

Flow Matching: A Step-by-Step Mathematical Derivation

Understanding velocity fields, continuity equations, conditional flow matching, and robotic action generation

October 2026

Contents

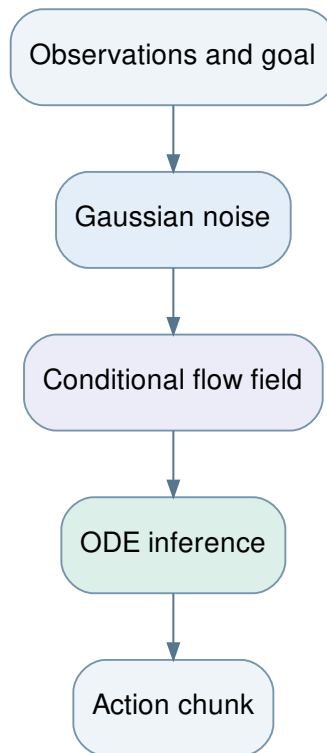
Step 1 – Why do we need Flow Matching?	2
Step 2 – Why introduce an ODE?	3
Step 3 – Why do we need the continuity equation?	4
Step 4 – What is the ideal Flow Matching objective?	6
Step 5 – How is Conditional Flow Matching constructed?	6
Step 6 – Why does conditional velocity regression recover the global field?	7
Step 7 – Why does MSE regression learn the conditional expectation?	9
Step 8 – Why introduce more general Gaussian conditional paths?	11
Step 9 – How are Flow Matching models trained and sampled?	12
Step 10 – Why can Flow Matching generate multimodal distributions?	14
Step 11 – How does $\pi_{0.5}$ use Flow Matching?	15
Step 12 – How can Flow Matching be fine-tuned with reinforcement learning?	17
Step 13 – Synthesis: the complete mathematical logic of Flow Matching	19

Course home · 简体中文 · PDF

Item	Details
Applicability	CFM theory; OpenPI pi0.py code reference pinned below; examples are not a tested fine-tuning run
Documentation review date	2026-10-18
License	CC BY-NC-SA 4.0 · Xiong Zeng and contributors

Step 1 — Why do we need Flow Matching?

Chapter concept map



1.1 What problem does a generative model solve?

Assume the training data follow an unknown distribution:

$$x \sim p_{\text{data}}(x).$$

In image generation, x represents an image; in video generation, it represents a video; in robot control, it may represent an **action chunk**.

Our goal is to learn a generative model such that

$$x_{\text{generated}} \sim p_{\text{data}}(x).$$

Real-world data distributions are often complicated. For example, a robot may grasp a cup from either the left or the right. The resulting action distribution can be multimodal:

$$p_{\text{data}}(x) = \frac{1}{2}p_{\text{left}}(x) + \frac{1}{2}p_{\text{right}}(x).$$

Suppose we instead train a deterministic network $f_{\theta}(o) \rightarrow x$ using squared error:

$$\mathcal{L}(\theta) = \mathbb{E} [\|f_\theta(o) - x\|^2].$$

Its population-optimal prediction is the conditional mean:

$$f_\theta^*(o) = \mathbb{E}[x \mid o].$$

That mean may correspond to neither a valid left grasp nor a valid right grasp.

Motivation. We want to learn the **full data distribution**, rather than output only its mean.

1.2 Start from an easily sampled Gaussian distribution

Sampling from a standard Gaussian is easy:

$$x_0 \sim p_0(x) = \mathcal{N}(0, I).$$

Instead of predicting a real sample directly, imagine learning a transformation

$$F_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d, \quad x_1 = F_\theta(x_0),$$

that satisfies

$$(F_\theta)_\# p_0 = p_{\text{data}}.$$

The symbol $\#$ denotes the **pushforward distribution**. Intuitively, if we transform many Gaussian samples through F_θ , the distribution of their outputs should match the data distribution:

$$\underbrace{\mathcal{N}(0, I)}_{\text{simple Gaussian}} \xrightarrow{F_\theta} \underbrace{p_{\text{data}}}_{\text{complex multimodal distribution}}.$$

The question now becomes: **How can we learn this distribution transformation?**

Step 2 – Why introduce an ODE?

