Lecture Script
Flow Matching

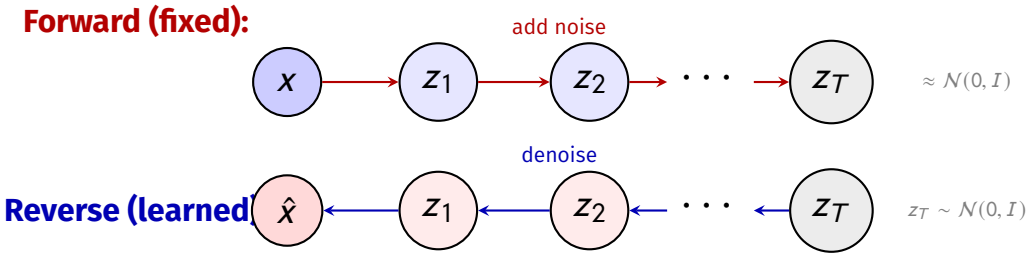
References:

- Lipman et al., *Flow Matching for Generative Modeling* (2022)
- Tong et al., *Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport* (2023)

Outline

1. Summary

Last lecture we looked at diffusion models:



The two key objects:

$$\textbf{Diffusion kernel:} \quad q(z_t \mid x) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\, x,\ (1 - \bar{\alpha}_t)\, I\big)$$

$$\textbf{Reverse conditional:} \quad p(z_{t-1} \mid z_t, x) = \mathcal{N}\big(\tilde{\mu}_t(z_t, x),\ \tilde{\beta}_t\, I\big)$$

The recipe:

1. Mix data and noise: $z_t = \sqrt{\bar{\alpha}_t}\, x + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$
2. Train a network to predict the noise: $\mathcal{L} = \|\epsilon - \hat{\epsilon}_\theta(z_t, t)\|^2$
3. Generate by denoising many steps backward

Today we'll see that there's an even simpler way to do this:

1. Interpolate between noise and data: $z_t = (1 - t)\, x + t\, \epsilon$
2. Train a network to predict the velocity: $\mathcal{L} = \|v_\theta(z_t, t) - v_t\|^2$
3. Generate by integrating the velocity field from noise to data

Same idea—move probability mass from noise to data—but with a straight-line interpolation instead of a noise schedule, and a velocity field instead of a noise predictor.

▹ Let's start from what we know.

