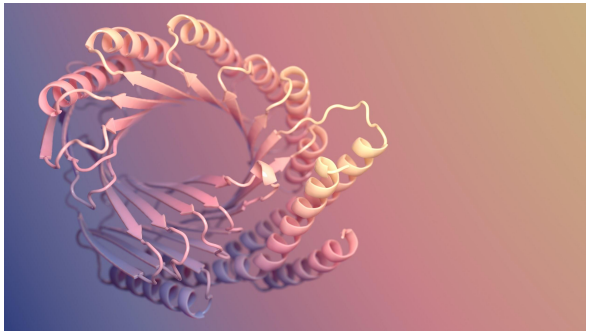
Flow Matching and Diffusion Models

Examples & Motivation

Pi0



RFDiffusion

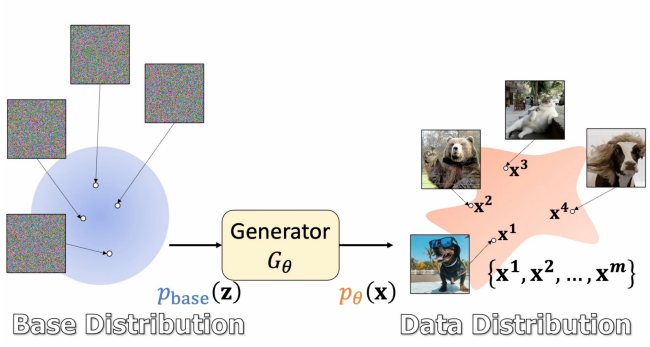
Stable Diffusion

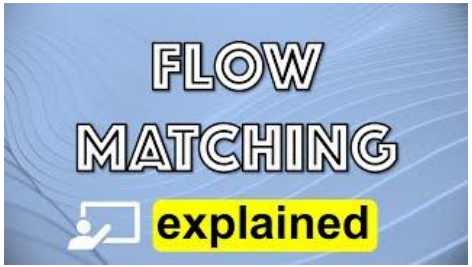
Table of Contents

Overview

General Idea

Goal: convert some known probability distribution into a one that describes data well



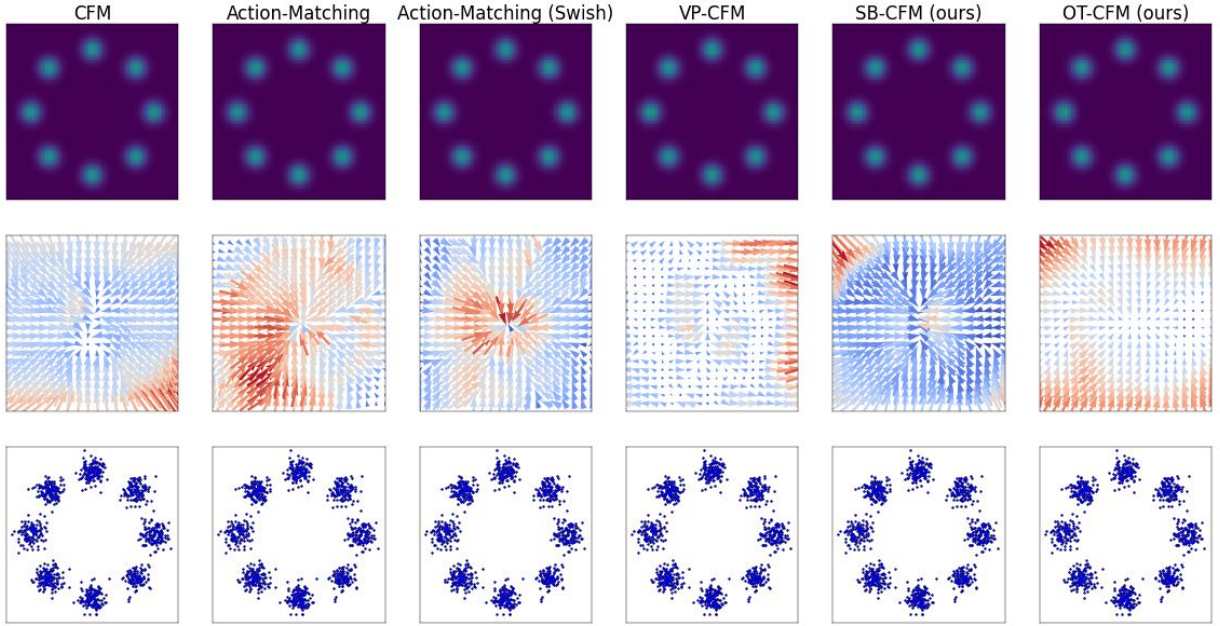


Imagine deforming substances

Probability is our substance

How do we learn to deform distributions?

Goal: Convert 8 two-dimensional Gaussians into “two Moons” shaped distribution



Flow

Underlying time-dependent vector field

We want to learn this vector field

Animations source: TorchCFM

How do we learn to deform distributions?

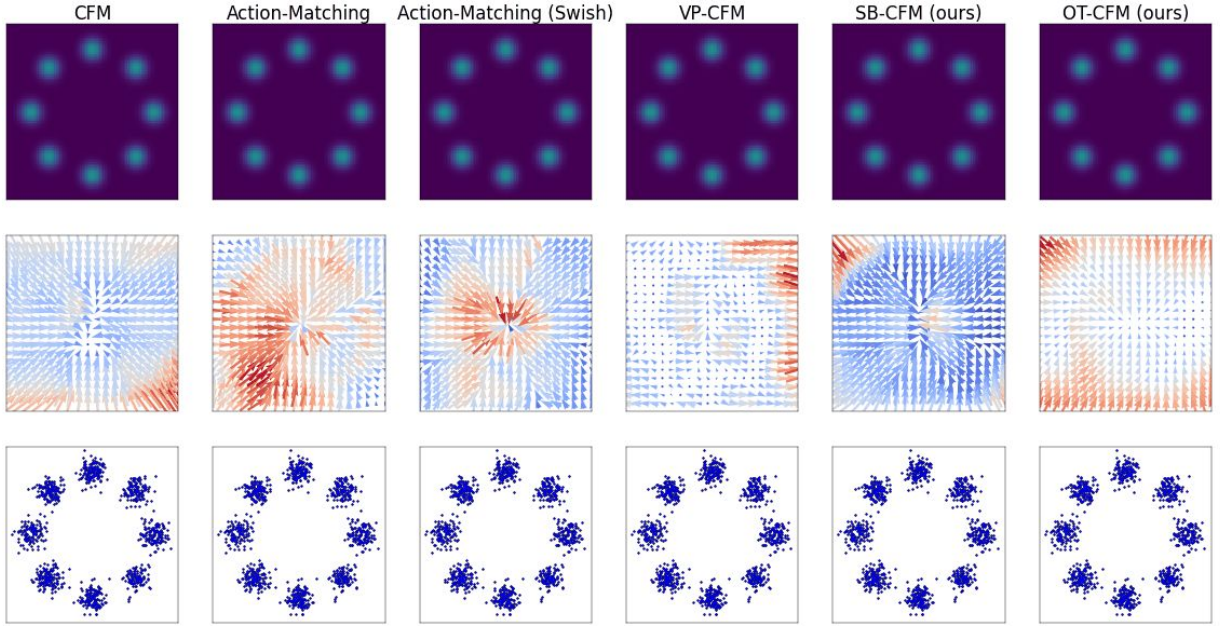
$$\phi : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$$

- ϕ: flow
- $[0, 1]$: time
- $\mathbb{R}^d$: points in space

$$\frac{\mathrm{d}\phi_t(x_t)}{\mathrm{d}t} = u_t(x_t)$$

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t,\, p_t(x)} ||v_{\theta,t}(X) - u_t(X)||$$

Goal: Convert 8 two-dimensional Gaussians into “two Moons” shaped distribution



Animations source: TorchCFM

Why do we learn to deform distributions?

Generation as sampling from the data distribution

Data distribution: Distribution of objects that we want to generate: p_{data}

Probability density:

$$p_{\text{data}} : \mathbb{R}^d \to \mathbb{R}_{\geq 0},$$

$$z \mapsto p_{\text{data}}(z)$$

Note: We don't know the probability density!

Generation means sampling the data distribution:

$$z \sim p_{\text{data}}$$

$z =$ 

So in the end we do not necessarily want to know the distribution, but we want to **be able to sample from it**

Why do we learn to deform distributions?

Conditional Generation allows us to condition on prompts

Data distribution p_{data}

Fixed prompt

"Dog"

Conditional data distribution $p_{\text{data}}(\cdot|y)$

Condition variable: **y**

y="Dog" y="Cat" y="Landscape"

Conditional generation means sampling the conditional data distribution:

$$z \sim p_{\text{data}}(\cdot|y)$$

We will first focus on unconditional generation and then learn how to translate an unconditional model to a conditional one.

Preliminaries

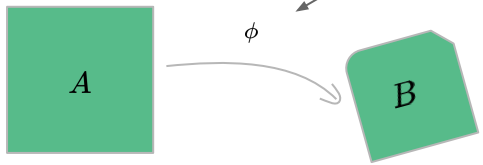
Deforming Probability

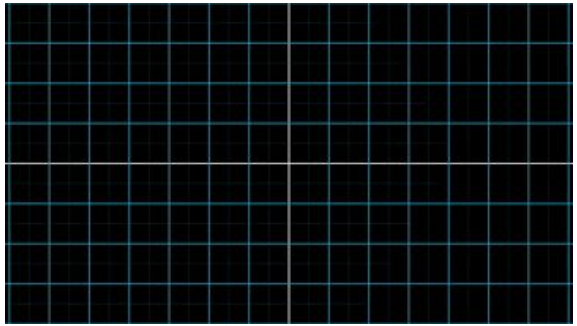
Consider invertible map:

$$Y = \phi(X)$$

Random variables

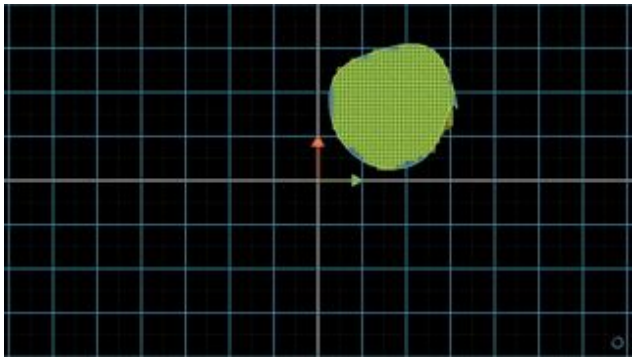
The map determines how space is deformed





$$\mathbb{P}[Y \in B] = \mathbb{P}[X \in A] \implies \int_B f_Y(y)dy = \int_A f_X(x)dx$$

Probabilities have to stay the same!