2.1 Decompose a complicated transformation into small continuous steps

Learning $x_1 = F_\theta(x_0)$ in a single step may be difficult. Instead, introduce a continuous **generative time** variable:

$$t \in [0, 1].$$

By convention, $t = 0$ corresponds to the Gaussian distribution, and $t = 1$ corresponds to the data distribution:

$$t = 0 : x_0 \sim p_0, \quad t = 1 : x_1 \sim p_{\text{data}}.$$

Between these endpoints, a sample moves along a continuous trajectory:

$$x_0 \longrightarrow x_t \longrightarrow x_1.$$

Define a time-dependent **velocity field**

$$v_\theta(x, t) : \mathbb{R}^d \times [0, 1] \rightarrow \mathbb{R}^d$$

and the ordinary differential equation (ODE)

$$\boxed{\frac{dx_t}{dt} = v_\theta(x_t, t)}. \quad (1)$$

With $x_0 \sim \mathcal{N}(0, I)$ as the initial condition, integration gives

$$x_1 = x_0 + \int_0^1 v_\theta(x_t, t) dt.$$

Motivation. Rather than predicting a final sample in one shot, the network answers a local question: *Given the current location x_t and time t , in which direction should the sample move?* A global generation task becomes local velocity prediction.

This construction is an instance of a **Continuous Normalizing Flow (CNF)**.

2.2 A control-theoretic interpretation

From a dynamical-systems perspective, Flow Matching learns the nonautonomous system

$$\dot{x} = v_\theta(x, t).$$

Its goal, however, is not to drive a single state toward an equilibrium. It aims to evolve an **entire probability distribution** along a desired path:

$$p_0 \longrightarrow p_t \longrightarrow p_1.$$

A conventional control problem often focuses on $x(0) \rightarrow x_{\text{target}}$; Flow Matching instead focuses on

$$\boxed{p_0(x) \rightarrow p_{\text{target}}(x)}.$$

We do not have to specify in advance which particular noise sample should correspond to which particular training example. Matching the generated **distribution** is the central requirement.

Step 3 – Why do we need the continuity equation?

3.1 Begin with moving particles

Consider many particles $x_t^{(1)}, \dots, x_t^{(N)}$, each satisfying

$$\frac{dx_t^{(i)}}{dt} = v_t(x_t^{(i)}).$$

Together, they induce a probability density $p_t(x)$. When particles flow into a region, its density increases; when they leave, its density decreases.

Thus, we need a precise connection between the **velocity field** and the **time evolution of probability density**.

3.2 Derivation of the continuity equation

Take an arbitrary smooth test function $\phi : \mathbb{R}^d \rightarrow \mathbb{R}$. Consider

$$\mathbb{E}[\phi(x_t)] = \int \phi(x) p_t(x) dx.$$

Differentiate with respect to time, assuming sufficient regularity to interchange differentiation and integration:

$$\frac{d}{dt} \mathbb{E}[\phi(x_t)] = \int \phi(x) \frac{\partial p_t(x)}{\partial t} dx.$$

On the other hand, the chain rule gives

$$\frac{d}{dt} \phi(x_t) = \nabla \phi(x_t)^\top \frac{dx_t}{dt} = \nabla \phi(x_t)^\top v_t(x_t).$$

Consequently,

$$\frac{d}{dt} \mathbb{E}[\phi(x_t)] = \int \nabla \phi(x)^\top v_t(x) p_t(x) dx.$$

Assuming the boundary term vanishes, integrate by parts:

$$= - \int \phi(x) \nabla \cdot [p_t(x) v_t(x)] dx.$$

Equating the two expressions yields

$$\int \phi(x) \left[\frac{\partial p_t(x)}{\partial t} + \nabla \cdot (p_t(x) v_t(x)) \right] dx = 0.$$

Because this holds for every suitable ϕ , we obtain

$$\boxed{\frac{\partial p_t(x)}{\partial t} + \nabla \cdot [p_t(x) v_t(x)] = 0.} \quad (2)$$

This is the **continuity equation**.

3.3 Physical meaning of the continuity equation

Expand the divergence term:

$$\frac{\partial p_t}{\partial t} + v_t^\top \nabla p_t + p_t \nabla \cdot v_t = 0.$$

The term $v_t^\top \nabla p_t$ describes advection of the density, while $\nabla \cdot v_t$ measures the local divergence of the velocity field.

If $\nabla \cdot v_t > 0$, nearby particles tend to spread out, and density tends to decrease. If $\nabla \cdot v_t < 0$, particles tend to converge, and density tends to increase.

Along an ODE trajectory, we also have

$$\boxed{\frac{d}{dt} \log p_t(x_t) = -\nabla \cdot v_t(x_t).} \quad (3)$$

This identity is the basis for evaluating sample log-densities in CNFs.

Motivation. An ODE describes the motion of an individual particle, whereas the continuity equation describes how the **distribution of particles** changes. Together they link generative dynamics to the target probability distribution.

Step 4 – What is the ideal Flow Matching objective?