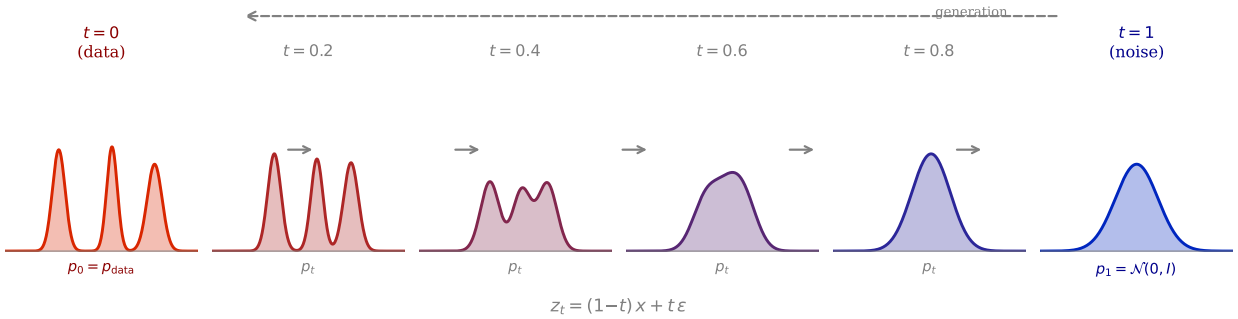
2. From denoising to velocity

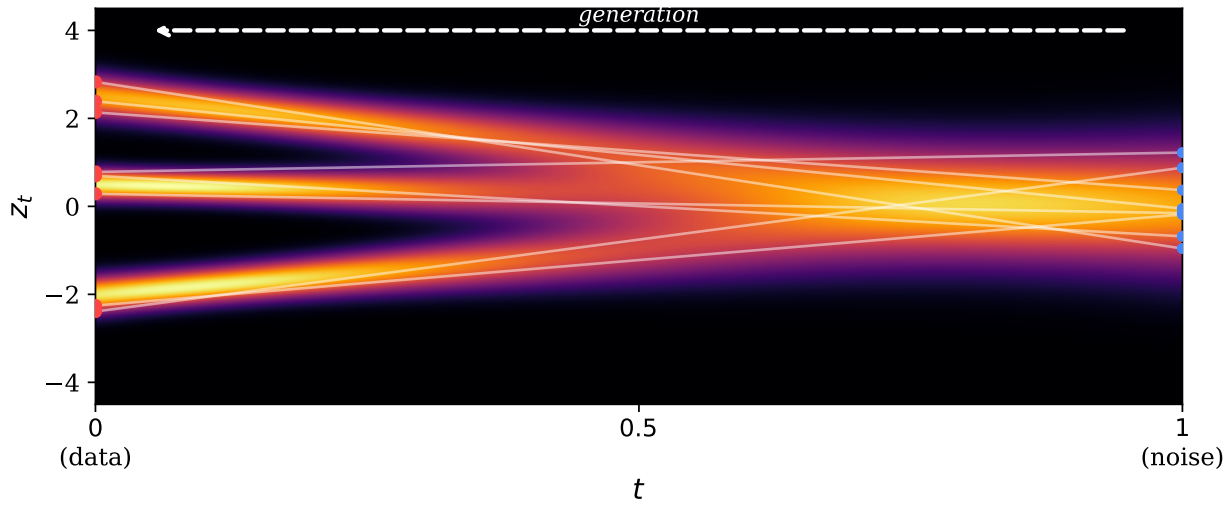
The velocity field

Instead of denoising step by step, think of generation as moving particles. Each particle starts at $z_1 \sim \mathcal{N}(0, I)$ and follows a **velocity field** $v_t(x)$:

$$\frac{dz}{dt} = v_t(z)$$

Start a particle at $z_1 \sim \mathcal{N}(0, I)$ and follow the velocity field from $t = 1$ to $t = 0$. If v_t is the right velocity field, the particle ends up as a sample from p_{data}.





This is **flow matching**: learn a velocity field and integrate from noise to data.

Sampling. In practice, we discretize with step size $\Delta t = 1/N$ and integrate from $t = 1$ (noise) to $t = 0$ (data):

1. Sample $z_1 \sim \mathcal{N}(0, I)$
2. For $t = 1, 1-\Delta t, 1-2\Delta t, \ldots, \Delta t$:

$$z_{t-\Delta t} = z_t + \Delta t \cdot v_\theta(z_t, t)$$

3. Return $z_0 \approx$ sample from p_{data}

This is Euler integration. Same direction as diffusion ($1 \to 0$, i.e. noise to data), but fully deterministic—no noise injection.

▸ OK, so we want to learn a velocity field. How?

3. Conditional flow matching

The marginal velocity is intractable

Ideally, we'd train the network by regression against the true velocity field $u_t(x)$ that transports p_0 to p_1:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,x\sim p_t}\left[\|v_\theta(x,t) - u_t(x)\|^2\right]$$

But what is $u_t(z_t)$? Suppose we knew, for each data point x, a conditional velocity $u_t(z_t \mid x)$ that transports noise toward x, generating a conditional density $p_t(z_t \mid x)$. The marginal density at time t is a mixture:

$$p_t(z_t) = \int p_t(z_t \mid x)\, q(x)\, dx$$

What velocity field generates this mixture? Each component $p_t(\cdot \mid x)$ is transported by its own $u_t(\cdot \mid x)$, weighted by how much it contributes to the density at z_t. The marginal velocity is their weighted average:

$$u_t(z_t) = \int u_t(z_t \mid x) \underbrace{\frac{p_t(z_t \mid x)\, q(x)}{p_t(z_t)}}_{q(x|z_t)} dx = \mathbb{E}_{q(x|z_t)}[u_t(z_t \mid x)]$$

The weights are the posterior $q(x \mid z_t)$—the probability that data point x "produced" this z_t. To compute $u_t(z_t)$, we'd need to evaluate this posterior over all data. Same intractability as in diffusion.