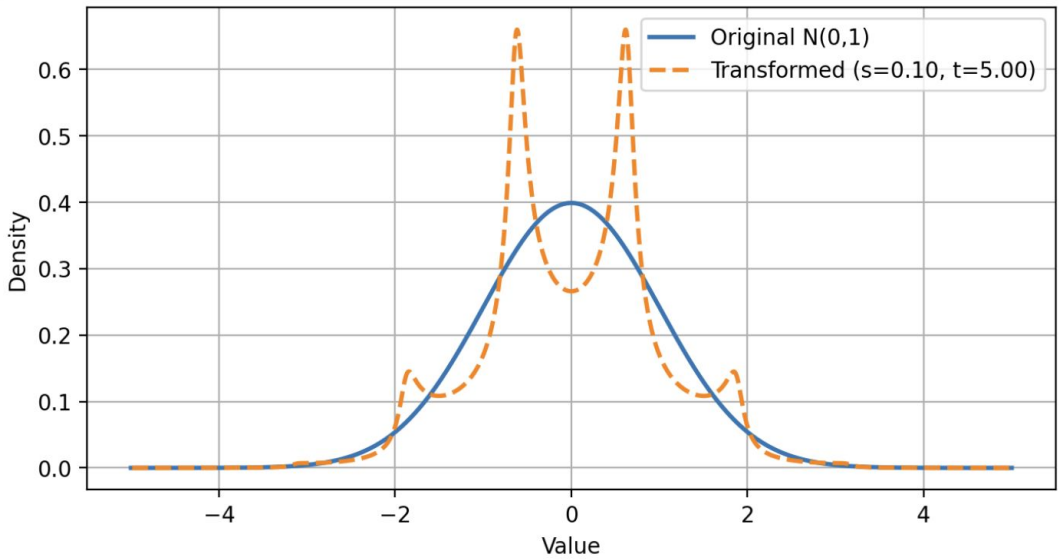


$$\int_A f_X(x)dx = \int_B f_X\left(\phi^{-1}(y)\right)\left|\det D\phi^{-1}(y)\right|dy$$

$$\implies f_Y(y) = f_X\left(\phi^{-1}(y)\right) \cdot \left|\det D\phi^{-1}(y)\right|$$

Deforming Probability: Example

$$X \sim \mathcal{N}(0,1),\ Y = \Phi(X) = X + s\sin(tX),\ \text{where } s,t \in \mathbb{R} \text{ and } |st| < 1$$



Managed to transform a unimodal PDF into a multimodal PDF!

How was this possible?

Deforming Probability: General Case

$$X \sim p_{\text{init}},\ \Phi_\theta(X) \sim p_d$$

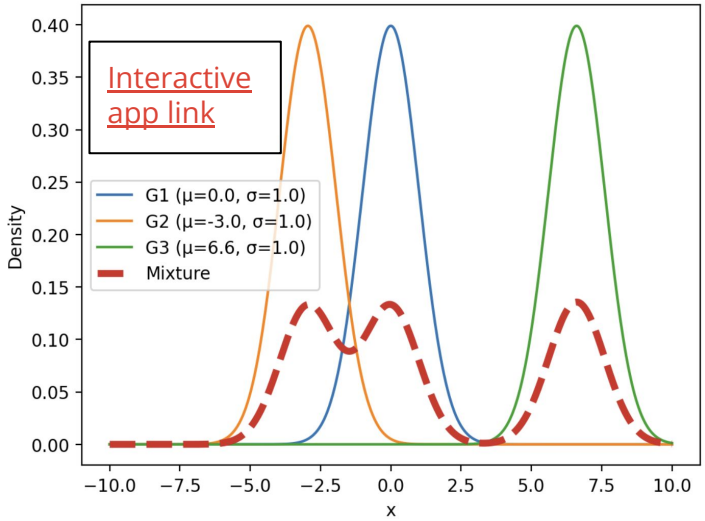
Some known distribution

Deformed distribution

When the deformed distribution will be good?

When it fits the data well!

For example, a Gaussian Mixture (GM) with Gaussians at each observed data point



Of course for such a GM one would need at least #parameters = #data points

So we could control the deformation with parameters. This brings us to…

Residual Flows

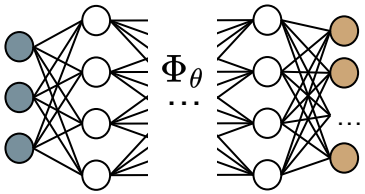
Learning the Deformation: Kullback-Leibler Divergence

$$X \sim p_{\text{init}},\ \Phi_\theta(X) \sim p_d$$

Some known distribution

Deformed distribution

Need to measure how similar deformed distribution is to the true data distribution q



Try to learn the deformation with a neural network

But how do we train? Loss = …?

Loss: $\mathcal{L}(\theta) = -\mathbb{E}_{X\sim q}[\log p_d(X)]$

KL divergence: $D_{KL}(q, p_d) = \int q(x) \log \frac{q(x)}{p_d(x)}$

Why???

Consider a coin:

$\begin{cases} q_1 & \text{heads} \\ q_2 & \text{tails} \end{cases}$ $\begin{cases} p_1 & \text{heads} \\ p_2 & \text{tails} \end{cases}$

N obs: H H T H T T H

Average likelihood ratio per observation:

$$\left(\prod_N \frac{q(x_i)}{p(x_i)}\right)^{\frac{1}{N}} = \left(\frac{q_1^{N_h} q_2^{N_t}}{p_1^{N_h} p_2^{N_t}}\right)^{\frac{1}{N}} \quad \text{take log: } \frac{N_h}{N}\log\frac{q_1}{p_1} + \frac{N_t}{N}\log\frac{q_2}{p_2}$$

Why???

$$D_{KL}(q, p_d) = \mathbb{E}_{X\sim q}\left[\log \frac{q(X)}{p_d(X)}\right]$$
$$= \mathbb{E}_{X\sim q}[\log q(X)] - \mathbb{E}_{X\sim q}[\log p_d(X)]$$

Does not depend on model parameters

So called **cross-entropy**

In the limit: $\sum_x q(x) \log \frac{q(x)}{p(x)}$

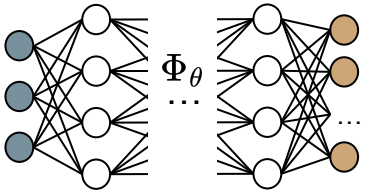
Monte Carlo Estimates

$$X \sim p_{\text{init}},\ \Phi_\theta(X) \sim p_d$$

Some known distribution

Deformed distribution

Try to learn the deformation with a neural network



Loss: $\mathcal{L}(\theta) = -\mathbb{E}_{X\sim q}[\log p_d(X)]$

An integral, possible over an infinite space…

How do we compute such a thing during training?

Approximate:

Samples drawn according to the distribution

$$\mathbb{E}[f(X)] \approx \frac{1}{N}\sum_i f(x_i)$$

How do we get samples?

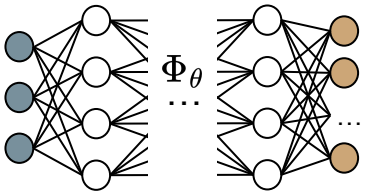
DATA!

Learning the Deformation: Residual Flows

$$X \sim p_{\text{init}},\ \Phi_\theta(X) \sim p_d$$

Some known distribution

Deformed distribution



Try to learn the deformation with a neural network

Can we take any network?

Loss: $\mathcal{L}(\theta) = -\mathbb{E}_{X\sim q}\left[\log p_d(X)\right]$

Have to make sure it is a useful probability deformation!

$$p_d(x) = p_{\text{init}}\left(\phi^{-1}(x)\right) \cdot \left|\det D\phi^{-1}(x)\right|$$

Has to be invertible…

… and we must be able to compute its determinant efficiently

One way to do it - Residual Networks:

$$\phi = \phi_N \circ \phi_{N-1} \circ \cdots \circ \phi_1$$

$$\text{where } \phi_k(x) = x + \delta u_k(x),\ \delta \in \mathbb{R}$$

Proper *u* can assure invertibility and easy computation of the determinant (e.g. make the jacobian triangular)

$$\begin{bmatrix} \frac{\partial \phi_1}{\partial x_1} & 0 & \dots & 0 \\ \frac{\partial \phi_2}{\partial x_1} & \frac{\partial \phi_2}{\partial x_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \phi_d}{\partial x_1} & \frac{\partial \phi_d}{\partial x_2} & \dots & \frac{\partial \phi_d}{\partial x_d} \end{bmatrix}$$

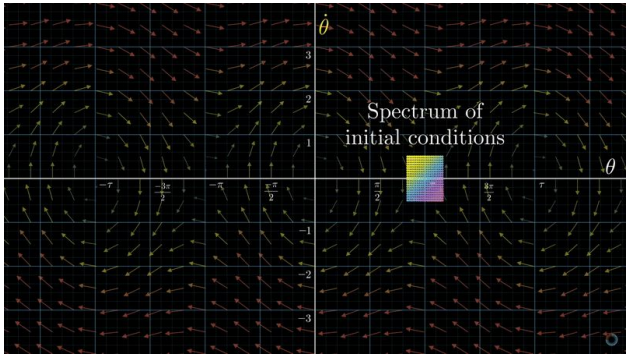
$$\mathcal{L}(\theta) = \mathbb{E}_{X\sim q}\left[\log p_{\text{init}}\left(\phi^{-1}(x)\right) + \sum_k \log\det \phi_k\right]$$

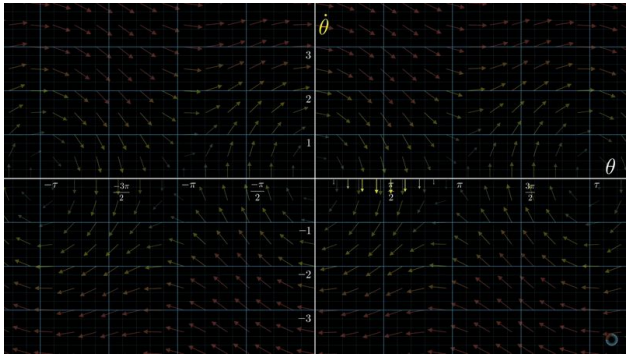
Residual Flows as Numerical Integration

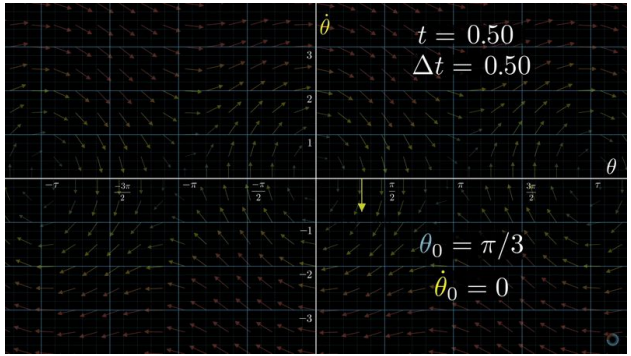
One can notice that this → $\phi_k(x) = x + \delta \cdot u_k(x),\ \delta \in \mathbb{R}$

Looks very similar to Forward Euler integration scheme → $x_{t+\Delta t} \approx x_t + \Delta t \cdot v(x_t, t)$

Numerical integration intuition: we track what's happening with points of a substance in space.