4.1 Suppose the correct velocity field were known

Assume we choose a probability path $p_t(x)$ in advance, satisfying

$$p_{t=0} = p_0, \quad p_{t=1} = p_{\text{data}},$$

and a velocity field $u_t(x)$ exists such that

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

We could then regress directly onto this field:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{\substack{t \sim \mathcal{U}[0,1] \\ x \sim p_t}} [\|v_\theta(x, t) - u_t(x)\|^2]. \quad (4)$$

If $v_\theta = u_t$ is learned exactly, the induced ODE realizes the selected distribution evolution.

4.2 Why cannot we train with this ideal objective directly?

In practice, we generally observe only

$$x_1 \sim p_{\text{data}}.$$

We do not observe the intermediate process by which Gaussian samples evolve into data samples. Therefore, neither the intermediate $p_t(x)$ nor the required $u_t(x)$ is usually known.

Central motivation. Can we construct a path that is easy to sample **and** whose velocity is directly computable, thereby obtaining a supervised learning signal for the network?

That question leads to **Conditional Flow Matching (CFM)**.

Step 5 – How is Conditional Flow Matching constructed?

5.1 Construct two endpoints

Sample independently from the Gaussian source and the data distribution:

$$x_0 \sim p_0 = \mathcal{N}(0, I), \quad x_1 \sim p_{\text{data}}.$$

The simplest pairing is independent:

$$x_0 \perp x_1.$$

Given an endpoint pair (x_0, x_1) , prescribe constant-speed linear motion:

$$x_t = (1 - t)x_0 + tx_1. \quad (5)$$

The boundary conditions are immediate: $x_t = x_0$ at $t = 0$, and $x_t = x_1$ at $t = 1$.

For example, let

$$x_0 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad x_1 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}.$$

Differentiating (5),

$$\boxed{\frac{dx_t}{dt} = x_1 - x_0.} \quad (6)$$

The target velocity in this example is $[4, 2]^\top$. It is independent of t because we deliberately chose constant-speed linear interpolation.

Why a straight line? For fixed endpoints, it is extremely easy to compute, has an explicit velocity label, and requires neither an auxiliary model nor simulation of an intermediate trajectory.

5.2 Obtain a directly trainable loss

During training, sample

$$x_0 \sim \mathcal{N}(0, I), \quad x_1 \sim p_{\text{data}}, \quad t \sim \mathcal{U}[0, 1].$$

Construct

$$x_t = (1 - t)x_0 + tx_1, \quad u_t = x_1 - x_0.$$

Train the neural network to predict the velocity:

$$v_\theta(x_t, t) \approx x_1 - x_0.$$

The resulting loss is

$$\boxed{\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t, x_0, x_1} [\|v_\theta((1 - t)x_0 + tx_1, t) - (x_1 - x_0)\|^2]}. \quad (7)$$

This is the familiar **straight-line Conditional Flow Matching objective**.

One training update requires just a data sample, Gaussian noise, a randomly chosen time, an interpolation, a target velocity, one network forward pass, and a mean-squared-error (MSE) loss.

Major advantage. There is no need to solve the generative ODE during training. This is commonly called **simulation-free training**.

A deeper question remains: the network receives only (x_t, t) , not the original endpoints (x_0, x_1) . Why does regressing onto $x_1 - x_0$ recover an appropriate global velocity field?

Step 6 — Why does conditional velocity regression recover the global field?

6.1 Individual-path velocity versus marginal velocity

For fixed endpoints (x_0, x_1) , the velocity of that individual path is

$$u_t(x \mid x_0, x_1) = x_1 - x_0.$$

It depends on the chosen endpoints, and we call it a **conditional velocity**.

At inference time, however, the final data endpoint is unknown. The network must predict a field that depends only on the current location and time:

$$v_t(x).$$

Many different straight paths can pass through the same (x, t) , potentially with different velocities. What should the network output? The answer is the **conditional expectation** of those path velocities.

6.2 A rigorous derivation of the marginal velocity field

Use capital letters for random variables. Define

$$Z = (X_0, X_1), \quad X_t = (1 - t)X_0 + tX_1,$$

and

$$U_t = X_1 - X_0.$$

Since $dX_t/dt = U_t$, for any smooth test function ϕ ,

$$\frac{d}{dt} \mathbb{E}[\phi(X_t)] = \mathbb{E}[\nabla \phi(X_t)^\top U_t].$$

Apply the tower property of conditional expectation:

$$= \mathbb{E}[\nabla \phi(X_t)^\top \mathbb{E}[U_t \mid X_t]].$$

Define

$$\boxed{v_t^*(x) = \mathbb{E}[U_t \mid X_t = x]}. \quad (8)$$

Then

$$\frac{d}{dt} \mathbb{E}[\phi(X_t)] = \int \nabla \phi(x)^\top v_t^*(x) p_t(x) dx.$$