Deriving the weighted average from the continuity equation

The link between a density and the velocity field that transports it is the **continuity equation**—conservation of probability:

$$\partial_t p_t(x) + \nabla \cdot (p_t(x)\, u_t(x)) = 0$$

Probability doesn't appear or disappear; it only flows. (Same equation as mass conservation in fluids.)
Each conditional pair $(p_t(z_t \mid x),\, u_t(z_t \mid x))$ satisfies its own continuity equation:

$$\partial_t p_t(z_t \mid x) = -\nabla \cdot (p_t(z_t \mid x)\, u_t(z_t \mid x))$$

Differentiate the mixture $p_t(z_t) = \int p_t(z_t \mid x)\, q(x)\, dx$ with respect to t, then substitute:

$$\partial_t p_t(z_t) = \int \partial_t p_t(z_t \mid x)\, q(x)\, dx = -\nabla \cdot \int p_t(z_t \mid x)\, u_t(z_t \mid x)\, q(x)\, dx$$

(The ∇ is over z_t and the integral is over x, so they commute.) This has the form $\partial_t p_t = -\nabla \cdot (p_t\, u_t)$, so we can identify:

$$p_t(z_t)\, u_t(z_t) = \int p_t(z_t \mid x)\, u_t(z_t \mid x)\, q(x)\, dx$$

Dividing by $p_t(z_t)$ and applying Bayes' rule gives the weighted average formula. The key insight: it is the *flux* $p_t u_t$ that is linear in the mixture components—not the velocity u_t alone—which is why the conditional density remains in the integrand and produces the posterior weighting.

Conditioning makes it tractable

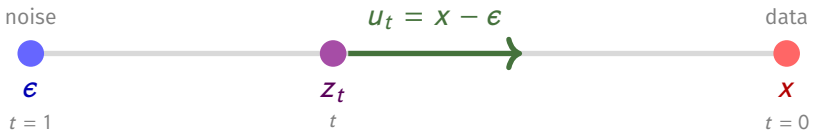
In diffusion, the fix was: condition on the clean data z_0. The reverse step $p(z_{t-1} \mid z_t, z_0)$ was a known Gaussian—the "answer key." We trained a network to approximate it without z_0.

Here the fix is the same: condition on the data point x. Pick a specific data point x and a noise sample $\epsilon \sim \mathcal{N}(0, I)$. Connect them with a **straight line**:

$$z_t = (1 - t)\, x + t\, \epsilon$$

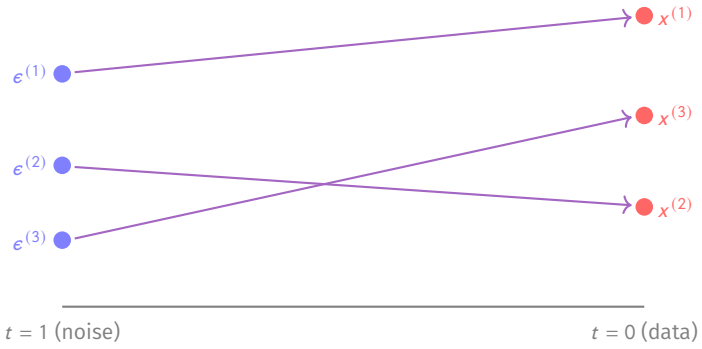
At $t = 0$: clean data x. At $t = 1$: pure noise ϵ. The velocity pointing from

noise toward data is: $u_t(z_t \mid x) = x - \epsilon$.



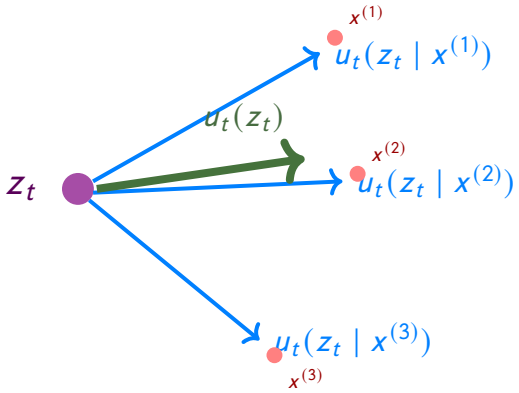
The conditional velocity $u_t(z_t \mid x) = x - \epsilon$ is constant along the path—it doesn't depend on t. This is our "answer key," the flow matching analog of $p(z_{t-1} \mid z_t, z_0)$ from diffusion.