In real life the vector field guides the points along continuous paths

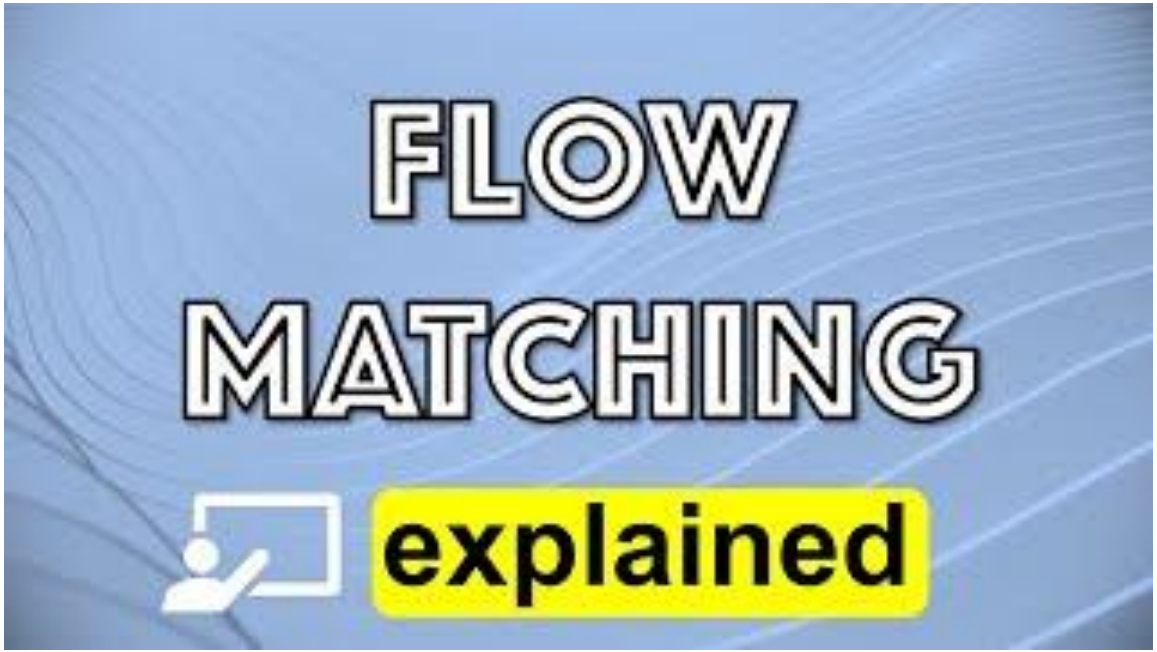
During a computation we use discrete time steps

Changing the timestep affects the result!

The core idea is approximating such Ordinary Differential Equation (ODE) → $v(x_t, t) = \frac{\mathrm{d}x}{\mathrm{d}t}(t) \approx \frac{x_{t+\Delta t} - x_t}{\Delta t}$

Flow Matching

Continuous Flows



So the idea now is to consider **continuous time** deformations:

$$\phi : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$$

- ϕ: flow
- $[0, 1]$: time
- $\mathbb{R}^d$: points in space

And learn vector fields that govern these deformations:

$$\frac{\mathrm{d}\phi\left(t, x\left(t\right)\right)}{\mathrm{d}t} = u\left(t, x\left(t\right)\right)$$

$$\iff \frac{\mathrm{d}\phi_t\left(x_t\right)}{\mathrm{d}t} = u_t\left(x_t\right)$$

Simplified notation

Note that we no longer have to specify:

- the structure for in-between deformations
- the number of ‘integration steps” N

This **extra freedom** is **why people started considering flows**

Probability Flow: Example

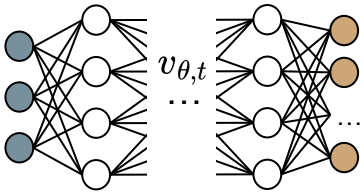
$$X \sim \mathcal{N}(0,1),\ Y = \Phi(X) = X + s\sin(tX),\ \text{where } s,t \in \mathbb{R} \text{ and } |st| < 1$$

Interactive app link

Fix *s* and think of *t* as time

Continuous Normalizing Flows (CNF)

Goal: learn the vector field which defines the flow



$$\frac{\mathrm{d}\phi_t(x_t)}{\mathrm{d}t} = u_t(x_t)$$

We could try to still use the following relation:

$$p_t(x) = p_{\text{init}}\left(\phi_t^{-1}(x)\right) \cdot \left|\det D\phi_t^{-1}(x)\right|$$

Probability path

to maximise the cross-entropy again:

$$\mathcal{L}(\theta) = -\mathbb{E}_{X\sim q}\left[\log p_1(X)\right]$$

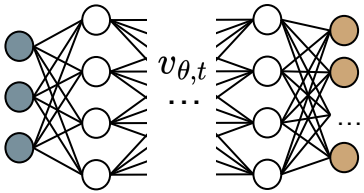
But then we have to:

- Perform numerical integration to find the final PDF of the deformation at each training step
- Somehow estimate the divergence of the vector field to compute the log (why?)

Long story short: **this is hard.** Instead…

Continuous Normalizing Flows (CNF)

Goal: learn the vector field which defines the flow

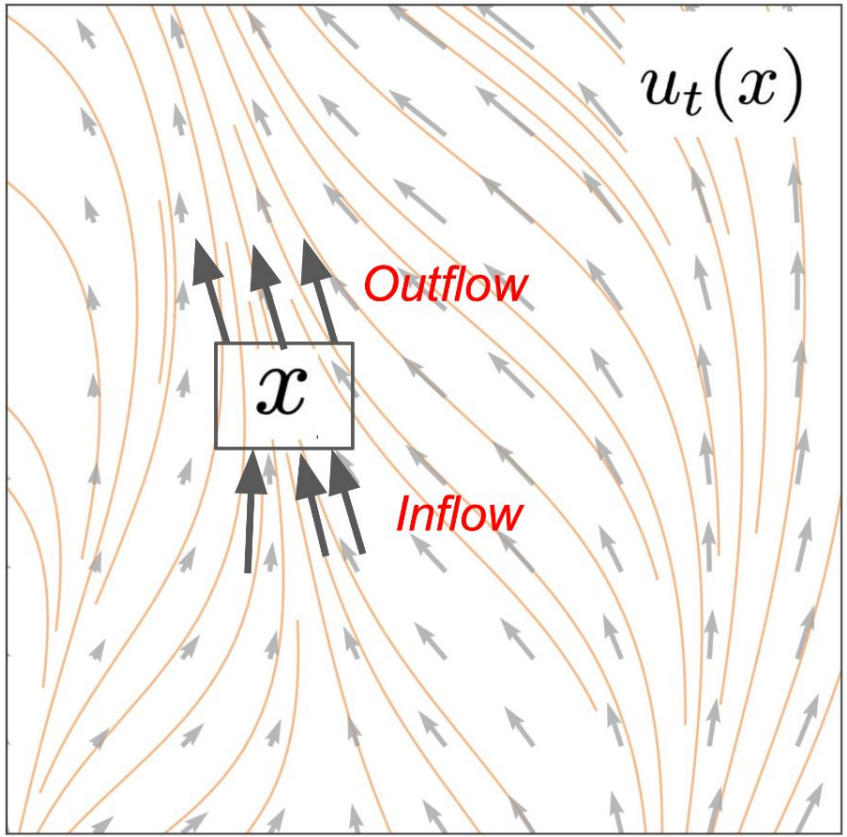


$$\frac{\mathrm{d}\phi_t(x_t)}{\mathrm{d}t} = u_t(x_t)$$

It turns out that if the following holds, a vector field induces a probability path (deformation):

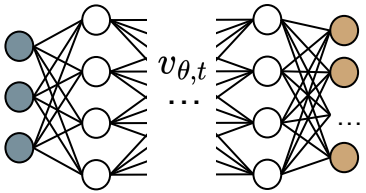
$$\frac{\partial p_t(x_t)}{\partial t} = \nabla \cdot u_t(x_t) p_t(x_t)$$

Continuity equation



Flow Matching

Goal: learn the vector field which defines the flow



$$\frac{\mathrm{d}\phi_t(x_t)}{\mathrm{d}t} = u_t(x_t)$$

It turns out that if the following holds, a vector field induces a probability path (deformation):

$$\frac{\partial p_t(x_t)}{\partial t} = \nabla \cdot u_t(x_t) p_t(x_t)$$

Just optimise distance between the vector fields?

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

uniform time distribution

Induced probability path

Neural network approximation

Vector field deforming probability correctly

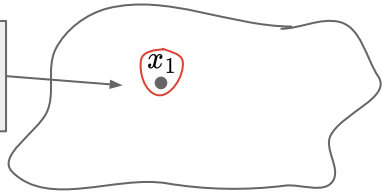
But we don't have access to neither a valid probability path nor the vector field *u*…

… it's also hard to construct / propose anything reasonable

Conditional Flow Matching

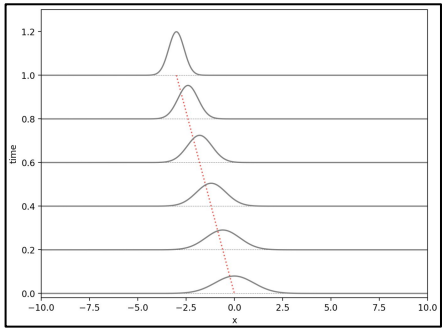
A trivial case when we would know how to construct a vector field is if the true distribution is ~deterministic:

Our samples are always very similar to a single data point



The vector field would have to simply deform init distribution into a distribution gathered around the data point, e.g. linearly:

$$p_1 = \mathcal{N}(x_1, \sigma_{\text{small}})$$
$$\uparrow$$
$$p_{\text{init}} = \mathcal{N}(0, 1)$$

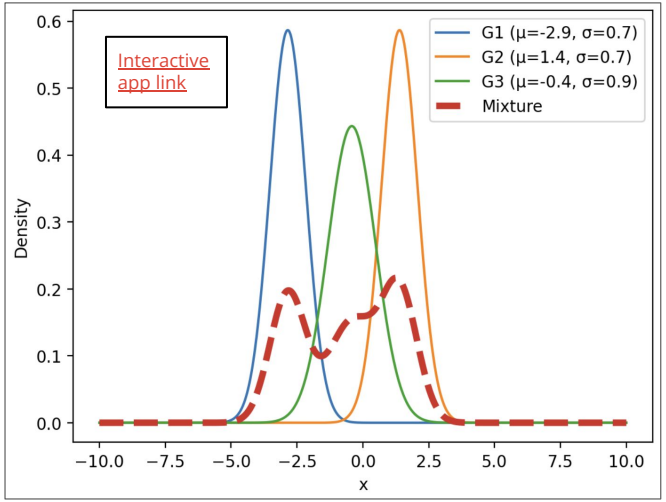


Let's rewrite a non-trivial distribution as a mixture of such simple distributions

$$p_1(x) = \int p_1(x \mid x_1) q(x_1) dx_1 \approx q(x)$$

We hope!

But should be true if we choose conditional probabilities well **and have enough data**



Conditional Flow Matching

QUESTION: Assuming having access to all possible data points, what conditional distributions would always give us exactly the real distribution here?

ANSWER: Dirac delta distributions

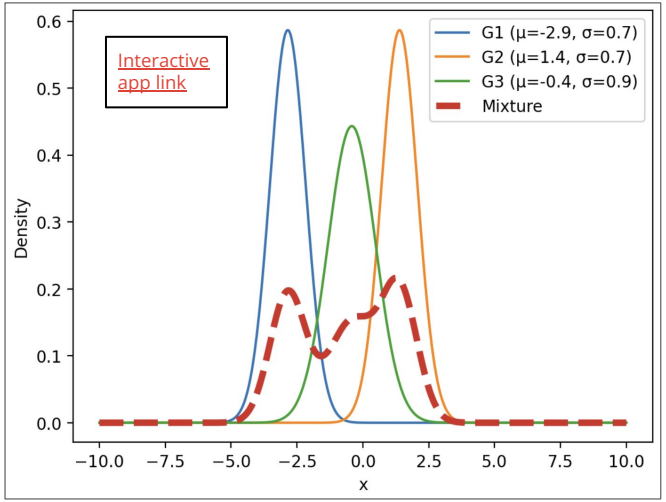
Sidenote: This procedure is kind of analogous to the discretization in space in classical physical simulations for deformable substances

Let's rewrite a non-trivial distribution as a mixture of such simple distributions

$$p_1(x) = \int p_1(x \mid x_1) q(x_1) dx_1 \approx q(x)$$

We hope!