Integration by parts gives

$$= - \int \phi(x) \nabla \cdot [p_t(x) v_t^*(x)] dx.$$

But we also have

$$\frac{d}{dt} \mathbb{E}[\phi(X_t)] = \int \phi(x) \partial_t p_t(x) dx.$$

Hence,

$$\partial_t p_t(x) + \nabla \cdot [p_t(x) v_t^*(x)] = 0. \quad (9)$$

In other words, the conditional expectation defines a velocity field that realizes the target **marginal probability path**:

$$v_t^*(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x]. \quad (10)$$

This is one of the central mathematical statements behind CFM.

6.3 Equivalent expression using probability densities

More generally, let z denote a conditioning variable, $p_t(x \mid z)$ a conditional probability path, and $u_t(x \mid z)$ its conditional velocity. The marginal distribution is

$$p_t(x) = \int p_t(x \mid z) q(z) dz.$$

We can define

$$v_t^*(x) = \frac{\int u_t(x \mid z) p_t(x \mid z) q(z) dz}{p_t(x)}. \quad (11)$$

Equivalently,

$$v_t^*(x) = \mathbb{E}[u_t(x \mid Z) \mid X_t = x].$$

For deterministic interpolation between fixed endpoints, the conditional path is a Dirac distribution concentrated at the interpolated point. The random-variable argument in the previous subsection is therefore more convenient.

Important subtlety. The continuity equation can admit more than one valid velocity field. Flow Matching does not discover a unique, path-independent “true velocity field.” We choose a probability path and learn a field that realizes it.

Step 7 – Why does MSE regression learn the conditional expectation?

7.1 Orthogonal decomposition of least squares

For a fixed t , define

$$m_t(X_t) = \mathbb{E}[U_t \mid X_t].$$

Consider the training loss

$$\mathcal{L}(\theta) = \mathbb{E}[\|v_\theta(X_t, t) - U_t\|^2].$$

Add and subtract m_t inside the difference:

$$v_\theta - U_t = (v_\theta - m_t) + (m_t - U_t).$$

Expand the squared norm:

$$\begin{aligned}\|v_\theta - U_t\|^2 &= \|v_\theta - m_t\|^2 + \|m_t - U_t\|^2 \\ &\quad + 2(v_\theta - m_t)^\top (m_t - U_t).\end{aligned}$$

By the defining property of conditional expectation,

$$\mathbb{E}[m_t - U_t \mid X_t] = 0.$$

Furthermore, $v_\theta(X_t, t) - m_t(X_t)$ is a function of X_t . Thus the expected cross term vanishes:

$$\mathbb{E}[(v_\theta - m_t)^\top (m_t - U_t)] = 0.$$

Therefore,

$$\boxed{\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}[\|v_\theta(X_t, t) - m_t(X_t)\|^2] + \mathbb{E}[\|U_t - m_t(X_t)\|^2].} \quad (12)$$

The second term does not depend on θ . Therefore, under the same sampling distribution and squared-error definition,

$$\boxed{\arg \min_{\theta} \mathcal{L}_{\text{CFM}}(\theta) = \arg \min_{\theta} \mathcal{L}_{\text{FM}}(\theta).} \quad (13)$$

Provided differentiation and expectation can be interchanged, we also have

$$\boxed{\nabla_{\theta} \mathcal{L}_{\text{CFM}} = \nabla_{\theta} \mathcal{L}_{\text{FM}}.} \quad (14)$$

7.2 Why is this such an important result?

What we want to learn is

$$v_t^*(x) = \mathbb{E}[U_t \mid X_t = x],$$

but computing this conditional expectation explicitly is difficult.

CFM avoids calculating it. Instead, it **regresses directly onto sampled values of U_t** . By the least-squares projection principle, the optimal predictor is automatically the conditional expectation:

$$\underbrace{v_t^*(x)}_{\text{hard-to-compute marginal velocity}} \quad \Longleftarrow \quad \underbrace{X_1 - X_0}_{\text{sampled conditional velocity}}.$$

This is the fundamental reason Conditional Flow Matching can be trained efficiently.

7.3 Why can generated trajectories curve if training paths are straight?

For each endpoint pair, the training path is indeed a straight line:

$$X_t = (1 - t)X_0 + tX_1.$$

But the network ultimately learns the conditional average

$$v_t^*(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x].$$

For example, suppose two paths cross at the same location with velocities

$$u_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad u_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

If both paths have equal conditional weight at that location,

$$v_t^*(x) = \frac{1}{2}u_1 + \frac{1}{2}u_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

This velocity equals neither individual path velocity. Therefore, the trajectory obtained by solving $\dot{x}_t = v_t^*(x_t)$ can be curved.

