Diffusion Models II: Fast Sampling, Guidance, and Scalable Architectures

Lecture 24 ▪ CSCI 4140/6960 ▪ Machine Learning & Optimization

Alex Gittens

Rensselaer Polytechnic Institute

Spring 2026

Course website: `cs.rpi.edu/.../mlandopt.html`

Learning Objectives

By the end of this lecture, you should be able to:

1. **Explain** why DDPM sampling requires $\sim$1000 steps and how DDIM achieves high-quality samples in $\sim$50 steps by reusing the noise estimate instead of drawing fresh noise at every step
2. **Apply** the DDIM update rule given a trained denoiser, and describe the role of the stochasticity parameter η
3. **Derive** the classifier-free guidance formula $\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\cdot, \varnothing) + w[\boldsymbol{\epsilon}_\theta(\cdot, c) - \boldsymbol{\epsilon}_\theta(\cdot, \varnothing)]$ from the score decomposition, and explain the diversity–fidelity tradeoff controlled by w
4. **Explain** why diffusion in pixel space is wasteful and how latent diffusion reduces the denoising dimension by $\sim 48\times$ while preserving perceptual quality
5. **Describe** the two-stage LDM pipeline: perceptual VAE compression followed by latent ϵ-prediction, and how cross-attention injects text conditioning into the U-Net
6. **Explain** why DiT replaces the U-Net for scalable generation: patchify, adaLN-Zero conditioning, and power-law FID scaling with compute

Lecture Roadmap

Recap

Fast Sampling

Guidance

Latent Diffusion

DiT

Putting It All Together

Recap

Where We Left Off

Lecture 23 in one slide

Denoising diffusion probabilistic model (DDPM) objective

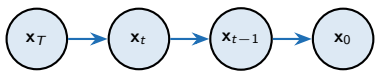
$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$$

DDPM sampling (1000-step reverse Gaussian chain)

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t\,\boldsymbol{\epsilon}, \quad \mathbf{x}_T \sim \mathcal{N}(0, I)$$

Architecture: U-Net with ResBlocks, GroupNorm + SiLU, sinusoidal time embedding.



Where We Left Off

Lecture 23 in one slide

Denoising diffusion probabilistic model (DDPM) objective

$$\mathcal{L}_{\text{DDPM}} = \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\right]$$

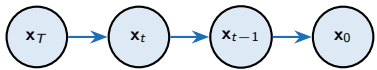
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}$$

DDPM sampling (1000-step reverse Gaussian chain)

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t\,\boldsymbol{\epsilon}, \quad \mathbf{x}_T \sim \mathcal{N}(0, I)$$

Architecture: U-Net with ResBlocks, GroupNorm + SiLU, sinusoidal time embedding.

reverse Markov chain



1000 sequential ϵ_θ evaluations per image



The Three Problems We Solve Today

DDPM works, but has three critical limitations. Each section of today's lecture resolves one problem.

Problem	Example	What we will introduce

The Three Problems We Solve Today

DDPM works, but has three critical limitations. Each section of today's lecture resolves one problem.

Problem	Example	What we will introduce
Speed	17 min on CPU for one image	Faster samplers: DDIM and DPM-Solver

The Three Problems We Solve Today

DDPM works, but has three critical limitations. Each section of today's lecture resolves one problem.

Problem	Example	What we will introduce
Speed	17 min on CPU for one image	Faster samplers: DDIM and DPM-Solver
Control	"a cat sitting on a chair" $\rightarrow$???	Classifier-free guidance so the prompt steers the image

The Three Problems We Solve Today

DDPM works, but has three critical limitations. Each section of today's lecture resolves one problem.

Problem	Example	What we will introduce
Speed	17 min on CPU for one image	Faster samplers: DDIM and DPM-Solver
Control	"a cat sitting on a chair" $\rightarrow$???	Classifier-free guidance so the prompt steers the image

DDIM = *Denoising Diffusion Implicit Models*; DiT = *Diffusion Transformer*.

The Three Problems We Solve Today

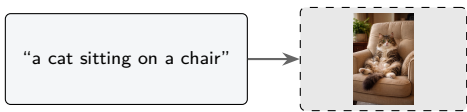
DDPM works, but has three critical limitations. Each section of today's lecture resolves one problem.

Problem	Example	What we will introduce
Speed	17 min on CPU for one image	Faster samplers: DDIM and DPM-Solver
Control	"a cat sitting on a chair" $\rightarrow$???	Classifier-free guidance so the prompt steers the image
Scale	$256^2 \times 3 = 196$K dims / step	Latent diffusion so we denoise in a smaller space

DDIM = *Denoising Diffusion Implicit Models*; DiT = *Diffusion Transformer*.

Bonus: once we have a latent denoiser, what architecture scales predictably? $\rightarrow$ **Diffusion Transformer (DiT)**.

All three together = Stable Diffusion.



Fast Sampling

Why DDPM Sampling Is Slow

The original DDPM sampler is a long reverse chain:



Why DDPM Sampling Is Slow

The original DDPM sampler is a long reverse chain:

each arrow = one neural network evaluation



Why DDPM Sampling Is Slow

The original DDPM sampler is a long reverse chain:

each arrow = one neural network evaluation



In the original sampler we usually take about 1000 small denoising steps before reaching $\mathbf{x}_0$.

- Each step is individually cheap, but 1000 of them adds up.

Why DDPM Sampling Is Slow

The original DDPM sampler is a long reverse chain:

each arrow = one neural network evaluation



In the original sampler we usually take about
1000 small denoising steps before reaching $\mathbf{x}_0$.

- Each step is individually cheap, but 1000 of them adds up.
- This is why sampling is much slower than the training objective might suggest.

Why DDPM Sampling Is Slow

The original DDPM sampler is a long reverse chain:

each arrow = one neural network evaluation



In the original sampler we usually take about
1000 small denoising steps before reaching $\mathbf{x}_0$.

- Each step is individually cheap, but 1000 of them adds up.
- This is why sampling is much slower than the training objective might suggest.
- The question for today: can we take *fewer*, *larger* steps without retraining the model?

Recap: What DDPM Is Doing

DDPM has a very simple training story:

1. Start with a clean image $\mathbf{x}_0$.

Recap: What DDPM Is Doing

DDPM has a very simple training story:

1. Start with a clean image $\mathbf{x}_0$.
2. Add Gaussian noise until you get a noisy image $\mathbf{x}_t$.

The basic forward relation is

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I).$$

Recap: What DDPM Is Doing

DDPM has a very simple training story:

1. Start with a clean image $\mathbf{x}_0$.
2. Add Gaussian noise until you get a noisy image $\mathbf{x}_t$.
3. Train a network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to predict the added noise.

The basic forward relation is

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I).$$

So at every noise level t, the network learns:

given a noisy image, what noise was added to it?

Why DDPM Adds Noise During Sampling

At sampling time, DDPM runs the reverse update

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(0, I).$$

Why DDPM Adds Noise During Sampling

At sampling time, DDPM runs the reverse update

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(0, I).$$

- $\mu_\theta(\mathbf{x}_t, t)$**:** the denoising step; it moves toward a cleaner image.

Why DDPM Adds Noise During Sampling

At sampling time, DDPM runs the reverse update

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(0, I).$$

- $\mu_\theta(\mathbf{x}_t, t)$**:** the denoising step; it moves toward a cleaner image.
- $\sigma_t \mathbf{z}$**:** a random kick; it keeps sampling stochastic so we generate many possible images, not one fixed path.

Why DDPM Adds Noise During Sampling

At sampling time, DDPM runs the reverse update

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(0, I).$$

- $\mu_\theta(\mathbf{x}_t, t)$**:** the denoising step; it moves toward a cleaner image.
- $\sigma_t \mathbf{z}$**:** a random kick; it keeps sampling stochastic so we generate many possible images, not one fixed path.

How is the noise size chosen? From the forward schedule:

- The network was trained at specific noise levels.

Why DDPM Adds Noise During Sampling

At sampling time, DDPM runs the reverse update

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(0, I).$$

- $\mu_\theta(\mathbf{x}_t, t)$**:** the denoising step; it moves toward a cleaner image.
- $\sigma_t \mathbf{z}$**:** a random kick; it keeps sampling stochastic so we generate many possible images, not one fixed path.

How is the noise size chosen? From the forward schedule:

- The network was trained at specific noise levels.
- The reverse step should land at the *next* noise level the network knows how to handle.

Why DDPM Adds Noise During Sampling

At sampling time, DDPM runs the reverse update

$$\mathbf{x}_{t-1} = \mu_\theta(\mathbf{x}_t, t) + \sigma_t \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(0, I).$$

- $\mu_\theta(\mathbf{x}_t, t)$**:** the denoising step; it moves toward a cleaner image.
- $\sigma_t \mathbf{z}$**:** a random kick; it keeps sampling stochastic so we generate many possible images, not one fixed path.

How is the noise size chosen? From the forward schedule:

- The network was trained at specific noise levels.
- The reverse step should land at the *next* noise level the network knows how to handle.
- So DDPM matches the forward schedule instead of inventing a new noise scale.

For reference, the standard DDPM variance is

$$\sigma_t^2 = \tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t.$$

What DDIM Changes

DDIM = **Denoising Diffusion Implicit Models**.

It keeps the *same trained network* and changes only the *sampling rule*.

Same as DDPM	**Changed in DDIM**

What DDIM Changes

DDIM = **Denoising Diffusion Implicit Models**.

It keeps the *same trained network* and changes only the *sampling rule*.

Same as DDPM	**Changed in DDIM**
same noisy training examples	reverse update rule

What DDIM Changes

DDIM = **Denoising Diffusion Implicit Models**.

It keeps the *same trained network* and changes only the *sampling rule*.

Same as DDPM	**Changed in DDIM**
same noisy training examples	reverse update rule
same denoiser ϵ_θ	amount of fresh noise per step

What DDIM Changes

DDIM = **Denoising Diffusion Implicit Models**.

It keeps the *same trained network* and changes only the *sampling rule*.

Same as DDPM	**Changed in DDIM**
same noisy training examples	reverse update rule
same denoiser ϵ_θ	amount of fresh noise per step
same training objective	number of timesteps visited at sampling time

What DDIM Changes

DDIM = **Denoising Diffusion Implicit Models**.

It keeps the *same trained network* and changes only the *sampling rule*.

Same as DDPM	**Changed in DDIM**
same noisy training examples	reverse update rule
same denoiser ϵ_θ	amount of fresh noise per step
same training objective	number of timesteps visited at sampling time

In short:

- DDPM says: add a fresh random kick at every reverse step.

What DDIM Changes

DDIM = **Denoising Diffusion Implicit Models**.

It keeps the *same trained network* and changes only the *sampling rule*.

Same as DDPM	**Changed in DDIM**
same noisy training examples	reverse update rule
same denoiser ϵ_θ	amount of fresh noise per step
same training objective	number of timesteps visited at sampling time

In short:

- DDPM says: add a fresh random kick at every reverse step.
- DDIM says: maybe we do *not* need a full fresh random kick every time.

Reusing the Noise Estimate

Suppose the network tells us what the clean image probably is:

$$\hat{\mathbf{x}}_0 = \text{network's guess for the clean image.}$$

Reusing the Noise Estimate

Suppose the network tells us what the clean image probably is:

$$\hat{\mathbf{x}}_0 = \text{network's guess for the clean image.}$$

The same network output also gives a noise estimate: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the noise the network thinks was added to $\hat{\mathbf{x}}_0$ to produce $\mathbf{x}_t$.

DDIM keeps $\hat{\epsilon}$ and uses it to set the direction of the next point, rather than discarding it and drawing fresh noise.

Reusing the Noise Estimate

Suppose the network tells us what the clean image probably is:

$$\hat{\mathbf{x}}_0 = \text{network's guess for the clean image.}$$

The same network output also gives a noise estimate: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the noise the network thinks was added to $\hat{\mathbf{x}}_0$ to produce $\mathbf{x}_t$.

DDIM keeps $\hat{\epsilon}$ and uses it to set the direction of the next point, rather than discarding it and drawing fresh noise.

So DDIM says:

- do not throw away the noise estimate $\hat{\epsilon}$,

Reusing the Noise Estimate

Suppose the network tells us what the clean image probably is:

$$\hat{\mathbf{x}}_0 = \text{network's guess for the clean image.}$$

The same network output also gives a noise estimate: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the noise the network thinks was added to $\hat{\mathbf{x}}_0$ to produce $\mathbf{x}_t$.

DDIM keeps $\hat{\epsilon}$ and uses it to set the direction of the next point, rather than discarding it and drawing fresh noise.

So DDIM says:

- do not throw away the noise estimate $\hat{\epsilon}$,
- keep following that direction as you reduce the noise level,

Reusing the Noise Estimate

Suppose the network tells us what the clean image probably is:

$$\hat{\mathbf{x}}_0 = \text{network's guess for the clean image.}$$

The same network output also gives a noise estimate: $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ is the noise the network thinks was added to $\hat{\mathbf{x}}_0$ to produce $\mathbf{x}_t$.

DDIM keeps $\hat{\epsilon}$ and uses it to set the direction of the next point, rather than discarding it and drawing fresh noise.

So DDIM says:

- do not throw away the noise estimate $\hat{\epsilon}$,
- keep following that direction as you reduce the noise level,
- and only add extra fresh randomness if you still want it.

That is what "reusing the noise" means here.

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

DDIM keeps track of two things instead:

- the network's clean-image estimate,

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

DDIM keeps track of two things instead:

- the network's clean-image estimate,
- the current noise direction.

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

DDIM keeps track of two things instead:

- the network's clean-image estimate,
- the current noise direction.

Once we have those, we can move directly to a *later* noise level without visiting every intermediate timestep.

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

DDIM keeps track of two things instead:

- the network's clean-image estimate,
- the current noise direction.

Once we have those, we can move directly to a *later* noise level without visiting every intermediate timestep.

That means:

- fewer timesteps kept,

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

DDIM keeps track of two things instead:

- the network's clean-image estimate,
- the current noise direction.

Once we have those, we can move directly to a *later* noise level without visiting every intermediate timestep.

That means:

- fewer timesteps kept,
- fewer denoiser evaluations,

Why This Can Be Faster

DDPM is tied to one particular reverse Markov chain, so it usually takes many tiny steps.

DDIM keeps track of two things instead:

- the network's clean-image estimate,
- the current noise direction.

Once we have those, we can move directly to a *later* noise level without visiting every intermediate timestep.

That means:

- fewer timesteps kept,
- fewer denoiser evaluations,
- faster sampling.

DDIM is faster because it can skip many intermediate reverse steps while still following a coherent denoising trajectory.

The DDIM Noise Knob

DDIM has a parameter that controls how much fresh randomness to add:

$$\sigma_t^2 = \eta^2 \, \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \left(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}} \right), \qquad \eta \in [0, 1].$$

The DDIM Noise Knob

DDIM has a parameter that controls how much fresh randomness to add:

$$\sigma_t^2 = \eta^2 \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\left(1-\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}\right), \qquad \eta \in [0,1].$$

You do not need to memorize this formula. The main idea is:

- $\eta = 0$: no fresh noise added $\Rightarrow$ deterministic DDIM.

The DDIM Noise Knob

DDIM has a parameter that controls how much fresh randomness to add:

$$\sigma_t^2 = \eta^2 \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\left(1-\frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}\right), \qquad \eta \in [0,1].$$

You do not need to memorize this formula. The main idea is:

- $\eta = 0$**:** no fresh noise added $\Rightarrow$ deterministic DDIM.
- **larger** η**:** add more fresh noise $\Rightarrow$ behavior closer to DDPM.

So DDIM gives us a *stochasticity knob*: from fully deterministic to more DDPM-like.

One Reverse Step: DDPM vs. DDIM

Both DDPM and DDIM use the denoiser to estimate where the clean sample should be. The difference is how the next point is placed once that estimate is available.

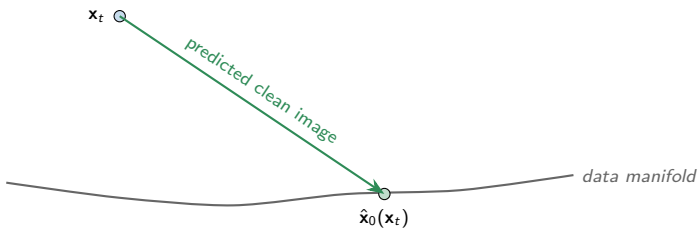
DDIM update = estimate clean sample, then move to the next noise level



One Reverse Step: DDPM vs. DDIM

Both DDPM and DDIM use the denoiser to estimate where the clean sample should be. The difference is how the next point is placed once that estimate is available.

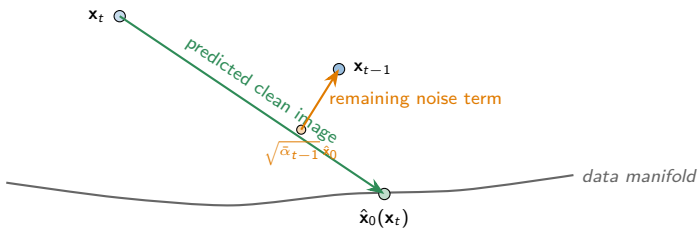
DDIM update = estimate clean sample, then move to the next noise level



One Reverse Step: DDPM vs. DDIM

Both DDPM and DDIM use the denoiser to estimate where the clean sample should be. The difference is how the next point is placed once that estimate is available.

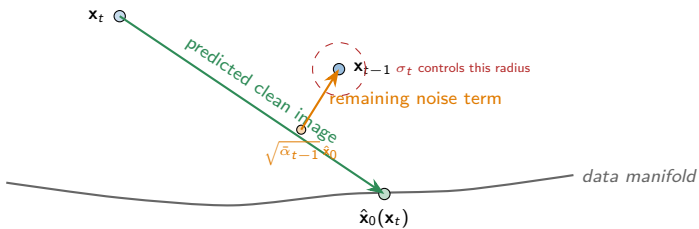
DDIM update = estimate clean sample, then move to the next noise level



One Reverse Step: DDPM vs. DDIM

Both DDPM and DDIM use the denoiser to estimate where the clean sample should be. The difference is how the next point is placed once that estimate is available.

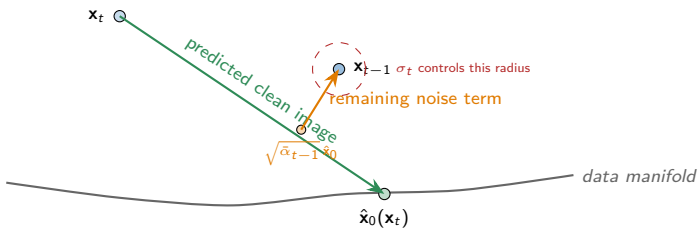
DDIM update = estimate clean sample, then move to the next noise level



One Reverse Step: DDPM vs. DDIM

Both DDPM and DDIM use the denoiser to estimate where the clean sample should be. The difference is how the next point is placed once that estimate is available.

DDIM update = estimate clean sample, then move to the next noise level

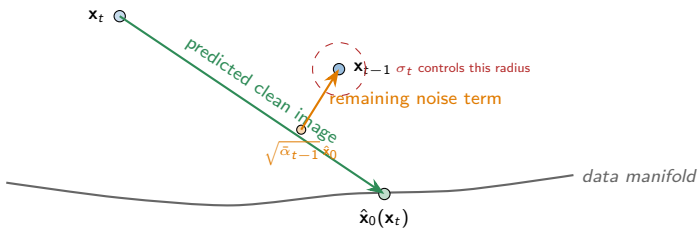


- **DDPM:** the reverse Gaussian model fixes both the mean and the fresh noise level from the schedule.

One Reverse Step: DDPM vs. DDIM

Both DDPM and DDIM use the denoiser to estimate where the clean sample should be. The difference is how the next point is placed once that estimate is available.

DDIM update = estimate clean sample, then move to the next noise level



- **DDPM:** the reverse Gaussian model fixes both the mean and the fresh noise level from the schedule.
- **DDIM:** use the same denoiser estimate, but choose how much fresh randomness to add through σ_t.