But should be true if we choose conditional probabilities well **and have enough data**



Conditional Flow Matching

This works at every moment on the probability path:

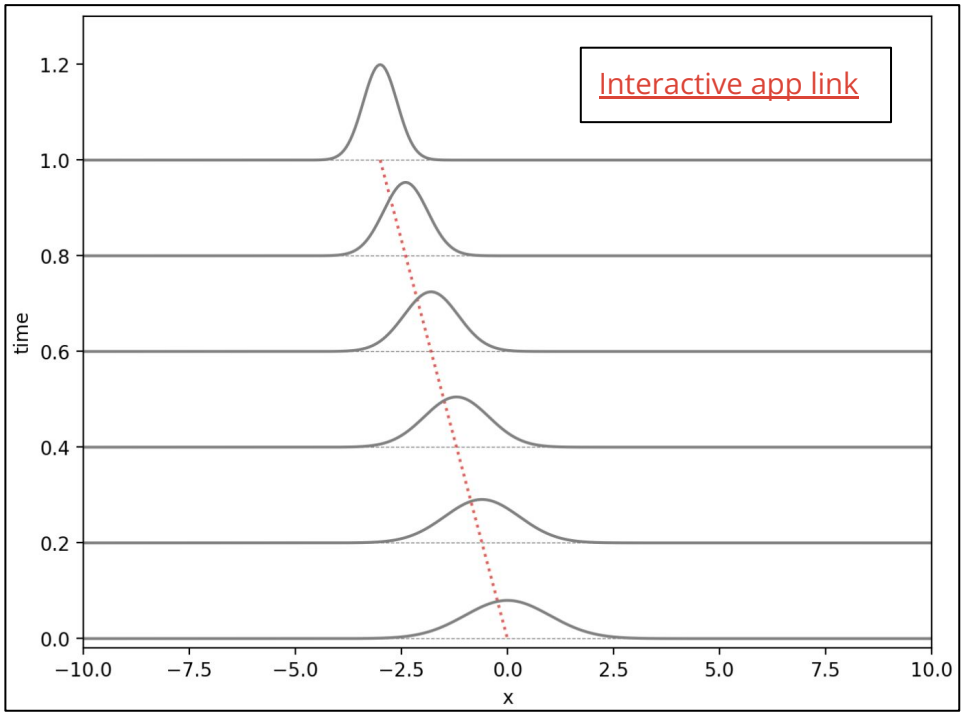
$$p_t(x) = \int p_t(x \mid x_1) q(x_1) dx_1$$

We end up with conditional probability paths

As before, we have a related conditional vector field that determines conditional deformation (continuity equation):

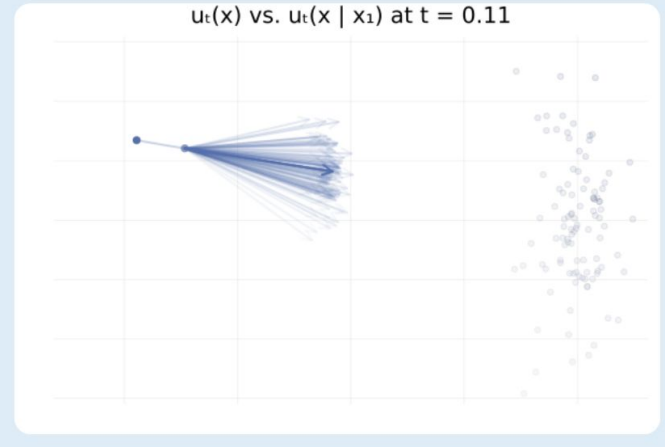
$$\frac{\partial p_t(x_t \mid x_1)}{\partial t} = \nabla \cdot u_t(x_t \mid x_1) p(x_t \mid x_1)$$

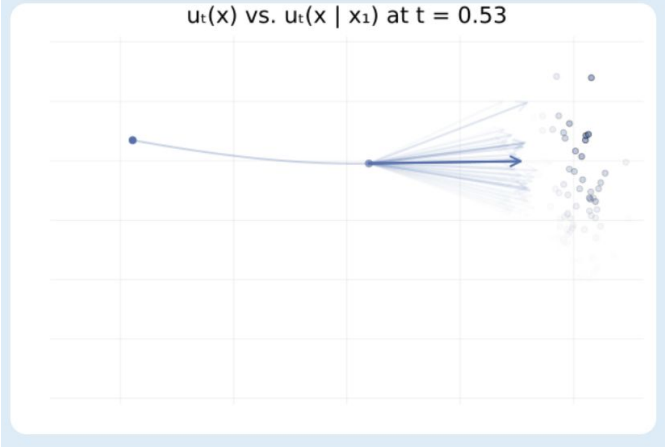
Example: Optimal Transport Gaussian conditional probability path.



Conditional Flow Matching

What happens if we average over multiple conditional vector field directions?





In other words, we look at:

$$u_t(x_t) = \mathbb{E}_{p_{1|t}}[u_t(x_t \mid x_1)]$$
$$= \int u_t(x_t \mid x_1) \frac{p_t(x_t \mid x_1) q(x_1)}{p_t(x_t)} dx_1$$
$$= \int u_t(x_t \mid x_1) p_t(x_1 \mid x_t) dx_1$$

Posterior probability

One can (relatively easily) show that such a vector field induces our overall deformation:

$$\frac{\partial p_t(x_t)}{\partial t} = \nabla \cdot u_t(x_t) p_t(x_t)$$

The problem is we **almost never have closed formulas for the posterior probability (WHY?)**

Conditional Flow Matching

Surprisingly, one can (again, relatively easily) show that the unconditional loss:

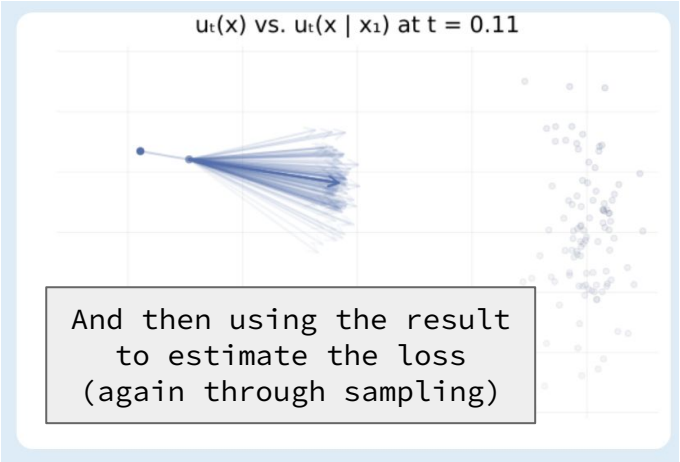
$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t,\, p_t(x)} ||v_{\theta,t}(X) - u_t(X)||^2$$

Is up to a constant equal to a conditional loss:

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t,\, p_t(x \mid x_1),\, q(x_1)} ||v_{\theta,t}(X) - u_t(X \mid X_1)||^2$$

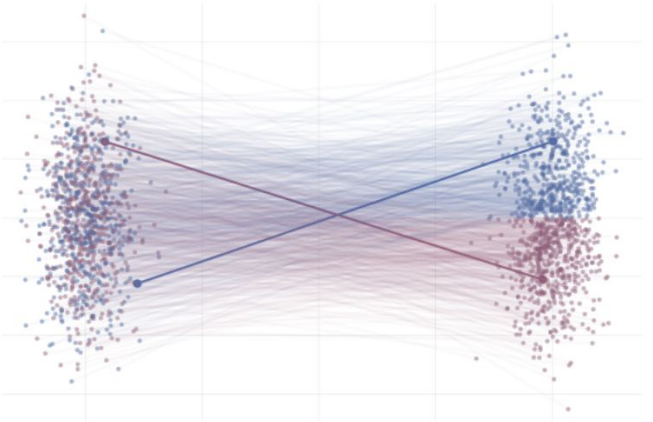
Think of changing the sampling scheme.

Instead of this:



And then using the result to estimate the loss (again through sampling)

We sample single "conditional arrows" all over the space



So **with reasonable conditional paths and enough data**, we **can estimate the conditional loss and compute gradients** without ever estimating the true vector field

Gaussian Conditional Probability Paths

In practice almost always gaussian conditional probability paths are used:

$$p_t(x \mid x_1) = \mathcal{N}\left(x \mid \mu_t(x_1), \sigma_t^2(x_1) I\right)$$

One can show that they have the corresponding conditional vector field

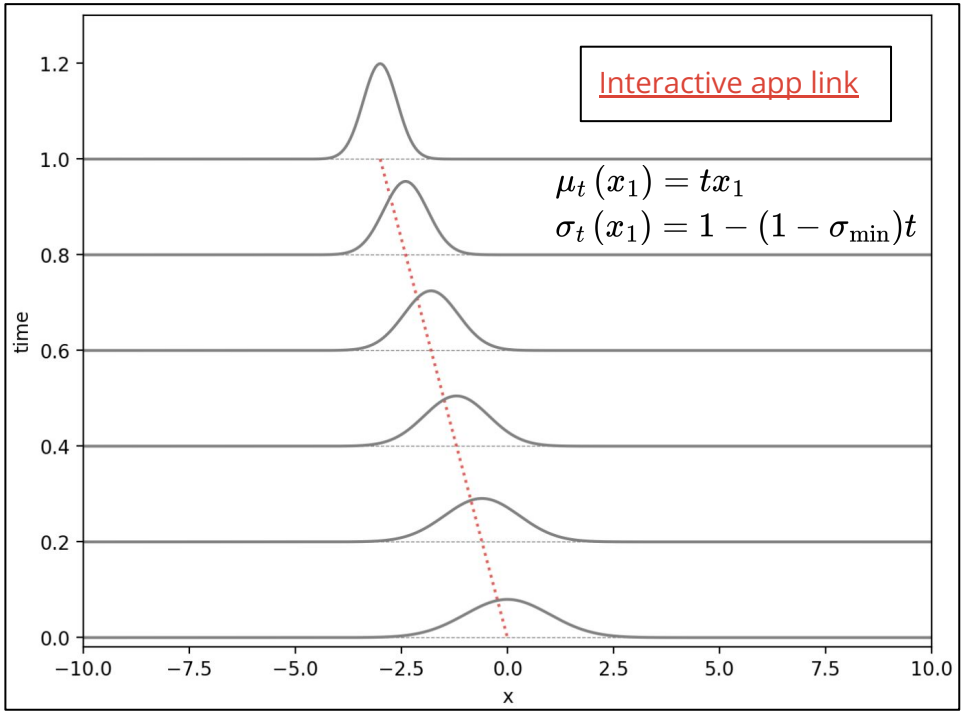
$$u_t(x \mid x_1) = \frac{\frac{d}{dt}\sigma_t(x_1)}{\sigma_t(x_1)}(x - \mu_t(x_1)) + \frac{d}{dt}\mu_t(x_1)$$

Example 2. Diffusion Process

$$\mu_t(x_1) = x_1$$
$$\sigma_t := f_{\downarrow} \text{ with } \sigma_1 = 0$$

Some decreasing function

Example: Optimal Transport Gaussian conditional probability path.