Key insight. Straight-line training interpolations do not imply straight-line generative ODE trajectories.

Step 8 – Why introduce more general Gaussian conditional paths?

Straight-line interpolation is only one special case of Flow Matching. The broader framework allows more general conditional probability paths.

8.1 A general Gaussian conditional path

For a given data sample x_1 , define

$$p_t(x \mid x_1) = \mathcal{N}(x; \mu_t(x_1), \sigma_t^2 I). \quad (15)$$

Here $\mu_t(x_1)$ specifies how the center of the distribution moves, while σ_t determines how the distribution expands or contracts.

We can sample from this distribution using

$$X_t = \mu_t(x_1) + \sigma_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \quad (16)$$

Differentiate with respect to t :

$$\frac{dX_t}{dt} = \dot{\mu}_t(x_1) + \dot{\sigma}_t \epsilon.$$

Equation (16) implies

$$\epsilon = \frac{X_t - \mu_t(x_1)}{\sigma_t}.$$

Substituting back gives the **conditional velocity field**

$$u_t(x \mid x_1) = \dot{\mu}_t(x_1) + \frac{\dot{\sigma}_t}{\sigma_t} [x - \mu_t(x_1)]. \quad (17)$$

This derivation assumes $\sigma_t > 0$ at the relevant time. If the endpoint variance approaches zero, the endpoint is understood

by taking a limit.

Motivation. Once we choose μ_t and σ_t , we can compute the conditional velocity explicitly and train with the same MSE framework. Straight-line interpolation is not the only available path.

8.2 The simplest linear Gaussian path

Choose

$$\mu_t(x_1) = tx_1, \quad \sigma_t = 1 - (1 - \sigma_{\min})t,$$

where $0 \leq \sigma_{\min} \ll 1$. Then

$$\boxed{X_t = tx_1 + [1 - (1 - \sigma_{\min})t]\epsilon.} \quad (18)$$

At $t = 0$, $X_0 = \epsilon \sim \mathcal{N}(0, I)$.

At $t = 1$, $X_1 = x_1 + \sigma_{\min}\epsilon$, so

$$X_1 \mid x_1 \sim \mathcal{N}(x_1, \sigma_{\min}^2 I).$$

When $\sigma_{\min} = 0$, the endpoint is exactly the data sample. When $\sigma_{\min} > 0$, the final marginal distribution is the data distribution convolved with a small Gaussian, not exactly the original data distribution.

8.3 Derive the velocity field

Differentiate (18):

$$\frac{dX_t}{dt} = x_1 - (1 - \sigma_{\min})\epsilon.$$

The target velocity is therefore

$$\boxed{u_t = x_1 - (1 - \sigma_{\min})\epsilon.} \quad (19)$$

Setting $\sigma_{\min} = 0$ gives

$$X_t = (1 - t)\epsilon + tx_1,$$

and

$$\boxed{u_t = x_1 - \epsilon.} \quad (20)$$

This recovers the earlier straight-line Flow Matching objective.

Takeaway. The familiar target $x_1 - \epsilon$ is a special case of Gaussian Conditional Flow Matching. Different probability paths change the supervised target, the learning problem, and the behavior of numerical integration.

Step 9 — How are Flow Matching models trained and sampled?

9.1 Training procedure

Given a data sample

$$x_1 \sim p_{\text{data}},$$

perform the following operations:

1. Sample Gaussian noise: $x_0 \sim \mathcal{N}(0, I)$.
2. Sample a random time: $t \sim \mathcal{U}(0, 1)$.
3. Construct the interpolated sample: $x_t = (1 - t)x_0 + tx_1$.
4. Construct the target velocity: $u_t = x_1 - x_0$.
5. Predict the velocity: $\hat{u}_t = v_\theta(x_t, t)$.
6. Optimize the parameters using $\mathcal{L} = \|\hat{u}_t - u_t\|^2$.

No numerical integration of the generative ODE is required during training.