DDIM Algorithm

Input: trained ϵ_θ; schedule $\{\bar{\alpha}_t\}$; subsequence $\tau_1 > \tau_2 > \cdots > \tau_S$; noise level $\sigma_{\tau_s} \geq 0$
Output: clean sample $\mathbf{x}_0$
$\mathbf{x}_{\tau_1} \sim \mathcal{N}(0, I)$
for $s = 1, \ldots, S-1$ **do**

end

Algorithm 1 — DDIM sampling

DDIM Algorithm

Input: trained ϵ_θ; schedule $\{\bar{\alpha}_t\}$; subsequence $\tau_1 > \tau_2 > \cdots > \tau_S$; noise level $\sigma_{\tau_s} \geq 0$
Output: clean sample $\mathbf{x}_0$
$\mathbf{x}_{\tau_1} \sim \mathcal{N}(0, I)$
for $s = 1, \ldots, S-1$ **do**
    $\hat{\epsilon} \leftarrow \epsilon_\theta(\mathbf{x}_{\tau_s}, \tau_s)$ // noise prediction
end

Algorithm 2 — DDIM sampling

DDIM Algorithm

Input: trained ϵ_θ; schedule $\{\bar{\alpha}_t\}$; subsequence $\tau_1 > \tau_2 > \cdots > \tau_S$; noise level $\sigma_{\tau_s} \geq 0$
Output: clean sample $\mathbf{x}_0$
$\mathbf{x}_{\tau_1} \sim \mathcal{N}(0, I)$
for $s = 1, \ldots, S-1$ **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(\mathbf{x}_{\tau_s}, \tau_s)$ // noise prediction
  $\hat{\mathbf{x}}_0 \leftarrow \left(\mathbf{x}_{\tau_s} - \sqrt{1-\bar{\alpha}_{\tau_s}}\,\hat{\epsilon}\right) / \sqrt{\bar{\alpha}_{\tau_s}}$ // implied clean image
end

Algorithm 3 — DDIM sampling

DDIM Algorithm

Input: trained ϵ_θ; schedule $\{\bar{\alpha}_t\}$; subsequence $\tau_1 > \tau_2 > \cdots > \tau_S$; noise level $\sigma_{\tau_s} \geq 0$
Output: clean sample $\mathbf{x}_0$
$\mathbf{x}_{\tau_1} \sim \mathcal{N}(0, I)$
for $s = 1, \ldots, S-1$ **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(\mathbf{x}_{\tau_s}, \tau_s)$ // noise prediction
  $\hat{\mathbf{x}}_0 \leftarrow \left(\mathbf{x}_{\tau_s} - \sqrt{1-\bar{\alpha}_{\tau_s}}\,\hat{\epsilon}\right) / \sqrt{\bar{\alpha}_{\tau_s}}$ // implied clean image
  $\epsilon \sim \mathcal{N}(0, I)$
  $\mathbf{x}_{\tau_{s+1}} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{\tau_{s+1}} - \sigma_{\tau_s}^2}\,\hat{\epsilon} + \sigma_{\tau_s}\,\epsilon$ // DDIM update
end

Algorithm 4 — DDIM sampling

DDIM Algorithm

Input: trained $\boldsymbol{\epsilon}_\theta$; schedule $\{\bar{\alpha}_t\}$; subsequence $\tau_1 > \tau_2 > \cdots > \tau_S$; noise level $\sigma_{\tau_s} \geq 0$
Output: clean sample $\mathbf{x}_0$
$\mathbf{x}_{\tau_1} \sim \mathcal{N}(0, I)$
for $s = 1, \ldots, S-1$ **do**
  $\hat{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{x}_{\tau_s}, \tau_s)$ // noise prediction
  $\hat{\mathbf{x}}_0 \leftarrow \left(\mathbf{x}_{\tau_s} - \sqrt{1-\bar{\alpha}_{\tau_s}}\,\hat{\boldsymbol{\epsilon}}\right) / \sqrt{\bar{\alpha}_{\tau_s}}$ // implied clean image
  $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$
  $\mathbf{x}_{\tau_{s+1}} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{\tau_{s+1}} - \sigma_{\tau_s}^2}\,\hat{\boldsymbol{\epsilon}} + \sigma_{\tau_s}\,\boldsymbol{\epsilon}$ // DDIM update
end
$\hat{\mathbf{x}}_0 \leftarrow \left(\mathbf{x}_{\tau_S} - \sqrt{1-\bar{\alpha}_{\tau_S}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_{\tau_S}, \tau_S)\right) / \sqrt{\bar{\alpha}_{\tau_S}}$
return $\hat{\mathbf{x}}_0$

Algorithm 5 — DDIM sampling

DDIM Algorithm

Input: trained ϵ_θ; schedule $\{\bar{\alpha}_t\}$; subsequence $\tau_1 > \tau_2 > \cdots > \tau_S$; noise level $\sigma_{\tau_s} \geq 0$
Output: clean sample $\mathbf{x}_0$
$\mathbf{x}_{\tau_1} \sim \mathcal{N}(0, I)$
for $s = 1, \ldots, S-1$ **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(\mathbf{x}_{\tau_s}, \tau_s)$ // noise prediction
  $\hat{\mathbf{x}}_0 \leftarrow \left(\mathbf{x}_{\tau_s} - \sqrt{1-\bar{\alpha}_{\tau_s}}\,\hat{\epsilon}\right) / \sqrt{\bar{\alpha}_{\tau_s}}$ // implied clean image
  $\epsilon \sim \mathcal{N}(0, I)$
  $\mathbf{x}_{\tau_{s+1}} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{\tau_{s+1}} - \sigma_{\tau_s}^2}\,\hat{\epsilon} + \sigma_{\tau_s}\,\epsilon$ // DDIM update
end
$\hat{\mathbf{x}}_0 \leftarrow \left(\mathbf{x}_{\tau_S} - \sqrt{1-\bar{\alpha}_{\tau_S}}\,\epsilon_\theta(\mathbf{x}_{\tau_S}, \tau_S)\right) / \sqrt{\bar{\alpha}_{\tau_S}}$
return $\hat{\mathbf{x}}_0$

Algorithm 6 — DDIM sampling

Step skipping. Marginal noise level means the quantity $1 - \bar{\alpha}_t$ in

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

Because DDIM is written directly in terms of these marginal noise levels, we can choose a coarse subsequence of timesteps instead of visiting all 1000.

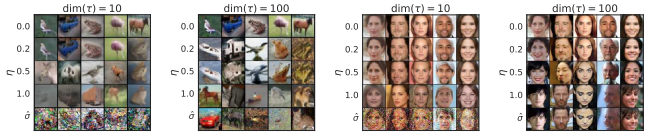
Typical rule of thumb: $S \approx 50$ often preserves most of the quality while cutting sampling time dramatically.

If S gets too small, quality starts to degrade. That motivates better solvers than plain DDIM.

DDIM Quality vs. Step Count

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
η	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
$\hat{\sigma}$		367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**

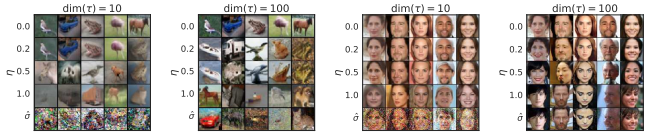


Source: Song, Meng, Ermon 2020 (DDIM), Figure 3.

DDIM Quality vs. Step Count

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
η	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
$\hat{\sigma}$		367.43	133.37	32.72	9.99	**3.17**	299.71	183.83	71.71	45.20	**3.26**



Source: Song, Meng, Ermon 2020 (DDIM), Figure 3.

Takeaway: the $\sigma_t = 0$ DDIM curve flattens by $S \approx 50$. The remaining gap between 50 and 1000 steps is small.

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

DPM-Solver family (Lu et al. 2022):

Sampler	**Steps to FID** ≤ 5 **(CIFAR-10)**	**Notes**
DDPM	~ 1000	Markovian; stochastic

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

DPM-Solver family (Lu et al. 2022):

Sampler	**Steps to FID ≤ 5 (CIFAR-10)**	**Notes**
DDPM	~ 1000	Markovian; stochastic
DDIM	~ 50–100	Deterministic or stochastic

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

DPM-Solver family (Lu et al. 2022):

Sampler	**Steps to FID ≤ 5 (CIFAR-10)**	**Notes**
DDPM	~ 1000	Markovian; stochastic
DDIM	~ 50–100	Deterministic or stochastic
DPM-Solver-2	~ 15–20	Second-order solver

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

DPM-Solver family (Lu et al. 2022):

Sampler	**Steps to FID ≤ 5 (CIFAR-10)**	**Notes**
DDPM	~ 1000	Markovian; stochastic
DDIM	~ 50–100	Deterministic or stochastic
DPM-Solver-2	~ 15–20	Second-order solver
DPM-Solver-3	~ 10–15	Third-order solver

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

DPM-Solver family (Lu et al. 2022):

Sampler	**Steps to FID ≤ 5 (CIFAR-10)**	**Notes**
DDPM	~ 1000	Markovian; stochastic
DDIM	~ 50–100	Deterministic or stochastic
DPM-Solver-2	~ 15–20	Second-order solver
DPM-Solver-3	~ 10–15	Third-order solver

Takeaway. DDIM is the simplest fast sampler; DPM-Solver keeps the same denoiser but uses a better numerical update rule.

Why Higher-Order Solvers Help

There is a continuous-time view of deterministic denoising in which DDIM with $\sigma_t = 0$ behaves like a *forward Euler* method.

That point of view matters because Euler is only a first-order numerical method. Once we recognize that, we can use better integrators.

DPM-Solver family (Lu et al. 2022):

Sampler	Steps to FID ≤ 5 (CIFAR-10)	Notes
DDPM	~ 1000	Markovian; stochastic
DDIM	~ 50–100	Deterministic or stochastic
DPM-Solver-2	~ 15–20	Second-order solver
DPM-Solver-3	~ 10–15	Third-order solver

Takeaway. DDIM is the simplest fast sampler; DPM-Solver keeps the same denoiser but uses a better numerical update rule.

Problem 1 (Speed) resolved. ✓ DDIM / DPM-Solver give high-quality samples in $\sim 10-50$ steps.

Guidance

How Do We Steer the Generation?

- Trained DDPM samples from the *unconditional* $p(\mathbf{x})$.

How Do We Steer the Generation?

- Trained DDPM samples from the *unconditional* $p(\mathbf{x})$.
- We want $p(\mathbf{x} \mid c)$ for conditioning c: text prompt, class label, reference image.

How Do We Steer the Generation?

- Trained DDPM samples from the *unconditional* $p(\mathbf{x})$.
- We want $p(\mathbf{x} \mid c)$ for conditioning c: text prompt, class label, reference image.
- Running example: $c =$ "a cat sitting on a chair" — we want this cat, not a random image.

How Do We Steer the Generation?

- Trained DDPM samples from the *unconditional* $p(\mathbf{x})$.
- We want $p(\mathbf{x} \mid c)$ for conditioning c: text prompt, class label, reference image.
- Running example: $c =$ "a cat sitting on a chair" — we want this cat, not a random image.

Score perspective. Bayes' rule gives

$$\log p(\mathbf{x} \mid c) = \log p(c \mid \mathbf{x}) + \log p(\mathbf{x}) - \log p(c).$$

How Do We Steer the Generation?

- Trained DDPM samples from the *unconditional* $p(\mathbf{x})$.
- We want $p(\mathbf{x} \mid c)$ for conditioning c: text prompt, class label, reference image.
- Running example: $c =$ "a cat sitting on a chair" — we want this cat, not a random image.

Score perspective. Bayes' rule gives

$$\log p(\mathbf{x} \mid c) = \log p(c \mid \mathbf{x}) + \log p(\mathbf{x}) - \log p(c).$$

Differentiating w.r.t. $\mathbf{x}$:

$$\boxed{\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) = \underbrace{\nabla_{\mathbf{x}} \log p(c \mid \mathbf{x})}_{\text{classifier gradient}} + \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x})}_{\text{unconditional score}}}$$

This decomposition is the foundation for both guidance methods we will study.

How Do We Steer the Generation?

- Trained DDPM samples from the *unconditional* $p(\mathbf{x})$.
- We want $p(\mathbf{x} \mid c)$ for conditioning c: text prompt, class label, reference image.
- Running example: $c =$ "a cat sitting on a chair" — we want this cat, not a random image.

Score perspective. Bayes' rule gives

$$\log p(\mathbf{x} \mid c) = \log p(c \mid \mathbf{x}) + \log p(\mathbf{x}) - \log p(c).$$

Differentiating w.r.t. $\mathbf{x}$:

$$\boxed{\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) = \underbrace{\nabla_{\mathbf{x}} \log p(c \mid \mathbf{x})}_{\text{classifier gradient}} + \underbrace{\nabla_{\mathbf{x}} \log p(\mathbf{x})}_{\text{unconditional score}}}$$

This decomposition is the foundation for both guidance methods we will study.

First we'll see how to train a conditional model directly—then show why sampling from it isn't quite enough—then study the two guidance methods.

Conditioning Without a Classifier

The simplest fix. Inject the condition c the same way we inject time t.

Conditioning Without a Classifier

The simplest fix. Inject the condition c the same way we inject time t.

For class labels: a learned embedding $\mathbf{e}_c \in \mathbb{R}^d$ added to the time embedding.
For text: a frozen or learned encoder feeds c into the denoiser via cross-attention.
Either way, the network becomes $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)$ and the training loss is unchanged:

$$\mathcal{L} = \mathbb{E}\big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)\|^2\big].$$

Conditioning Without a Classifier

The simplest fix. Inject the condition c the same way we inject time t.

For class labels: a learned embedding $\mathbf{e}_c \in \mathbb{R}^d$ added to the time embedding.
For text: a frozen or learned encoder feeds c into the denoiser via cross-attention.
Either way, the network becomes $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)$ and the training loss is unchanged:

$$\mathcal{L} = \mathbb{E}\big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)\|^2\big].$$

Analogy. Time t is embedded, scaled, and added to the residual stream. Condition c is embedded the same way—no architectural change, no extra model.

Conditioning Without a Classifier

The simplest fix. Inject the condition c the same way we inject time t.

For class labels: a learned embedding $\mathbf{e}_c \in \mathbb{R}^d$ added to the time embedding.
For text: a frozen or learned encoder feeds c into the denoiser via cross-attention.
Either way, the network becomes $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)$ and the training loss is unchanged:

$$\mathcal{L} = \mathbb{E}\big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)\|^2\big].$$

Analogy. Time t is embedded, scaled, and added to the residual stream. Condition c is embedded the same way—no architectural change, no extra model.

What this buys you. At sampling time, fix any c and run the reverse process. The sampler draws from the *true conditional distribution* $p(\mathbf{x} \mid c)$.

Conditioning Without a Classifier

The simplest fix. Inject the condition c the same way we inject time t.

For class labels: a learned embedding $\mathbf{e}_c \in \mathbb{R}^d$ added to the time embedding.
For text: a frozen or learned encoder feeds c into the denoiser via cross-attention.
Either way, the network becomes $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)$ and the training loss is unchanged:

$$\mathcal{L} = \mathbb{E}\big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)\|^2\big].$$

Analogy. Time t is embedded, scaled, and added to the residual stream. Condition c is embedded the same way—no architectural change, no extra model.

What this buys you. At sampling time, fix any c and run the reverse process. The sampler draws from the *true conditional distribution* $p(\mathbf{x} \mid c)$.

Is that enough? Not always.

Why Sampling from $p(\mathbf{x} \mid c)$ Isn't Enough

The problem. A perfectly trained $\epsilon_\theta(\mathbf{x}_t, t, c)$ samples from the *true* $p(\mathbf{x} \mid c)$. That distribution includes *all* modes:

Why Sampling from $p(\mathbf{x} \mid c)$ Isn't Enough

The problem. A perfectly trained $\epsilon_\theta(\mathbf{x}_t, t, c)$ samples from the *true* $p(\mathbf{x} \mid c)$. That distribution includes *all* modes:

- rare and ambiguous examples
- cluttered scenes where c is present but not prominent
- noisy correlations from the training data

Why Sampling from $p(\mathbf{x} \mid c)$ Isn't Enough

The problem. A perfectly trained $\epsilon_\theta(\mathbf{x}_t, t, c)$ samples from the *true* $p(\mathbf{x} \mid c)$. That distribution includes *all* modes:

- rare and ambiguous examples
- cluttered scenes where c is present but not prominent
- noisy correlations from the training data

In practice: ImageNet class-conditional models produce *correct but messy* images; text-to-image models produce *loosely aligned* outputs.

Why Sampling from $p(\mathbf{x} \mid c)$ Isn't Enough

The problem. A perfectly trained $\epsilon_\theta(\mathbf{x}_t, t, c)$ samples from the *true* $p(\mathbf{x} \mid c)$. That distribution includes *all* modes:

- rare and ambiguous examples
- cluttered scenes where c is present but not prominent
- noisy correlations from the training data

In practice: ImageNet class-conditional models produce *correct but messy* images; text-to-image models produce *loosely aligned* outputs.

What we actually want. Samples that are *prototypical*—high-fidelity, unambiguous, clearly of class c. That is closer to a MAP target:

$$\mathbf{x}^* = \arg\max \big[\log p(\mathbf{x}) + w \log p(c \mid \mathbf{x})\big].$$

Equivalently, we want to sample from the *tilted* distribution $\tilde{p}(\mathbf{x} \mid c) \propto p(\mathbf{x})\, p(c \mid \mathbf{x})^w$, where $w > 1$ sharpens toward high-likelihood-c regions.

Why Sampling from $p(\mathbf{x} \mid c)$ Isn't Enough

The problem. A perfectly trained $\epsilon_\theta(\mathbf{x}_t, t, c)$ samples from the *true* $p(\mathbf{x} \mid c)$. That distribution includes *all* modes:

- rare and ambiguous examples
- cluttered scenes where c is present but not prominent
- noisy correlations from the training data

In practice: ImageNet class-conditional models produce *correct but messy* images; text-to-image models produce *loosely aligned* outputs.

What we actually want. Samples that are *prototypical*—high-fidelity, unambiguous, clearly of class c. That is closer to a MAP target:

$$\mathbf{x}^* = \arg\max \big[\log p(\mathbf{x}) + w \log p(c \mid \mathbf{x})\big].$$

Equivalently, we want to sample from the *tilted* distribution $\tilde{p}(\mathbf{x} \mid c) \propto p(\mathbf{x})\, p(c \mid \mathbf{x})^w$, where $w > 1$ sharpens toward high-likelihood-c regions.

This motivates guidance: a mechanism to tilt the sampling distribution without retraining from scratch.

Diversity vs. Fidelity



Left: classifier scale 1.0 (FID 33.0) – high *diversity*, low *fidelity*. **Right:** classifier scale 10.0 (FID 12.0) – low diversity, high *fidelity*.
Source: Dhariwal & Nichol 2021 (Classifier Guidance), Figure 3.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Why modify the noise prediction? In a diffusion model the noise predictor $\epsilon_\theta(\mathbf{x}_t, t)$ represents the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. Adding $\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t)$ to the score is equivalent to steering the noise prediction, giving a guided predictor.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Why modify the noise prediction? In a diffusion model the noise predictor $\epsilon_\theta(\mathbf{x}_t, t)$ represents the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. Adding $\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t)$ to the score is equivalent to steering the noise prediction, giving a guided predictor.

That gives the guided prediction

$$\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t) - w\sqrt{1-\bar{\alpha}_t}\,\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t).$$

- $w = 0$: no steering.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Why modify the noise prediction? In a diffusion model the noise predictor $\epsilon_\theta(\mathbf{x}_t, t)$ represents the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. Adding $\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t)$ to the score is equivalent to steering the noise prediction, giving a guided predictor.

That gives the guided prediction

$$\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t) - w\sqrt{1-\bar{\alpha}_t}\,\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t).$$

- $w = 0$: no steering.
- larger w: push harder toward images the classifier thinks match class c.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Why modify the noise prediction? In a diffusion model the noise predictor $\epsilon_\theta(\mathbf{x}_t, t)$ represents the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. Adding $\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t)$ to the score is equivalent to steering the noise prediction, giving a guided predictor.

That gives the guided prediction

$$\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t) - w\sqrt{1-\bar{\alpha}_t}\,\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t).$$

- $w = 0$: no steering.
- larger w: push harder toward images the classifier thinks match class c.
- Guidance scale trades **diversity** for **fidelity**.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Why modify the noise prediction? In a diffusion model the noise predictor $\epsilon_\theta(\mathbf{x}_t, t)$ represents the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. Adding $\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t)$ to the score is equivalent to steering the noise prediction, giving a guided predictor.

That gives the guided prediction

$$\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t) - w\sqrt{1-\bar{\alpha}_t}\,\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t).$$

- $w = 0$: no steering.
- larger w: push harder toward images the classifier thinks match class c.
- Guidance scale trades **diversity** for **fidelity**.

Failure mode 1. A standard ImageNet classifier trained on clean images *does not work*. The gradient $\nabla_{\mathbf{x}} \log p_\phi(c \mid \mathbf{x}_t)$ is meaningless for heavily noised $\mathbf{x}_t$. The classifier must be trained (or finetuned) on noisy images at every σ level — significant extra compute.

Classifier Guidance (Dhariwal & Nichol 2021)

As motivated: we want to tilt sampling toward higher $p(c \mid \mathbf{x}_t)$. The explicit approach—train a classifier $p_\phi(c \mid \mathbf{x}_t)$ on *noisy* images at every level t, and use its gradient directly.

Why modify the noise prediction? In a diffusion model the noise predictor $\epsilon_\theta(\mathbf{x}_t, t)$ represents the unconditional score $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t)$. Adding $\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t)$ to the score is equivalent to steering the noise prediction, giving a guided predictor.