With many pairs, each (x, ϵ) traces its own straight line:



Why conditional velocities are enough

At any point z_t, multiple data points contribute conditional velocities. The network learns their weighted average—the marginal velocity:



Each $x^{(i)}$ pulls z_t toward itself.
Closer data points get higher weight $q(x^{(i)} \mid z_t)$.
The green arrow is their weighted average.

The **conditional flow matching** (CFM) loss is:

$$\mathcal{L}_{\text{CFM}} = \mathbb{E}_{t,x,\epsilon}\left[\| v_\theta(z_t, t) - \underbrace{(x-\epsilon)}_{\text{conditional velocity}} \|^2\right]$$

This has the **same gradients** w.r.t. θ as the intractable marginal loss $\mathcal{L}_{\text{FM}}$. Because MSE regression converges to the conditional mean of the targets. At a point z_t, the network sees different velocity targets from different pairs—but their conditional mean is exactly the marginal velocity:

$$\mathbb{E}_{q(x|z_t)}[\, u_t(z_t \mid x)\,] = \mathbb{E}_{q(x|z_t)}[\, x - \epsilon\,] = u_t(z_t)$$

The first equality substitutes our straight-line conditional velocity $u_t(z_t \mid x) = x - \epsilon$. The second is the definition of the marginal velocity from earlier. The network never computes the posterior $q(x \mid z_t)$—regression does the averaging implicitly.

	Diffusion	**Flow Matching**
Marginal (intractable)	$p(z_{t-1} \mid z_t)$	$u_t(z_t)$
Conditional (known)	$p(z_{t-1} \mid z_t, z_0)$	$u_t(z_t \mid x) = x - \epsilon$

Condition on what you're trying to generate (z_0 or x), get a trivial target, and regression handles the marginalization.

The training algorithm

For each mini-batch:

1. Sample data $x \sim p_{\text{data}}$
2. Sample noise $\epsilon \sim \mathcal{N}(0, I)$
3. Sample time $t \sim \text{Uniform}(0, 1)$
4. Interpolate: $z_t = (1 - t)\, x + t\, \epsilon$
5. Compute target velocity: $u = x - \epsilon$
6. Loss: $\|v_\theta(z_t, t) - u\|^2$

That's it! The network v_θ has the same architecture as a diffusion noise predictor (e.g., a U-Net or transformer)—it takes (z_t, t) and outputs a vector the same size as z_t. Only the training target changes.

▹ Let's put the two methods side by side.

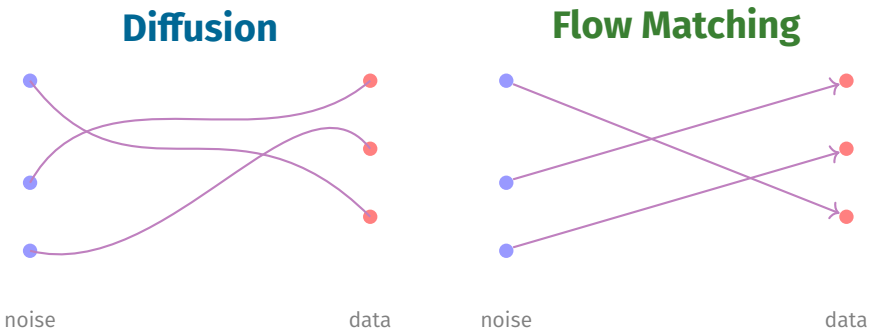
4. Diffusion vs. flow matching

	Diffusion	**Flow Matching**
Interpolation	$z_t = \sqrt{\bar{\alpha}_t}\, x + \sqrt{1-\bar{\alpha}_t}\, \epsilon$	$z_t = (1-t)\, x + t\, \epsilon$
Network predicts	noise $\hat{\epsilon}_\theta(z_t, t)$	velocity $v_\theta(z_t, t)$
Training target	ϵ	$x - \epsilon$
Loss	$\|\epsilon - \hat{\epsilon}_\theta\|^2$	$\|v_\theta - (x-\epsilon)\|^2$
Generation	$T \to 0$, stochastic	$1 \to 0$, deterministic ODE
Schedule	β_t (needs tuning)	none

Training cost is identical, but flow matching typically needs far fewer steps for generation.

Why fewer steps?