How to generate objects with a Flow Model

Algorithm 1 Sampling from a Flow Model with Euler method

Require: Neural network vector field u_t^θ, number of steps n

1: Set $t = 0$
2: Set step size $h = \frac{1}{n}$
3: Draw a sample $X_0 \sim p_{\text{init}}$ *Random initialization!*
4: **for** $i = 1, \ldots, n-1$ **do**
5: $\quad X_{t+h} = X_t + h u_t^\theta(X_t)$
6: $\quad$ Update $t \leftarrow t + h$
7: **end for**
8: **return** X_1 *Return final point*

Example: Pi0

Flow Matching

$\mathbf{A}_t^0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

$\cdots$

$\mathbf{A}_t^\tau$

$\cdots$

$\mathbf{A}_t^1 \sim p_1$

Flow Matching in PiO Context

Extra notation just to easily differentiate from conditional probability paths from Flow Matching

Real distribution

$$q(\mathbf{A}_t \mid \mathbf{o}_t) =: q_{\mathbf{o}_t}(\mathbf{A}_t)$$

Action chunk

Observation

Subscripts denote robotic timesteps

Superscripts correspond to the flow matching timesteps, i.e. the one used for integration

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t,\, q(x_1),\, p_t(x \mid x_1)} ||v_{\theta,t}(X) - u_t(X \mid X_1)||^2$$

$$\mathcal{L}^\tau(\theta) = \mathbb{E}_{q_{\mathbf{o}_t}(\mathbf{A}_t),\, p(\mathbf{A}_t^\tau \mid \mathbf{A}_t)} ||v_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t) - u_t(\mathbf{A}_t^\tau \mid \mathbf{A}_t)||^2$$

Optimal Transport conditional probability paths:

$$p(\mathbf{A}_t^\tau \mid \mathbf{A}_t) = \mathcal{N}(\tau \mathbf{A}_t, (1-\tau)\mathbf{I})$$

Forward Euler integration to find continuous actions (starting from noise)

$$\mathbf{A}_t^{\tau+\delta} = \mathbf{A}_t^\tau + \delta \mathbf{v}_\theta(\mathbf{A}_t^\tau, \mathbf{o}_t)$$


autonomous, 1x speed

Architecture Overview

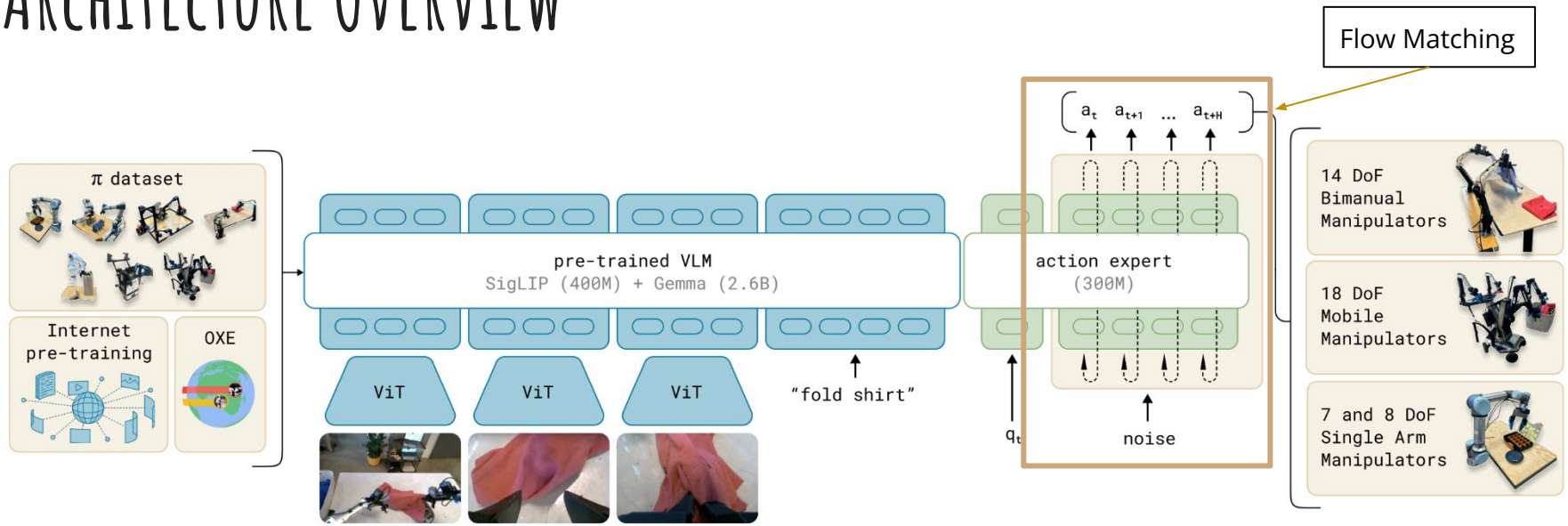
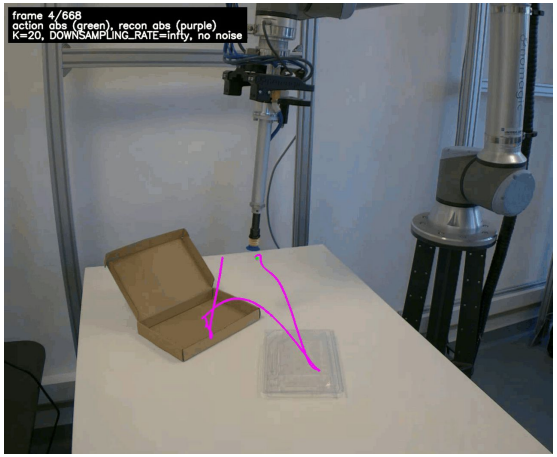
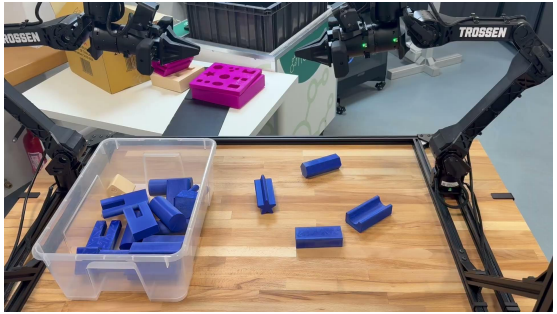
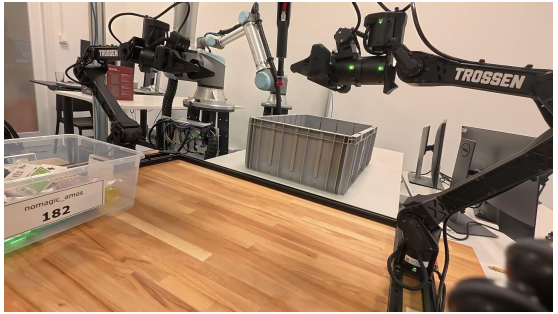


Fig. 3: **Overview of our framework.** We start with a pre-training mixture, which consists of both our own dexterous manipulation datasets and open-source data. We use this mixture to train our flow matching VLA model, which consists of a larger VLM backbone and a smaller *action expert* for processing robot states and actions. The VLM backbone weights are initialized from PaliGemma [5], providing representations learned from large-scale Internet pre-training. The resulting π_0 model can be used to control multiple robot embodiments with differing action spaces to accomplish a wide variety of tasks.

We Are Doing Research in these Areas at MIMUW!



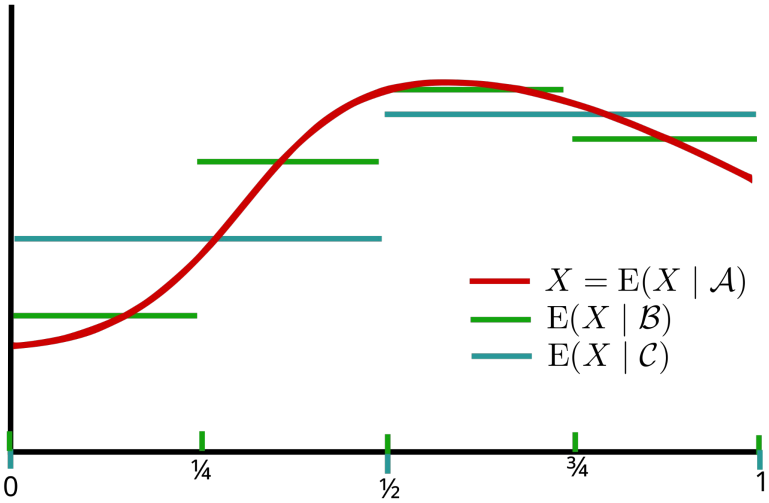




Note About the Proofs: Conditional Expectation

Intuitive definition:
a **random variable** that takes values that are the "averages" on sets which you specify as "measurable"

A good metaphor is a camera: the higher "resolution" sigma field (more measurable sets) allows you to draw more "specific" samples



With this intuition one can clearly see that:

$$\mathbb{E}\left[\mathbb{E}\left[X\,|\mathcal{B}|\right]|\,\mathcal{A}\right] = \mathbb{E}\left[X\,|\,\mathcal{A}\right] \text{ whenever } \mathcal{A} \subseteq \mathcal{B}$$

We are taking a low resolution picture of a higher resolution picture

In particular for trivial outer sigma field:

$$\mathcal{A} = \{\emptyset, \Omega\} \implies \mathbb{E}\left[\mathbb{E}\left[X\,|\mathcal{B}|\right]|\,\mathcal{A}\right] = \mathbb{E}\left[X\right]$$

Diffusion

Flow Matching: Summary

$$\phi : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$$

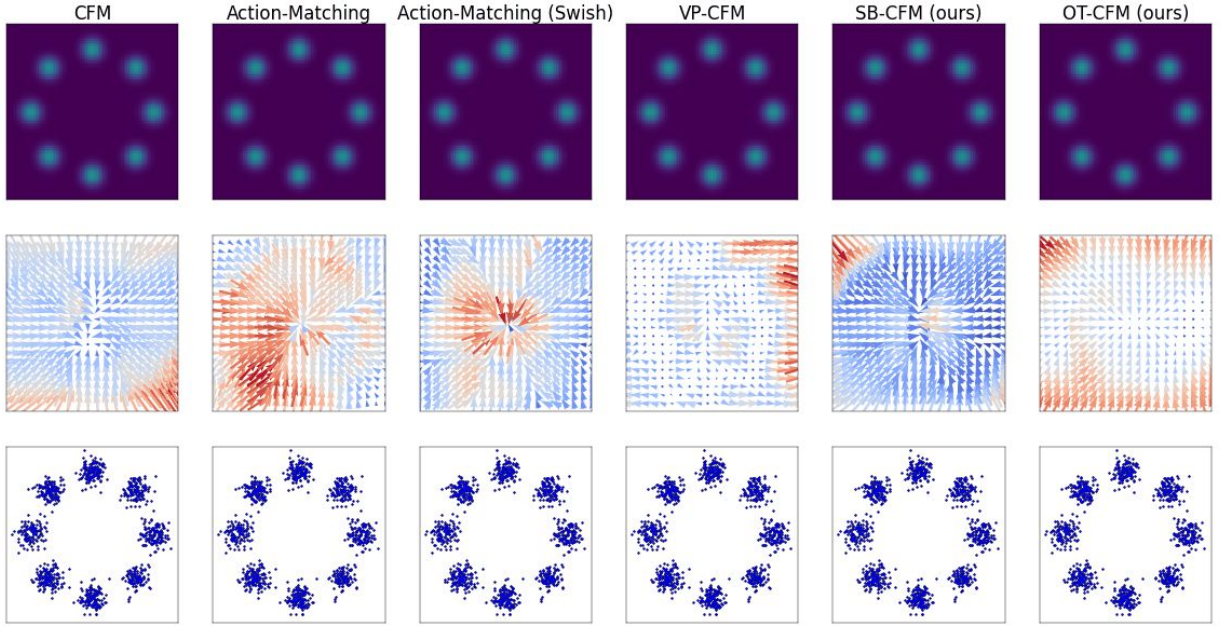
flow — time — points in space

$$\frac{\mathrm{d}\phi_t(x_t)}{\mathrm{d}t} = u_t(x_t)$$

FM summarised: Numerically solve an ODE to reach the data distribution

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t,\, p_t(x)} ||v_{\theta,t}(X) - u_t(X)||$$