That gives the guided prediction

$$\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t) - w\sqrt{1-\bar{\alpha}_t}\,\nabla_{\mathbf{x}_t} \log p_\phi(c \mid \mathbf{x}_t).$$

- $w = 0$: no steering.
- larger w: push harder toward images the classifier thinks match class c.
- Guidance scale trades **diversity** for **fidelity**.

Failure mode 1. A standard ImageNet classifier trained on clean images *does not work*. The gradient $\nabla_{\mathbf{x}} \log p_\phi(c \mid \mathbf{x}_t)$ is meaningless for heavily noised $\mathbf{x}_t$. The classifier must be trained (or finetuned) on noisy images at every σ level — significant extra compute.

Failure mode 2. At large w, the sampler can exploit classifier shortcuts: it may create a weird high-contrast texture or exaggerated class cue that makes the classifier say "yes, that's a cat" even though the image looks unnatural to a human.

Classifier-Free Guidance: The Key Insight

Ho & Salimans 2022

Instead of a separate classifier, run the *same* diffusion model twice — with and without conditioning.

Classifier-Free Guidance: The Key Insight

Ho & Salimans 2022

Instead of a separate classifier, run the *same* diffusion model twice — with and without conditioning.

Training. With probability $p_{\text{drop}} \approx 0.1$, replace c with a null token $\varnothing$:

the network learns both $\epsilon_\theta(\mathbf{x}_t, t, c)$ and $\epsilon_\theta(\mathbf{x}_t, t, \varnothing)$ in a single model.

Classifier-Free Guidance: The Key Insight

Ho & Salimans 2022

Instead of a separate classifier, run the *same* diffusion model twice — with and without conditioning.

Training. With probability $p_{\text{drop}} \approx 0.1$, replace c with a null token $\varnothing$:

the network learns both $\epsilon_\theta(\mathbf{x}_t, t, c)$ and $\epsilon_\theta(\mathbf{x}_t, t, \varnothing)$ in a single model.

Input: dataset $\{(\mathbf{x}^{(i)}, c^{(i)})\}$; dropout probability $p_{\text{drop}} = 0.1$
for *each minibatch* $(\mathbf{x}_0, c)$ **do**
  $t \sim \text{Unif}\{1, \ldots, T\}$; $\epsilon \sim \mathcal{N}(0, I)$
  $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\,\epsilon$
end

Algorithm 9 — CFG training (standard DDPM loss + random condition dropout)

Classifier-Free Guidance: The Key Insight

Ho & Salimans 2022

Instead of a separate classifier, run the *same* diffusion model twice — with and without conditioning.

Training. With probability $p_{\text{drop}} \approx 0.1$, replace c with a null token $\varnothing$:

the network learns both $\epsilon_\theta(\mathbf{x}_t, t, c)$ and $\epsilon_\theta(\mathbf{x}_t, t, \varnothing)$ in a single model.

Input: dataset $\{(\mathbf{x}^{(i)}, c^{(i)})\}$; dropout probability $p_{\text{drop}} = 0.1$
for *each minibatch* $(\mathbf{x}_0, c)$ **do**
  $t \sim \text{Unif}\{1, \ldots, T\}$; $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$
  $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\,\boldsymbol{\epsilon}$
  if $u \sim \mathit{Unif}[0, 1] < p_{\mathit{drop}}$ **then**
    $c \leftarrow \varnothing$ // null token
  end
end

Algorithm 10 — CFG training (standard DDPM loss + random condition dropout)

Classifier-Free Guidance: The Key Insight

Ho & Salimans 2022

Instead of a separate classifier, run the *same* diffusion model twice — with and without conditioning.

Training. With probability $p_{\text{drop}} \approx 0.1$, replace c with a null token $\varnothing$:

the network learns both $\epsilon_\theta(\mathbf{x}_t, t, c)$ and $\epsilon_\theta(\mathbf{x}_t, t, \varnothing)$ in a single model.

Input: dataset $\{(\mathbf{x}^{(i)}, c^{(i)})\}$; dropout probability $p_{\text{drop}} = 0.1$
for *each minibatch* $(\mathbf{x}_0, c)$ **do**
  $t \sim \text{Unif}\{1, \ldots, T\}$; $\epsilon \sim \mathcal{N}(0, I)$
  $\mathbf{x}_t \leftarrow \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon$
  if $u \sim \mathit{Unif}[0,1] < p_{\mathit{drop}}$ **then**
    $c \leftarrow \varnothing$ // null token
  end
  $\mathcal{L} \leftarrow \|\epsilon - \epsilon_\theta(\mathbf{x}_t, t, c)\|^2$
  update θ by $\nabla_\theta \mathcal{L}$
end

Algorithm 11 — CFG training (standard DDPM loss + random condition dropout)

CFG Sampling Formula

Sampling. Combine the two predictions at every step:

$$\boxed{\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t, \varnothing) + w \cdot \left[\epsilon_\theta(\mathbf{x}_t, t, c) - \epsilon_\theta(\mathbf{x}_t, t, \varnothing)\right]}$$

CFG Sampling Formula

Sampling. Combine the two predictions at every step:

$$\boxed{\tilde{\epsilon}(\mathbf{x}_t, t, c) = \epsilon_\theta(\mathbf{x}_t, t, \varnothing) + w \cdot \big[\epsilon_\theta(\mathbf{x}_t, t, c) - \epsilon_\theta(\mathbf{x}_t, t, \varnothing)\big]}$$

Equivalently: $\tilde{\epsilon} = (1 + w)\, \epsilon_\theta(\mathbf{x}_t, t, c) - w\, \epsilon_\theta(\mathbf{x}_t, t, \varnothing)$.

CFG Sampling Formula

Sampling. Combine the two predictions at every step:

$$\boxed{\tilde{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, c) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing) + w \cdot \big[\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)\big]}$$

Equivalently: $\tilde{\boldsymbol{\epsilon}} = (1 + w)\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c) - w\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)$.

Plug $\tilde{\boldsymbol{\epsilon}}$ into any sampler. CFG is an inference-time modification; it composes cleanly with DDIM, DPM-Solver, or any other sampler. At each step s:

$$\hat{\mathbf{x}}_0 \leftarrow \big(\mathbf{x}_{\tau_s} - \sqrt{1 - \bar{\alpha}_{\tau_s}}\, \tilde{\boldsymbol{\epsilon}}\big) / \sqrt{\bar{\alpha}_{\tau_s}}, \qquad \mathbf{x}_{\tau_{s+1}} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\, \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{\tau_{s+1}}}\, \tilde{\boldsymbol{\epsilon}}.$$

CFG Sampling Formula

Sampling. Combine the two predictions at every step:

$$\boxed{\tilde{\boldsymbol{\epsilon}}(\mathbf{x}_t, t, c) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing) + w \cdot \big[\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c) - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)\big]}$$

Equivalently: $\tilde{\boldsymbol{\epsilon}} = (1 + w)\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c) - w\, \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)$.

Plug $\tilde{\boldsymbol{\epsilon}}$ into any sampler. CFG is an inference-time modification; it composes cleanly with DDIM, DPM-Solver, or any other sampler. At each step s:

$$\hat{\mathbf{x}}_0 \leftarrow \big(\mathbf{x}_{\tau_s} - \sqrt{1 - \bar{\alpha}_{\tau_s}}\, \tilde{\boldsymbol{\epsilon}}\big) / \sqrt{\bar{\alpha}_{\tau_s}}, \qquad \mathbf{x}_{\tau_{s+1}} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\, \hat{\mathbf{x}}_0 + \sqrt{1 - \bar{\alpha}_{\tau_{s+1}}}\, \tilde{\boldsymbol{\epsilon}}.$$

Cost. Two denoiser evaluations per step (conditional and unconditional). That is slower than unguided sampling, but still affordable once we are using fast samplers.

Why It Works: Extrapolation in Score Space

Using $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)/\sqrt{1-\bar{\alpha}_t}$ and similarly for $\log p(\mathbf{x} \mid c)$, the CFG direction is

$$\tilde{\boldsymbol{\epsilon}} \;\propto\; -\nabla_{\mathbf{x}} \log p(\mathbf{x}) - w\big[\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\big]$$

— extrapolation past the conditional score, in the direction away from the unconditional.



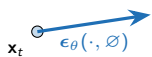
CFG extrapolates past the conditional prediction

Why It Works: Extrapolation in Score Space

Using $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)/\sqrt{1-\bar{\alpha}_t}$ and similarly for $\log p(\mathbf{x} \mid c)$, the CFG direction is

$$\tilde{\boldsymbol{\epsilon}} \;\propto\; -\nabla_{\mathbf{x}} \log p(\mathbf{x}) - w\big[\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\big]$$

— extrapolation past the conditional score, in the direction away from the unconditional.



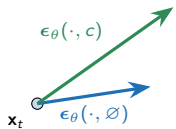
CFG extrapolates past the conditional prediction

Why It Works: Extrapolation in Score Space

Using $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)/\sqrt{1-\bar{\alpha}_t}$ and similarly for $\log p(\mathbf{x} \mid c)$, the CFG direction is

$$\tilde{\boldsymbol{\epsilon}} \;\propto\; -\nabla_{\mathbf{x}} \log p(\mathbf{x}) - w\big[\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\big]$$

— extrapolation past the conditional score, in the direction away from the unconditional.



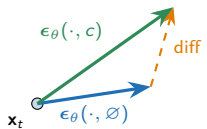
CFG extrapolates past the conditional prediction

Why It Works: Extrapolation in Score Space

Using $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)/\sqrt{1-\bar{\alpha}_t}$ and similarly for $\log p(\mathbf{x} \mid c)$, the CFG direction is

$$\tilde{\boldsymbol{\epsilon}} \;\propto\; -\nabla_{\mathbf{x}} \log p(\mathbf{x}) - w\big[\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\big]$$

— extrapolation past the conditional score, in the direction away from the unconditional.



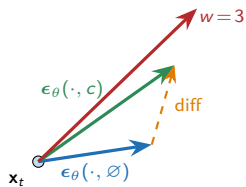
CFG extrapolates past the conditional prediction

Why It Works: Extrapolation in Score Space

Using $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)/\sqrt{1-\bar{\alpha}_t}$ and similarly for $\log p(\mathbf{x} \mid c)$, the CFG direction is

$$\tilde{\boldsymbol{\epsilon}} \;\propto\; -\nabla_{\mathbf{x}} \log p(\mathbf{x}) - w\big[\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\big]$$

— extrapolation past the conditional score, in the direction away from the unconditional.



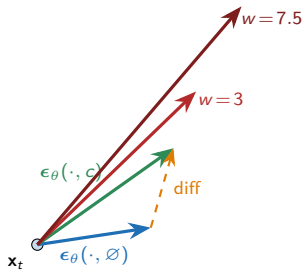
CFG extrapolates past the conditional prediction

Why It Works: Extrapolation in Score Space

Using $\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx -\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \varnothing)/\sqrt{1-\bar{\alpha}_t}$ and similarly for $\log p(\mathbf{x} \mid c)$, the CFG direction is

$$\tilde{\boldsymbol{\epsilon}} \;\propto\; -\nabla_{\mathbf{x}} \log p(\mathbf{x}) - w\big[\nabla_{\mathbf{x}} \log p(\mathbf{x} \mid c) - \nabla_{\mathbf{x}} \log p(\mathbf{x})\big]$$

— extrapolation past the conditional score, in the direction away from the unconditional.



$\tilde{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\varnothing + w(\boldsymbol{\epsilon}_c - \boldsymbol{\epsilon}_\varnothing)$

$w = 0$: *unconditional*
$w = 1$: *exactly* $\boldsymbol{\epsilon}_c$
$w > 1$: *extrapolate*

CFG extrapolates past the conditional prediction

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.
- $w = 0$: unconditional. $w \to \infty$: mode-seeking, adversarial artifacts.

Guidance Scale w: Concrete Tradeoff



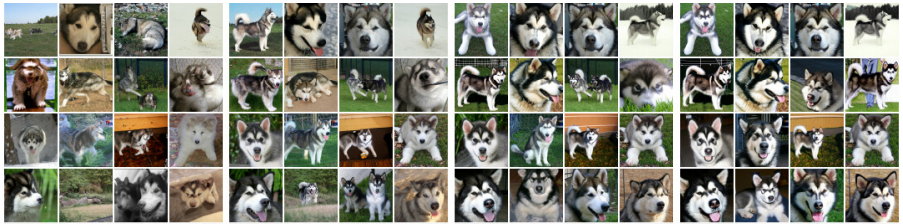
Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.
- $w = 0$: unconditional. $w \to \infty$: mode-seeking, adversarial artifacts.

Advantages over classifier guidance.

1. No separate classifier — one model, two modes.

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.
- $w = 0$: unconditional. $w \to \infty$: mode-seeking, adversarial artifacts.

Advantages over classifier guidance.

1. No separate classifier — one model, two modes.
2. Works for any c (text, image, audio) without changing machinery.

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.
- $w = 0$: unconditional. $w \to \infty$: mode-seeking, adversarial artifacts.

Advantages over classifier guidance.

1. No separate classifier — one model, two modes.
2. Works for any c (text, image, audio) without changing machinery.
3. Swap c at inference: zero retraining.

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.
- $w = 0$: unconditional. $w \to \infty$: mode-seeking, adversarial artifacts.

Advantages over classifier guidance.

1. No separate classifier — one model, two modes.
2. Works for any c (text, image, audio) without changing machinery.
3. Swap c at inference: zero retraining.

Caveat. For $w > 1$ this is an extrapolation trick, not exact conditional sampling. In practice it is very effective.

Guidance Scale w: Concrete Tradeoff



Left: no guidance ($w = 1$) – high *diversity*, low *fidelity*. **Right:** heavy guidance ($w = 7.5$) – low diversity, high *fidelity*.
Source: Ho & Salimans 2022 (CFG), Figure 1.

- Typical in practice: $w \approx 7$–8 for Stable Diffusion text-to-image.
- $w = 0$: unconditional. $w \to \infty$: mode-seeking, adversarial artifacts.

Advantages over classifier guidance.

1. No separate classifier — one model, two modes.
2. Works for any c (text, image, audio) without changing machinery.
3. Swap c at inference: zero retraining.

Caveat. For $w > 1$ this is an extrapolation trick, not exact conditional sampling. In practice it is very effective.

Problem 2 (Control) resolved. ✓ CFG gives text/class conditioning from a single model.

Mid-lecture reset

Problem	Solution	Key formula / idea	Status
Speed	DDIM / DPM-Solver	non-Markovian; ODE Euler	✓

Mid-lecture reset

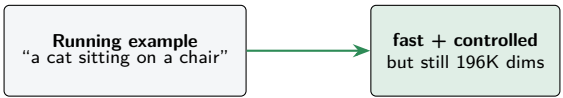
Problem	Solution	Key formula / idea	Status
Speed	DDIM / DPM-Solver	non-Markovian; ODE Euler	✓
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_{\varnothing} + w(\epsilon_c - \epsilon_{\varnothing})$	✓

Mid-lecture reset

Problem	Solution	Key formula / idea	Status
Speed	DDIM / DPM-Solver	non-Markovian; ODE Euler	✓
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_{\varnothing} + w(\epsilon_c - \epsilon_{\varnothing})$	✓
Scale	Latent diffusion	coming now	○

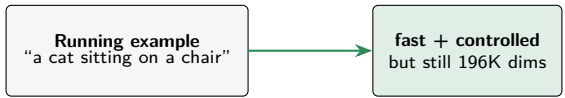
Mid-lecture reset

Problem	Solution	Key formula / idea	Status
Speed	DDIM / DPM-Solver	non-Markovian; ODE Euler	✓
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\varnothing + w(\epsilon_c - \epsilon_\varnothing)$	✓
Scale	Latent diffusion	coming now	○



Mid-lecture reset

Problem	Solution	Key formula / idea	Status
Speed	DDIM / DPM-Solver	non-Markovian; ODE Euler	✓
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\varnothing + w(\epsilon_c - \epsilon_\varnothing)$	✓
Scale	Latent diffusion	coming now	○



Remaining question. Diffusing in pixel space is expensive. Can we compress to a smaller space first?

Latent Diffusion

Why Pixel Space Is Wasteful

- $256 \times 256 \times 3 \approx 196{,}608$ dimensions per image.

Why Pixel Space Is Wasteful

- $256 \times 256 \times 3 \approx 196{,}608$ dimensions per image.
- Most of this is perceptually redundant: high-frequency texture, imperceptible color variation.

Why Pixel Space Is Wasteful

- $256 \times 256 \times 3 \approx 196{,}608$ dimensions per image.
- Most of this is perceptually redundant: high-frequency texture, imperceptible color variation.
- DDPM at 256^2: each U-Net pass is expensive; $\times 1000$ steps $=$ prohibitive at scale.

Why Pixel Space Is Wasteful

- $256 \times 256 \times 3 \approx 196{,}608$ dimensions per image.
- Most of this is perceptually redundant: high-frequency texture, imperceptible color variation.
- DDPM at 256^2: each U-Net pass is expensive; $\times 1000$ steps $=$ prohibitive at scale.

Rombach et al. (2022) insight:

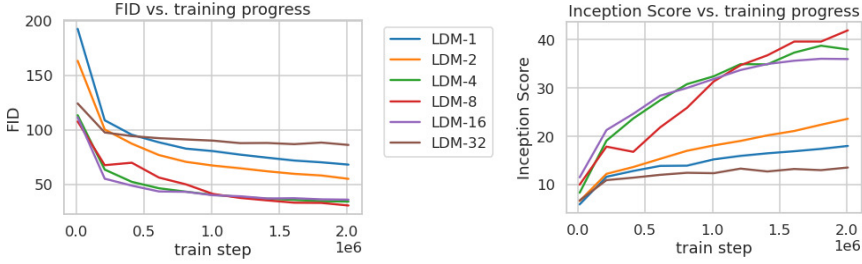
Diffusion doesn't need to work in pixel space.
It only needs a space that preserves perceptually important structure.

Why Pixel Space Is Wasteful

- $256 \times 256 \times 3 \approx 196{,}608$ dimensions per image.
- Most of this is perceptually redundant: high-frequency texture, imperceptible color variation.
- DDPM at 256^2: each U-Net pass is expensive; $\times 1000$ steps = prohibitive at scale.

Rombach et al. (2022) insight:

Diffusion doesn't need to work in pixel space.
It only needs a space that preserves perceptually important structure.



Left: FID (lower is better) vs. training steps. **Right:** Inception Score (higher is better) vs. training steps. Downsampling factors $f = 1$ (pixel space) to $f = 32$. LDM-8 matches LDM-1 in FID and beats it in IS – a good sweet spot.
Source: Rombach et al. 2022 (LDM), Figure 6.

VAE Recap: Latent Space as a Perceptual Bottleneck

Standard VAE training objective (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{x} \mid \mathbf{z})\big]}_{\text{reconstruction}} - D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big).$$

The encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ compresses $\mathbf{x}$ into a low-dimensional posterior; the decoder $p_\psi(\mathbf{x} \mid \mathbf{z})$ reconstructs from $\mathbf{z}$; the KL term regularizes the posterior toward a standard Gaussian so that sampling $\mathbf{z} \sim \mathcal{N}(0, I)$ makes sense.

VAE Recap: Latent Space as a Perceptual Bottleneck

Standard VAE training objective (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{x} \mid \mathbf{z})\big]}_{\text{reconstruction}} - D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big).$$

The encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ compresses $\mathbf{x}$ into a low-dimensional posterior; the decoder $p_\psi(\mathbf{x} \mid \mathbf{z})$ reconstructs from $\mathbf{z}$; the KL term regularizes the posterior toward a standard Gaussian so that sampling $\mathbf{z} \sim \mathcal{N}(0, I)$ makes sense.

What the bottleneck forces. Because $\mathbf{z}$ is much smaller than $\mathbf{x}$, the encoder must discard low-level noise and preserve structure that the decoder needs to reconstruct faithfully:

- coarse shape and layout (needed to reconstruct at all)

VAE Recap: Latent Space as a Perceptual Bottleneck

Standard VAE training objective (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{x} \mid \mathbf{z})\big]}_{\text{reconstruction}} - D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big).$$

The encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ compresses $\mathbf{x}$ into a low-dimensional posterior; the decoder $p_\psi(\mathbf{x} \mid \mathbf{z})$ reconstructs from $\mathbf{z}$; the KL term regularizes the posterior toward a standard Gaussian so that sampling $\mathbf{z} \sim \mathcal{N}(0, I)$ makes sense.