A minimal PyTorch implementation:

```
import torch

def flow_matching_loss(model, x1, cond=None):
    # x1 shape: [B, D] or [B, H, D]
    B = x1.shape[0]

    # 1. Gaussian noise
    x0 = torch.randn_like(x1)

    # 2. Random flow timestep
    t = torch.rand(B, device=x1.device)
    t_view = t.view(B, *[1] * (x1.ndim - 1))

    # 3. Linear interpolation
    xt = (1 - t_view) * x0 + t_view * x1

    # 4. Target velocity
    target_v = x1 - x0

    # 5. Predict velocity
    pred_v = model(xt, t, cond)

    # 6. Mean squared error
    loss = ((pred_v - target_v) ** 2).mean()
    return loss
```

9.2 Sampling (inference) procedure

At inference time, a real sample x_1 is no longer needed. Start by sampling

$$x_0 \sim \mathcal{N}(0, I),$$

then solve

$$\frac{dx_t}{dt} = v_\theta(x_t, t).$$

The simplest numerical integrator is **forward Euler**. Divide $[0, 1]$ into K steps with $\Delta t = 1/K$:

$$x_{k+1} = x_k + \Delta t v_{\theta}(x_k, k\Delta t). \quad (21)$$

The final state x_K approximates the desired generated sample x_1 .

PyTorch pseudocode:

```
@torch.no_grad()
def sample_flow(model, shape, cond=None, K=10, device="cuda"):
    x = torch.randn(shape, device=device)
    dt = 1.0 / K

    for k in range(K):
        t = torch.full((shape[0],), k * dt, device=device)
        v = model(x, t, cond)
        x = x + dt * v

    return x
```

Other options include Heun, midpoint, Runge–Kutta, and adaptive ODE solvers. Higher-order methods can reduce numerical error, but they often require more neural-network evaluations per step. The design trade-off is between accuracy and the **number of function evaluations (NFE)**.

Step 10 — Why can Flow Matching generate multimodal distributions?

Suppose an observation o permits two different grasps:

$$p(A \mid o) = \frac{1}{2}p_{\text{left}}(A \mid o) + \frac{1}{2}p_{\text{right}}(A \mid o).$$

A conventional MSE policy $f_{\theta}(o)$ sees only the observation. Its optimal prediction is the conditional mean $\mathbb{E}[A \mid o]$, potentially falling between the two valid actions.

In contrast, a Flow Matching policy has the form

$$v_{\theta}(x_t, t, o),$$

with a random initial sample

$$x_0 \sim \mathcal{N}(0, I).$$

Different initial noise samples can be transported to different action modes.

Even if the ODE output is deterministic once x_0 and o are fixed, X_0 remains random. Therefore

$$X_1 = \Phi_1(X_0; o)$$

is also a random variable, with conditional distribution

$$p_{\theta}(A \mid o) = (\Phi_1(\cdot; o))_{\#} \mathcal{N}(0, I). \quad (22)$$

Key distinction. Flow Matching also uses MSE, but it regresses onto a **transport velocity field**, not onto a single final-action mean. This permits modeling multimodal action distributions.

Step 11 — How does $\pi 0.5$ use Flow Matching?

11.1 The generated object is an action chunk

This chapter treats action chunks as high-dimensional random variables for generative modelling. Demonstration datasets, chunk execution horizons, temporal alignment and replanning belong in [Data and Imitation Learning](#).

A robot action model aims to learn

$$p(A \mid o, \ell),$$

where

$$A = [a_0, a_1, \dots, a_{H-1}]$$

is a sequence of future actions, o denotes visual observations and robot state, and ℓ denotes language or higher-level task context.

If each action has dimension d_a , then

$$A \in \mathbb{R}^{H \times d_a}.$$

For example, $H = 50$ and $d_a = 18$ give an action chunk with 900 continuous variables. Flow Matching generates this entire high-dimensional action chunk jointly rather than autoregressively sampling every action component.

11.2 Conditional Flow Matching for action chunks

First use the time convention adopted throughout these notes: $t = 0$ means noise and $t = 1$ means data. Let

$$\epsilon \sim \mathcal{N}(0, I),$$

and define

$$A_t = (1 - t)\epsilon + tA. \tag{23}$$

The target velocity is

$$u_t = A - \epsilon. \tag{24}$$

Train the **Action Expert** using

$$\mathcal{L}_{\text{FM}} = \mathbb{E} [\|v_\theta(A_t, t, o, \ell) - (A - \epsilon)\|^2]. \tag{25}$$

Compared with unconditional Flow Matching, we additionally supply the observation and language condition to the velocity network. The ideal conditional velocity is

$$v_t(x \mid o, \ell) = \mathbb{E}[U_t \mid X_t = x, o, \ell]. \tag{26}$$

The network therefore learns how to convert Gaussian noise into plausible robot action chunks **given the current observation and task**.

Deployment connection: Learning the generative velocity field is separate from handling chunk-handoff latency. See [Real-Time Chunking in Data and Imitation Learning](#) for asynchronous execution, prefix inpainting, training-time conditioning, and timing metrics.

11.3 Important implementation detail: OpenPI reverses the time direction

When reproducing $\pi_{0.5}$, be careful to distinguish the explanatory time convention above from the convention used in actual code.