Diffusion's $\sqrt{\bar{\alpha}_t}$ schedule creates **curved** paths—the signal and noise evolve on different schedules, and undoing this requires many small corrections. Flow matching's linear interpolation gives **straight** paths. The Euler method $z_{t-\Delta t} = z_t + \Delta t \cdot v_\theta$ is exact for constant velocity, so straighter paths need fewer discretization steps.



The connection

The two methods are closely related. The diffusion forward process $z_t = \sqrt{\bar{\alpha}_t}\, x + \sqrt{1-\bar{\alpha}_t}\, \epsilon$ is itself a path from data to noise—just a curved one. Predicting noise $\hat{\epsilon}$ and predicting velocity v are reparameterizations of each other. For any diffusion schedule there's an equivalent flow matching formulation. The linear interpolation $z_t = (1-t)x + t\epsilon$ is the simplest choice, and gives the straightest paths.

▹ The basics are done. Now some practical tricks and design choices.

5. Practical tricks

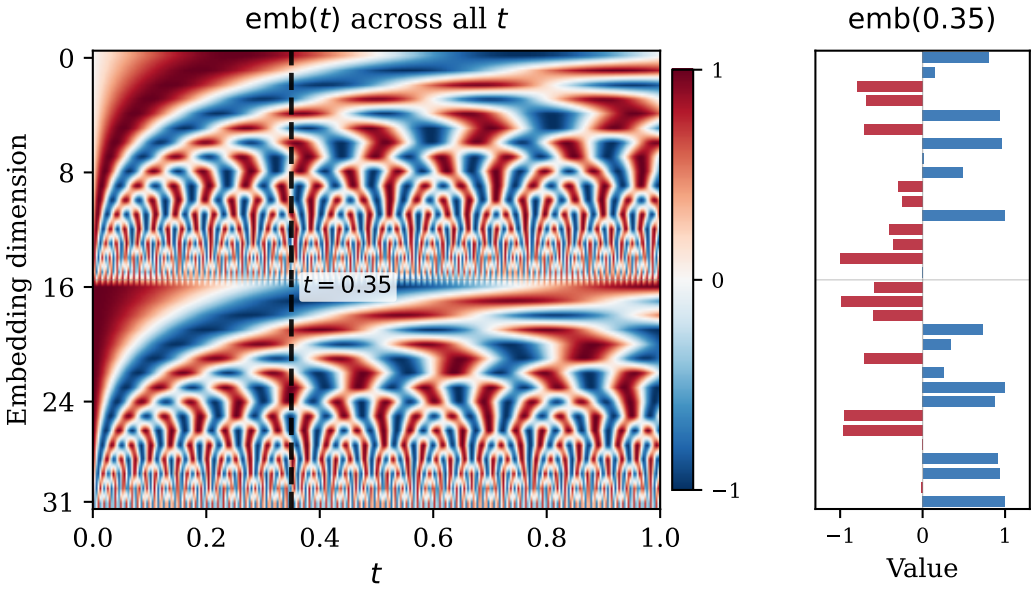
A few tricks can make a large difference in practice. We'll implement each of these in the notebook.

Better architectures. Goes without saying – a better architecture (given your problem/data) will lead to better results!

Time conditioning. The network needs to know t—the same input z_t at different times requires different velocities. A scalar $t \in [0, 1]$ is a poor input: neural networks struggle with single numbers. Instead, project t into a high-dimensional vector using **sinusoidal embeddings** (same idea as positional encodings in transformers):

$$\text{emb}(t) = \Big[\sin(\omega_1 t),\ \cos(\omega_1 t),\ \sin(\omega_2 t),\ \cos(\omega_2 t),\ \ldots\Big]$$

with frequencies ω_k spanning several orders of magnitude. This gives the network a rich, smooth representation of time that it can easily distinguish across different t values.



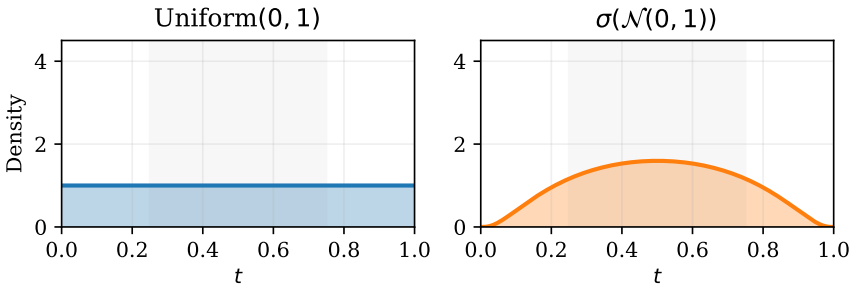
The embedding is then passed through an MLP and used to modulate intermediate features in the network (scale and shift), so the network's behavior smoothly adapts to the noise level.