What the bottleneck forces. Because $\mathbf{z}$ is much smaller than $\mathbf{x}$, the encoder must discard low-level noise and preserve structure that the decoder needs to reconstruct faithfully:

- coarse shape and layout (needed to reconstruct at all)
- color statistics and texture (costly to lose)

VAE Recap: Latent Space as a Perceptual Bottleneck

Standard VAE training objective (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{x} \mid \mathbf{z})\big]}_{\text{reconstruction}} - D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big).$$

The encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ compresses $\mathbf{x}$ into a low-dimensional posterior; the decoder $p_\psi(\mathbf{x} \mid \mathbf{z})$ reconstructs from $\mathbf{z}$; the KL term regularizes the posterior toward a standard Gaussian so that sampling $\mathbf{z} \sim \mathcal{N}(0, I)$ makes sense.

What the bottleneck forces. Because $\mathbf{z}$ is much smaller than $\mathbf{x}$, the encoder must discard low-level noise and preserve structure that the decoder needs to reconstruct faithfully:

- coarse shape and layout (needed to reconstruct at all)
- color statistics and texture (costly to lose)
- fine pixel noise and imperceptible high-frequency variation $\rightarrow$ discarded

VAE Recap: Latent Space as a Perceptual Bottleneck

Standard VAE training objective (ELBO):

$$\mathcal{L}_{\text{ELBO}} = \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\psi(\mathbf{x} \mid \mathbf{z})\big]}_{\text{reconstruction}} - D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big).$$

The encoder $q_\phi(\mathbf{z} \mid \mathbf{x})$ compresses $\mathbf{x}$ into a low-dimensional posterior; the decoder $p_\psi(\mathbf{x} \mid \mathbf{z})$ reconstructs from $\mathbf{z}$; the KL term regularizes the posterior toward a standard Gaussian so that sampling $\mathbf{z} \sim \mathcal{N}(0, I)$ makes sense.

What the bottleneck forces. Because $\mathbf{z}$ is much smaller than $\mathbf{x}$, the encoder must discard low-level noise and preserve structure that the decoder needs to reconstruct faithfully:

- coarse shape and layout (needed to reconstruct at all)
- color statistics and texture (costly to lose)
- fine pixel noise and imperceptible high-frequency variation $\rightarrow$ discarded

Why this is exactly the space we want to diffuse in. Diffusion needs to model structure that matters perceptually. The VAE latent $\mathbf{z}$ concentrates that structure compactly, removing the redundant dimensions that make pixel-space diffusion expensive.

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stable Diffusion: $f = 8$, $c_z = 4$. $256\times256\times3 \to 32\times32\times4$: **48**$\times$ **fewer dimensions**.

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stable Diffusion: $f = 8$, $c_z = 4$. $\quad 256\times256\times3 \to 32\times32\times4$: **48**× **fewer dimensions**.

Training objective. The standard VAE uses pixel MSE as the reconstruction term, treating all pixels equally. This VAE replaces it with a *perceptual* reconstruction loss:

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathcal{L}_{\text{perceptual}}\big(\mathbf{x},\, \mathcal{D}(\mathcal{E}(\mathbf{x}))\big)}_{\text{feature-space distance}} + \lambda\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big), \quad \lambda \ll 1.$$

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stable Diffusion: $f = 8$, $c_z = 4$. $\quad 256\times256\times3 \to 32\times32\times4$: **48× fewer dimensions**.

Training objective. The standard VAE uses pixel MSE as the reconstruction term, treating all pixels equally. This VAE replaces it with a *perceptual* reconstruction loss:

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathcal{L}_{\text{perceptual}}\big(\mathbf{x},\, \mathcal{D}(\mathcal{E}(\mathbf{x}))\big)}_{\text{feature-space distance}} + \lambda\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big), \quad \lambda \ll 1.$$

What is $\mathcal{L}_{\text{perceptual}}$? Pass both $\mathbf{x}$ and $\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$ through a pretrained VGG-16 network frozen at its convolutional features. Compute the ℓ_2 distance between their intermediate feature maps (e.g. `relu2_2`, `relu3_3`):

$$\mathcal{L}_{\text{perceptual}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{\ell} w_\ell \, \|\phi_\ell(\mathbf{x}) - \phi_\ell(\tilde{\mathbf{x}})\|_2^2.$$

These features respond to edges, textures, and object parts. Matching them yields reconstructions that look sharp to a human even when pixel values differ slightly. This is the **LPIPS** metric *(Zhang et al., CVPR 2018)*.

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stable Diffusion: $f = 8$, $c_z = 4$. $\quad 256\times256\times3 \to 32\times32\times4$: **48× fewer dimensions**.

Training objective. The standard VAE uses pixel MSE as the reconstruction term, treating all pixels equally. This VAE replaces it with a *perceptual* reconstruction loss:

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathcal{L}_{\text{perceptual}}(\mathbf{x},\, \mathcal{D}(\mathcal{E}(\mathbf{x})))}_{\text{feature-space distance}} + \lambda\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big), \quad \lambda \ll 1.$$

What is $\mathcal{L}_{\text{perceptual}}$? Pass both $\mathbf{x}$ and $\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$ through a pretrained VGG-16 network frozen at its convolutional features. Compute the ℓ_2 distance between their intermediate feature maps (e.g. `relu2_2`, `relu3_3`):

$$\mathcal{L}_{\text{perceptual}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{\ell} w_\ell \, \|\phi_\ell(\mathbf{x}) - \phi_\ell(\tilde{\mathbf{x}})\|_2^2.$$

These features respond to edges, textures, and object parts. Matching them yields reconstructions that look sharp to a human even when pixel values differ slightly. This is the **LPIPS** metric *(Zhang et al., CVPR 2018)*.

Why the tiny KL weight?

- Standard VAE: large KL weight forces $p(\mathbf{z}) \approx \mathcal{N}(0, I)$ so the prior can be sampled at test time.

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stable Diffusion: $f = 8$, $c_z = 4$. $\quad 256\times256\times3 \to 32\times32\times4$: **48× fewer dimensions**.

Training objective. The standard VAE uses pixel MSE as the reconstruction term, treating all pixels equally. This VAE replaces it with a *perceptual* reconstruction loss:

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathcal{L}_{\text{perceptual}}(\mathbf{x},\, \mathcal{D}(\mathcal{E}(\mathbf{x})))}_{\text{feature-space distance}} + \lambda\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big), \quad \lambda \ll 1.$$

What is $\mathcal{L}_{\text{perceptual}}$? Pass both $\mathbf{x}$ and $\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$ through a pretrained VGG-16 network frozen at its convolutional features. Compute the ℓ_2 distance between their intermediate feature maps (e.g. `relu2_2`, `relu3_3`):

$$\mathcal{L}_{\text{perceptual}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{\ell} w_\ell \, \|\phi_\ell(\mathbf{x}) - \phi_\ell(\tilde{\mathbf{x}})\|_2^2.$$

These features respond to edges, textures, and object parts. Matching them yields reconstructions that look sharp to a human even when pixel values differ slightly. This is the **LPIPS** metric *(Zhang et al., CVPR 2018)*.

Why the tiny KL weight?

- Standard VAE: large KL weight forces $p(\mathbf{z}) \approx \mathcal{N}(0, I)$ so the prior can be sampled at test time.

Stage 1: VAE for Perceptual Compression

Trained first, then frozen. The VAE encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained *independently* from the diffusion model; frozen afterward.

$$\mathbf{z} = \mathcal{E}(\mathbf{x}) \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}, \qquad \tilde{\mathbf{x}} = \mathcal{D}(\mathbf{z}).$$

Stable Diffusion: $f = 8$, $c_z = 4$. $\quad 256\times256\times3 \to 32\times32\times4$: **48× fewer dimensions**.

Training objective. The standard VAE uses pixel MSE as the reconstruction term, treating all pixels equally. This VAE replaces it with a *perceptual* reconstruction loss:

$$\mathcal{L}_{\text{VAE}} = \underbrace{\mathcal{L}_{\text{perceptual}}(\mathbf{x},\, \mathcal{D}(\mathcal{E}(\mathbf{x})))}_{\text{feature-space distance}} + \lambda\, D_{\text{KL}}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, \mathcal{N}(0, I)\big), \quad \lambda \ll 1.$$

What is $\mathcal{L}_{\text{perceptual}}$? Pass both $\mathbf{x}$ and $\tilde{\mathbf{x}} = \mathcal{D}(\mathcal{E}(\mathbf{x}))$ through a pretrained VGG-16 network frozen at its convolutional features. Compute the ℓ_2 distance between their intermediate feature maps (e.g. `relu2_2`, `relu3_3`):

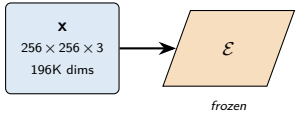
$$\mathcal{L}_{\text{perceptual}}(\mathbf{x}, \tilde{\mathbf{x}}) = \sum_{\ell} w_\ell \, \|\phi_\ell(\mathbf{x}) - \phi_\ell(\tilde{\mathbf{x}})\|_2^2.$$

These features respond to edges, textures, and object parts. Matching them yields reconstructions that look sharp to a human even when pixel values differ slightly. This is the **LPIPS** metric *(Zhang et al., CVPR 2018)*.

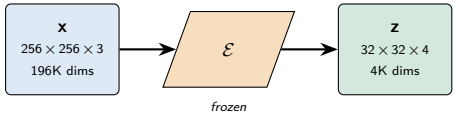
Why the tiny KL weight?

- Standard VAE: large KL weight forces $p(\mathbf{z}) \approx \mathcal{N}(0, I)$ so the prior can be sampled at test time.

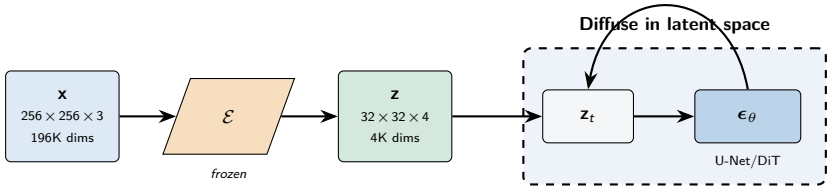
LDM Architecture: Two-Stage Pipeline



LDM Architecture: Two-Stage Pipeline

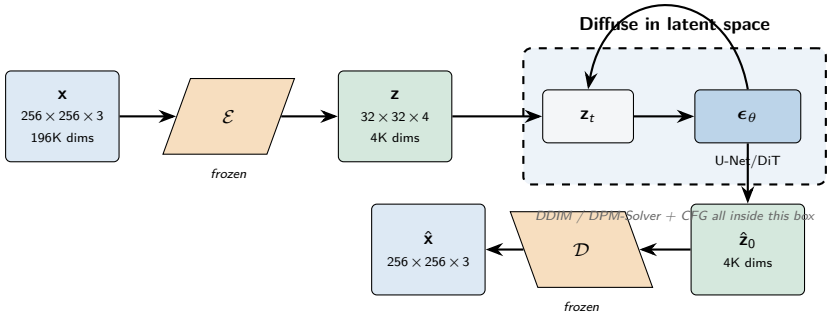


LDM Architecture: Two-Stage Pipeline

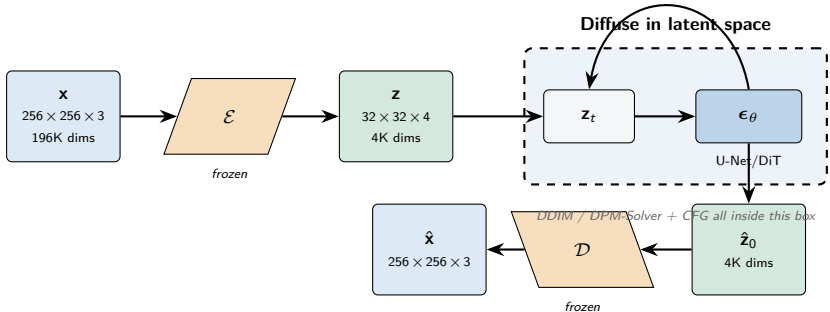


DDIM / DPM-Solver + CFG all inside this box

LDM Architecture: Two-Stage Pipeline



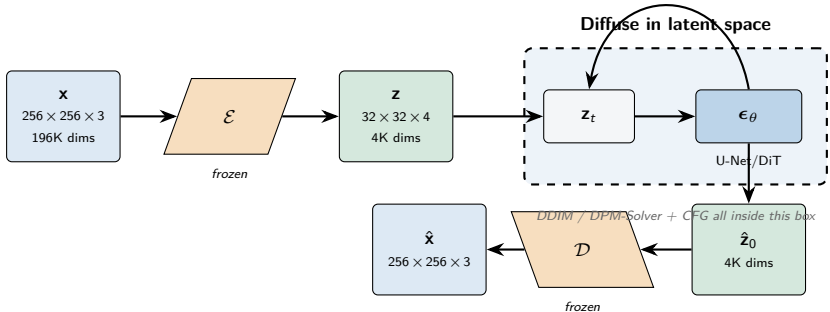
LDM Architecture: Two-Stage Pipeline



Stage 2 objective. Identical ϵ-prediction loss, just on $\mathbf{z}_t$:

$$\mathcal{L}_{\text{LDM}} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\big[\,\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, c)\|^2\,\big], \quad \mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0), \quad \mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}.$$

LDM Architecture: Two-Stage Pipeline



Stage 2 objective. Identical ϵ-prediction loss, just on $\mathbf{z}_t$:

$$\mathcal{L}_{\text{LDM}} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\big[\,\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t, c)\|^2\,\big], \quad \mathbf{z}_0 = \mathcal{E}(\mathbf{x}_0), \quad \mathbf{z}_t = \sqrt{\bar{\alpha}_t}\mathbf{z}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}.$$

Sampling: DDIM on $\mathbf{z}$, then one decoder pass $\hat{\mathbf{x}} = \mathcal{D}(\hat{\mathbf{z}}_0)$.

The U-Net Denoiser with Cross-Attention

U-Net structure (recall). ResBlocks with GroupNorm + SiLU; sinusoidal time embedding added at each block; encoder-decoder with skip connections at multiple spatial resolutions.

The U-Net Denoiser with Cross-Attention

U-Net structure (recall). ResBlocks with GroupNorm + SiLU; sinusoidal time embedding added at each block; encoder-decoder with skip connections at multiple spatial resolutions.

Text conditioning via cross-attention (Rombach et al., 2022). A frozen text encoder τ_θ (e.g., CLIP—Contrastive Language–Image Pretraining; see Appendix C) maps the prompt c to a sequence of tokens:

$$\tau_\theta(c) \in \mathbb{R}^{L_{\text{seq}} \times d_{\text{text}}}.$$

At selected resolutions in the U-Net, a cross-attention layer is inserted *after* each ResBlock. Let $\phi_i(\mathbf{z}_t) \in \mathbb{R}^{h_i w_i \times d_i}$ denote the **flattened spatial feature map** at U-Net level i: for each of the $h_i \times w_i$ spatial positions, a channel vector of dimension d_i.

The U-Net Denoiser with Cross-Attention

U-Net structure (recall). ResBlocks with GroupNorm + SiLU; sinusoidal time embedding added at each block; encoder-decoder with skip connections at multiple spatial resolutions.

Text conditioning via cross-attention (Rombach et al., 2022). A frozen text encoder τ_θ (e.g., CLIP—Contrastive Language–Image Pretraining; see Appendix C) maps the prompt c to a sequence of tokens:

$$\tau_\theta(c) \in \mathbb{R}^{L_{\text{seq}} \times d_{\text{text}}}.$$

At selected resolutions in the U-Net, a cross-attention layer is inserted *after* each ResBlock. Let $\phi_i(\mathbf{z}_t) \in \mathbb{R}^{h_i w_i \times d_i}$ denote the **flattened spatial feature map** at U-Net level i: for each of the $h_i \times w_i$ spatial positions, a channel vector of dimension d_i.
The cross-attention queries come from the spatial features; keys and values come from the text tokens:

$$Q = W_Q\, \phi_i(\mathbf{z}_t), \quad K = W_K\, \tau_\theta(c), \quad V = W_V\, \tau_\theta(c).$$

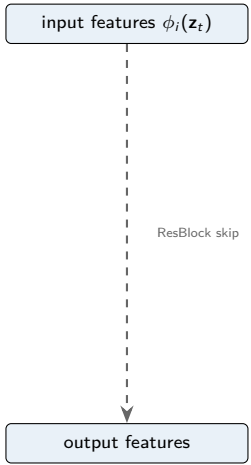
The output is added back **residually** to the spatial features:

$$\phi_i(\mathbf{z}_t) \;\leftarrow\; \phi_i(\mathbf{z}_t) + \mathrm{softmax}\!\left(\frac{QK^\top}{\sqrt{d_k}}\right) V.$$

Each spatial position attends over all text tokens and updates its channel vector accordingly. The residual ensures the update is an incremental conditioning, not a full replacement.

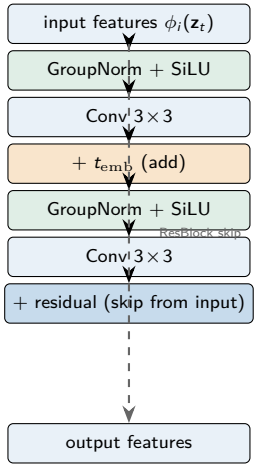
Cross-Attention in the U-Net: Block Structure

Where cross-attention sits inside a single U-Net block at a conditioned resolution:



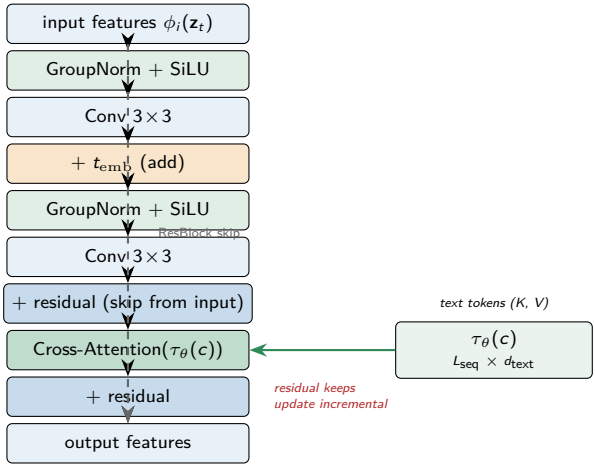
Cross-Attention in the U-Net: Block Structure

Where cross-attention sits inside a single U-Net block at a conditioned resolution:



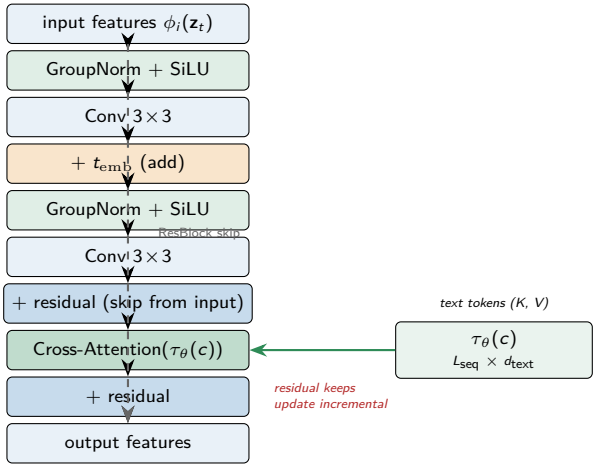
Cross-Attention in the U-Net: Block Structure

Where cross-attention sits inside a single U-Net block at a conditioned resolution:



Cross-Attention in the U-Net: Block Structure

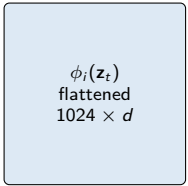
Where cross-attention sits inside a single U-Net block at a conditioned resolution:



At each spatial location, the attention row answers: *which words in the prompt matter here?* The residual ensures the spatial feature retains its prior content and receives the text-conditioned update additively.