In the relevant Flow Matching implementation of OpenPI’s official `pi0.py`, the convention is

$$t = 1 : \text{noise}, \quad t = 0 : \text{real action}.$$

The interpolation is

$$A_t = t\epsilon + (1 - t)A. \quad (27)$$

Consequently,

$$\frac{dA_t}{dt} = \epsilon - A. \quad (28)$$

The velocity regression target is

$$\mathcal{L} = \|v_\theta(A_t, t, o) - (\epsilon - A)\|^2. \quad (29)$$

The model predicts **noise minus action**, not **action minus noise**. Both conventions are mathematically valid, but the velocity target and direction of integration must be consistent.

Sampling starts with $A_1 \sim \mathcal{N}(0, I)$ and integrates in the direction $t : 1 \rightarrow 0$. With K Euler steps, $\Delta t = -1/K$:

$$A_{t+\Delta t} = A_t + \Delta t v_\theta(A_t, t, o),$$

or equivalently,

$$A_{t-1/K} = A_t - \frac{1}{K} v_\theta(A_t, t, o). \quad (30)$$

The terminal state A_0 is the generated action chunk. Relevant OpenPI configurations commonly use 10 integration steps and can cache the visual-language backbone’s prefix key/value tensors, avoiding repeated full-prefix computation at every flow step. Exact settings depend on the software version and configuration.

Flow Matching velocity is not robot joint velocity

It is essential to distinguish two different notions of time:

- **Generative time t :** the artificial ODE time used to turn noise into an action chunk.
- **Physical time:** the time during which the robot actually executes $a_0, a_1, \dots, a_{H-1}$.

If $A_t \in \mathbb{R}^{50 \times 18}$, then

$$v_\theta(A_t, t, o) \in \mathbb{R}^{50 \times 18}.$$

This velocity describes how the entire **noisy action chunk** changes with generative time; it is not the physical joint velocity of the robot arm.

Step 12 — How can Flow Matching be fine-tuned with reinforcement learning?

This section connects the derivation to research on $\pi 0.5$, DSRL, and residual velocity adaptation.

12.1 What does pretrained Flow Matching learn?

Imitation learning produces a velocity field

$$v_{\text{pre}}(x, t, o),$$

whose generated distribution approximately matches the demonstration action distribution $p_{\text{demo}}(A \mid o)$.

Solving

$$\frac{dA_t}{dt} = v_{\text{pre}}(A_t, t, o)$$

produces actions resembling those in the demonstrations, but these actions need not maximize reward on the current task.

For example, several grasps may all be valid, but differ in success rate, stability, or execution time. The goal of RL post-training is to shift the generated action distribution toward **higher-return behavior**.

12.2 Why learn a residual velocity?

One natural approach freezes the pretrained velocity field and trains a smaller residual correction:

$$v_{\text{new}}(x, t, o) = v_{\text{pre}}(x, t, o) + \Delta v_{\phi}(x, t, o). \quad (31)$$

The generative ODE becomes

$$\frac{dA_t}{dt} = v_{\text{pre}}(A_t, t, o) + \Delta v_{\phi}(A_t, t, o). \quad (32)$$

Motivation. This resembles **residual control**: preserve useful behavior learned by a pretrained system while making incremental adjustments for a new reward or task objective.

This is a proposed post-training algorithmic choice—not a structural requirement of Flow Matching itself.

12.3 Why cannot reward be used directly as a velocity label?

During pretraining, the conditional target

$$u_t = A - \epsilon$$

comes from differentiating an explicitly defined path equation.

Reinforcement learning typically provides

$$R(A, o),$$

which evaluates a final action or executed trajectory. It does **not** directly specify the appropriate residual velocity at each intermediate generative state:

$$\Delta v_\phi(A_t, t, o).$$

We therefore cannot simply interpret an advantage estimate as a velocity label in generative space. A principled connection is required:

Reward $\rightarrow$ Target action distribution $\rightarrow$ Velocity-field update.

12.4 A natural design: Advantage-Weighted Flow Matching

Suppose online exploration yields a dataset

$$\mathcal{D} = \{(o_i, A_i, R_i)\}_{i=1}^N.$$

Assign larger **positive weights** to higher-return actions:

$$w_i = \exp\left(\frac{\widehat{\text{Adv}}_i}{\beta}\right), \quad \beta > 0.$$

Consider the following weighted residual-velocity regression objective:

$$\mathcal{L}(\phi) = \mathbb{E}_{\mathcal{D}, \epsilon, t} \left[w(A, o) \|v_{\text{pre}}(A_t, t, o) + \Delta v_\phi(A_t, t, o) - (A - \epsilon)\|^2 \right], \quad (33)$$

where

$$A_t = (1 - t)\epsilon + tA.$$

A residual regularizer may be added:

$$\lambda \mathbb{E} [\|\Delta v_\phi(A_t, t, o)\|^2]$$

to penalize deviations from the pretrained velocity field.

