Yale University

Latent Diffusion, Flow Matching, and Text

S&DS 265 · Introductory Machine Learning · Lecture 19

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

Learning Objectives

In this lecture you will learn:

1. How conditioning enters the denoiser, and why guidance trades coverage for prompt fidelity;
2. How moving the process into a learned code buys a large factor in compute, and what ceiling that imposes;
3. Why a trained diffusion model can be sampled in far fewer, deterministic steps without retraining;
4. Why a straight path from noise to data gives a constant velocity to regress, and how the same conditioning trick justifies it;
5. How replacing Gaussian noise with masking gives a generative model of text.

Outline

Three Generations of Text-to-Image Models

Stable Diffusion, 2022

512 × 512

SDXL, 2023

1024 × 1024

Stable Diffusion 3.5, 2024

flow-matching backbone

FLUX 1.1 Pro, 2024

flow-matching backbone

One prompt, four model generations, three years. The 2022 sample is recognizable but soft; by 2024 they are at photographic resolution. Every system here trains with the loss of Lecture 18. What changed is where the process runs, how it is steered, and — on the right — the shape of the path.

Wikimedia Commons; CC0 and public domain. Prompt and model recorded on each file page.

Problem Setup

We want to draw from a conditional distribution,

$$\tilde{x} \sim p(x \mid c), \qquad c = \text{“an astronaut riding a horse”},$$

and we want it in seconds, not minutes.

Problem Setup

We want to draw from a conditional distribution,

$$\tilde{x} \sim p(x \mid c), \qquad c = \text{“an astronaut riding a horse”},$$

and we want it in seconds, not minutes.

DDPM as derived is slow, for two separate reasons: images are large, so one network call is expensive; and the sampler takes many steps, $T = 1000$ of them, strictly in order.

What This Lecture Adds

1. **Conditional generation.** How the prompt c enters the denoiser, and how to amplify it. Still DDPM — the loss and the sampler are the ones we already have.
2. **Cheaper steps.** Run the whole process on a learned code instead of on pixels — an autoencoder wrapped around the diffusion model of Lecture 18.
3. **Fewer steps.** DDIM reuses the trained network with a shorter schedule; flow matching then asks whether the path itself was well chosen.
4. **Discrete data.** What replaces Gaussian noise when the state is a sequence of tokens.

What This Lecture Adds

1. **Conditional generation.** How the prompt c enters the denoiser, and how to amplify it. Still DDPM — the loss and the sampler are the ones we already have.
2. **Cheaper steps.** Run the whole process on a learned code instead of on pixels — an autoencoder wrapped around the diffusion model of Lecture 18.
3. **Fewer steps.** DDIM reuses the trained network with a shorter schedule; flow matching then asks whether the path itself was well chosen.
4. **Discrete data.** What replaces Gaussian noise when the state is a sequence of tokens.

Only DDIM leaves the training untouched — it is a different way to use the network of Lecture 18. The others change what is trained, but not the loss it is trained with.

Outline

Conditioning the Denoiser

Everything so far sampled from $p(x)$. To sample from $p(x \mid c)$, give the denoiser the condition variable c and train on the same loss:

$$\min_{\theta} \; \mathbb{E}_{x,c,\varepsilon,t}[\|\varepsilon - \epsilon_\theta(x_t, t, c)\|^2].$$

For a class label, this needs no new machinery at all: embed the label and add a per-channel shift, exactly as we did for the level t.

$$h \;\longleftarrow\; h + \mathrm{Linear}(\mathrm{emb}(t)) + \mathrm{Linear}(\mathrm{emb}(c)).$$

A sentence is not a label: it is a variable-length sequence, and different parts of the image should listen to different words. Systems use **cross-attention** for this — the mechanism of Lecture 21. Read it for now as "each position can look up the words relevant to it".

An Honest Sample Is Not What We Want

Train $\epsilon_\theta(x_t, t, c)$ this way and it does sample from $p(x \mid c)$ — which is what we asked for, and in practice disappointing.

An Honest Sample Is Not What We Want

Train $\epsilon_\theta(x_t, t, c)$ this way and it does sample from $p(x \mid c)$ — which is what we asked for, and in practice disappointing.

The trouble is that $p(x \mid c)$ is broad. A great many images would be accepted as "an astronaut riding a horse": dark ones, badly cropped ones, ones where the horse is a smudge in the corner. All of them carry probability mass, so an honest sample is often one of them.

An Honest Sample Is Not What We Want

Train $\epsilon_\theta(x_t, t, c)$ this way and it does sample from $p(x \mid c)$ — which is what we asked for, and in practice disappointing.

The trouble is that $p(x \mid c)$ is broad. A great many images would be accepted as "an astronaut riding a horse": dark ones, badly cropped ones, ones where the horse is a smudge in the corner. All of them carry probability mass, so an honest sample is often one of them.

We do not want a *typical* image given the prompt. We want one the prompt describes especially well — an image a captioner would label with c confidently, so that $p(c \mid x)$ is large. That is a different distribution, and the next slide writes it down.

The Guided Density

Consider the reweighted density

$$p_w(x \mid c) \;\propto\; p(x) \cdot p(c \mid x)^w, \qquad w \geq 0.$$

[1] Dhariwal and Nichol (2021), arXiv:2105.05233 — *classifier guidance*.

The Guided Density

Consider the reweighted density

$$p_w(x \mid c) \;\propto\; p(x) \cdot p(c \mid x)^w, \qquad w \geq 0.$$

At $w = 0$ this is the unconditional $p(x)$; at $w = 1$ Bayes' rule gives exactly $p(x \mid c)$. For $w > 1$ it over-weights images the prompt explains well.

Here $p(c \mid x)$ is a **classifier**: it scores how well the prompt fits the image. Training one on noisy images was the original route.[1] We can avoid it.

[1] Dhariwal and Nichol (2021), arXiv:2105.05233 — *classifier guidance*.

The Score of the Forward Marginal

derive

Sampling sees x_t and never x_0, so the object we need is the score of the marginal $q(x_t) = \int q(x_t \mid x_0) \cdot q(x_0)\, dx_0$. Differentiate under the integral and use $\nabla q = q \cdot \nabla \log q$:

$$\nabla_{x_t} \log q(x_t) = \frac{\int \nabla_{x_t} q(x_t \mid x_0) \cdot q(x_0)\, dx_0}{q(x_t)} = \int \underbrace{\frac{q(x_t \mid x_0) \cdot q(x_0)}{q(x_t)}}_{= q(x_0 \mid x_t)} \cdot \nabla_{x_t} \log q(x_t \mid x_0)\, dx_0.$$

The Score of the Forward Marginal

derive

Sampling sees x_t and never x_0, so the object we need is the score of the marginal $q(x_t) = \int q(x_t \mid x_0) \cdot q(x_0)\, dx_0$. Differentiate under the integral and use $\nabla q = q \cdot \nabla \log q$:

$$\nabla_{x_t} \log q(x_t) = \frac{\int \nabla_{x_t} q(x_t \mid x_0) \cdot q(x_0)\, dx_0}{q(x_t)} = \int \underbrace{\frac{q(x_t \mid x_0) \cdot q(x_0)}{q(x_t)}}_{= q(x_0 \mid x_t)} \cdot \nabla_{x_t} \log q(x_t \mid x_0)\, dx_0.$$

So $\nabla_{x_t} \log q(x_t) = \mathbb{E}[\nabla_{x_t} \log q(x_t \mid x_0) \mid x_t]$: the marginal score is the posterior average of the conditional score — and the conditional one is Gaussian.