Cross-Attention, Concretely

Queries (spatial)



Cross-Attention, Concretely

Queries (spatial)

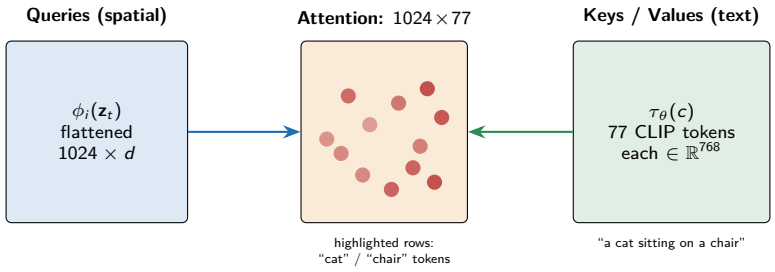
$\phi_i(\mathbf{z}_t)$
flattened
$1024 \times d$

Keys / Values (text)

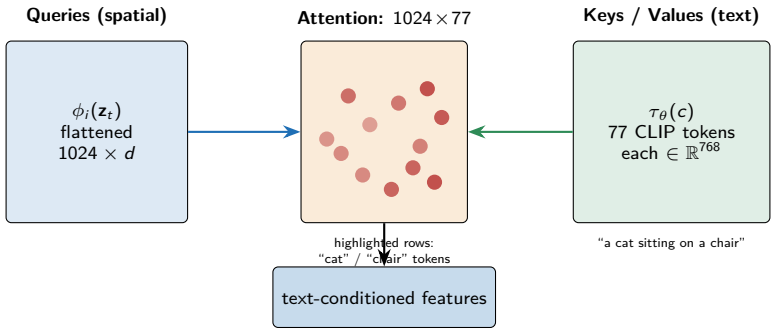
$\tau_\theta(c)$
77 CLIP tokens
each $\in \mathbb{R}^{768}$

"a cat sitting on a chair"

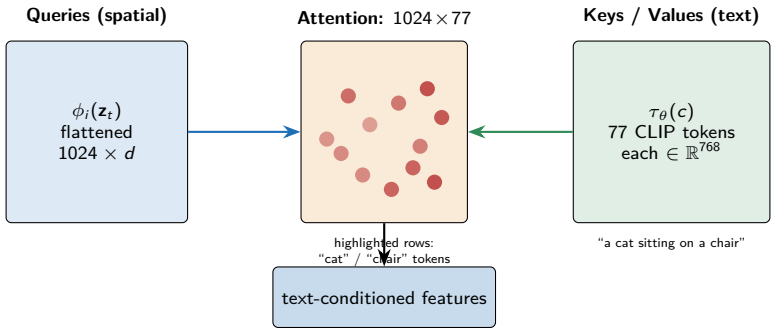
Cross-Attention, Concretely



Cross-Attention, Concretely

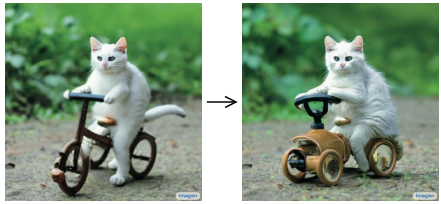


Cross-Attention, Concretely



Each row of the 1024×77 attention matrix is a distribution over prompt tokens: *at this spatial location, which words matter most?*

Cross-Attention in Practice

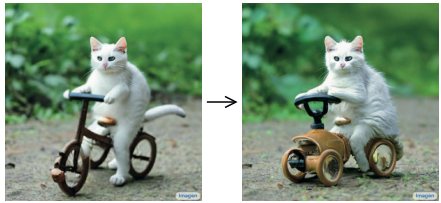


"Photo of a cat riding on a bicycle."

car

Source: Hertz et al. 2022 (Prompt-to-Prompt), Figure 1.

Cross-Attention in Practice



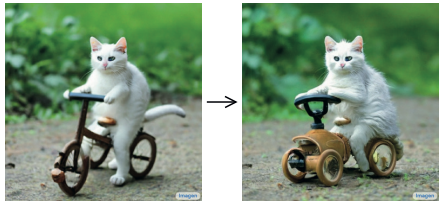
"Photo of a cat riding on a bicycle." car

Source: Hertz et al. 2022 (Prompt-to-Prompt), Figure 1.

Key downstream applications.

- *Prompt-to-Prompt editing:* sampling still runs, but reusing attention maps for unchanged words keeps the new sample close to the old one while editing the changed region.

Cross-Attention in Practice



"Photo of a cat riding on a bicycle."

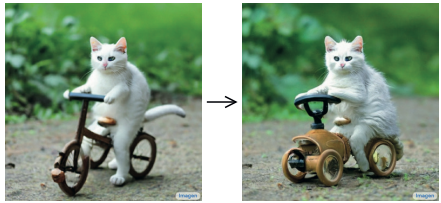
car

Source: Hertz et al. 2022 (Prompt-to-Prompt), Figure 1.

Key downstream applications.

- *Prompt-to-Prompt editing:* sampling still runs, but reusing attention maps for unchanged words keeps the new sample close to the old one while editing the changed region.
- *Inpainting and region control:* mask attention rows to force certain regions to attend to specific prompt tokens.

Cross-Attention in Practice



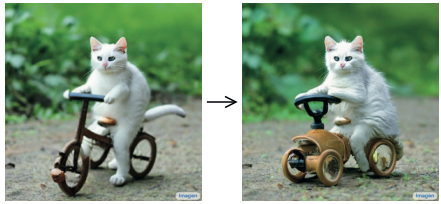
"Photo of a cat riding on a ~~bicycle~~ car."

Source: Hertz et al. 2022 (Prompt-to-Prompt), Figure 1.

Key downstream applications.

- *Prompt-to-Prompt editing:* sampling still runs, but reusing attention maps for unchanged words keeps the new sample close to the old one while editing the changed region.
- *Inpainting and region control:* mask attention rows to force certain regions to attend to specific prompt tokens.
- *ControlNet / IP-Adapter:* additional cross-attention inputs beyond text (depth, pose, reference image).

Cross-Attention in Practice



"Photo of a cat riding on a bicycle." car

Source: Hertz et al. 2022 (Prompt-to-Prompt), Figure 1.

Key downstream applications.

- *Prompt-to-Prompt editing:* sampling still runs, but reusing attention maps for unchanged words keeps the new sample close to the old one while editing the changed region.
- *Inpainting and region control:* mask attention rows to force certain regions to attend to specific prompt tokens.
- *ControlNet / IP-Adapter:* additional cross-attention inputs beyond text (depth, pose, reference image).

Problem 3 (Scale) resolved. ✓ Diffuse in a $32 \times 32 \times 4$ latent; decode once at the end.

DiT

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Central problem for scaling. U-Net compute $\rightarrow$ FID does *not* follow a clean power law. Hard to predict which variant will be optimal at larger scale — you must run many ablations.

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Central problem for scaling. U-Net compute $\rightarrow$ FID does *not* follow a clean power law. Hard to predict which variant will be optimal at larger scale — you must run many ablations.

Contrast. We saw earlier in this course that transformers follow clean power-law scaling with compute (Kaplan et al., 2020). This is bankable: you predict returns before training.

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Central problem for scaling. U-Net compute $\rightarrow$ FID does *not* follow a clean power law. Hard to predict which variant will be optimal at larger scale — you must run many ablations.

Contrast. We saw earlier in this course that transformers follow clean power-law scaling with compute (Kaplan et al., 2020). This is bankable: you predict returns before training.

What would a scalable denoiser need?

1. Standard transformer so existing scaling insights transfer.

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Central problem for scaling. U-Net compute $\rightarrow$ FID does *not* follow a clean power law. Hard to predict which variant will be optimal at larger scale — you must run many ablations.

Contrast. We saw earlier in this course that transformers follow clean power-law scaling with compute (Kaplan et al., 2020). This is bankable: you predict returns before training.

What would a scalable denoiser need?

1. Standard transformer so existing scaling insights transfer.
2. Efficient conditioning on both timestep t and condition c.

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Central problem for scaling. U-Net compute $\rightarrow$ FID does *not* follow a clean power law. Hard to predict which variant will be optimal at larger scale — you must run many ablations.

Contrast. We saw earlier in this course that transformers follow clean power-law scaling with compute (Kaplan et al., 2020). This is bankable: you predict returns before training.

What would a scalable denoiser need?

1. Standard transformer so existing scaling insights transfer.
2. Efficient conditioning on both timestep t and condition c.
3. Handles 2D spatial structure (images are not free token sequences).

Why Replace the U-Net?

U-Net's inductive biases are a tradeoff.

Inductive bias	Benefit	Cost
Local convolutions	Parameter efficiency, translation equivariance	Limited long-range interaction at coarse scales
Multi-scale hierarchy	Captures structure at multiple resolutions	Fixed architecture; hard to add scale cleanly
ResBlocks + GroupNorm	Stable training	Irregular FLOP scaling as model grows

Central problem for scaling. U-Net compute $\rightarrow$ FID does *not* follow a clean power law. Hard to predict which variant will be optimal at larger scale — you must run many ablations.

Contrast. We saw earlier in this course that transformers follow clean power-law scaling with compute (Kaplan et al., 2020). This is bankable: you predict returns before training.

What would a scalable denoiser need?

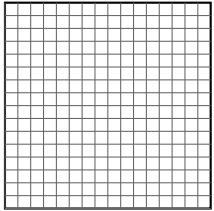
1. Standard transformer so existing scaling insights transfer.
2. Efficient conditioning on both timestep t and condition c.
3. Handles 2D spatial structure (images are not free token sequences).

Peebles & Xie (2023), "Scalable Diffusion Models with Transformers" (DiT) is the standard answer.

DiT Step 1: Patchify the Latent

Input: $\mathbf{z}_t \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$. Example: $32\times32\times4$.

$\mathbf{z}_t$: $32\times32\times4$



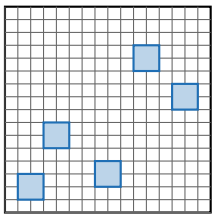
DiT Step 1: Patchify the Latent

Input: $\mathbf{z}_t \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$. Example: $32\times32\times4$.

Divide into non-overlapping $p\times p$ patches (typically $p=2$). Each patch spans $p\times p$ spatial positions and all c_z channels, giving a raw vector of p^2c_z values per patch (for $p=2$, $c_z=4$: $2\times2\times4=16$ values). A learned linear projection maps this to $\mathbb{R}^d$. Add 2D sinusoidal positional embeddings.

$$N=\left(\frac{H/f}{p}\right)^2=\left(\frac{32}{2}\right)^2=256 \quad \text{tokens, each in } \mathbb{R}^d.$$

$\mathbf{z}_t$: $32\times32\times4$



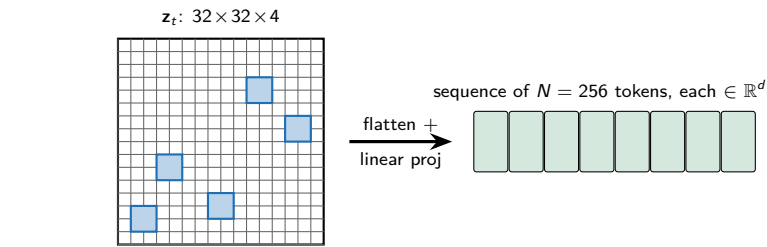
non-overlapping $2\times2\times c_z$ patches ($=2\times2\times4$ here)

DiT Step 1: Patchify the Latent

Input: $\mathbf{z}_t \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$. Example: $32\times32\times4$.

Divide into non-overlapping $p\times p$ patches (typically $p=2$). Each patch spans $p\times p$ spatial positions and all c_z channels, giving a raw vector of $p^2 c_z$ values per patch (for $p=2$, $c_z=4$: $2\times2\times4=16$ values). A learned linear projection maps this to $\mathbb{R}^d$. Add 2D sinusoidal positional embeddings.

$$N = \left(\frac{H/f}{p}\right)^2 = \left(\frac{32}{2}\right)^2 = 256 \quad \text{tokens, each in } \mathbb{R}^d.$$

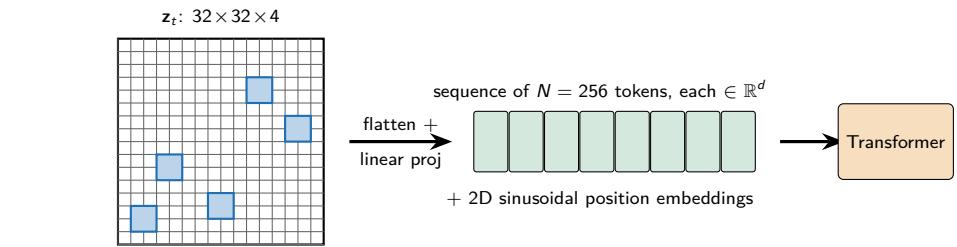


DiT Step 1: Patchify the Latent

Input: $\mathbf{z}_t \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$. Example: $32\times32\times4$.

Divide into non-overlapping $p\times p$ patches (typically $p=2$). Each patch spans $p\times p$ spatial positions and all c_z channels, giving a raw vector of $p^2 c_z$ values per patch (for $p=2$, $c_z=4$: $2\times2\times4=16$ values). A learned linear projection maps this to $\mathbb{R}^d$. Add 2D sinusoidal positional embeddings.

$$N = \left(\frac{H/f}{p}\right)^2 = \left(\frac{32}{2}\right)^2 = 256 \quad \text{tokens, each in } \mathbb{R}^d.$$

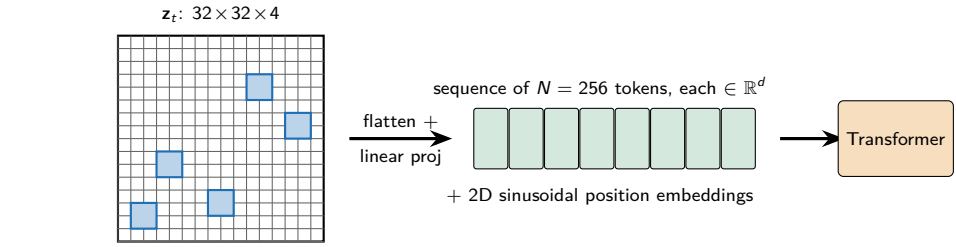


DiT Step 1: Patchify the Latent

Input: $\mathbf{z}_t \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$. Example: $32\times32\times4$.

Divide into non-overlapping $p\times p$ patches (typically $p=2$). Each patch spans $p\times p$ spatial positions and all c_z channels, giving a raw vector of p^2c_z values per patch (for $p=2$, $c_z=4$: $2\times2\times4=16$ values). A learned linear projection maps this to $\mathbb{R}^d$. Add 2D sinusoidal positional embeddings.

$$N = \left(\frac{H/f}{p}\right)^2 = \left(\frac{32}{2}\right)^2 = 256 \quad \text{tokens, each in } \mathbb{R}^d.$$



Contrast to U-Net. U-Net processes the full spatial grid with local convolutions. DiT treats every patch as a token with *global* self-attention — all N^2 pairwise interactions at every layer.

DiT Step 2: Conditioning via adaLN-Zero

Core question. How to inject t and c into each transformer block?

Standard transformer block:

$$\mathbf{x}' = \mathbf{x} + \mathrm{Attn}(\mathrm{LN}(\mathbf{x})), \qquad \mathbf{x}'' = \mathbf{x}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{x}')).$$

DiT Step 2: Conditioning via adaLN-Zero

Core question. How to inject t and c into each transformer block?

Standard transformer block:

$$\mathbf{x}' = \mathbf{x} + \mathrm{Attn}(\mathrm{LN}(\mathbf{x})), \qquad \mathbf{x}'' = \mathbf{x}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{x}')).$$

DiT: adaLN. Replace fixed LN scale/shift (γ, β) with values predicted from conditioning by a shallow 2-layer MLP (linear $\to$ SiLU $\to$ linear):

$$(\gamma_1, \beta_1, \alpha_1, \gamma_2, \beta_2, \alpha_2) = W_2\,\mathrm{SiLU}(W_1(t_{\mathrm{emb}} + c_{\mathrm{emb}})).$$

Here $c_{\mathrm{emb}} \in \mathbb{R}^d$ is a learned embedding for the class label (a single integer for class-conditional ImageNet generation; contrast with the U-Net's sequence of text tokens via cross-attention).

DiT Step 2: Conditioning via adaLN-Zero

Core question. How to inject t and c into each transformer block?

Standard transformer block:

$$\mathbf{x}' = \mathbf{x} + \mathrm{Attn}(\mathrm{LN}(\mathbf{x})), \qquad \mathbf{x}'' = \mathbf{x}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{x}')).$$

DiT: adaLN. Replace fixed LN scale/shift (γ, β) with values predicted from conditioning by a shallow 2-layer MLP (linear $\to$ SiLU $\to$ linear):

$$(\gamma_1, \beta_1, \alpha_1, \gamma_2, \beta_2, \alpha_2) = W_2\,\mathrm{SiLU}(W_1(t_{\mathrm{emb}} + c_{\mathrm{emb}})).$$

Here $c_{\mathrm{emb}} \in \mathbb{R}^d$ is a learned embedding for the class label (a single integer for class-conditional ImageNet generation; contrast with the U-Net's sequence of text tokens via cross-attention).

$$\mathbf{x}' = \mathbf{x} + \alpha_1 \cdot \mathrm{Attn}\big(\mathrm{adaLN}(\mathbf{x}; \gamma_1, \beta_1)\big), \qquad \mathbf{x}'' = \mathbf{x}' + \alpha_2 \cdot \mathrm{FFN}\big(\mathrm{adaLN}(\mathbf{x}'; \gamma_2, \beta_2)\big).$$

The α_k are learned *residual scale factors*: they modulate how much each block transforms the input.

DiT Step 2: Conditioning via adaLN-Zero

Core question. How to inject t and c into each transformer block?

Standard transformer block:

$$\mathbf{x}' = \mathbf{x} + \mathrm{Attn}(\mathrm{LN}(\mathbf{x})), \qquad \mathbf{x}'' = \mathbf{x}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{x}')).$$

DiT: adaLN. Replace fixed LN scale/shift (γ, β) with values predicted from conditioning by a shallow 2-layer MLP (linear $\to$ SiLU $\to$ linear):

$$(\gamma_1, \beta_1, \alpha_1, \gamma_2, \beta_2, \alpha_2) = W_2\,\mathrm{SiLU}(W_1(t_{\mathrm{emb}} + c_{\mathrm{emb}})).$$

Here $c_{\mathrm{emb}} \in \mathbb{R}^d$ is a learned embedding for the class label (a single integer for class-conditional ImageNet generation; contrast with the U-Net's sequence of text tokens via cross-attention).

$$\mathbf{x}' = \mathbf{x} + \alpha_1 \cdot \mathrm{Attn}\big(\mathrm{adaLN}(\mathbf{x}; \gamma_1, \beta_1)\big), \qquad \mathbf{x}'' = \mathbf{x}' + \alpha_2 \cdot \mathrm{FFN}\big(\mathrm{adaLN}(\mathbf{x}'; \gamma_2, \beta_2)\big).$$

The α_k are learned *residual scale factors*: they modulate how much each block transforms the input.

Zero-init trick. Initialize W_2 to zero so that $\alpha_1, \alpha_2 = 0$ at the start of training.

- At step 0: each block is identity: $\mathbf{x}' = \mathbf{x} + 0 \cdot \mathrm{Attn}(\cdots) = \mathbf{x}$.

DiT Step 2: Conditioning via adaLN-Zero

Core question. How to inject t and c into each transformer block?

Standard transformer block:

$$\mathbf{x}' = \mathbf{x} + \mathrm{Attn}(\mathrm{LN}(\mathbf{x})), \qquad \mathbf{x}'' = \mathbf{x}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{x}')).$$

DiT: adaLN. Replace fixed LN scale/shift (γ, β) with values predicted from conditioning by a shallow 2-layer MLP (linear $\to$ SiLU $\to$ linear):

$$(\gamma_1, \beta_1, \alpha_1, \gamma_2, \beta_2, \alpha_2) = W_2\,\mathrm{SiLU}(W_1(t_{\mathrm{emb}} + c_{\mathrm{emb}})).$$

Here $c_{\mathrm{emb}} \in \mathbb{R}^d$ is a learned embedding for the class label (a single integer for class-conditional ImageNet generation; contrast with the U-Net's sequence of text tokens via cross-attention).

$$\mathbf{x}' = \mathbf{x} + \alpha_1 \cdot \mathrm{Attn}\big(\mathrm{adaLN}(\mathbf{x}; \gamma_1, \beta_1)\big), \qquad \mathbf{x}'' = \mathbf{x}' + \alpha_2 \cdot \mathrm{FFN}\big(\mathrm{adaLN}(\mathbf{x}'; \gamma_2, \beta_2)\big).$$

The α_k are learned *residual scale factors*: they modulate how much each block transforms the input.

Zero-init trick. Initialize W_2 to zero so that $\alpha_1, \alpha_2 = 0$ at the start of training.

- At step 0: each block is identity: $\mathbf{x}' = \mathbf{x} + 0 \cdot \mathrm{Attn}(\cdots) = \mathbf{x}$.
- Gradients flow unchanged through the residual path, enabling stable training of deep stacks (DiT uses up to $L = 28$ layers; the principle extends further).

DiT Step 2: Conditioning via adaLN-Zero

Core question. How to inject t and c into each transformer block?

Standard transformer block:

$$\mathbf{x}' = \mathbf{x} + \mathrm{Attn}(\mathrm{LN}(\mathbf{x})), \qquad \mathbf{x}'' = \mathbf{x}' + \mathrm{FFN}(\mathrm{LN}(\mathbf{x}')).$$

DiT: adaLN. Replace fixed LN scale/shift (γ, β) with values predicted from conditioning by a shallow 2-layer MLP (linear $\to$ SiLU $\to$ linear):

$$(\gamma_1, \beta_1, \alpha_1, \gamma_2, \beta_2, \alpha_2) = W_2\,\mathrm{SiLU}(W_1(t_{\mathrm{emb}} + c_{\mathrm{emb}})).$$

Here $c_{\mathrm{emb}} \in \mathbb{R}^d$ is a learned embedding for the class label (a single integer for class-conditional ImageNet generation; contrast with the U-Net's sequence of text tokens via cross-attention).

$$\mathbf{x}' = \mathbf{x} + \alpha_1 \cdot \mathrm{Attn}\big(\mathrm{adaLN}(\mathbf{x}; \gamma_1, \beta_1)\big), \qquad \mathbf{x}'' = \mathbf{x}' + \alpha_2 \cdot \mathrm{FFN}\big(\mathrm{adaLN}(\mathbf{x}'; \gamma_2, \beta_2)\big).$$

The α_k are learned *residual scale factors*: they modulate how much each block transforms the input.