Goal: Convert 8 two-dimensional Gaussians into “two Moons” shaped distribution



Animations source: TorchCFM

Diffusion: Summary

$$\phi : [0, 1] \times \mathbb{R}^d \rightarrow \mathbb{R}^d$$

flow — time — points in space

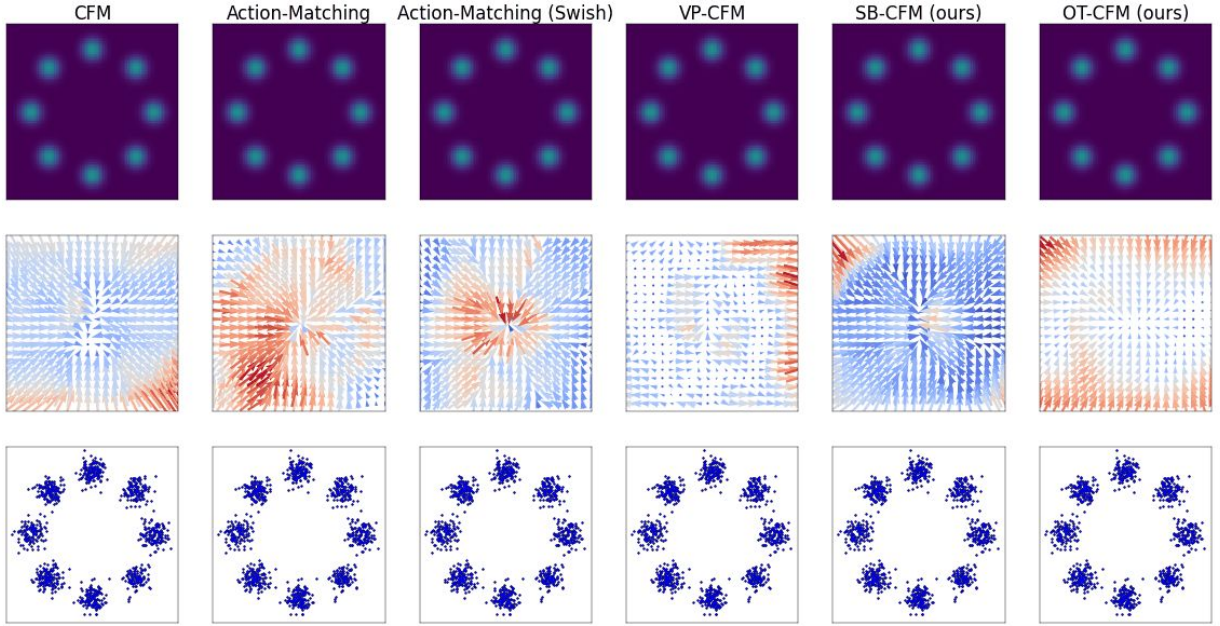
$$d\phi_t(x_t) = u_t(x_t)dt + \sigma_t dW_t$$

$$\frac{\mathrm{d}\phi_t(x_t)}{\mathrm{d}t} = u_t(x_t)$$

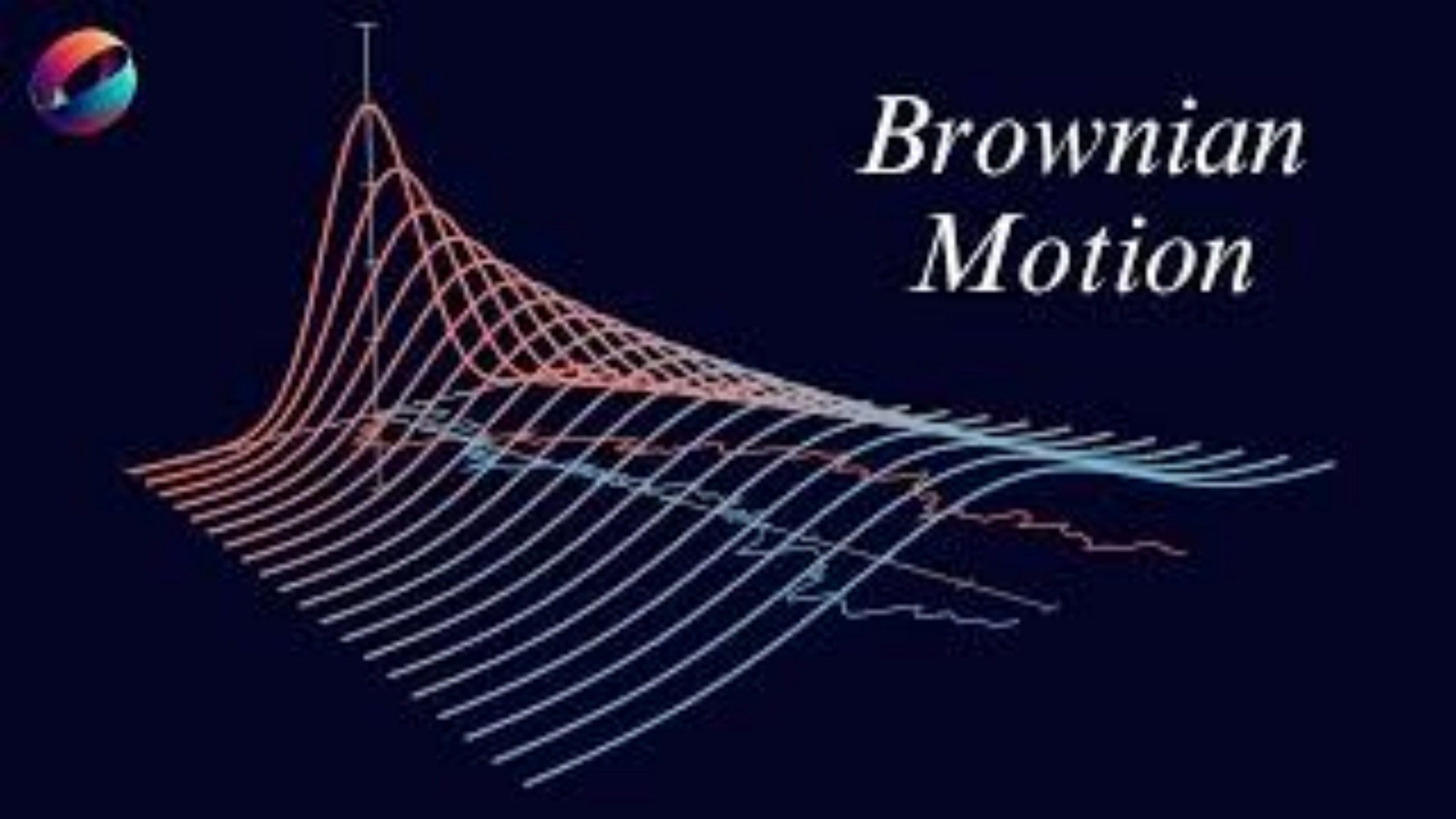
FM summarised: Numerically solve an ODE to reach the data distribution

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t,\, p_t(x)} ||v_{\theta,t}(X) - u_t(X)||$$

Goal: Convert 8 two-dimensional Gaussians into “two Moons” shaped distribution



Animations source: TorchCFM


Brownian
Motion

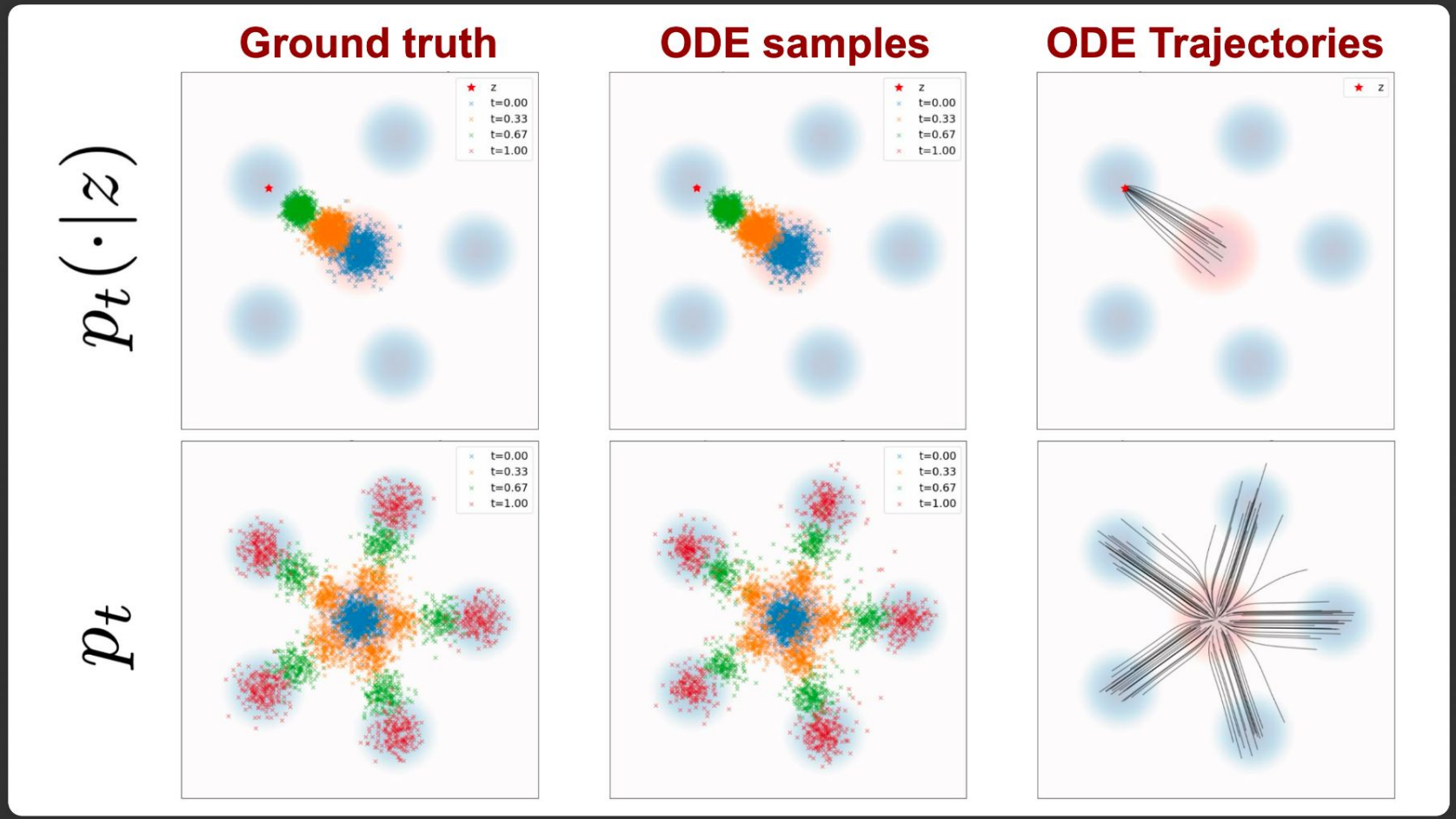
Numerical SDE simulation (Euler-Maruyama method)