What is the mathematical motivation?

Recall that the optimal MSE regressor is the conditional expectation of velocities under the chosen joint distribution.

For fixed observation o , define a reward-weighted action distribution

$$p_w(A | o) = \frac{w(A, o)p_{\mathcal{D}}(A | o)}{Z(o)}, \quad (34)$$

where

$$Z(o) = \mathbb{E}_{A \sim p_{\mathcal{D}}(\cdot | o)}[w(A, o)].$$

If the dataset has sufficient coverage, the weights are nonnegative and integrable, and the model and optimization are sufficiently expressive, weighted CFM can be interpreted as learning the velocity field associated with the **reweighted target action distribution**.

The research rationale is therefore

Advantage $\rightarrow$ weighted action distribution $\rightarrow$ Flow Matching $\rightarrow$ new velocity field.

Important limitation. Weighted Flow Matching is not automatically equivalent to an online policy-gradient method, and it does not guarantee monotonic policy improvement. Practical performance still depends on data coverage, advantage estimation, distribution shift, numerical integration, and the design of update constraints.

Step 13 — Synthesis: the complete mathematical logic of Flow Matching

13.1 Eight essential steps

1. Define the generative goal. Transport a Gaussian distribution into the data distribution:

$$p_0 = \mathcal{N}(0, I) \rightarrow p_1 = p_{\text{data}}.$$

2. Represent continuous transport with an ODE.

$$\dot{x}_t = v_\theta(x_t, t).$$

Its induced probability path should obey

$$\partial_t p_t + \nabla \cdot (p_t v_\theta) = 0.$$

3. The ideal objective regresses onto a global velocity field.

$$\mathcal{L}_{\text{FM}} = \mathbb{E}[\|v_\theta(x, t) - v_t^*(x)\|^2],$$

but v_t^* is generally unknown.

4. Construct a computable conditional path.

$$x_t = (1 - t)x_0 + tx_1, \quad u_t = x_1 - x_0.$$

5. Prove that the marginal velocity is a conditional expectation.

$$v_t^*(x) = \mathbb{E}[X_1 - X_0 \mid X_t = x].$$

6. Use the L^2 projection property of MSE.

$$\mathbb{E}\|v_\theta - U_t\|^2 = \mathbb{E}\|v_\theta - v_t^*\|^2 + C.$$

7. Arrive at the simple training objective.

$$\min_{\theta} \mathbb{E}_{\substack{t \sim \mathcal{U}[0,1] \\ x_0 \sim \mathcal{N}(0, I) \\ x_1 \sim p_{\text{data}}}} \left[\|v_\theta((1 - t)x_0 + tx_1, t) - (x_1 - x_0)\|^2 \right].$$

8. Generate samples by integrating the ODE.

$$\begin{cases} x_0 \sim \mathcal{N}(0, I), \\ \dot{x}_t = v_\theta(x_t, t), \\ x_1 \sim p_\theta \approx p_{\text{data}}. \end{cases}$$

The last line means that the **distribution of generated samples** approximates the data distribution; it does not imply that every generated sample must equal an existing training example.

13.2 Three ideas to remember

First, Flow Matching learns a dynamical system for probability distributions.

$$\dot{x}_t = v_t(x_t) \iff \partial_t p_t = -\nabla \cdot (p_t v_t).$$

This links generative modeling to continuous-time dynamical systems.

Second, the essence of CFM is conditional expectation and L^2 projection.

$$v_t^*(x) = \mathbb{E}[U_t \mid X_t = x].$$

This explains why simple supervision from random conditional paths can train a velocity field for complicated distributions.

Third, straight-line training interpolation does not imply a straight-line sampling trajectory. The neural network learns a global marginal velocity field determined jointly by conditional paths and their probability distribution.

For online post-training of Flow Matching policies, a particularly useful research direction is

$$\text{changing the action distribution} \iff \text{changing the transport velocity field.}$$

How to use rewards or advantages to modify the velocity field **stably**, while retaining the generative capabilities of a pretrained model, remains an optimization problem worth studying in its own right.

Document scope. This English edition preserves the streamlined 13-section structure: foundational derivations, conditional flow matching, training and inference, $\pi 0.5$, and residual-velocity RL fine-tuning. The previously requested chapters on *Optimal Transport* and on the *Flow Matching–Diffusion mathematical connection* remain excluded. Later sections and numbered equations have been renumbered continuously.