.47	**83.49**	**70.89**	**80.97**

Table 2. **Quantitative comparison on VBench-2.0. Abbreviations: Common.**: Commonsense; **Control.**: Controllability; **Hum. Fid.**: Human Fidelity. The best and second-best results are highlighted in **bold** and underlined.

Method	VBench 2.0 Dimensions					Total
	Creativity	Common.	Control.	Hum. Fid.	Physics	
Wan2.1-T2V-1.3B	45.92	59.17	16.81	76.09	**55.85**	50.77
Wan2.1-T2V-1.3B(FT)	43.13	62.80	**18.41**	77.09	54.51	51.18
VideoJAM	49.32	**64.89**	16.33	78.68	52.42	52.33
DreamWorld (Ours)	**50.89**	61.82	16.95	**80.11**	55.07	**52.97**

Table 3. **Quantitative comparison on VideoPhy benchmark.** We report Semantic Adherence (SA) and Physical Commonsense (PC) scores. The best results are highlighted in **bold**.

Method	Solid-Solid		Solid-Fluid		Fluid-Fluid		Overall	
	SA	PC	SA	PC	SA	PC	SA	PC
Wan2.1-T2V-1.3B	51.1	22.3	45.2	19.1	45.4	23.6	47.7	21.2
Wan2.1-T2V-1.3B(FT)	46.2	18.2	47.8	22.6	50.9	23.6	45.1	20.9
VideoJAM	51.7	29.4	38.4	23.3	47.3	20.1	45.3	25.3
DreamWorld (Ours)	**54.5**	**24.5**	**48.6**	**25.4**	**60.1**	**32.7**	**52.9**	**26.2**

Advanced Diffusion Models: WorldForge

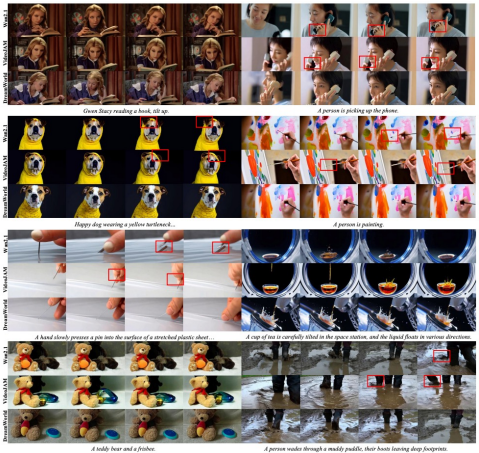


Figure 3. Qualitative comparison of world consistency. DreamWorld outperforms baselines by maintaining semantic realism, spatial integrity, and temporal precision. In contrast, competitor models frequently exhibit geometric penetrations and uncanny distortions

Advanced Diffusion Models: H3-World

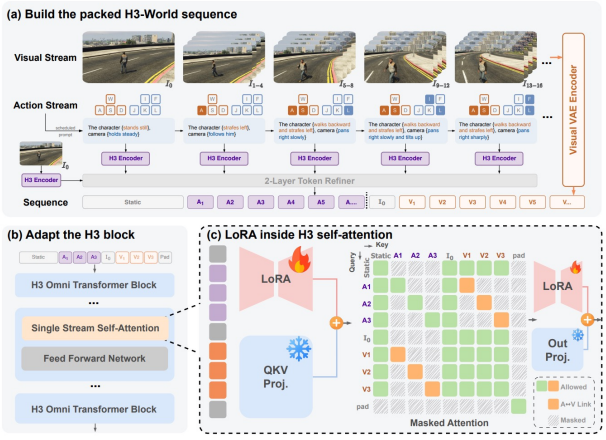


Figure 3. Overview of H3-WORLD. (a) Latent-aligned action prompts are independently encoded and packed with the static semantic condition, initial observation, and video latents. (b) MiniMax-H3 processes the packed sequence through single-stream self-attention. (c) LoRA adapts the attention projections under single-egress routing, which connects each action span directly to its matched video latent and retains bidirectional attention among video latents.

Advanced Diffusion Models: H3-World

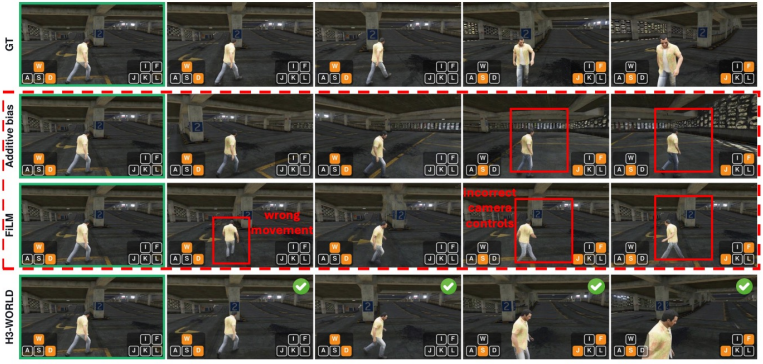
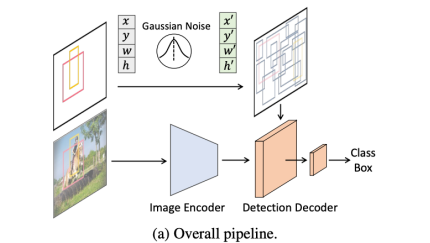


Figure 6. Action-conditioning interfaces on a held-out clip. GT provides the recorded reference motion. Direct additive-bias and FiLM conditioning yield limited or inconsistent responses to the recorded controls, while text-based H3-WORLD produces coordinated character and camera changes.