The Score Is the Noise Prediction

derive

Differentiating $q(x_t \mid x_0) = \mathcal{N}(\sqrt{\overline{\alpha}_t} \cdot x_0,\ (1 - \overline{\alpha}_t)I)$, then substituting $x_t - \sqrt{\overline{\alpha}_t} \cdot x_0 = \sqrt{1 - \overline{\alpha}_t} \cdot \varepsilon$,

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{x_t - \sqrt{\overline{\alpha}_t} \cdot x_0}{1 - \overline{\alpha}_t} = -\frac{\varepsilon}{\sqrt{1 - \overline{\alpha}_t}}.$$

The Score Is the Noise Prediction

derive

Differentiating $q(x_t \mid x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t} \cdot x_0,\ (1 - \bar{\alpha}_t)I)$, then substituting $x_t - \sqrt{\bar{\alpha}_t} \cdot x_0 = \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon$,

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{x_t - \sqrt{\bar{\alpha}_t} \cdot x_0}{1 - \bar{\alpha}_t} = -\frac{\varepsilon}{\sqrt{1 - \bar{\alpha}_t}}.$$

Take the posterior average. Only ε is random, and its mean is what squared-error training fits:

$$\boxed{\nabla_{x_t} \log q(x_t) \;=\; -\frac{\mathbb{E}[\varepsilon \mid x_t]}{\sqrt{1 - \bar{\alpha}_t}} \;\approx\; -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1 - \bar{\alpha}_t}}}$$

The Score Is the Noise Prediction

derive

Differentiating $q(x_t \mid x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t} \cdot x_0,\ (1 - \bar{\alpha}_t)I)$, then substituting $x_t - \sqrt{\bar{\alpha}_t} \cdot x_0 = \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon$,

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{x_t - \sqrt{\bar{\alpha}_t} \cdot x_0}{1 - \bar{\alpha}_t} = -\frac{\varepsilon}{\sqrt{1 - \bar{\alpha}_t}}.$$

Take the posterior average. Only ε is random, and its mean is what squared-error training fits:

$$\boxed{\nabla_{x_t} \log q(x_t) \;=\; -\frac{\mathbb{E}[\varepsilon \mid x_t]}{\sqrt{1 - \bar{\alpha}_t}} \;\approx\; -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1 - \bar{\alpha}_t}}}$$

A score and a noise prediction are the same object.

The Guided Score

derive

Reverse sampling needs the reweighting at every noise level, so apply it to the marginal at t: $q_w(x_t \mid c) \propto q(x_t) \cdot q(c \mid x_t)^w$. Take ∇_{x_t} log; the product becomes a sum, and $\log q(c \mid x_t) = \log q(x_t \mid c) - \log q(x_t) + \text{const}$ removes the classifier:

$$\nabla_{x_t} \log q_w(x_t \mid c) = \nabla_{x_t} \log q(x_t) + w \cdot (\nabla_{x_t} \log q(x_t \mid c) - \nabla_{x_t} \log q(x_t)).$$

The Guided Score

derive

Reverse sampling needs the reweighting at every noise level, so apply it to the marginal at t: $q_w(x_t \mid c) \propto q(x_t) \cdot q(c \mid x_t)^w$. Take ∇_{x_t} log; the product becomes a sum, and $\log q(c \mid x_t) = \log q(x_t \mid c) - \log q(x_t) + \text{const}$ removes the classifier:

$$\nabla_{x_t} \log q_w(x_t \mid c) = \nabla_{x_t} \log q(x_t) + w \cdot (\nabla_{x_t} \log q(x_t \mid c) - \nabla_{x_t} \log q(x_t)).$$

Every term on the right is a score — one unconditional, one conditional on c. And a score is a noise prediction.

Classifier-Free Guidance

Replace each score by $-\epsilon_\theta / \sqrt{1-\overline{\alpha}_t}$. The factor is the same in all three places, so it cancels, and a statement about densities becomes one about noise predictions:

$$\boxed{\epsilon_{\text{guide}} = \epsilon_\theta(x_t, t, \emptyset) + w \cdot (\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \emptyset))}$$

where $\emptyset$ is the empty prompt.

Classifier-Free Guidance

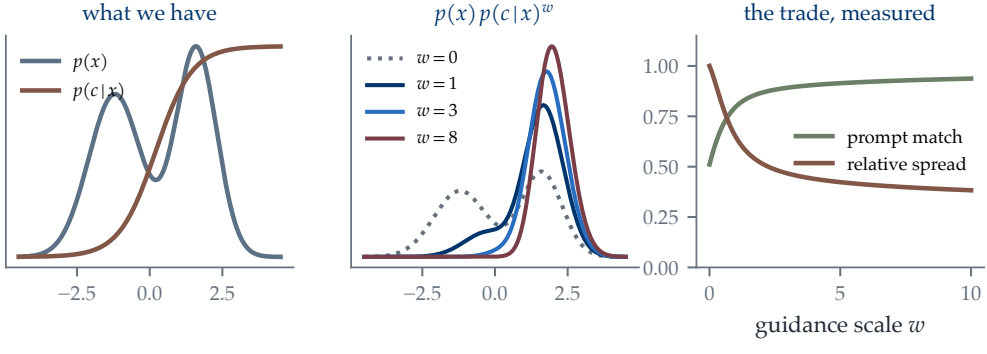
Replace each score by $-\epsilon_\theta/\sqrt{1-\overline{\alpha}_t}$. The factor is the same in all three places, so it cancels, and a statement about densities becomes one about noise predictions:

$$\boxed{\epsilon_{\text{guide}} = \epsilon_\theta(x_t, t, \emptyset) + w \cdot (\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \emptyset))}$$

where $\emptyset$ is the empty prompt.

- The classifier is gone: Bayes' rule traded $\nabla_x \log p(c \mid x)$ for two things we already have. Hence **classifier-free guidance**.
- One network serves both terms: during training the prompt is dropped to $\emptyset$ perhaps 10% of the time.
- The price is two network calls per step instead of one.

Effect of the Guidance Scale



An exact one-dimensional calculation. Left: a two-mode prior and a prompt likelihood. Middle: raising the likelihood to the power w sharpens the favoured mode — and by $w = 8$ has deleted the other one. Right: prompt match rises and spread collapses. Typical image systems use w between 5 and 8.

Two-component prior, logistic likelihood, exact normalisation.

The Guided Sampling Loop in PyTorch

```
x = torch.randn(n, 3, 512, 512)                  # pure noise, full resolution
c, null = text_encoder(prompt), text_encoder("")

for t in reversed(range(1000)):                  # every level, as in Lecture 18
    e_cond = unet(x, t, c)                       # two calls per step
    e_null = unet(x, t, null)
    eps = e_null + w * (e_cond - e_null)         # classifier-free guidance

    mu = (x - (1 - a[t]) / (1 - a_bar[t]).sqrt() * eps) / a[t].sqrt()
    x  = mu + sigma[t] * torch.randn_like(x) * (t > 0)      # DDPM step
```

Lecture 18's sampler unchanged, except that the prompt enters twice and the two predictions are combined before the step.

Cost of One Generated Image

Sampling as written above, at full resolution:

$$\underbrace{1000}_{\text{steps}} \times \underbrace{2}_{\text{guidance}} = 2000 \text{ U-Net calls on } 3 \times 512^2.$$

Cost of One Generated Image

Sampling as written above, at full resolution:

$$\underbrace{1000}_{\text{steps}} \times \underbrace{2}_{\text{guidance}} = 2000 \text{ U-Net calls on } 3 \times 512^2.$$

Minutes on a good GPU for a single picture. Conditioning works, and it is far too slow to use.

Two factors are separately attackable, and the rest of the lecture attacks both:

- Make each call cheaper — run the diffusion on a small latent instead of on pixels;
- Make fewer calls — take 50 steps instead of 1000.

Outline

Diffusion in the Latent Space of an Autoencoder

An audoencoder gives us a trained pair of encoder and decoder for compression:

$$\underbrace{E : x \mapsto z}_{\text{encoder}} \quad \text{and} \quad \underbrace{D : z \mapsto \tilde{x}}_{\text{decoder}}, \qquad x \in \mathbb{R}^{3\times 512^2}, \quad z \in \mathbb{R}^{4\times 64^2},$$

with $D(E(x)) \approx x$. The code z is a much smaller description of the same picture.

Diffusion in the Latent Space of an Autoencoder

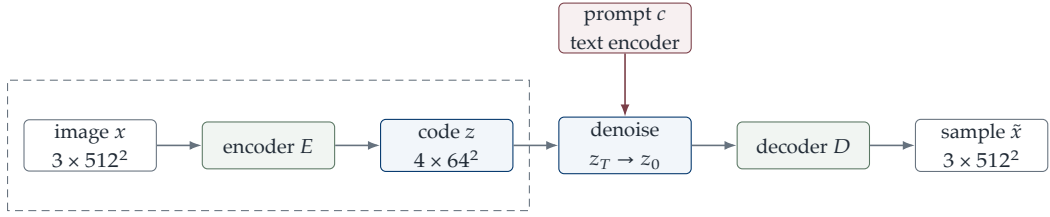
An audoencoder gives us a trained pair of encoder and decoder for compression:

$$\underbrace{E:\ x \mapsto z}_{\text{encoder}} \qquad \text{and} \qquad \underbrace{D:\ z \mapsto \tilde{x}}_{\text{decoder}}, \qquad x \in \mathbb{R}^{3\times 512^2}, \quad z \in \mathbb{R}^{4\times 64^2},$$

with $D(E(x)) \approx x$. The code z is a much smaller description of the same picture. **Nothing in DDPM required the state to be pixels.** Freeze a good E and D, and run the whole construction on z:

- Training: encode each image once, then the same forward process, the same U-Net, the same noise-prediction loss — on $z = E(x)$;
- Sampling: draw $z_T \sim \mathcal{N}(0, I)$, take the same reverse steps to get $z_{T-1}, \dots, z_0$, and decode once at at end — output $x = D(z_0)$.