Zero-init trick. Initialize W_2 to zero so that $\alpha_1, \alpha_2 = 0$ at the start of training.

- At step 0: each block is identity: $\mathbf{x}' = \mathbf{x} + 0 \cdot \mathrm{Attn}(\cdots) = \mathbf{x}$.
- Gradients flow unchanged through the residual path, enabling stable training of deep stacks (DiT uses up to $L = 28$ layers; the principle extends further).

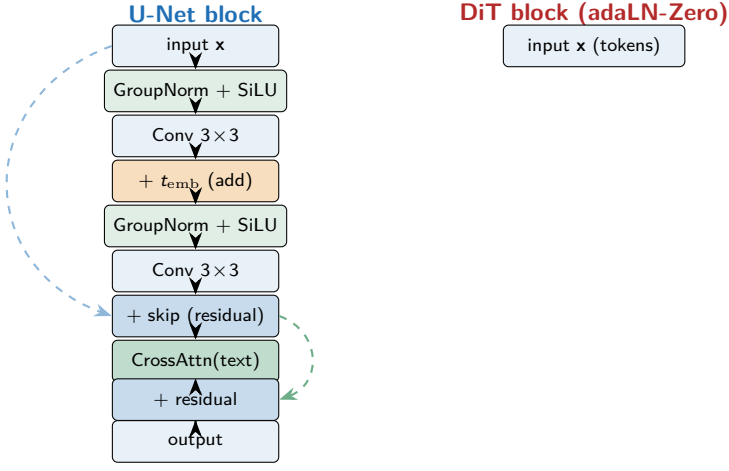
Parameter cost per block. The adaLN-Zero MLP maps $\mathbb{R}^d \to \mathbb{R}^{6d}$, costing $d^2 + 6d^2 = 7d^2$ parameters per block, or $7d^2 L$ total across L blocks.

U-Net Block vs. DiT Block



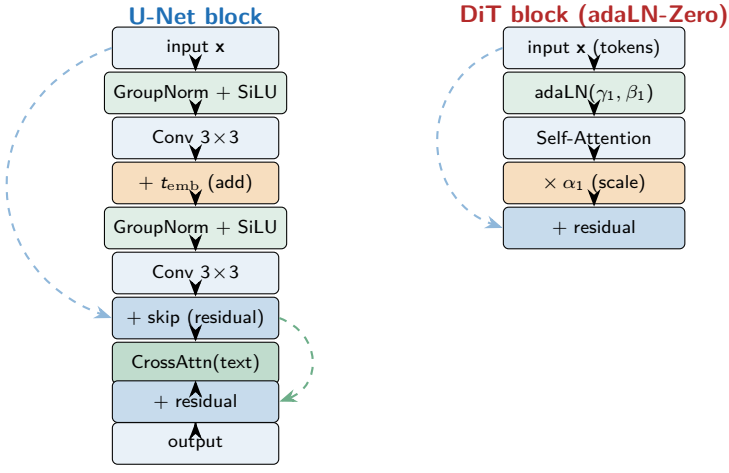
Dashed arrows = residual (skip) connections. **Cf.** Peebles & Xie 2023, Figure 3.

U-Net Block vs. DiT Block



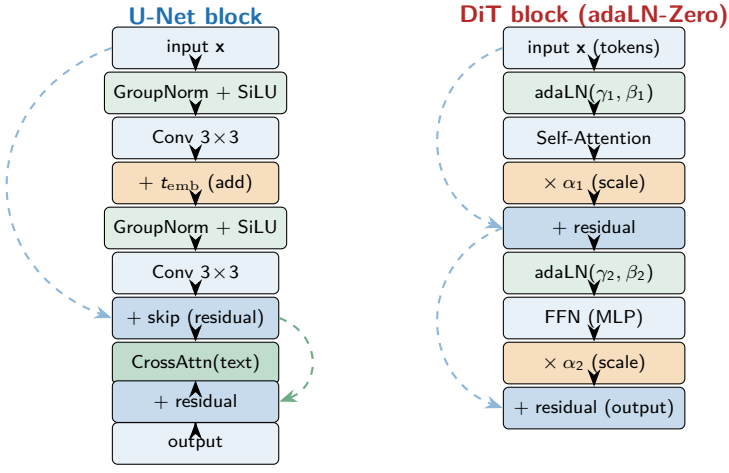
Dashed arrows = residual (skip) connections. **Cf.** Peebles & Xie 2023, Figure 3.

U-Net Block vs. DiT Block



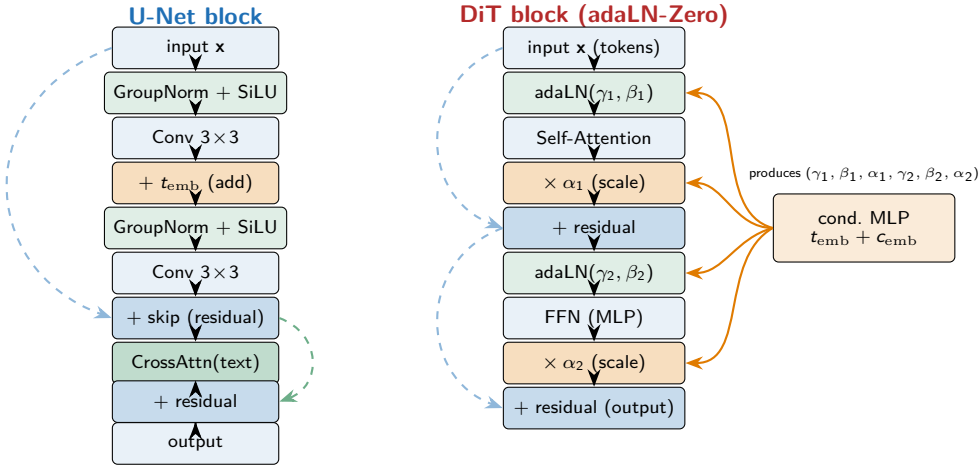
Dashed arrows = residual (skip) connections. **Cf.** Peebles & Xie 2023, Figure 3.

U-Net Block vs. DiT Block



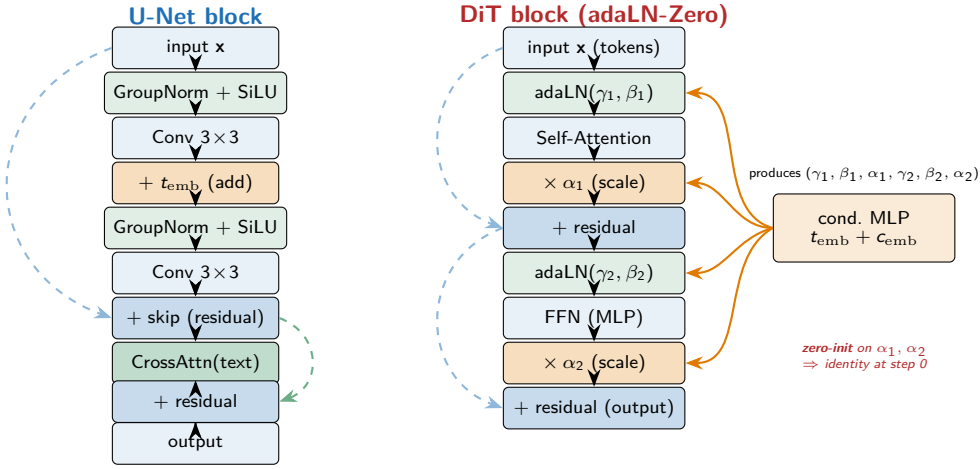
Dashed arrows = residual (skip) connections. **Cf.** Peebles & Xie 2023, Figure 3.

U-Net Block vs. DiT Block



Dashed arrows = residual (skip) connections. **Cf.** Peebles & Xie 2023, Figure 3.

U-Net Block vs. DiT Block



Dashed arrows = residual (skip) connections. **Cf.** Peebles & Xie 2023, Figure 3.

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.
2. Linear layer predicts $\epsilon \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$ at each spatial position.

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.
2. Linear layer predicts $\boldsymbol{\epsilon} \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$ at each spatial position.

Training objective. Identical to before:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta^{\mathrm{DiT}}(\mathbf{z}_t, t, c)\right\|^2\right].$$

Same loss; different architecture for the denoiser.

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.
2. Linear layer predicts $\boldsymbol{\epsilon} \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$ at each spatial position.

Training objective. Identical to before:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta^{\mathrm{DiT}}(\mathbf{z}_t, t, c)\right\|^2\right].$$

Same loss; different architecture for the denoiser.

Model family. Varying (d, L, heads) gives DiT-S/2, DiT-B/2, DiT-L/2, DiT-XL/2 (all at patch size $p = 2$).

Model	hidden d	layers L	heads	params
DiT-S/2	384	12	6	33M

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.
2. Linear layer predicts $\boldsymbol{\epsilon} \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$ at each spatial position.

Training objective. Identical to before:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta^{\mathrm{DiT}}(\mathbf{z}_t, t, c)\right\|^2\right].$$

Same loss; different architecture for the denoiser.

Model family. Varying (d, L, heads) gives DiT-S/2, DiT-B/2, DiT-L/2, DiT-XL/2 (all at patch size $p = 2$).

Model	hidden d	layers L	heads	params
DiT-S/2	384	12	6	33M
DiT-B/2	768	12	12	130M

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.
2. Linear layer predicts $\boldsymbol{\epsilon} \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$ at each spatial position.

Training objective. Identical to before:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta^{\mathrm{DiT}}(\mathbf{z}_t, t, c)\right\|^2\right].$$

Same loss; different architecture for the denoiser.

Model family. Varying (d, L, heads) gives DiT-S/2, DiT-B/2, DiT-L/2, DiT-XL/2 (all at patch size $p = 2$).

Model	hidden d	layers L	heads	params
DiT-S/2	384	12	6	33M
DiT-B/2	768	12	12	130M
DiT-L/2	1024	24	16	458M

DiT Step 3: Output Head

After L transformer blocks, the N token outputs are:

1. Unpatchified back to spatial layout $(H/f) \times (W/f) \times c_z$.
2. Linear layer predicts $\boldsymbol{\epsilon} \in \mathbb{R}^{(H/f)\times(W/f)\times c_z}$ at each spatial position.

Training objective. Identical to before:

$$\mathcal{L} = \mathbb{E}_{t,\mathbf{z}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta^{\mathrm{DiT}}(\mathbf{z}_t, t, c)\right\|^2\right].$$

Same loss; different architecture for the denoiser.

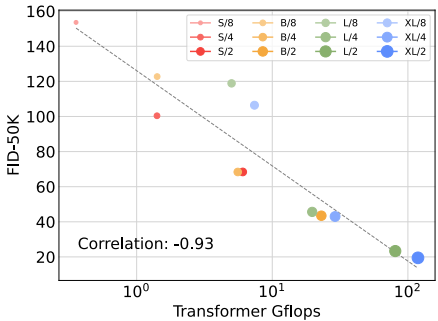
Model family. Varying (d, L, heads) gives DiT-S/2, DiT-B/2, DiT-L/2, DiT-XL/2 (all at patch size $p = 2$).

Model	hidden d	layers L	heads	params
DiT-S/2	384	12	6	33M
DiT-B/2	768	12	12	130M
DiT-L/2	1024	24	16	458M
DiT-XL/2	1152	28	16	675M

DiT Follows Transformer Scaling Laws

Peebles & Xie (2023): DiT FID scales as a clean power law in training compute (GFLOPs):

$$\text{FID} \propto C^{-\alpha}.$$



Source: Peebles & Xie 2023 (DiT), Figure 8. FID vs. training compute (GFLOPs) for DiT-S/B/L/XL at patch size $p = 2$. DiT follows a clean power law across 3+ orders of magnitude; U-Net variants show no comparable trend.

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM
Sora (2024)	Spatial-temporal DiT	Extends patch tokens to video

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM
Sora (2024)	Spatial-temporal DiT	Extends patch tokens to video

Practical consequence. Fit a power law on small DiT-S/B/L, predict the FID you'll get at DiT-XL scale, decide whether to commit. This is what makes DiT *investable* in a way U-Net was not.

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM
Sora (2024)	Spatial-temporal DiT	Extends patch tokens to video

Practical consequence. Fit a power law on small DiT-S/B/L, predict the FID you'll get at DiT-XL scale, decide whether to commit. This is what makes DiT *investable* in a way U-Net was not.

Message.

- U-Net was the right tool for 2020–2022.

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM
Sora (2024)	Spatial-temporal DiT	Extends patch tokens to video

Practical consequence. Fit a power law on small DiT-S/B/L, predict the FID you'll get at DiT-XL scale, decide whether to commit. This is what makes DiT *investable* in a way U-Net was not.

Message.

- U-Net was the right tool for 2020–2022.
- DiT is the right tool for 2023 onward.

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM
Sora (2024)	Spatial-temporal DiT	Extends patch tokens to video

Practical consequence. Fit a power law on small DiT-S/B/L, predict the FID you'll get at DiT-XL scale, decide whether to commit. This is what makes DiT *investable* in a way U-Net was not.

Message.

- U-Net was the right tool for 2020–2022.
- DiT is the right tool for 2023 onward.
- If you read the FLUX or Sora paper today, you should recognize every component of the denoiser.

Where DiT Is Used Today

Model	Denoiser	Notes
DALL·E 3 (2023)	DiT + U-Net hybrid	
Stable Diffusion 3 (2024)	MM-DiT	Multi-modal cross-attn in transformer
FLUX.1 (2024)	DiT (flow matching)	Rectified flow instead of DDPM
Sora (2024)	Spatial-temporal DiT	Extends patch tokens to video

Practical consequence. Fit a power law on small DiT-S/B/L, predict the FID you'll get at DiT-XL scale, decide whether to commit. This is what makes DiT *investable* in a way U-Net was not.

Message.

- U-Net was the right tool for 2020–2022.
- DiT is the right tool for 2023 onward.
- If you read the FLUX or Sora paper today, you should recognize every component of the denoiser.

Bonus problem (scalable denoiser) resolved. ✓ DiT = patchify + adaLN-Zero + power-law scaling.

DiT + Text Conditioning: Beyond Classes

adaLN-Zero conditions each DiT block on a *single vector* (class label or timestep). A text prompt is an *arbitrary-length token sequence* — a fundamentally different conditioning structure.

Two architectural responses have emerged:

DiT + Text Conditioning: Beyond Classes

adaLN-Zero conditions each DiT block on a *single vector* (class label or timestep). A text prompt is an *arbitrary-length token sequence* — a fundamentally different conditioning structure.

Two architectural responses have emerged:

Strategy	**Mechanism**	**Models**
Cross-attention	Extra sub-layer per block; Q from image tokens, K, V from text tokens; adaLN-Zero still handles timestep	PixArt-α (ICLR 2024), FLUX.1
Joint self-attention	Image and text tokens concatenated into one sequence; single self-attention over both; separate weight matrices per modality (MM-DiT)	Stable Diffusion 3 (ICML 2024)

DiT + Text Conditioning: Beyond Classes

adaLN-Zero conditions each DiT block on a *single vector* (class label or timestep). A text prompt is an *arbitrary-length token sequence* — a fundamentally different conditioning structure.

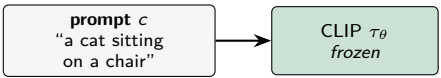
Two architectural responses have emerged:

Strategy	**Mechanism**	**Models**
Cross-attention	Extra sub-layer per block; Q from image tokens, K, V from text tokens; adaLN-Zero still handles timestep	PixArt-α (ICLR 2024), FLUX.1
Joint self-attention	Image and text tokens concatenated into one sequence; single self-attention over both; separate weight matrices per modality (MM-DiT)	Stable Diffusion 3 (ICML 2024)

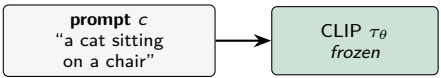
See Appendix C for why cross-attention conditioning requires CLIP text embeddings rather than BERT embeddings.

Putting It All Together

Stable Diffusion: Full Inference Pipeline

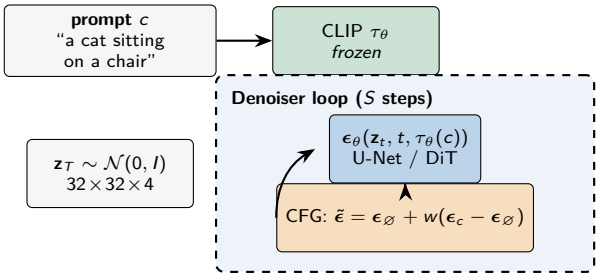


Stable Diffusion: Full Inference Pipeline

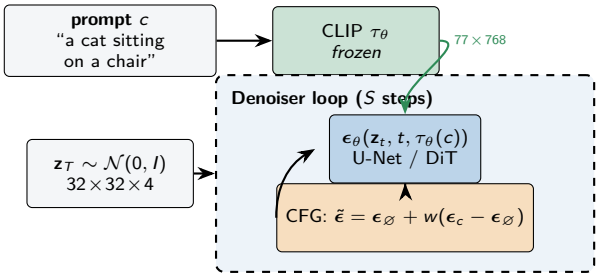


$\mathbf{z}_T \sim \mathcal{N}(0, I)$
$32 \times 32 \times 4$

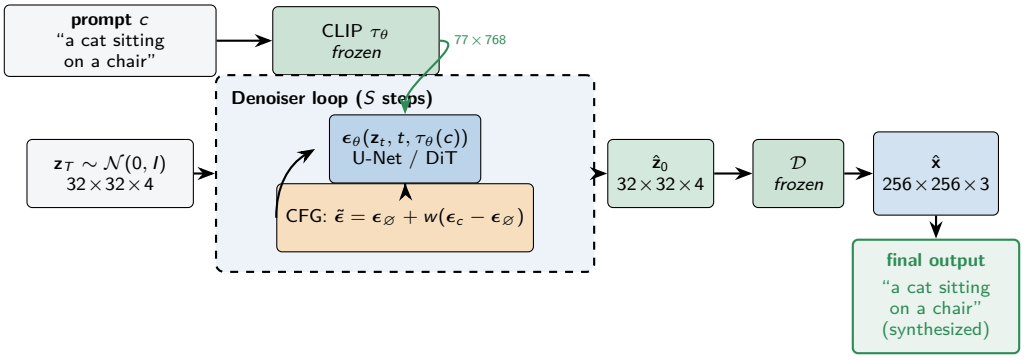
Stable Diffusion: Full Inference Pipeline



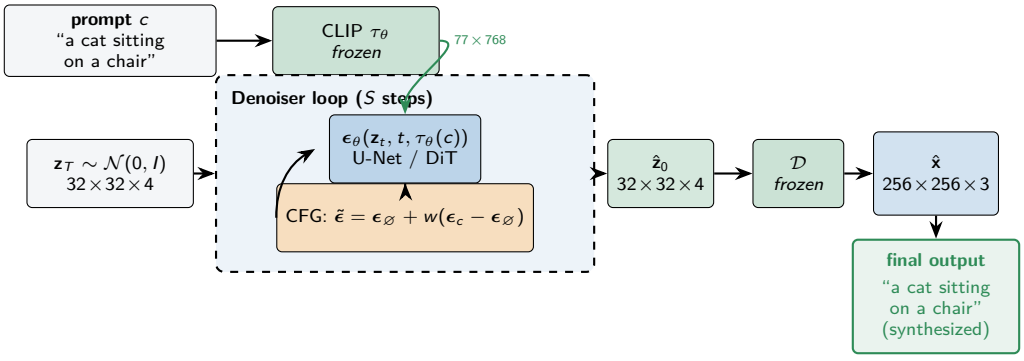
Stable Diffusion: Full Inference Pipeline



Stable Diffusion: Full Inference Pipeline



Stable Diffusion: Full Inference Pipeline



Typical settings. $S = 30-50$ (DPM-Solver), $w = 7.5$, $f = 8$, CLIP ViT-L/14.