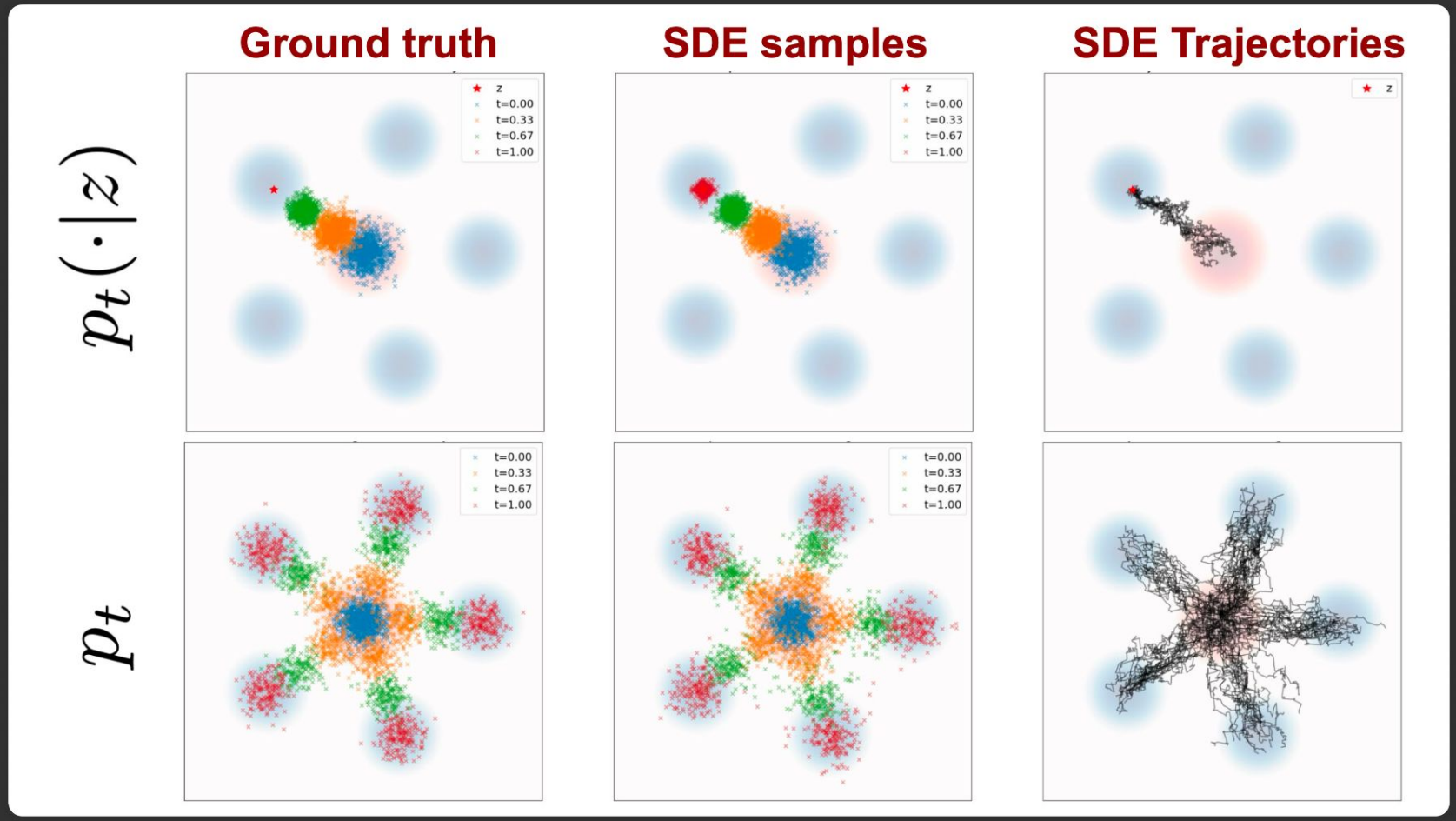
Algorithm 2 Sampling from a SDE (Euler-Maruyama method)

Require: Vector field u_t, number of steps n, diffusion coefficient σ_t

Set $t = 0$
Set step size $h = \frac{1}{n}$
Set $X_0 = x_0$
for $i = 1, \ldots, n-1$ **do**
  Draw a sample $\epsilon \sim \mathcal{N}(0, I_d)$
  $X_{t+h} = X_t + h u_t(X_t) + \sigma_t \sqrt{h} \epsilon$
  Update $t \leftarrow t + h$
end for
return $X_0, X_h, X_{2h}, X_{3h}, \ldots, X_1$

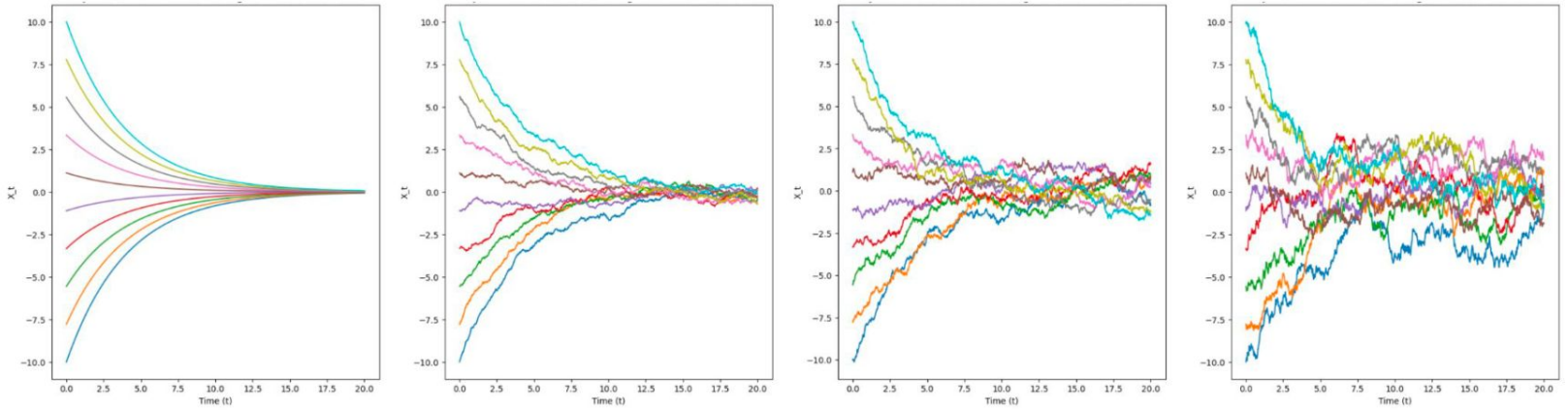
Add additional noise with var=h scaled by diffusion coefficient σ_t


Ground truth
ODE samples
ODE Trajectories
$p_t(\cdot|z)$
p_t
z
t=0.00
t=0.33
t=0.67
t=1.00


Ground truth
SDE samples
SDE Trajectories
$p_t(\cdot|z)$
p_t
z
t=0.00
t=0.33
t=0.67
t=1.00

Ornstein-Uhlenbeck Process

$$\mathrm{d}X_t = -\theta X_t \mathrm{d}t + \sigma dW_t$$



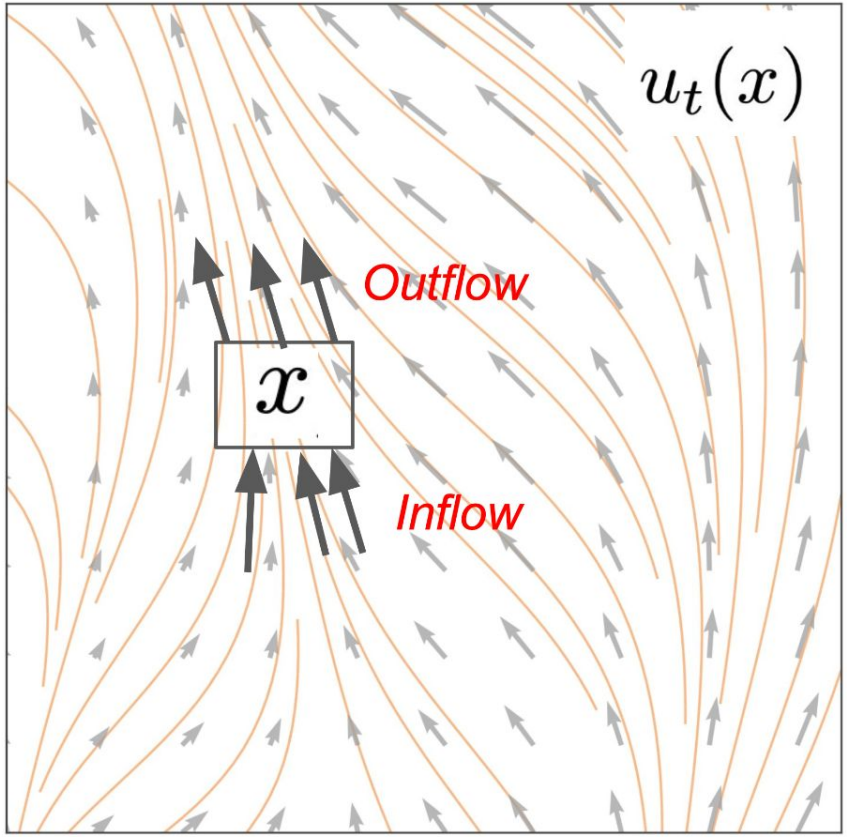
Increasing diffusion coefficient σ

Continuity Equation

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = -\mathrm{div}(p_t u_t)(x)$$

Change of probability mass at x

Outflow - inflow of probability mass from u

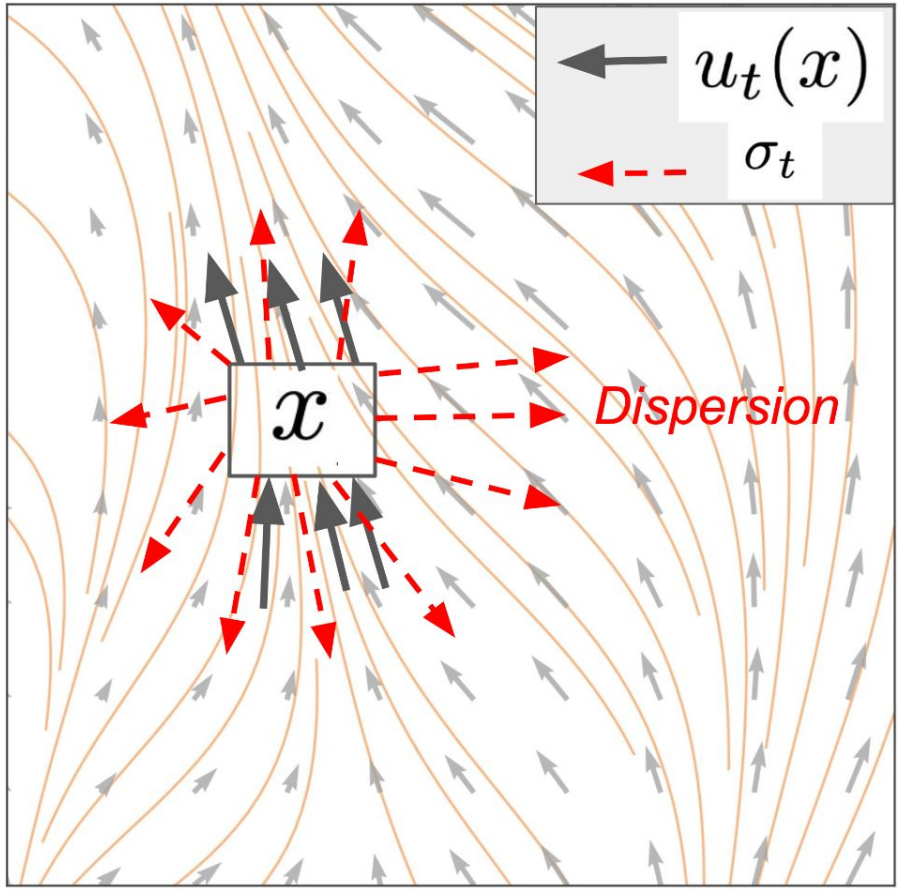


Fokker-Planck Equation

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = -\mathrm{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x)$$

Change of probability mass at x

Heat dispersion



Outlook (Next class) - Score Matching Loss

The Score Matching loss is a mean squared error between the neural network and the marginal score function:

$$L_{\text{sm}}(\theta) = \mathbb{E}_{z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|s_t^\theta(x) - \nabla \log p_t(x)\|^2]$$

Why this score here?