The Latent Diffusion Pipeline



E and D are trained a priori on images, and then frozen during DDPM training.

Training only uses encoder and generation only uses decoder.

Perceptual and Semantic Information

A photograph carries two very different kinds of information:

Semantic content

there is a horse; it is chestnut; the light comes from the left

Perceptual detail

the exact grain of the sand, the sensor noise, the sub-pixel edge positions

A pixel-space diffusion model spends almost all of its compute on the second column — detail a viewer cannot distinguish and an autoencoder can regenerate.

Perceptual and Semantic Information

A photograph carries two very different kinds of information:

Semantic content

there is a horse; it is chestnut; the light comes from the left

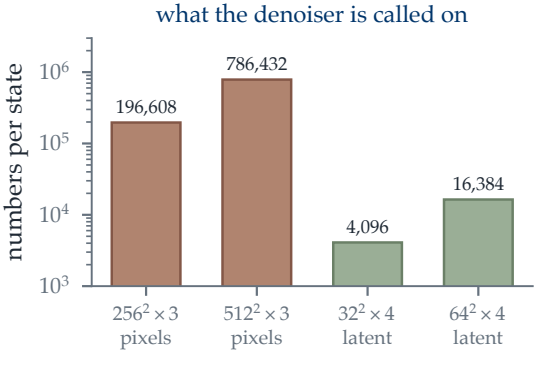
Perceptual detail

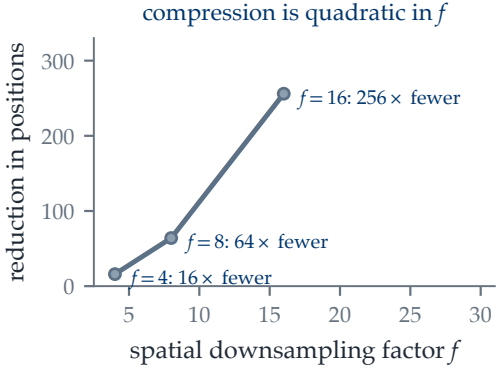
the exact grain of the sand, the sensor noise, the sub-pixel edge positions

A pixel-space diffusion model spends almost all of its compute on the second column — detail a viewer cannot distinguish and an autoencoder can regenerate. **This suggests separating the two tasks:** an autoencoder handles perceptual detail, and diffusion models focus on semantic content.

Source: The *perceptual* versus *semantic compression* split is the argument of Rombach et al. (2022), §1 and Figure 2.

State Size and Cost per Step

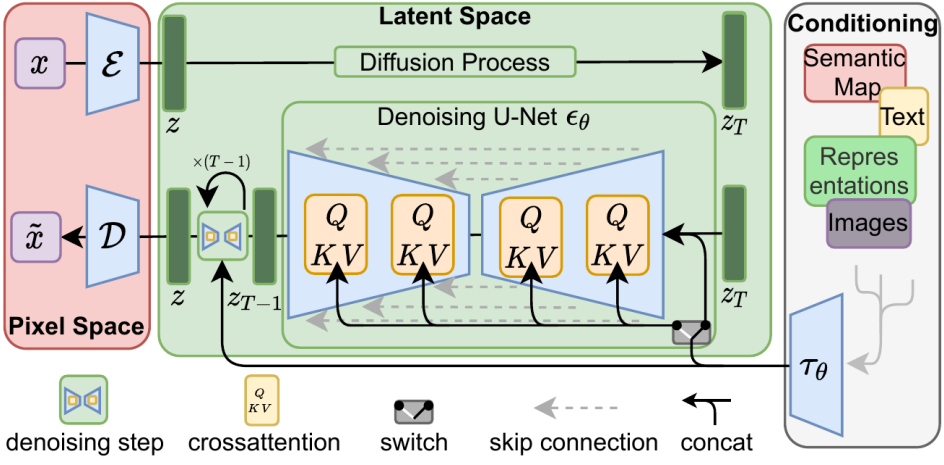




A 512 × 512 image has 262,144 positions; its $f = 8$ latent code has 4,096. Because compression is quadratic in the downsampling factor, the same U-Net at the same width becomes roughly 64× cheaper per step, and a step count that was unaffordable becomes routine.

Shapes from the Stable Diffusion v1 configuration.

The Latent Diffusion Architecture



The Q, K, V boxes are **cross-attention**, the route by which the prompt reaches the denoiser. We will see attention in the next lecture. Rombach et al., *High-Resolution Image Synthesis with Latent Diffusion Models*.

Is the Latent Code Discrete?

VQ-VAE snaps each encoder output to the nearest of K codebook vectors, so an image becomes a grid of integers — and Gaussian noise on an integer index is meaningless. Two ways out:

Is the Latent Code Discrete?

VQ-VAE snaps each encoder output to the nearest of K codebook vectors, so an image becomes a grid of integers — and Gaussian noise on an integer index is meaningless. Two ways out:

1. Keep the diffusion Gaussian. Run it on the encoder output before quantization. At generation, sample a continuous z_0 and quantize on the way into the decoder: the quantizer is simply the decoder's first layer. This is latent diffusion.

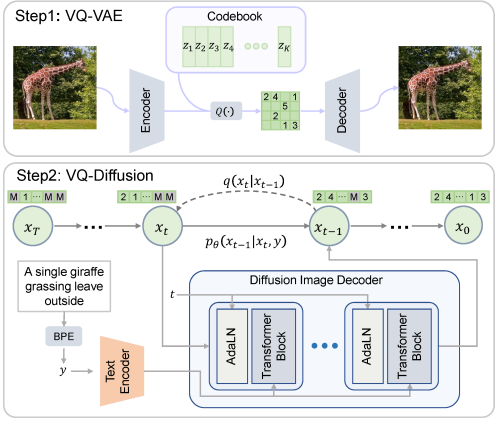
Is the Latent Code Discrete?

VQ-VAE snaps each encoder output to the nearest of K codebook vectors, so an image becomes a grid of integers — and Gaussian noise on an integer index is meaningless. Two ways out:

1. Keep the diffusion Gaussian. Run it on the encoder output before quantization. At generation, sample a continuous z_0 and quantize on the way into the decoder: the quantizer is simply the decoder's first layer. This is latent diffusion.
2. Make the diffusion discrete. Diffuse the integers themselves, corrupting tokens towards a mask symbol instead of towards Gaussian noise. This is VQ-Diffusion.

Source: In the latent-diffusion code, `VQModelInterface.decode` quantizes before decoding.

VQ-Diffusion



Diffusion run on a grid of integers. The only change from Lecture 18: the corruption replaces tokens with a mask symbol `M` instead of adding Gaussian noise.

Gu et al., *Vector Quantized Diffusion Model for Text-to-Image Synthesis.*

The Latent Diffusion Objective

Pixel space, from Lecture 18:

$$\mathcal{L}_{\text{DM}} = \mathbb{E}_{x,\varepsilon,t}[\|\varepsilon - \epsilon_\theta(x_t, t)\|^2].$$

Latent space, with a frozen encoder E and a prompt c:

$$\boxed{\mathcal{L}_{\text{LDM}} = \mathbb{E}_{E(x),\varepsilon,t}[\|\varepsilon - \epsilon_\theta(z_t, t, c)\|^2]}$$

Two substitutions: $x \mapsto E(x)$, and one extra argument to the network. The derivation, the schedule, and the sampler are untouched.

Source: Rombach et al. (2022), equations (1) and (2).

The Reconstruction Ceiling

The same image encoded and decoded at three compression factors, with no diffusion involved. At $f = 4$ the reconstruction is faithful; by $f = 16$ fine structure is gone. No amount of denoising in the latent space can put back detail the decoder cannot draw.

Rombach et al. (2022), Figure 1 panels; credited educational excerpt.

Photorealistic Surreal Images

"A photo of a Shiba Inu dog with a backpack riding a bike, wearing sunglasses and a beach hat"; "a very happy fuzzy panda dressed as a chef making dough"; "a cute corgi lives in a house made out of sushi".