Full Inference Algorithm

Input: prompt c; guidance scale w; sampler steps S; trained ϵ_θ, frozen $\tau_\theta, \mathcal{E}, \mathcal{D}$
Output: image $\hat{\mathbf{x}} \in \mathbb{R}^{256\times256\times3}$
tokens $\leftarrow \tau_\theta(c)$ // CLIP, 77×768
$\mathbf{z} \sim \mathcal{N}(0, I)$, $\mathbf{z} \in \mathbb{R}^{32\times32\times4}$
pick subsequence $\tau_1 > \cdots > \tau_S$

Algorithm 12 — Stable Diffusion inference, end to end

Full Inference Algorithm

Input: prompt c; guidance scale w; sampler steps S; trained ϵ_θ, frozen $\tau_\theta, \mathcal{E}, \mathcal{D}$
Output: image $\hat{\mathbf{x}} \in \mathbb{R}^{256\times256\times3}$
tokens $\leftarrow \tau_\theta(c)$ // CLIP, 77×768
$\mathbf{z} \sim \mathcal{N}(0, I)$, $\mathbf{z} \in \mathbb{R}^{32\times32\times4}$
pick subsequence $\tau_1 > \cdots > \tau_S$
for $s = 1, \ldots, S-1$ **do**
  $\boldsymbol{\epsilon}_c \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \text{tokens})$ // conditional prediction
  $\boldsymbol{\epsilon}_\varnothing \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \varnothing)$ // unconditional prediction
end

Algorithm 13 — Stable Diffusion inference, end to end

Full Inference Algorithm

Input: prompt c; guidance scale w; sampler steps S; trained ϵ_θ, frozen $\tau_\theta, \mathcal{E}, \mathcal{D}$
Output: image $\hat{\mathbf{x}} \in \mathbb{R}^{256\times256\times3}$
tokens $\leftarrow \tau_\theta(c)$ // CLIP, 77×768
$\mathbf{z} \sim \mathcal{N}(0, I)$, $\mathbf{z} \in \mathbb{R}^{32\times32\times4}$
pick subsequence $\tau_1 > \cdots > \tau_S$
for $s = 1, \ldots, S-1$ **do**
  $\boldsymbol{\epsilon}_c \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \text{tokens})$ // conditional prediction
  $\boldsymbol{\epsilon}_\varnothing \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \varnothing)$ // unconditional prediction
  $\tilde{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}_\varnothing + w\,(\boldsymbol{\epsilon}_c - \boldsymbol{\epsilon}_\varnothing)$ // CFG
end

Algorithm 14 — Stable Diffusion inference, end to end

Full Inference Algorithm

Input: prompt c; guidance scale w; sampler steps S; trained ϵ_θ, frozen $\tau_\theta, \mathcal{E}, \mathcal{D}$
Output: image $\hat{\mathbf{x}} \in \mathbb{R}^{256\times256\times3}$

tokens $\leftarrow \tau_\theta(c)$ // CLIP, 77×768

$\mathbf{z} \sim \mathcal{N}(0, I)$, $\mathbf{z} \in \mathbb{R}^{32\times32\times4}$

pick subsequence $\tau_1 > \cdots > \tau_S$

for $s = 1, \ldots, S-1$ **do**

- $\boldsymbol{\epsilon}_c \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \text{tokens})$ // conditional prediction
- $\boldsymbol{\epsilon}_\varnothing \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \varnothing)$ // unconditional prediction
- $\tilde{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}_\varnothing + w\,(\boldsymbol{\epsilon}_c - \boldsymbol{\epsilon}_\varnothing)$ // CFG
- $\hat{\mathbf{z}}_0 \leftarrow (\mathbf{z} - \sqrt{1-\bar{\alpha}_{\tau_s}}\,\tilde{\boldsymbol{\epsilon}})/\sqrt{\bar{\alpha}_{\tau_s}}$
- $\mathbf{z} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\,\hat{\mathbf{z}}_0 + \sqrt{1-\bar{\alpha}_{\tau_{s+1}}}\,\tilde{\boldsymbol{\epsilon}}$ // DDIM step ($\sigma = 0$)

end

Algorithm 15 — Stable Diffusion inference, end to end

Full Inference Algorithm

Input: prompt c; guidance scale w; sampler steps S; trained $\boldsymbol{\epsilon}_\theta$, frozen $\tau_\theta, \mathcal{E}, \mathcal{D}$
Output: image $\hat{\mathbf{x}} \in \mathbb{R}^{256\times256\times3}$

tokens $\leftarrow \tau_\theta(c)$ // CLIP, 77×768

$\mathbf{z} \sim \mathcal{N}(0, I)$, $\mathbf{z} \in \mathbb{R}^{32\times32\times4}$

pick subsequence $\tau_1 > \cdots > \tau_S$

for $s = 1, \ldots, S-1$ **do**

- $\boldsymbol{\epsilon}_c \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \text{tokens})$ // conditional prediction
- $\boldsymbol{\epsilon}_\varnothing \leftarrow \boldsymbol{\epsilon}_\theta(\mathbf{z}, \tau_s, \varnothing)$ // unconditional prediction
- $\tilde{\boldsymbol{\epsilon}} \leftarrow \boldsymbol{\epsilon}_\varnothing + w\,(\boldsymbol{\epsilon}_c - \boldsymbol{\epsilon}_\varnothing)$ // CFG
- $\hat{\mathbf{z}}_0 \leftarrow (\mathbf{z} - \sqrt{1-\bar{\alpha}_{\tau_s}}\,\tilde{\boldsymbol{\epsilon}})/\sqrt{\bar{\alpha}_{\tau_s}}$
- $\mathbf{z} \leftarrow \sqrt{\bar{\alpha}_{\tau_{s+1}}}\,\hat{\mathbf{z}}_0 + \sqrt{1-\bar{\alpha}_{\tau_{s+1}}}\,\tilde{\boldsymbol{\epsilon}}$ // DDIM step ($\sigma = 0$)

end

return $\hat{\mathbf{x}} \leftarrow \mathcal{D}(\mathbf{z})$

Algorithm 16 — Stable Diffusion inference, end to end

Per-step cost: 2 denoiser evaluations (for CFG). Total: $2S$ denoiser calls, plus one CLIP pass, plus one VAE decode.

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\rightarrow$ tokens	contrastive (text-image pairs)	yes

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\rightarrow$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\rightarrow$ latent	perceptual + KL	yes

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\rightarrow$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\rightarrow$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\rightarrow$ image	perceptual + KL	yes

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\rightarrow$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\rightarrow$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\rightarrow$ image	perceptual + KL	yes
U-Net / DiT $\boldsymbol{\epsilon}_\theta$	denoise in latent	ϵ-prediction	**no** (generative core)

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\rightarrow$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\rightarrow$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\rightarrow$ image	perceptual + KL	yes
U-Net / DiT $\boldsymbol{\epsilon}_\theta$	denoise in latent	ϵ-prediction	**no** (generative core)
Sampler (DDIM / DPM-Solver)	step schedule	— (an algorithm)	—

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\rightarrow$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\rightarrow$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\rightarrow$ image	perceptual + KL	yes
U-Net / DiT $\boldsymbol{\epsilon}_\theta$	denoise in latent	ϵ-prediction	**no** (generative core)
Sampler (DDIM / DPM-Solver)	step schedule	— (an algorithm)	—
CFG (w)	amplify condition	— (inference knob)	—

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\to$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\to$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\to$ image	perceptual + KL	yes
U-Net / DiT $\boldsymbol{\epsilon}_\theta$	denoise in latent	ϵ-prediction	**no** (generative core)
Sampler (DDIM / DPM-Solver)	step schedule	— (an algorithm)	—
CFG (w)	amplify condition	— (inference knob)	—

Cross-references to earlier lectures.

- *CLIP cross-attention* $\to$ Lecture 18 (transformers)

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\to$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\to$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\to$ image	perceptual + KL	yes
U-Net / DiT ϵ_θ	denoise in latent	ϵ-prediction	**no** (generative core)
Sampler (DDIM / DPM-Solver)	step schedule	— (an algorithm)	—
CFG (w)	amplify condition	— (inference knob)	—

Cross-references to earlier lectures.

- *CLIP cross-attention* $\to$ Lecture 18 (transformers)
- *VAE* $\to$ Lecture 21 (ELBO)

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\to$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\to$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\to$ image	perceptual + KL	yes
U-Net / DiT ϵ_θ	denoise in latent	ϵ-prediction	**no** (generative core)
Sampler (DDIM / DPM-Solver)	step schedule	— (an algorithm)	—
CFG (w)	amplify condition	— (inference knob)	—

Cross-references to earlier lectures.

- *CLIP cross-attention* $\to$ Lecture 18 (transformers)
- *VAE* $\to$ Lecture 21 (ELBO)
- *ϵ-prediction / score matching* $\to$ Lecture 23

Component Summary

Component	Role	Training signal	Frozen at inference?
CLIP text enc. τ_θ	text $\to$ tokens	contrastive (text-image pairs)	yes
VAE encoder $\mathcal{E}$	image $\to$ latent	perceptual + KL	yes
VAE decoder $\mathcal{D}$	latent $\to$ image	perceptual + KL	yes
U-Net / DiT $\boldsymbol{\epsilon}_\theta$	denoise in latent	ϵ-prediction	**no** (generative core)
Sampler (DDIM / DPM-Solver)	step schedule	— (an algorithm)	—
CFG (w)	amplify condition	— (inference knob)	—

Cross-references to earlier lectures.

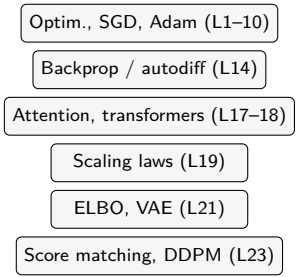
- *CLIP cross-attention* $\to$ Lecture 18 (transformers)
- *VAE* $\to$ Lecture 21 (ELBO)
- ϵ*-prediction / score matching* $\to$ Lecture 23
- *Scaling laws* $\to$ Lecture 19

Closing the Course Arc

Today's lecture ties together almost every thread in the course.

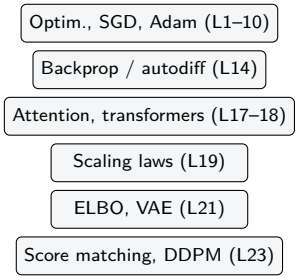
Closing the Course Arc

Today's lecture ties together almost every thread in the course.



Closing the Course Arc

Today's lecture ties together almost every thread in the course.

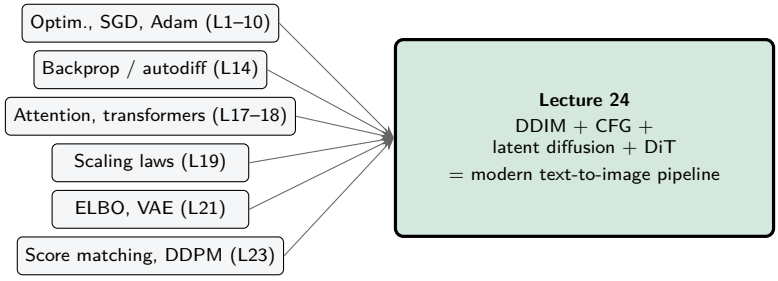


Lecture 24

DDIM + CFG +
latent diffusion + DiT
= modern text-to-image pipeline

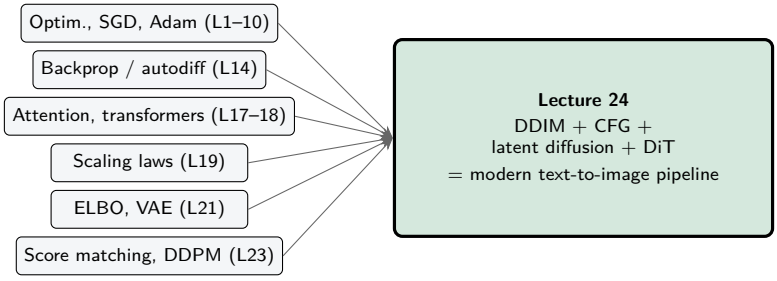
Closing the Course Arc

Today's lecture ties together almost every thread in the course.



Closing the Course Arc

Today's lecture ties together almost every thread in the course.

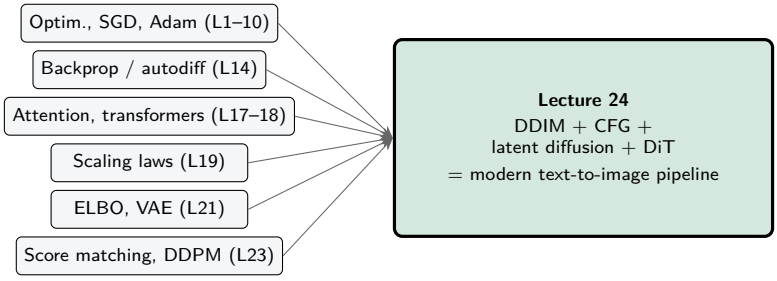


What you can now do.

- Read the FLUX or Sora paper and recognize every architectural component.

Closing the Course Arc

Today's lecture ties together almost every thread in the course.

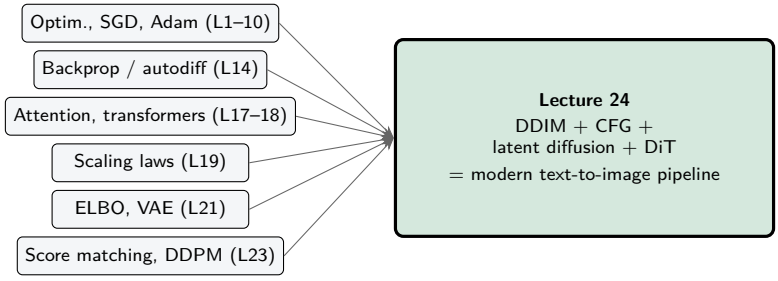


What you can now do.

- Read the FLUX or Sora paper and recognize every architectural component.
- Implement a DDIM sampler from scratch.

Closing the Course Arc

Today's lecture ties together almost every thread in the course.

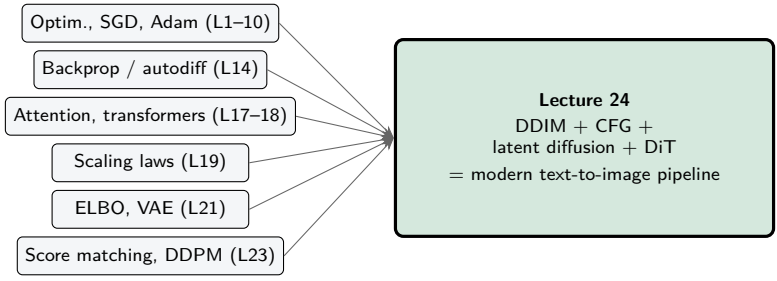


What you can now do.

- Read the FLUX or Sora paper and recognize every architectural component.
- Implement a DDIM sampler from scratch.
- Add classifier-free guidance to a class-conditional model.

Closing the Course Arc

Today's lecture ties together almost every thread in the course.



What you can now do.

- Read the FLUX or Sora paper and recognize every architectural component.
- Implement a DDIM sampler from scratch.
- Add classifier-free guidance to a class-conditional model.
- Explain why Stable Diffusion works in latent space and what a DiT block does.

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea
Speed	DDIM, DPM-Solver	non-Markovian; Euler / higher-order on probability-flow ODE

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea
Speed	DDIM, DPM-Solver	non-Markovian; Euler / higher-order on probability-flow ODE
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\theta(\cdot, \varnothing) + w\big[\epsilon_\theta(\cdot, c) - \epsilon_\theta(\cdot, \varnothing)\big]$

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea
Speed	DDIM, DPM-Solver	non-Markovian; Euler / higher-order on probability-flow ODE
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\theta(\cdot, \varnothing) + w\big[\epsilon_\theta(\cdot, c) - \epsilon_\theta(\cdot, \varnothing)\big]$
Scale	Latent diffusion	$\mathbf{z} = \mathcal{E}(\mathbf{x})$; diffuse $\mathbf{z}_t$; decode $\hat{\mathbf{x}} = \mathcal{D}(\hat{\mathbf{z}}_0)$

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea
Speed	DDIM, DPM-Solver	non-Markovian; Euler / higher-order on probability-flow ODE
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\theta(\cdot, \varnothing) + w\big[\epsilon_\theta(\cdot, c) - \epsilon_\theta(\cdot, \varnothing)\big]$
Scale	Latent diffusion	$\mathbf{z} = \mathcal{E}(\mathbf{x})$; diffuse $\mathbf{z}_t$; decode $\hat{\mathbf{x}} = \mathcal{D}(\hat{\mathbf{z}}_0)$
Bonus	DiT	patchify + adaLN-Zero + power-law scaling

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea
Speed	DDIM, DPM-Solver	non-Markovian; Euler / higher-order on probability-flow ODE
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\theta(\cdot, \varnothing) + w\big[\epsilon_\theta(\cdot, c) - \epsilon_\theta(\cdot, \varnothing)\big]$
Scale	Latent diffusion	$\mathbf{z} = \mathcal{E}(\mathbf{x})$; diffuse $\mathbf{z}_t$; decode $\hat{\mathbf{x}} = \mathcal{D}(\hat{\mathbf{z}}_0)$
Bonus	DiT	patchify + adaLN-Zero + power-law scaling

Stable Diffusion = DDIM + CFG + latent diffusion + (U-Net or DiT).

Summary: The Three Problems, Resolved

Problem	Solution	Key formula / idea
Speed	DDIM, DPM-Solver	non-Markovian; Euler / higher-order on probability-flow ODE
Control	Classifier-free guidance	$\tilde{\epsilon} = \epsilon_\theta(\cdot, \varnothing) + w\big[\epsilon_\theta(\cdot, c) - \epsilon_\theta(\cdot, \varnothing)\big]$
Scale	Latent diffusion	$\mathbf{z} = \mathcal{E}(\mathbf{x})$; diffuse $\mathbf{z}_t$; decode $\hat{\mathbf{x}} = \mathcal{D}(\hat{\mathbf{z}}_0)$
Bonus	DiT	patchify + adaLN-Zero + power-law scaling

Stable Diffusion = DDIM + CFG + latent diffusion + (U-Net or DiT).

Next lecture: flow matching and rectified flow — the current frontier.

Appendix Overview

Appendix Overview

The appendices provide deeper dives into supporting material. Here's what each covers:

Appendix	Content
A: Distributional Metrics	FID: The standard metric for comparing entire distributions of generated vs real images. Covers Fréchet distance, feature extraction, and practical usage.
B: Per-Image Metrics	LPIPS: A perceptual metric that measures human-noticeable differences between individual images. Explains why MSE/PSNR fail and how feature-space distances work.
C: CLIP Details	How CLIP aligns images and text in a shared embedding space. Covers the contrastive training objective and how it enables text-to-image conditioning.
D: Transformer Details	Vision Transformer architecture: patch embedding, positional encoding, and the complete DiT block structure. Essential for understanding how transformers process images.

Appendix A: Distributional Metrics

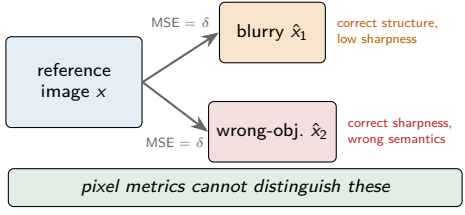
A1: Two Evaluation Questions

Evaluating a generative model requires answering *two distinct questions*:

1. **Distributional realism.** Does the *collection* of generated images look like the collection of real images?
2. **Per-image quality.** Given a generated image $\hat{x}$ and a reference x, does $\hat{x}$ look like x?

These are *not* interchangeable: a model can produce individually faithful images yet miss entire modes of p_{real}, or cover all modes yet produce individually poor samples.

Why pixel metrics fail at Question 1. MSE and PSNR average squared error over pixels: $\text{MSE}(\hat{x}, x) = \frac{1}{HWC}\|\hat{x} - x\|^2$. Two failure modes with equal MSE:



This appendix addresses Question 1 by developing a *distributional* distance, **FID**. The next appendix addresses Question 2 by developing a per-image *perceptual* distance, **LPIPS**.

A2: Feature Space and the Fréchet Distance

Why features, not pixels. A classifier trained on ImageNet learns to separate object categories; distances in its feature space correlate with human perceptual differences.

Fix the **Inception-v3 feature map** $\phi : \mathbb{R}^{H\times W\times 3} \to \mathbb{R}^{2048}$ (pool3 layer, Szegedy et al. 2016). Apply ϕ to every image in the real and generated sets.

Gaussian approximation. Model $\phi(p_{\text{real}})$ and $\phi(p_{\text{gen}})$ as Gaussians:

$$\phi(p_{\text{real}}) \approx \mathcal{N}(\mu_r, \Sigma_r), \qquad \phi(p_{\text{gen}}) \approx \mathcal{N}(\mu_g, \Sigma_g).$$

Justification: Gaussians have a closed-form Wasserstein distance, and empirically, Heusel et al. (NeurIPS 2017) found that fitting just the mean and covariance of the Inception features is enough to rank generative models consistently with human evaluations.