EMA (exponential moving average). During training, each gradient step updates the weights based on a single mini-batch—noisy by construction. At any given training step, the current weights may be in a temporarily bad region of parameter space. Maintaining a running average smooths this out:

$$\theta_{\text{ema}} \leftarrow \beta\,\theta_{\text{ema}} + (1-\beta)\,\theta, \qquad \beta \approx 0.999$$

Use the EMA weights at inference instead of the raw training weights. The effect is visible: EMA samples are consistently sharper and more coherent.

Logit-normal time sampling. Instead of $t \sim \text{Uniform}(0,1)$, sample $t = \sigma(z)$ where $z \sim \mathcal{N}(0,1)$ and σ is the sigmoid. This concentrates samples near $t = 0.5$. Near $t = 0$ or $t = 1$ the prediction is nearly trivial (output the data, or output the noise). The hard work is in the middle, so spend more training budget there.



σ_{min}. At $t = 0$, the interpolation gives $z_0 = x$ exactly—zero noise, which can cause numerical issues. Adding $\sigma_{\text{min}} \approx 10^{-4}$ keeps a trace of noise everywhere:

$$z_t = (1 - t)\, x + (t + \sigma_{\text{min}})\, \epsilon$$

▹ One more design choice hidden in the training algorithm.

6. The data-noise coupling

Look again at the training algorithm. Each step samples a pair (x, ϵ) and draws a straight line between them. So far we've sampled $x \sim p_{\text{data}}$ and $\epsilon \sim \mathcal{N}(0, I)$ **independently**. But nothing requires that. The CFM loss is valid for any joint distribution $\pi(x, \epsilon)$ whose marginals are p_{data} and $\mathcal{N}(0, I)$.

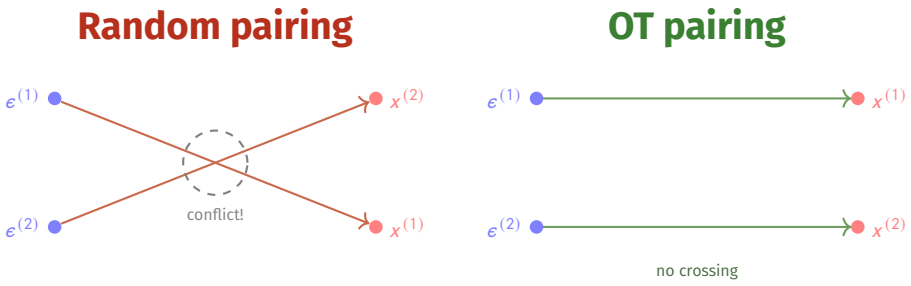
This joint $\pi(x, \epsilon)$ is called a **coupling**—it specifies how noise and data are paired. Two design choices:

1. **How to pair** (the coupling π): independent is simplest, but we can do better.
2. **What to start from** (the base p_{noise}): Gaussian is standard, but p_{noise} can be anything we can sample from.

Both affect path geometry, and therefore how easy the velocity field is to learn.

Better pairing: mini-batch optimal transport

With independent pairing, conditional paths can **cross**: a noise sample at the top pairs with data at the bottom, while a nearby noise sample goes to the top. At the crossing, the network sees conflicting targets.



The network averages these, producing a sideways velocity—curved marginal paths, higher gradient variance, slower convergence.

Fix: mini-batch OT. Within each mini-batch, pair noise and data to minimize total squared distance:

$$\pi^* = \arg\min_{\pi} \sum_{i=1}^{B} \|x^{(i)} - \epsilon^{(\pi(i))}\|^2$$

where π is a permutation of the mini-batch indices and the cost is squared Euclidean distance $\|x^{(i)} - \epsilon^{(j)}\|^2$. This is a **linear assignment problem**: given a cost matrix of all pairwise distances, find the one-to-one matching that minimizes total cost. Standard solvers (e.g. the Hungarian algorithm) handle this efficiently per mini-batch.