Saharia et al., *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*.

Photorealistic Surreal Images

"Teddy bears swimming at the Olympics 400m Butterfly event"; "a cute sloth holding a small treasure chest, a bright golden glow coming from the chest"; "a strawberry mug floating in a dark chocolate sea". No training image contains any of these: the model learned the parts, and the prompt says how to compose them.

Saharia et al., *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*.

Outline

Limitations of Ancestral Sampling

DDPM sampling must go through all T levels $(x_T, x_{T-1}, \dots, x_0)$, and at each level it injects fresh noise. It relies on the Markov chain structure.

- Skipping levels breaks the Markov chain and the distribution of generated x_0 is no longer correct;
- Injected noise means the same x_T gives a different image every time;
- A thousand sequential network calls per image is expensive.

Limitations of Ancestral Sampling

DDPM sampling must go through all T levels $(x_T, x_{T-1}, \dots, x_0)$, and at each level it injects fresh noise. It relies on the Markov chain structure.

- Skipping levels breaks the Markov chain and the distribution of generated x_0 is no longer correct;
- Injected noise means the same x_T gives a different image every time;
- A thousand sequential network calls per image is expensive.

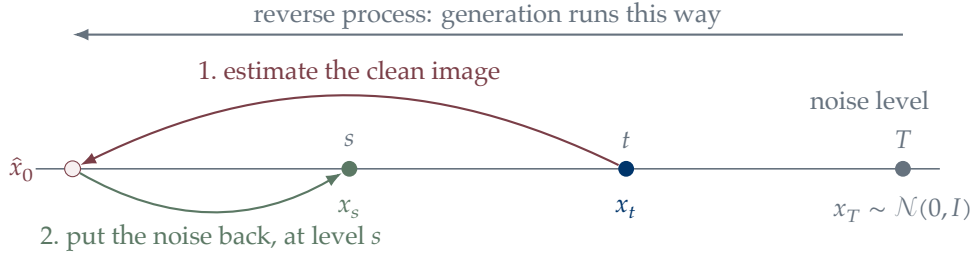
DDIM (denoising diffusion implicit models) fixes all three issue, reusing the network ε_θ already trained.

Idea of DDIM Generation

In the forward process x_t depends on x_0 alone, through one formula: $x_t = \sqrt{\overline{\alpha}_t} \cdot x_0 + \sqrt{1 - \overline{\alpha}_t} \cdot \varepsilon$. There is no chain in it.

Idea of DDIM Generation

In the forward process x_t depends on x_0 alone, through one formula:
$x_t = \sqrt{\overline{\alpha}_t} \cdot x_0 + \sqrt{1 - \overline{\alpha}_t} \cdot \varepsilon$. There is no chain in it.



$$\hat{x}_0 = \frac{x_t - \sqrt{1-\overline{\alpha}_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\overline{\alpha}_t}}, \qquad \boxed{x_s = \sqrt{\overline{\alpha}_s} \cdot \hat{x}_0 + \sqrt{1 - \overline{\alpha}_s} \cdot \epsilon_\theta(x_t, t)}$$

Properties of the DDIM Update

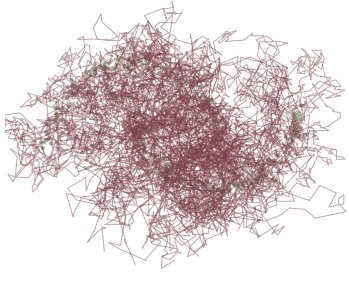
With $\hat{x}_0$ computed from the current x_t and the network's noise estimate, $\hat{x}_0 = (x_t - \sqrt{1-\overline{\alpha}_t} \cdot \epsilon_\theta(x_t, t))/\sqrt{\overline{\alpha}_t}$, the update reads

$$x_s = \underbrace{\sqrt{\overline{\alpha}_s} \cdot \hat{x}_0}_{\text{point at the guess}} + \underbrace{\sqrt{1-\overline{\alpha}_s} \cdot \epsilon_\theta(x_t, t)}_{\text{put back the } \textit{right amount} \text{ of the } \textit{same} \text{ noise}} .$$

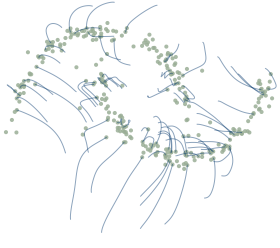
- No fresh randomness. Given x_T, the output is a deterministic function.
- Levels may be skipped. The update holds for any $s < t$, so generation can visit fifty levels instead of all thousand.
- Same neural network. ϵ_θ is the network trained by the DDPM loss.

DDPM and DDIM Sampling Paths

DDPM: noise injected at every step



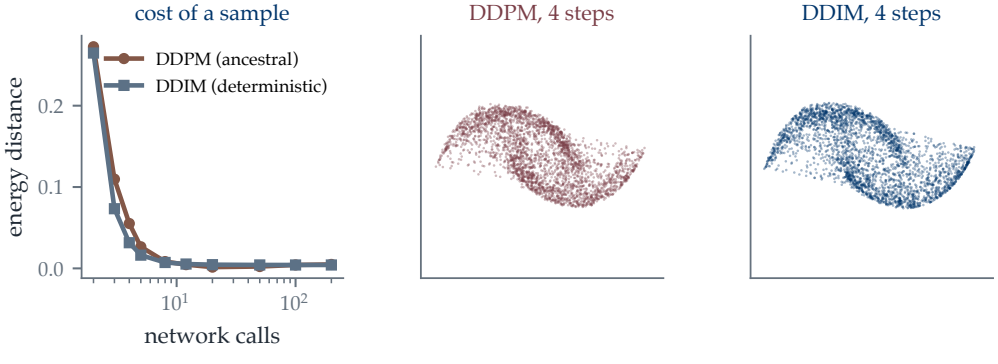
DDIM: no noise injected



The same trained network, the same 300 starting points, the same 120 network calls. Left: **DDPM** injects noise at every step, so each path is a random walk that happens to drift onto the data. Right: **DDIM** follows a smooth curve. Both arrive; only one is reproducible.

Ancestral and deterministic samplers from one two-moons model, seed 11.

Sample Quality against the Number of Network Calls



Energy distance to the target against network calls, one trained model. Few calls: DDIM wins — at four calls its error is 0.032 against 0.055, because a step with no injected noise is simply more accurate. Many calls: DDPM overtakes it, and both settle near the floor.

Two-moons model, 3,000 samples per point, seed 3. The same crossover holds at scale: on CIFAR-10, FID at 10 steps is 13.4 for DDIM against 41.1 for DDPM, but at 1000 steps DDPM reaches 3.2 against DDIM's 4.0 (Song et al., Table 1).

Which Sampler to Use

Both samplers run the same trained network; they differ only in whether noise is injected and whether levels may be skipped.

- Ten to fifty calls — the regime anyone actually generates in: **DDIM** is far better, because DDPM cannot skip levels without breaking its chain.
- A thousand calls: **DDPM** edges ahead, at 20× the compute.
- DDIM is also reproducible: with no injected noise, a seed fixes the image exactly.

Which Sampler to Use

Both samplers run the same trained network; they differ only in whether noise is injected and whether levels may be skipped.

- Ten to fifty calls — the regime anyone actually generates in: **DDIM** is far better, because DDPM cannot skip levels without breaking its chain.
- A thousand calls: **DDPM** edges ahead, at 20× the compute.
- DDIM is also reproducible: with no injected noise, a seed fixes the image exactly.

Practical image systems ship with 20–50 DDIM steps.

Outline

Curvature of the Diffusion Path

All the forward process ever had to do was supply a known path joining p_{data} to $\mathcal{N}(0, I)$, which generation then walks backwards. Lecture 18 built one from

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon,$$

chosen to hold the variance fixed and keep $q(x_t \mid x_0)$ Gaussian.

Curvature of the Diffusion Path

All the forward process ever had to do was supply a known path joining p_{data} to $\mathcal{N}(0, I)$, which generation then walks backwards. Lecture 18 built one from

$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon,$$

chosen to hold the variance fixed and keep $q(x_t \mid x_0)$ Gaussian.

That choice pins the weights to $(\sqrt{\bar{\alpha}_t})^2 + (\sqrt{1 - \bar{\alpha}_t})^2 = 1$, so they sweep a quarter circle, and a curved path is expensive to integrate in few steps.