Advanced Diffusion Models: Other Application



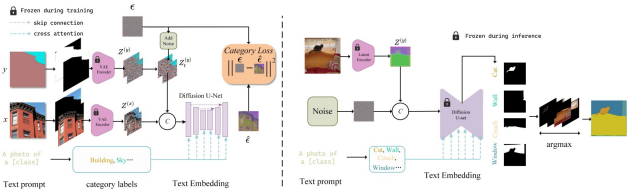


Fig. 2: DiGSeg pipeline overview, which presents the training and inference pipelines of our diffusion-based generation model. In training, paired images are encoded into latent space and, together with text prompts, guide the diffusion U-Net to predict noise using an MSE objective. In inference, noise is sampled and progressively denoised under text conditioning, after which the VAE decoder reconstructs the final image.

Advanced Diffusion Models: Application

Method	VLM	Backbone	Training Dataset	Additional Dataset	A-847	PC-459	A-150	PC-59	Cityscapes
ODISE [99]	CLIP ViT-L/14	Stable Diffusion	COCO-Panoptic	✗	11.0	13.8	28.7	55.3	-
OVSeg [56]	CLIP ViT-L/14	Swin-B	COCO-Stuff	✓	9.0	12.4	29.6	55.7	-
SAN [100]	CLIP ViT-L/14	Side Adapter	COCO-Stuff	✗	12.4	15.7	29.9	51.8	-
SCAN [58]	CLIP ViT-L/14	Swin-B	COCO-Stuff	✗	14.0	16.7	33.5	59.3	-
CAT-Seg [17]	CLIP ViT-L/14	-	COCO-Stuff	✗	16.0	23.8	31.5	62.0	-
MAFT+ [43]	ConvNeXt-L	-	COCO-Stuff	✗	15.1	21.6	36.1	59.4	-
SED [96]	CLIP ConvNeXt-L	-	COCO-Stuff	✗	13.9	22.6	35.2	60.6	-
Mask-Adapter [55]	CLIP ConvNeXt-L	-	COCO-Stuff	✗	16.2	22.7	38.2	60.4	37.9
Seg4Diff [46]	CLIP ViT-L/14	Stable Diffusion	COCO-Stuff	✗	-	-	35.2	51.2	26.0
HyperCLIP [65]	CLIP ViT-L/14	-	COCO-Stuff	✗	16.3	24.1	38.2	64.2	-
OVSNet [59]	CLIP ViT-L/14	ResNet-101c	COCO-Stuff	✗	16.2	23.5	37.1	62.0	-
DPSeg [112]	CLIP ConvNeXt-L	-	COCO-Stuff	✗	15.7	24.1	37.1	62.3	-
SemLA [66]	CLIP ConvNeXt-L	-	COCO-Stuff	✗	-	-	36.9	62.2	-
ESC-Net [50]	CLIP ViT-L/14	-	COCO-Stuff	✗	18.1	27.0	41.8	65.6	-
DiGSeg (ours)	CLIP ViT-L/14	Stable Diffusion	COCO-Stuff	✗	**19.9** (+1.8)	**29.2** (+2.2)	**43.2** (+1.4)	**68.4** (+2.8)	**38.5** (+0.6)
OVSeg [56]	CLIP ViT-B/16	ResNet-101c	COCO-Stuff	✓	7.1	11.0	24.8	53.3	-
SCAN [58]	CLIP ViT-B/16	Swin-B	COCO-Stuff	✗	10.8	13.2	30.8	58.4	-
EBSeg [72]	CLIP ViT-B/16	SAM ViT-B	COCO-Stuff	✗	11.1	17.3	30.0	56.7	-
SED [96]	ConvNeXt-B	-	COCO-Stuff	✗	11.4	18.6	31.6	57.3	-
CAT-Seg [17]	CLIP ViT-B/16	-	COCO-Stuff	✗	12.0	19.0	31.8	57.5	-
OPMapper [89]	CLIP ViT-B/16	Swin-B	COCO-Stuff	✗	-	-	31.0	58.3	-
ESC-Net [50]	CLIP ViT-B/16	-	COCO-Stuff	✗	13.3	21.1	35.6	59.0	-
HyperCLIP [65]	CLIP ViT-B/16	-	COCO-Stuff	✗	12.3	19.2	32.1	58.5	-
Mask-Adapter [55]	CLIP ConvNeXt-B	-	COCO-Stuff	✗	14.2	17.9	35.6	58.4	35.2
DPSeg [112]	CLIP ConvNeXt-B	-	COCO-Stuff	✗	12.5	20.1	33.3	58.4	-
DiGSeg (ours)	CLIP ViT-B/16	Stable Diffusion	COCO-Stuff	✗	**17.5** (+3.3)	**23.1** (+2.0)	**37.2** (+1.6)	**62.7** (+3.7)	**36.5** (+1.3)

Table 1: Quantitative evaluation on open-vocabulary segmentation benchmarks All methods are evaluated on A-847, PC-459, A-150, PC-59, and Cityscapes. We highlight the **best**, while the second-best results are underlined. Improvements over the second-best are highlighted in green.

Thanks

Welcome to join US

Sherrylone.github.io