Better starting point: informed base distributions

In diffusion, $p_1 = \mathcal{N}(0, I)$ is baked into the forward process. In flow matching, p_{noise} is free. If it already captures some structure of the data—spatial correlations, symmetries, known marginals—paths are shorter and the velocity field has less to learn.

This is especially useful in scientific applications where data has known statistical properties. We'll see concrete examples in the notebook.

7. Conditional generation and guidance

To generate x conditioned on some label c (class, text prompt, molecular property, …):

Training. Feed c to the velocity network: $v_\theta(z_t, t, c)$. Randomly drop $c \to \varnothing$ with probability 10–20% so the network also learns the unconditional velocity.

Sampling. Extrapolate between unconditional and conditional predictions (**guidance**):

$$\tilde{v} = v_\theta(z_t, t, \varnothing) + w \cdot \big(v_\theta(z_t, t, c) - v_\theta(z_t, t, \varnothing)\big)$$

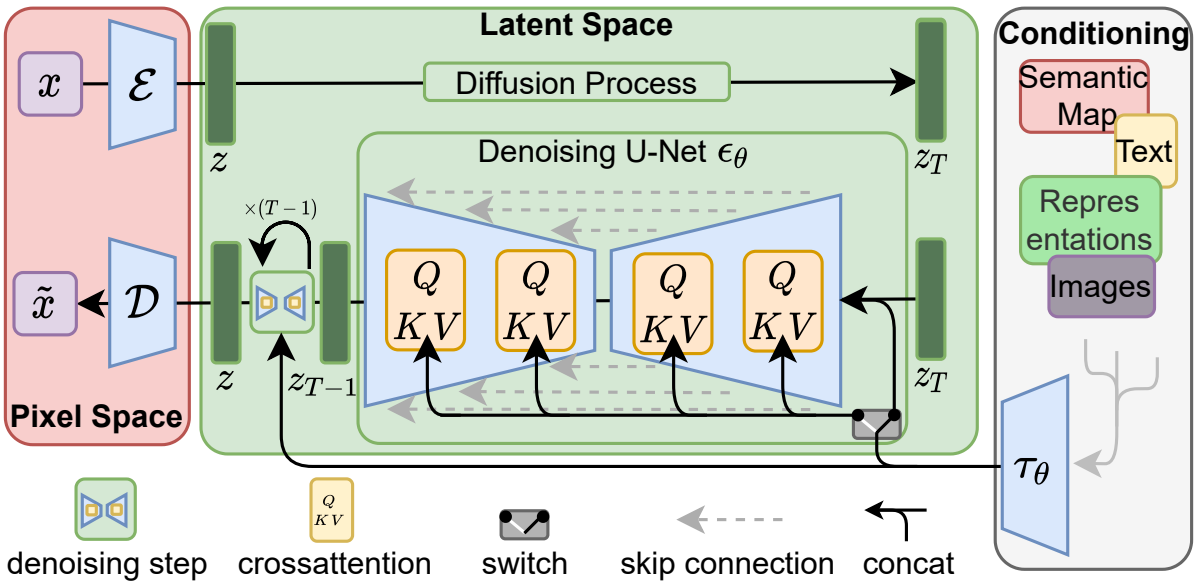
$w = 1$: standard conditional generation. $w > 1$: sharper samples, stronger adherence to c, less diversity.

The whole recipe, one more time

Train: Sample $x \sim$ data, $\epsilon \sim \mathcal{N}(0, I)$, $t \sim \text{Uniform}(0, 1)$. Set $z_t = (1-t)x + t\epsilon$. Minimize $\|v_\theta(z_t, t) - (x - \epsilon)\|^2$.
Generate: Start at $z_1 \sim \mathcal{N}(0, I)$. Step backward: $z_{t-\Delta t} = z_t + \Delta t \cdot v_\theta(z_t, t)$. Return z_0.

8. Latent diffusion

Running flow matching directly in data space becomes expensive as dimensionality grows. **Latent flow matching**: compress the data to a lower-dimensional latent space with an autoencoder, then run flow matching there.



Latent Diffusion Models (Rombach et al., 2022): the flow/diffusion model ϵ_θ operates in the compressed latent space z. Conditioning is injected via cross-attention.