Curvature of the Diffusion Path

All the forward process ever had to do was supply a known path joining p_{data} to $\mathcal{N}(0, I)$, which generation then walks backwards. Lecture 18 built one from

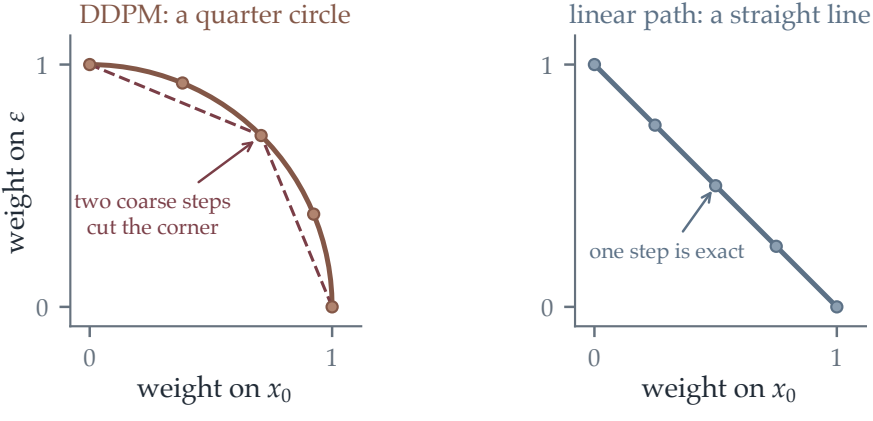
$$x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon,$$

chosen to hold the variance fixed and keep $q(x_t \mid x_0)$ Gaussian.

That choice pins the weights to $(\sqrt{\bar{\alpha}_t})^2 + (\sqrt{1 - \bar{\alpha}_t})^2 = 1$, so they sweep a quarter circle, and a curved path is expensive to integrate in few steps.

Nothing forced that choice. Could a **straight** path from noise to data do the same job?

Two Ways to Join Noise to Data



Both paths carry the same endpoints, x_0 at one end and ε at the other. DDPM holds the variance fixed, which pins its weights to the unit circle. The linear path takes the chord instead, so its velocity never changes and one step lands exactly on the answer.

The Linear Interpolation Path

Pair a noise draw with a data draw and interpolate:

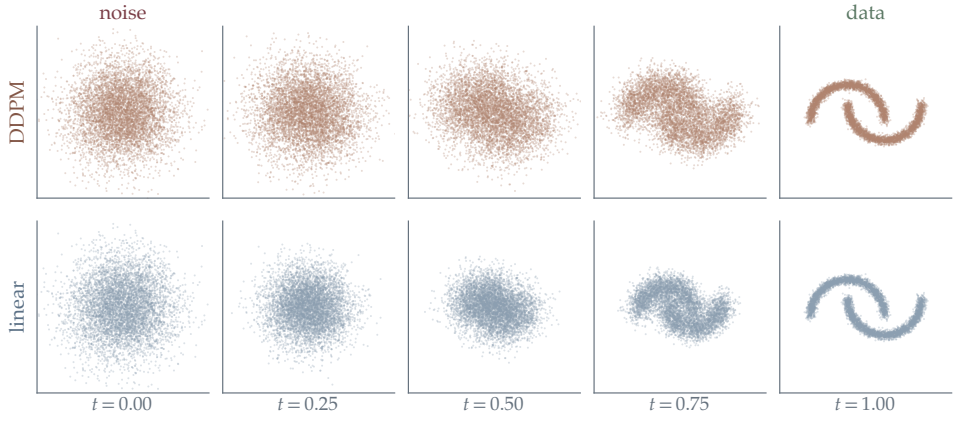
$$x_t = (1-t)\cdot x_0 + t\, x_1, \qquad x_0 \sim \mathcal{N}(0, I),\ x_1 \sim p_{\text{data}}, \quad \forall t \in [0,1].$$

Differentiating in t, we have an ODE

$$\frac{dx_t}{dt} = \underbrace{x_1 - x_0}_{\text{constant along the pair}}.$$

No schedule, no coefficients: the velocity of this path is exactly $x_1 - x_0$, the same at every t. It is the analogue of "the noise" in the diffusion loss — a target we can write down from the two endpoints alone.

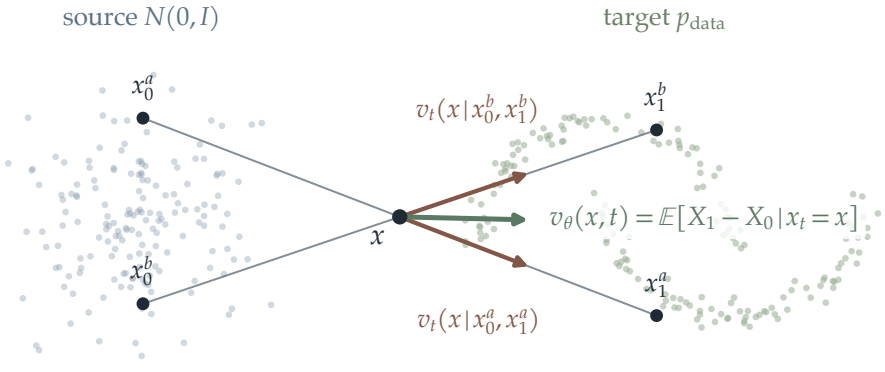
Marginals along the Linear Path



The same five times under both schedules. Under DDPM the moons appear only at the very end; along the linear path the structure arrives at a steady rate.

Two-moons target, independent pairing, seed 26521.

Estimation Target: Averaged Velocity at x_t



Two pairs pass through the same x at the same t, each wanting its own velocity (orange). A function of x cannot return both, so we take their average direction — and that is the velocity field we train.

After Helbling, *A Visual Introduction to Rectified Flows*, `https://alechelbling.com/blog/rectified-flow/`.

The Conditional Flow-Matching Loss

derive

The field we want is that average, one velocity per point:

$$\boxed{v^\star(x,t) = \mathbb{E}[\, x_1 - x_0 \mid x_t = x \,]}$$

The Conditional Flow-Matching Loss

derive

The field we want is that average, one velocity per point:

$$\boxed{v^\star(x,t) = \mathbb{E}[\, x_1 - x_0 \mid x_t = x \,]}$$

We cannot evaluate that expectation. But squared loss regresses onto the conditional mean, so it is the minimiser of an objective built from the pair velocity — which we can write down:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,x_0,x_1}[\|v_\theta(x_t,t) - (x_1 - x_0)\|^2].$$

The Conditional Flow-Matching Loss

derive

The field we want is that average, one velocity per point:

$$\boxed{v^\star(x,t) = \mathbb{E}[\, x_1 - x_0 \mid x_t = x \,]}$$

We cannot evaluate that expectation. But squared loss regresses onto the conditional mean, so it is the minimiser of an objective built from the pair velocity — which we can write down:

$$\mathcal{L}_{\mathrm{CFM}}(\theta) = \mathbb{E}_{t,x_0,x_1}[\|v_\theta(x_t,t) - (x_1 - x_0)\|^2].$$

The minimiser is not an approximation to the velocity we want: it is $v^\star$.

Computing It in Practice

Every term is something we can sample, so a training step is three draws:

1. An image $x_1 \sim p_{\text{data}}$ and noise $x_0 \sim \mathcal{N}(0, I)$, drawn independently;
2. A time $t \sim \text{Unif}[0, 1]$, giving $x_t = (1 - t) \cdot x_0 + t \cdot x_1$;
3. A gradient step on $\|v_\theta(x_t, t) - (x_1 - x_0)\|^2$.

Computing It in Practice

Every term is something we can sample, so a training step is three draws:

1. An image $x_1 \sim p_{\text{data}}$ and noise $x_0 \sim \mathcal{N}(0, I)$, drawn independently;
2. A time $t \sim \text{Unif}[0, 1]$, giving $x_t = (1 - t) \cdot x_0 + t \cdot x_1$;
3. A gradient step on $\|v_\theta(x_t, t) - (x_1 - x_0)\|^2$.

No schedule, no variance formula — and the expectation is never formed. Once v_θ is fitted, generation starts from $x \sim \mathcal{N}(0, I)$ and follows the field with an **ODE solver**, one network call per step; appendix B.2 works through a step.

Where This Objective Comes From

Lecture 18's loss ended a long argument: write the ELBO, decompose it, reduce the KL to mean matching, drop the weights. The bound did the work.