To train a diffusion model, we need to train the score network by minimizing the score matching loss (How? Next class!)

Reminder - Marginal score function:

$$\nabla \log p_t(x) = \int \nabla \log p_t(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z$$

Conditional score function

SDE extension trick: The marginal score function allows to extend the ODE to a SDE with arbitrary diffusion coefficient:

It's the correction we have to make to follow the path in presence of noise

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2} \nabla \log p_t(X_t) \right] \mathrm{d}t + \sigma_t \mathrm{d}W_t$$

$$\Rightarrow X_t \sim p_t$$

"Go where the log-likelihood increases the most!"

Extra Stuff

In-painting

You can run the inference in parallel to the robot execution thread.

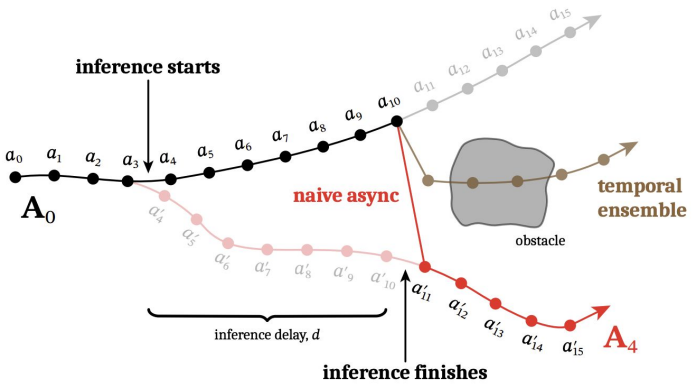
That’s why we no longer have jerky movement like this:

But smooth movement like this:

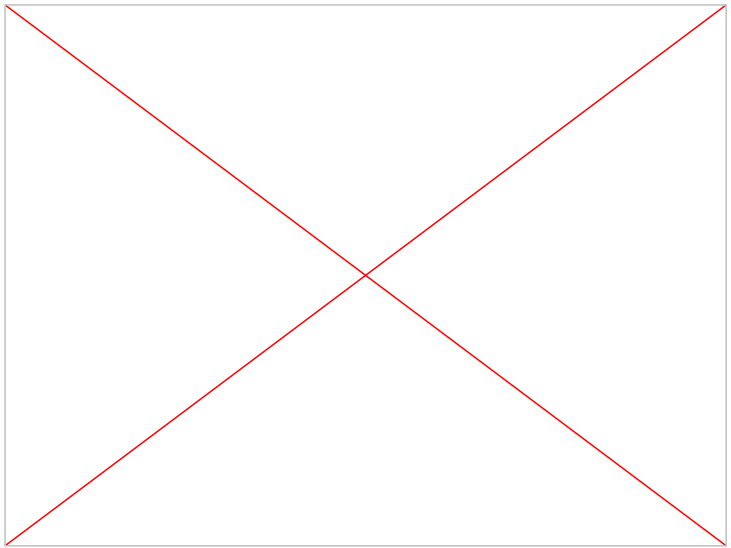
In-painting

You can run the inference in parallel to the robot execution thread.

But this requires for the inferences to be consistent:



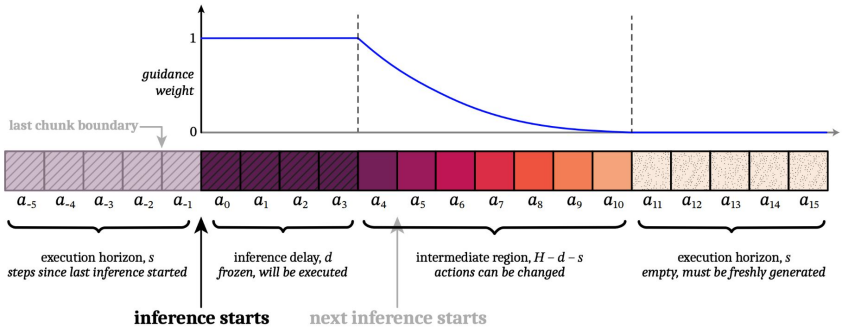
But smooth movement like this:



IN-PAINTING

You can run the inference in parallel to the robot execution thread.

But this requires for the inferences to be consistent:

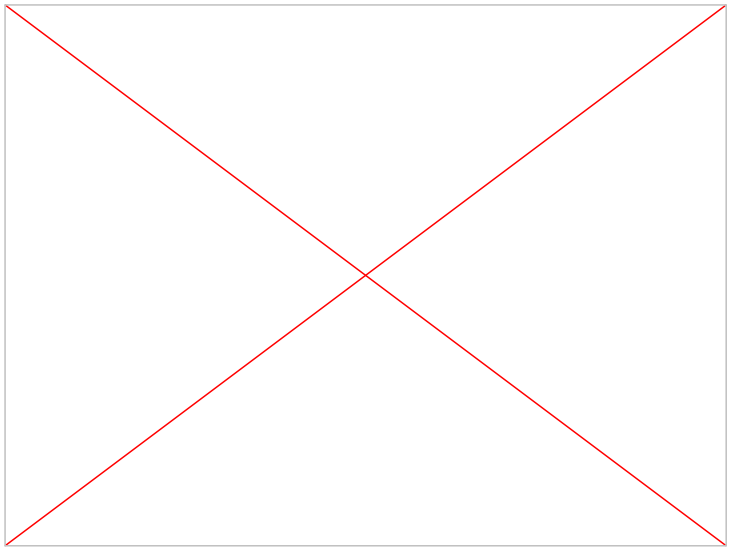


$$\mathbf{v}_{\Pi\text{GDM}}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) = \mathbf{v}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right)\left(\mathbf{Y} - \widehat{\mathbf{A}_t^1}\right)^\top \text{diag}(\mathbf{W})\,\frac{\partial \widehat{\mathbf{A}_t^1}}{\partial \mathbf{A}_t^\tau}$$

$$\text{where } \widehat{\mathbf{A}_t^1} = \mathbf{A}_t^\tau + (1-\tau)\mathbf{v}(\mathbf{A}_t^\tau, \mathbf{o}_t, \tau),$$

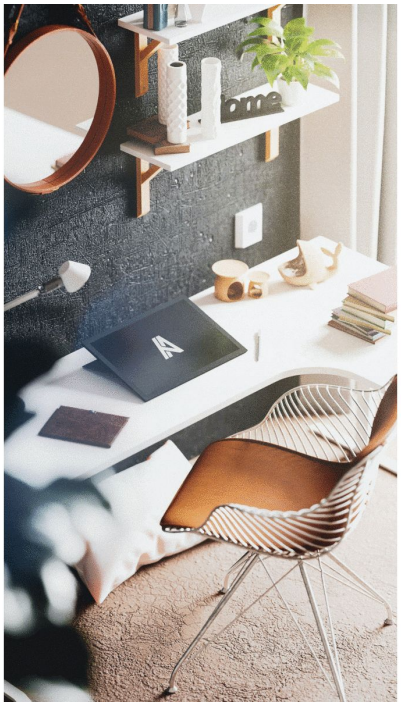
$$r_\tau^2 = \frac{(1-\tau)^2}{\tau^2 + (1-\tau)^2}.$$

But smooth movement like this:



In-painting

Similarly can be used for images



a b c

Image source

Functional Flow Matching

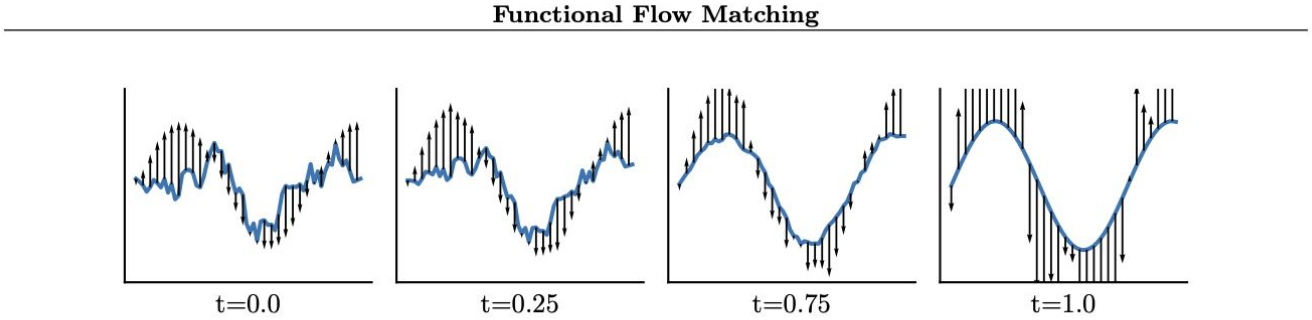


Figure 1: An illustration of our FFM method. The vector field $v_t(f) \in \mathcal{F}$ (in black) transforms a noise sample $g \sim \mu_0 = \mathcal{N}(0, C_0)$ drawn from a Gaussian process with a Matérn kernel (at $t = 0$) to the function $f(x) = \sin(x)$ (at $t = 1$) via solving a function-space ODE. By sampling many such $g \sim \mu_0$, we define a conditional path of measures μ_t^f approximately interpolating between $\mathcal{N}(0, C_0)$ and the function f, which we marginalize over samples $f \sim \nu$ from the data distribution in order to obtain a path of measures approximately interpolating between μ_0 and ν.

With proper care, FM can be generalised to work on probability measures

You can then use it to deform functions, not only finite-dimensional vectors

Hubert Lawenda's presentation during the last robot learning seminar

Additional Resources

Additional resources

- To learn more about diffusion and flow matching models:
  - Presentation was mostly built on:
    - An introduction to Flow Matching · Cambridge MLG Blog, and
    - MIT 6.S184: Flow Matching and Diffusion Models lectures
  - TorchCFM - a python package which implements Flow Matching
  - How I Understand Flow Matching
  - Flow Matching for Generative Modeling (Paper Explained)
  - What are Normalizing Flows?

Thank you