2-Wasserstein distance between Gaussians.

$$W_2^2\big(\mathcal{N}(\mu_1,\Sigma_1),\,\mathcal{N}(\mu_2,\Sigma_2)\big) = \|\mu_1-\mu_2\|^2 + \operatorname{tr}\Big(\Sigma_1+\Sigma_2-2\big(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2}\big)^{1/2}\Big).$$

Substituting the Inception feature statistics gives the **Fréchet Inception Distance**:

$$\boxed{\mathrm{FID} \;=\; \|\mu_r-\mu_g\|^2 + \operatorname{tr}\Big(\Sigma_r+\Sigma_g-2\big(\Sigma_r^{1/2}\Sigma_g\Sigma_r^{1/2}\big)^{1/2}\Big).}$$

Lower FID = closer distributions. **State-of-the-art diffusion models trained on ImageNet 256×256: FID ≈ 2–10. Uniform random noise vs. real ImageNet statistics: FID ≈ 300.**

A3: FID in Practice

Sample-size requirement. The empirical covariance $\hat{\Sigma} \in \mathbb{R}^{2048\times 2048}$ is rank-deficient for $n < 2048$.

- Convention: $n \geq 50{,}000$ for publication-grade FID; $n \geq 10{,}000$ acceptable for ablations.
- The MLE covariance $\hat{\Sigma} = \frac{1}{n}\sum_i (x_i - \bar{x})(x_i - \bar{x})^\top$ has $\mathbb{E}[\hat{\Sigma}] = \frac{n-1}{n}\Sigma$: it underestimates variances, making FID too large. Dividing by $n-1$ removes this bias.

What the two terms detect.

$\|\mu_r - \mu_g\|^2$
mean shift $\Rightarrow$ **mode dropping**

gen. distribution misses regions of p_{real}

$+$

$\text{tr}(\cdots)$ term
covariance mismatch $\Rightarrow$ **mode collapse**

Σ_g too small: generated images cluster together

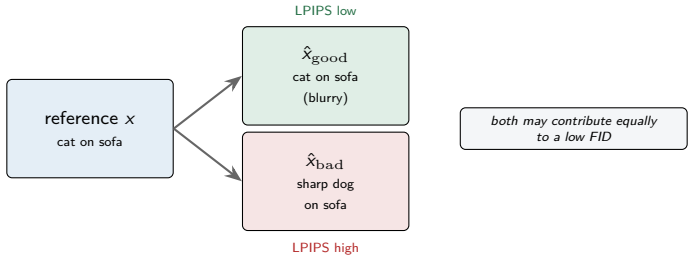
FID is a *distributional* statistic. A model that generates structurally diverse but individually incorrect images (wrong object at every pixel, but varied) can achieve low FID. This motivates the use of LPIPS, a per-image perceptual quality metric.

Appendix B: Per-Image Metrics

B1: Per-Image Quality

FID measures whether the *distribution* of generated images looks real. A separate question: does *this* generated image look like *its target*?

Failure mode FID misses.

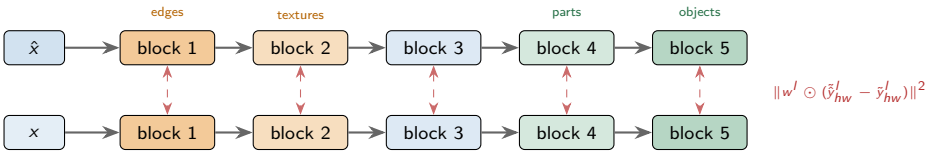


Correct formulation. Given generated image $\hat{x}$ and reference x, define a **perceptual distance** $d_{\text{perc}}(\hat{x}, x)$ that correlates with human similarity judgments.

Why VGG features? Zhang et al. (CVPR 2018) showed distances in deep feature spaces (AlexNet, VGG, ResNet) all outperform pixel-based metrics like MSE at predicting which of two distorted images humans find more similar to a reference. VGG is the conventional choice for LPIPS; AlexNet and ResNet work similarly well, but VGG was already standard in the image-quality literature when LPIPS was introduced.

B2: VGG Feature Pyramid and the LPIPS Formula

VGG-16 feature pyramid. Extract activations at each of the 5 convolutional blocks: $y^l \in \mathbb{R}^{H_l \times W_l \times C_l}$, $l = 1, \ldots, 5$.



Channel normalization: $\tilde{y}^l_{hwc} = y^l_{hwc} / \|y^l_{hw\cdot}\|_2$ (unit-normalize across channels).

LPIPS formula:

$$\mathrm{LPIPS}(\hat{x}, x) = \sum_{l=1}^{5} \frac{1}{H_l W_l} \sum_{h,w} \|w^l \odot (\tilde{y}^l_{hw} - \tilde{y}^l_{hw})\|^2$$

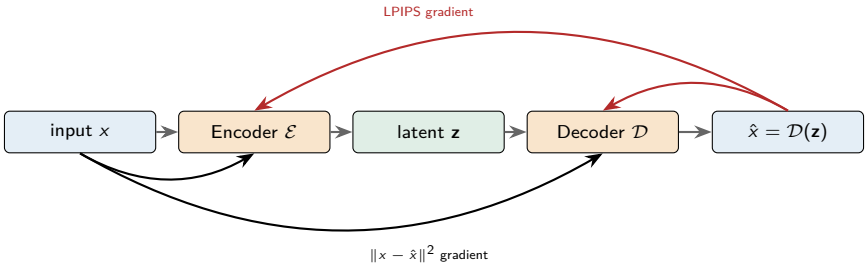
$w^l \in \mathbb{R}^{C_l}$ are **fit by supervised training**: VGG is frozen; the w^l are learned to make LPIPS rankings agree with human two-alternative forced-choice judgments of perceptual similarity.

B3: LPIPS in the Latent Diffusion Pipeline

The full VAE training loss in Stage 1 of LDM is:

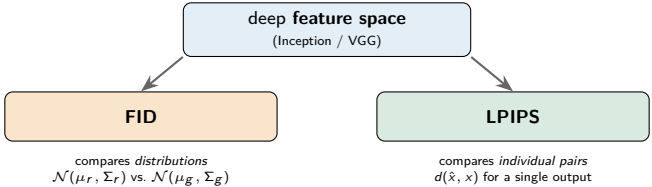
$$\mathcal{L}_{\mathrm{VAE}}(x) = \underbrace{\|x - \hat{x}\|^2 + \lambda\,\mathrm{LPIPS}(x, \hat{x})}_{\text{reconstruction loss}} + \beta\ \underbrace{D_{\mathrm{KL}}\big(q(\mathbf{z} \mid x)\ \|\ \mathcal{N}(0, I)\big)}_{\text{prior-matching regularization}}$$

where $\hat{x} = \mathcal{D}(\mathbf{z})$ is the decoder output and $\mathbf{z} \sim q(\mathbf{z} \mid x) = \mathcal{E}(x)$. Without the LPIPS term, VAE decoders trained with pixel-only loss produce *blurry* reconstructions: MSE is minimized by predicting the mean, a blurred average of plausible outputs. LPIPS provides a gradient signal that preserves sharp edges and high-frequency textures.



FID answers "is the *model* good?"; LPIPS answers "how good is *this sample*?"

B4: FID vs. LPIPS — When to Use Which



Question	**Use**
Is the output distribution close to real images overall?	FID
Does this specific output resemble this specific target?	LPIPS
Is the VAE reconstruction perceptually faithful?	LPIPS
Is a new sampler competitive with DDIM on ImageNet-256?	FID

Appendix C: CLIP

C1: The Image–Text Correspondence Problem

CLIP (Contrastive Language–Image Pretraining) learns a joint embedding of images and text via contrastive learning on web-scale image–caption pairs.

Question. Given an image and a text description, do they correspond?

Why supervised classification does not solve this.

- Requires a fixed, human-annotated label set.
- Cannot generalize to new concepts without relabeling.
- Scale is bottlenecked by annotation cost.

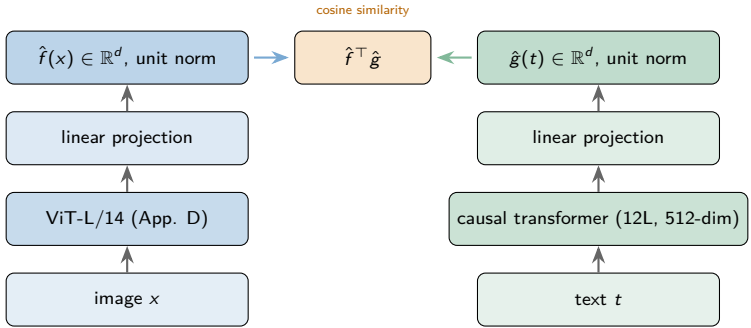
Key observation. The web contains billions of image-caption pairs (alt-text, descriptions, tweets). These pairs provide a supervision signal without manual annotation, and the signal scales with data collection.

Goal. Learn a joint embedding space:

$$f : \text{image} \to \mathbb{R}^d, \qquad g : \text{text} \to \mathbb{R}^d$$

such that matched pairs (x_i, t_i) satisfy $f(x_i) \approx g(t_i)$ and unmatched pairs (x_i, t_j), $j \neq i$ are pushed apart under the cosine similarity.

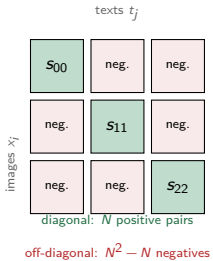
C2: Dual-Encoder Architecture



Both outputs are L2-normalized: $\hat{f}(x) = f(x)/\|f(x)\|$, $\hat{g}(t) = g(t)/\|g(t)\|$. After normalization the dot product equals cosine similarity $\in [-1, 1]$. We train so that $\hat{f}^\top \hat{g}$ is large when (x, t) is a training pair, and small when x and t do not occur together in the training data.

C3: InfoNCE Contrastive Loss

Training batch. N image-text pairs $(x_i, t_i)_{i=1}^N$. Compute N^2 similarities $s_{ij} = \hat{f}(x_i)^\top \hat{g}(t_j)$.



Loss (InfoNCE / symmetric cross-entropy):

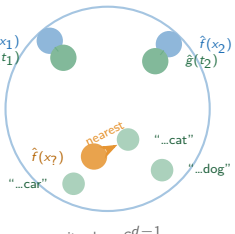
$$\mathcal{L}_{\text{CLIP}} = -\frac{1}{2N}\sum_{i=1}^{N}\Bigg[\underbrace{\log\frac{e^{s_{ii}/\tau}}{\sum_j e^{s_{ij}/\tau}}}_{\text{image } i \text{ finds its text}} + \underbrace{\log\frac{e^{s_{ii}/\tau}}{\sum_j e^{s_{ji}/\tau}}}_{\text{text } i \text{ finds its image}}\Bigg].$$

Each term is an N-way softmax classification. Large N is essential: off-diagonal negatives provide the contrastive signal that prevents all embeddings collapsing to one point.

Temperature $\tau > 0$ is a learned hyperparameter. Setting $\tau \to \infty$ makes the softmax uniform (loss $\to \log N$, no learning); $\tau \to 0$ makes the softmax one-hot (gradients explode). CLIP initializes $\tau = 0.07$, tuned to balance concentration of the embedding space against stable gradients, and learns it subject to a clamping to keep it from scaling the logits too much.

C4a: Learned Geometry and Zero-Shot Classification

After training: matched pairs (x, t) land nearby on the unit sphere; unmatched pairs are pushed apart.



Zero-shot classification (no fine-tuning required):

1. Choose a set of candidate classes $\mathcal{C} = \{\text{cat, dog, car, ...}\}$.
2. For each class $c \in \mathcal{C}$, form the prompt "a photo of a c" and embed it: $\hat{g}_c = \hat{g}(\text{"a photo of a } c\text{"})$.
3. Embed the query image: $\hat{f}(x)$.
4. Assign x to the class with the **highest cosine similarity**:

$$\hat{c} = \arg\max_{c \in \mathcal{C}} \hat{f}(x)^\top \hat{g}_c.$$

Because the embedding space is shared, any class expressible as a text prompt can be recognized — even classes unseen during training.

C4b: CLIP Score and Text Token Sequences

CLIP score. After generating an image $\hat{x}$ from a text prompt t, how do we measure whether the image actually matches the prompt? Compute

$$\text{CLIP score}(\hat{x}, t) = \hat{f}(\hat{x})^\top \hat{g}(t) \ \in [-1, 1].$$

A score near $+1$ means the generated image and the prompt land close together in the shared embedding space — i.e. the image looks like what the prompt describes. This is used as an *automatic* proxy metric when evaluating text-to-image models, alongside FID.

Text token sequence for cross-attention. CLIP's text encoder does not produce a single vector; it outputs a *sequence* of 77 token embeddings $\in \mathbb{R}^{77\times 768}$.

Why 77? CLIP tokenizes text with Byte-Pair Encoding (BPE), a subword algorithm that splits words into frequent sub-units (e.g. "running" $\rightarrow$ "run", "ning"). The tokenizer was trained with a fixed context window of **77 tokens** — a design choice made to keep inference fast and memory small at CLIP's original training scale. Any prompt longer than $\approx$50–60 words is silently truncated.

Why use the sequence, not just the pooled embedding? Cross-attention in a DiT block lets each image-patch token attend *separately* to each text token. A single pooled vector collapses all the text into one point, losing the distinction between, say, "red" and "ball". The full 77-token sequence lets each patch specialize: patches depicting color attend more to color tokens; patches depicting objects attend more to noun tokens.

Long prompts. 77 tokens is a real bottleneck for detailed prompts (roughly 50–60 words). See **D6** for how FLUX/SD3 address this using T5-XXL alongside CLIP.

C5: Why CLIP Embeddings for Cross-Attention in DiTs?

Empirical result. Rombach et al. (LDM, CVPR 2022) ablate BERT-large vs. CLIP ViT-L/14 as the text encoder for cross-attention conditioning; CLIP consistently achieves lower FID and higher CLIP score on COCO and LAION-400M.

Best available explanation — inductive bias of the text token space. The cross-attention weight $a_{ij} = \mathrm{softmax}(QK^\top/\sqrt{d})_{ij}$ measures how strongly image patch i attends to text token j. What structure in the K, V representations makes that learning tractable?

CLIP token space

Contrastive training aligns text tokens with image regions; visually salient tokens (nouns, adjectives, spatial relations) cluster apart from function words.

$\Rightarrow$ cross-attn learns a *simple* rule

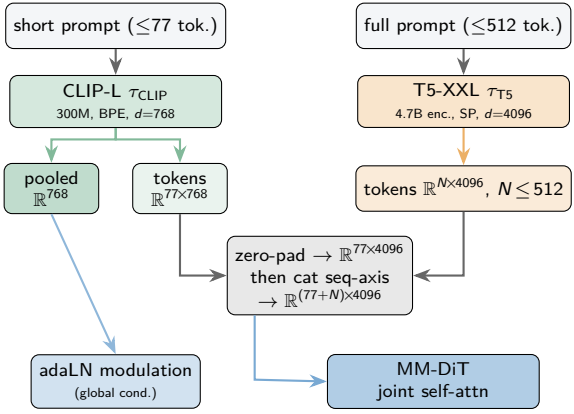
BERT token space

MLM training organizes tokens by linguistic co-occurrence; visually salient tokens are *not* distinguished from function words ("the", "of").

$\Rightarrow$ model must learn salience from scratch

Caveat. Inductive-bias argument, not a proof. The cross-attention query Q comes from noisy diffusion latents; the claim concerns only the *text* token representations. At larger scale, CLIP's 77-token limit becomes a practical bottleneck for complex prompts — see D6.

C6: Longer Prompts and Multi-Encoder Text Conditioning



CLIP's hard limit. BPE tokenizer hard-truncates at 77 tokens ($\approx$75 usable). Long prompts silently lose their tail.

T5-XXL. Encoder-decoder LM trained on span-corruption (text only, no images). Only the *encoder* is used: one 4096-dim vector per SentencePiece token, up to $\sim$512 tokens. Rich syntactic structure; *no* visual alignment.

How FLUX combines them.

- CLIP-L token seq. (77×768) zero-padded to 77×4096, then **cat along the sequence axis** with T5 tokens: $(77+N)\times4096$.
- Combined text tokens + image patch tokens enter **MM-DiT joint self-attention** — every patch token attends to every text token and vice versa.
- CLIP pooled embedding (768-dim) feeds **adaLN modulation** alongside the timestep: a fast, global text signal.

Upshot. Route a short, visually precise prompt to CLIP and the full detailed prompt to T5. CLIP supplies visual-semantic geometry; T5 supplies long-context coverage.

SD3 note. Adds CLIP-G (1280-dim). The two CLIP token seqs. concat channel-wise to 77×2048, then zero-padded to 77×4096; pooled vecs. concat to a 2048-dim modulation vector. Combined context: 154×4096.

Appendix D: Vision Transformers (ViT)

Note: DiT is a ViT

A **Diffusion Transformer (DiT)** is architecturally a Vision Transformer: it uses the identical patch-embed + transformer-encoder front end, but applied to image *latents* rather than pixels, and with diffusion-specific adaLN timestep modulation added on top.

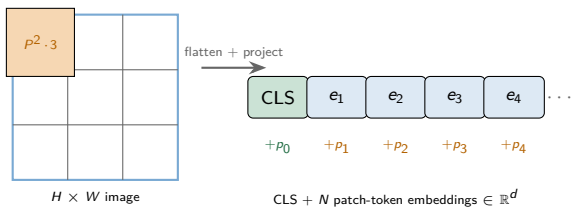
Understanding ViT therefore gives you DiT for free.

One difference: positional embeddings. The original ViT uses *learned* positional embeddings (trainable parameters). DiT uses *sinusoidal* (frequency-based) positional embeddings following the original transformer convention; some later variants (FLUX, SD3) replace these with RoPE. The choice of positional encoding is independent of the rest of the ViT/DiT architecture.

D1: Patch Embedding

Motivation. CLIP's image encoder is ViT-L/14. This appendix shows how a Vision Transformer converts an image into a patch token sequence. DiT (Diffusion Transformer) is *also* a ViT — the identical patchify-then-project front end, applied to image *latents* instead of pixels, with diffusion-specific adaLN timestep conditioning added. In a DiT the patch tokens self-attend and, depending on the variant, cross-attend to text embeddings or jointly self-attend with text tokens (MM-DiT) — *not* to the outputs of a separate ViT image encoder.

Patchify. Divide $x \in \mathbb{R}^{H \times W \times 3}$ into $N = HW/P^2$ non-overlapping $P \times P$ patches. For ViT-L/14: $P = 14$, giving $N = 256$ patches for a 224×224 image.

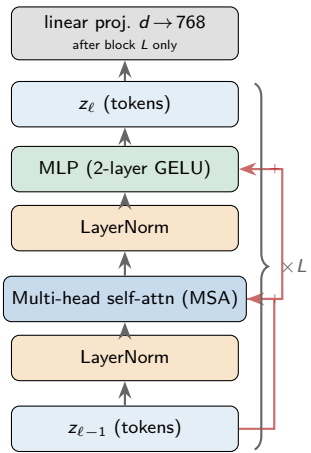


Positional encoding. Learned embeddings $p_0, p_1, \ldots, p_N \in \mathbb{R}^d$ break permutation symmetry.

$$z_0 = \left[x_{\text{cls}} + p_0;\ E\,x_p^1 + p_1;\ \ldots;\ E\,x_p^N + p_N\right], \qquad E \in \mathbb{R}^{d \times 3P^2}.$$

D2: Transformer Encoder

Stack of L *pre-LayerNorm* blocks:



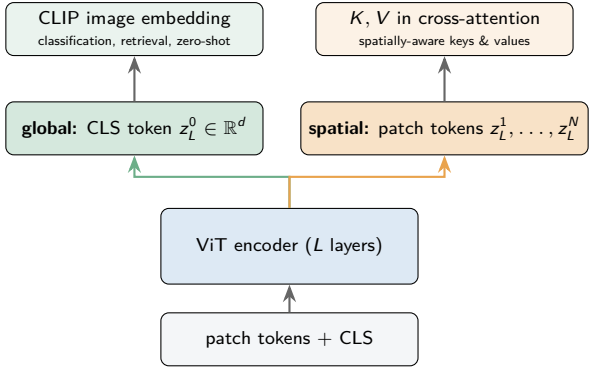
Pre-LN rationale. Normalizing *before* attention keeps residual-stream magnitude stable during training at large depth.

ViT-L/14 (CLIP image encoder):

- $L = 24$ layers, $d = 1024$, 16 heads
- Patch size $P = 14$
- Linear projection at the top maps the CLS token from $d = 1024$ to the 768-dim CLIP shared embedding space (applied once, after the final block)

D3: Two Output Modes

The same ViT forward pass produces two useful outputs.



CLS token: in every layer the CLS token attends to all patch tokens; by layer L it holds global context from the entire image. **Used by CLIP** as the image-side embedding.

Patch token sequence: retains spatial structure. When DiT uses cross-attention for text conditioning, these tokens provide spatially-aware keys and values so each diffusion token can attend to different regions of the visual field.