Where This Objective Comes From

Lecture 18's loss ended a long argument: write the ELBO, decompose it, reduce the KL to mean matching, drop the weights. The bound did the work.

Here there is no bound at all. The objective is a plain least-squares regression, justified entirely by the line above: its minimiser is $\mathbb{E}[x_1 - x_0 \mid x_t]$, and that field carries $\mathcal{N}(0, I)$ to p_{data} along the chosen marginals.

Where This Objective Comes From

Lecture 18's loss ended a long argument: write the ELBO, decompose it, reduce the KL to mean matching, drop the weights. The bound did the work.

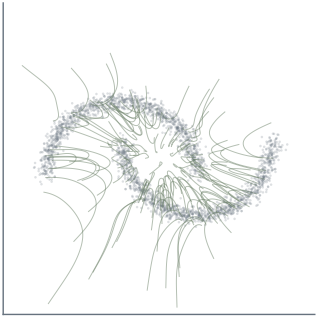
Here there is no bound at all. The objective is a plain least-squares regression, justified entirely by the line above: its minimiser is $\mathbb{E}[x_1 - x_0 \mid x_t]$, and that field carries $\mathcal{N}(0, I)$ to p_{data} along the chosen marginals.

Regressing on the marginal field directly is impossible. Conditioning on the pair gives a target we can write down, and the two objectives differ by a constant, so they have the same gradients.

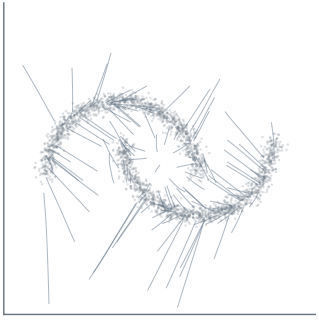
How Straight Are the Sampling Paths?

DDIM
length / distance = 1.26

flow matching
length / distance = 2.18



flow matching + one reflow
length / distance = 1.00

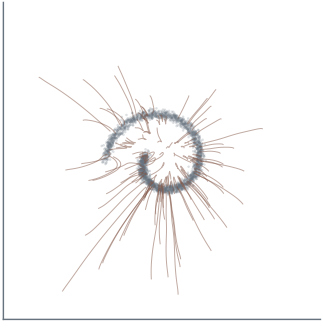


Path length over endpoint distance; 1.00 is exactly straight. A straight target does not give a straight sampling path: flow matching is in fact the most curved of the three, because the field must average over pairs that cross. **Rectified flow** straightens it.

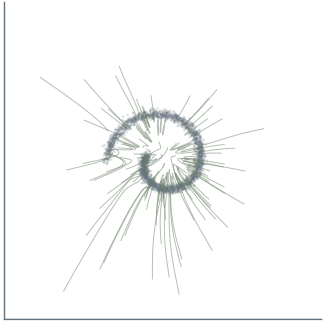
Same target, same network, 360 shared starting points, 60 steps, seed 11. Rectified flow is derived in appendix B.3.

The Same Comparison on a Spiral

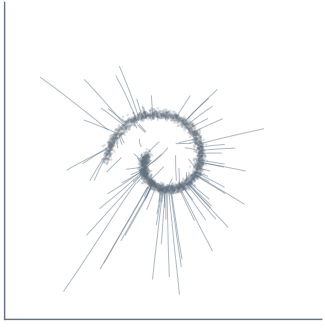
DDIM
length / distance = 1.91



flow matching
length / distance = 3.11



flow matching + one reflow
length / distance = 1.01



A harder target, and the same ordering, more sharply.

Identical network and budget; 360 shared starting points, 60 steps, seed 11.

Outline

When the Data Is Intrinsically Discrete

Gaussian noise needs the data to have magnitudes. A codebook entry, or a word, is a label: there is no useful meaning to

$$\text{“horse”} + 0.3 \cdot \varepsilon,$$

and the halfway point between two vocabulary entries is not a word.

When the Data Is Intrinsically Discrete

Gaussian noise needs the data to have magnitudes. A codebook entry, or a word, is a label: there is no useful meaning to

$$\text{“horse”} + 0.3 \cdot \varepsilon,$$

and the halfway point between two vocabulary entries is not a word. But the recipe never mentioned Gaussians. It needs a corruption we control that ends somewhere we can sample, a closed form at level t, and a learnable rule for undoing a step.

Any corruption with those three properties defines a diffusion model.

Diffusion Language Models

Take the data to be a sequence of words from a finite vocabulary $\mathcal{V}$:

$$x_0 = (x_0^1, \dots, x_0^L), \qquad x_0^i \in \mathcal{V}.$$

- Unconditional: generate a sequence of L words;
- Conditional: the prompt is the condition c, and the model generates the continuation — the same c that carried a text embedding earlier, now a prefix of words. Conditional diffusion can be applied.

Diffusion Language Models

Take the data to be a sequence of words from a finite vocabulary $\mathcal{V}$:

$$x_0 = (x_0^1, \dots, x_0^L), \qquad x_0^i \in \mathcal{V}.$$

- Unconditional: generate a sequence of L words;
- Conditional: the prompt is the condition c, and the model generates the continuation — the same c that carried a text embedding earlier, now a prefix of words. Conditional diffusion can be applied.

We need a corruption that destroys a sequence of words and can be undone.
Replacing words with a **mask** symbol does both.

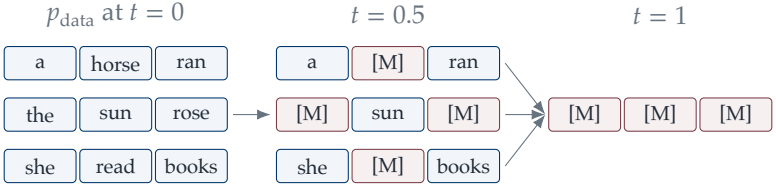
Masking as the Forward Process

Let `[M]` be a symbol outside the vocabulary. At level $t \in [0, 1]$, replace each position independently with `[M]` with probability t:

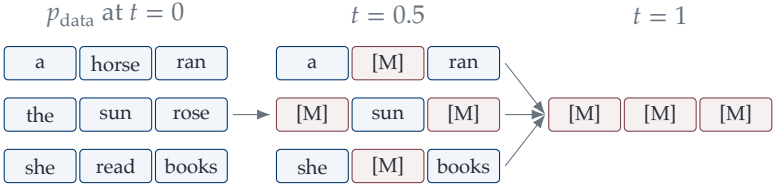
$$q(x_t^i \mid x_0^i) = \begin{cases} \texttt{[M]} & \text{with probability } t, \\ x_0^i & \text{with probability } 1 - t. \end{cases}$$

- At $t = 0$ the sentence is intact;
- Any level is one draw away from x_0 — the closed form that $q(x_t \mid x_0)$ gave us before;
- At $t = 1$ every position is `[M]`, whatever the sentence was.

Where the Forward Process Ends



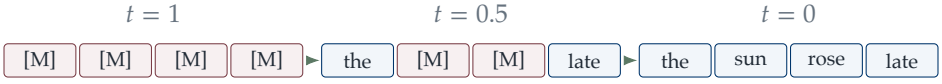
Where the Forward Process Ends



Gaussian diffusion ended at $\mathcal{N}(0, I)$ — still a distribution to draw from. Masking ends at a single state: every sentence collapses to the same all-`[M]` sequence.

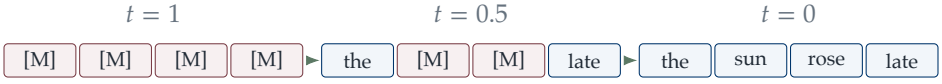
So the endpoint is free to sample. Generation starts from the all-mask sequence and unmasks.

The Reverse Process: Unmasking



Each step predicts a word at every masked position at once, then fills in some of them.

The Reverse Process: Unmasking

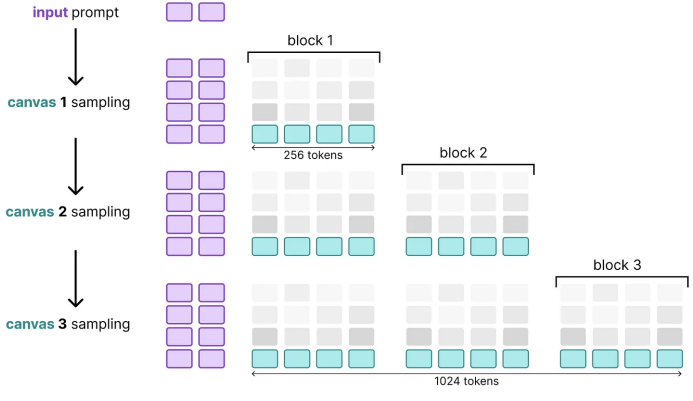


Each step predicts a word at every masked position at once, then fills in some of them.

Conditioning needs nothing new: put the prompt in front and never mask it.

The loss is a cross-entropy on the masked positions; appendix C.3.

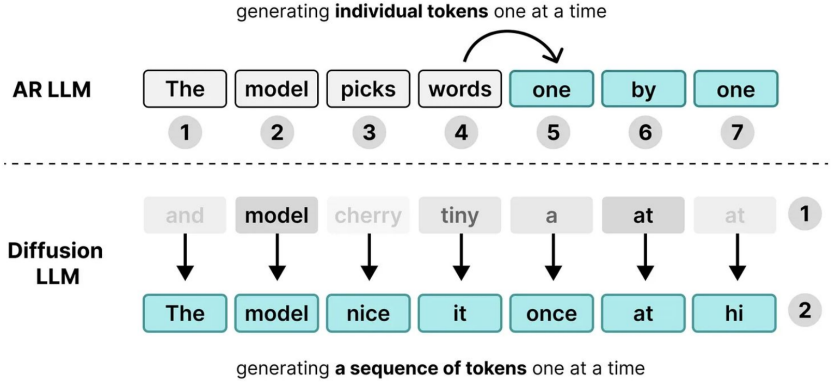
DiffusionGemma (Google's DLLM)



The prompt (purple) is never masked — it is the condition c. The model unmasks a canvas of 256 words at once, then repeats block by block for a longer answer.

Kuleshov and Arriola, *How to Build a Diffusion Language Model*; walked through in Grootendorst, *A Visual Guide to DiffusionGemma*. Both in appendix D.3.

Parallel Generation



A conventional model writes one word at a time, so a 500-word answer costs 500 sequential calls. A masked diffusion model fills in many positions per call, and the number of rounds is a choice — the same trade as the step count for images.

The figure says "tokens" where we say words. Kuleshov and Arriola, *How to Build a Diffusion Language Model*.

Summary: Making Generation Practical

$$\epsilon_{\text{guide}} = \epsilon_\theta(z_t, t, \emptyset) + w(\epsilon_\theta(z_t, t, c) - \epsilon_\theta(z_t, t, \emptyset)), \qquad z = E(x).$$

1. **DDIM** reuses the trained network with a noise-free sampler that may skip levels: same weights, far fewer steps, and best when the budget is small.
2. **Latent diffusion** runs the identical process on a learned code, paying a reconstruction ceiling for a large factor in cost.
3. **Guidance** tilts the density toward the prompt, buying fidelity with coverage, at two network calls per step.

Summary: Choosing the Path

$$x_t = (1-t)x_0 + tx_1, \qquad \min_\theta \ \mathbb{E}_{t,x_0,x_1}\|v_\theta(x_t,t) - (x_1 - x_0)\|^2.$$

1. **Flow matching** keeps the ODE but designs a straighter one: replace the noising schedule with a straight interpolation and regress its constant velocity.
2. Its correctness rests on the same move as Lecture 18: condition on the endpoints to get a writable target, and let squared loss marginalise them out.
3. **Masked diffusion** swaps Gaussian noise for hiding tokens, giving generative language models that fill in many positions per call.

Next Lecture

Generation is now practical: conditioned on a prompt, cheap per step, and short. Every model in the course so far has learned from a data set someone else collected.

Lecture 20 — Reinforcement Learning changes that. An agent that acts decides what it will see next, and that single change breaks most of the statistics we have been relying on.

Reading for this lecture: Rombach et al. (2022); Lipman et al. (2023).

Appendix

Notes skipped in the lecture hour, in four groups.

A. **Diffusion models.** The family of samplers between DDPM and DDIM, and the noise prediction read as a score.

B. **Flow matching and rectified flow.** Why the conditional loss is legitimate, why the paths curve, and how reflow straightens them.

C. **Discrete diffusion.** The forward and reverse processes for masking, and the objective.

D. **Further reading.**

A.1 The Stochastic Family between DDPM and DDIM

Song et al. (2021) introduce $\eta \in [0, 1]$ and the update

$$x_s = \sqrt{\bar{\alpha}_s} \cdot \hat{x}_0 + \sqrt{1 - \bar{\alpha}_s - \sigma^2} \cdot \epsilon_\theta(x_t, t) + \sigma z,$$

$$\sigma = \eta \sqrt{\frac{1 - \bar{\alpha}_s}{1 - \bar{\alpha}_t}} \sqrt{1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_s}}, \qquad z \sim \mathcal{N}(0, I).$$

At $\eta = 0$ this is the deterministic DDIM step; at $\eta = 1$ with $s = t - 1$ it is the DDPM ancestral step. The trained network is the same in both cases.

A.2 The Noise Prediction as a Score

Differentiating the Gaussian forward marginal and substituting $x_t - \sqrt{\bar{\alpha}_t} \cdot x_0 = \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon$,

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{\varepsilon}{\sqrt{1 - \bar{\alpha}_t}}, \qquad \text{so} \qquad \nabla_x \log q(x_t) \approx -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1 - \bar{\alpha}_t}}.$$

The network never sees the word “score”, but that is what it estimates: the direction in which the noisy density rises fastest. Moving along it carries a random point toward regions of higher data density, which is the geometric content of the reverse process.

Source: Song and Ermon (2019); Song et al. (2021), *Score-Based Generative Modeling through Stochastic Differential Equations*.

A.3 Interfaces at a Glance

	diffusion	latent diffusion	flow matching
state	pixels x_t	codes z_t	pixels or codes
target	noise ε	noise, in latent space	velocity $x_1 - x_0$
sample	reverse steps	reverse steps, decode	integrate an ODE
added error	sampler	compression + sampler	field + solver
conditioning	network argument	network argument	network argument

All three train one network by regression and call it repeatedly at sampling time, which is why they are compared at a fixed number of calls.

B.1 The Marginal Field and the Continuity Equation

Write p_t for the law of $x_t = (1-t)x_0 + tx_1$. Mass moved by a velocity field obeys the **continuity equation**

$$\frac{\partial p_t(x)}{\partial t} + \nabla \cdot (p_t(x) v_t(x)) = 0.$$

The conditional field $u_t(x \mid x_0, x_1) = x_1 - x_0$ transports the conditional path by construction. Averaging the conditional fields with the posterior over endpoints,

$$v^\star(x,t) = \mathbb{E}[\, x_1 - x_0 \mid x_t = x \,],$$

gives a field that satisfies the continuity equation for the marginal path. This is the theorem that makes the conditional loss legitimate.

B.2 Generation: Solving the ODE

A fitted velocity field defines an initial-value problem:

$$\frac{dx}{dt} = v_\theta(x, t), \qquad x(0) \sim \mathcal{N}(0, I), \qquad \text{report } x(1).$$

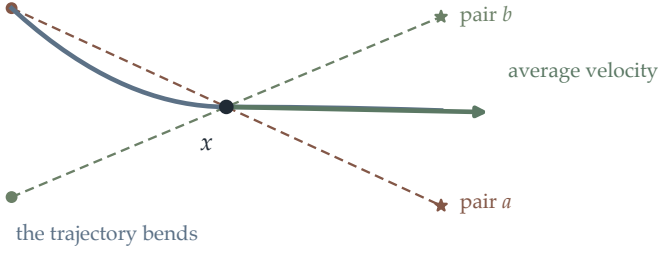
Over a short interval, $x(t + \Delta t) = x(t) + \int_t^{t+\Delta t} v_\theta(x(s), s)\, ds \approx x_k + \Delta t \cdot v_\theta(x_k, t_k)$. With $x_k = (1, -1)$, $\Delta t = 0.2$ and $v_\theta(x_k, t_k) = (0.5, 2)$,

$$x_{k+1} = (1, -1) + 0.2 \cdot (0.5, 2) = \boxed{(1.1,\ -0.6)}.$$

Ten such steps from $t = 0$ to $t = 1$ is a complete sampler: ten network calls, no schedule and no variance formula.

DDIM already integrates an ODE. Flow matching changes which one — it designs a straight path from the start rather than inheriting the curve a noising schedule implies.

B.3 Why Flow Matching Does Not Give Straight Paths

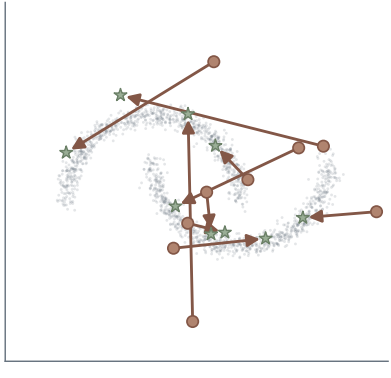


Two pairs cross at x. The field can return only one velocity there, so the trajectory follows neither dashed line.

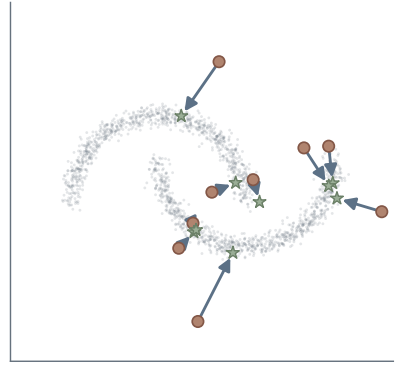
Fix the **pairing**, not the loss.

B.4 Reflow Removes the Crossings

independent pairing: 6 crossings



pairing induced by the flow: 0 crossings



Left: nine noise points paired at random with nine data points, as flow matching trains. Right: the same nine noise points paired with where the trained field actually sends them. Two ODE trajectories through one point would have to coincide, so the induced pairing cannot cross.

Crossings counted, not asserted; same trained field, seed 5.

B.5 Implementing Reflow

It is a retraining step, not a change to the sampler. The loss and the network are untouched; only the data fed to them changes. Given a fitted v_θ:

1. Draw $x_0 \sim \mathcal{N}(0, I)$ in bulk, and run the sampler to get $x_1 = \mathrm{ODE}_{v_\theta}(x_0)$ — this is inference, done once;
2. Store the pairs (x_0, x_1): they are now a **fixed data set**;
3. Train a fresh field on the same objective $\|v_\phi(x_t, t) - (x_1 - x_0)\|^2$ — drawing pairs from that data set instead of pairing independently.

B.5 Implementing Reflow

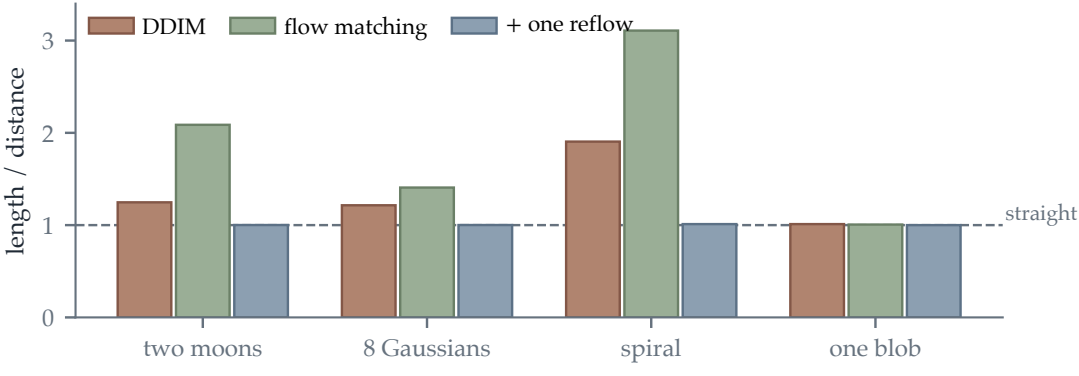
It is a retraining step, not a change to the sampler. The loss and the network are untouched; only the data fed to them changes. Given a fitted v_θ:

1. Draw $x_0 \sim \mathcal{N}(0, I)$ in bulk, and run the sampler to get $x_1 = \mathrm{ODE}_{v_\theta}(x_0)$ — this is inference, done once;
2. Store the pairs (x_0, x_1): they are now a **fixed data set**;
3. Train a fresh field on the same objective $\|v_\phi(x_t, t) - (x_1 - x_0)\|^2$ — drawing pairs from that data set instead of pairing independently.

Iterating straightens the paths, and the marginals are unchanged at every round.

Source: Liu, Gong, Liu (2023), *Flow Straight and Fast*.

B.6 What Reflow Buys



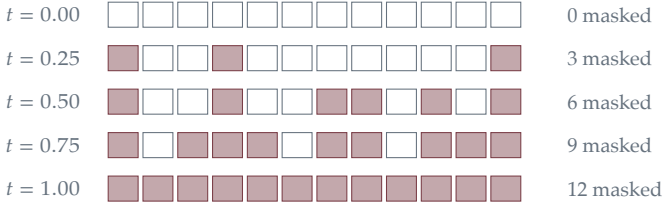
Path length over endpoint distance on four targets, 1.00 being straight. Reflow reaches 1.00 everywhere. Only for a single distant blob — where nothing crosses in the first place — do all three agree without it.

Twelve fits, identical network and budget; numbers in `straightness-survey.csv`.

C.1 The Masking Forward Process

Each position is masked independently with probability t:

$$q(x_t \mid x_0) = \prod_{i=1}^{L} q(x_t^i \mid x_0^i), \qquad q(x_t^i \mid x_0^i) = (1-t)\cdot \delta_{x_0^i} + t \cdot \delta_{\texttt{[M]}}.$$



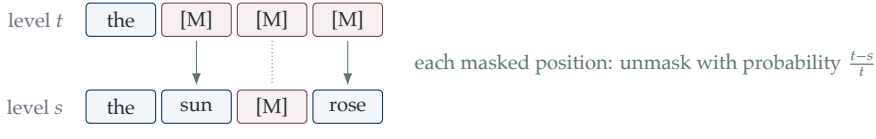
The masked count is Binomial(L, t), and any level is one draw from x_0 — no chain to unroll. At $t = 1$ the state is a single sequence, not a distribution.

C.2 The Reverse Process

derive

Masking is monotone, so an unmasked position never changes. A position masked at t should already be unmasked at $s < t$ with probability

$$\Pr[\text{unmask at } s \mid \text{masked at } t] = \frac{t - s}{t}.$$



A word that is unmasked is drawn from the network's prediction $p_\theta(\cdot \mid x_t)$ at that position.

C.3 The Objective

$$\mathcal{L}(\theta) = \mathbb{E}_{t,x_0,x_t}\left[\frac{1}{t}\sum_{i:\,x_t^i=\texttt{[M]}} -\log p_\theta(x_0^i \mid x_t)\right]$$

- Cross-entropy: the target is a category, not a magnitude;
- The sum runs over masked positions only; $1/t$ corrects for how many there are on average;
- To condition, leave the prompt unmasked and sum over the continuation.

Source: Nie et al. (2025), *Large Language Diffusion Models*, equation (3).

D.1 Further Reading: Diffusion Models

- Song, Meng, Ermon (2021), *Denoising Diffusion Implicit Models* — DDIM.
- Rombach et al. (2022), *High-Resolution Image Synthesis with Latent Diffusion Models* — Stable Diffusion.
- Ho, Salimans (2022), *Classifier-Free Diffusion Guidance*.
- Gu et al. (2022), *Vector Quantized Diffusion Model for Text-to-Image Synthesis* — diffusion on a discrete code.
- Alammar, *The Illustrated Stable Diffusion* — `https://jalammar.github.io/illustrated-stable-diffusion/`
- Hugging Face `diffusers`, latent-diffusion pipelines — `https://huggingface.co/docs/diffusers/en/api/pipelines/latent_diffusion`
- Rogge and Rasul, *The Annotated Diffusion Model* — Lecture 18 in code.

D.2 Further Reading: Flow Matching

- Lipman et al. (2023), *Flow Matching for Generative Modeling*; and Lipman et al. (2024), *Flow Matching Guide and Code*.
- Liu, Gong, Liu (2023), *Flow Straight and Fast* — rectified flow and reflow.
- Helbling, *A Visual Introduction to Rectified Flows* — `https://alechelbling.com/blog/rectified-flow/`
- Liu et al., *Rectified Flow* — why sampling trajectories curve — `https://rectifiedflow.github.io/blog/2024/intro/`

D.3 Further Reading: Diffusion Language Models

- Nie et al. (2025), *Large Language Diffusion Models* — LLaDA.
- Kuleshov and Arriola (2026), *How to Build a Diffusion Language Model* — `https://kuleshov-group.github.io/blog/blog/2026/how-to-build-a-diffusion-language-model/`
- Grootendorst, *A Visual Guide to DiffusionGemma* — `https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-diffusiongemma`