

Lecture 5: Diffusion Models

From <Noise> to Data: Diffusion Models through Flow Matching

MAE 6291 / CSCI 6907 - Large Language and Vision Models (LLVM)

Guest Lecturer: Alex Zongo, Fall 2026

Generative AI - A new generation of AI systems



Artistic Images



Realistic Videos



Draft Texts

These systems are “creative”: they generate new objects.

Generative AI - A new generation of AI systems



Artistic Images



Realistic Videos



Draft Texts

These systems are "creative": they generate new objects.

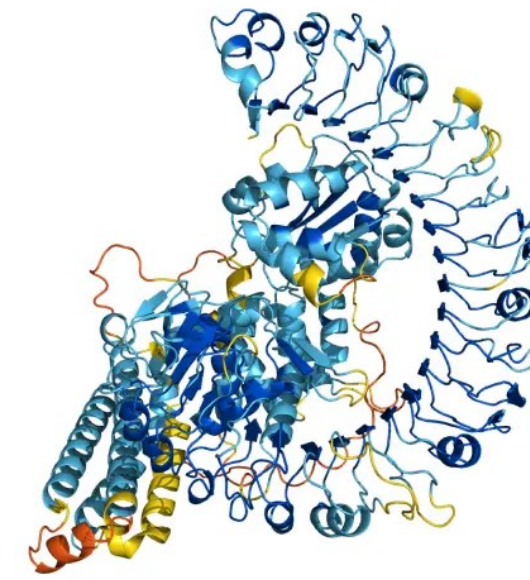
Flow and Diffusion Models generate structured continuous outputs: images, videos, molecular structures, robot actions



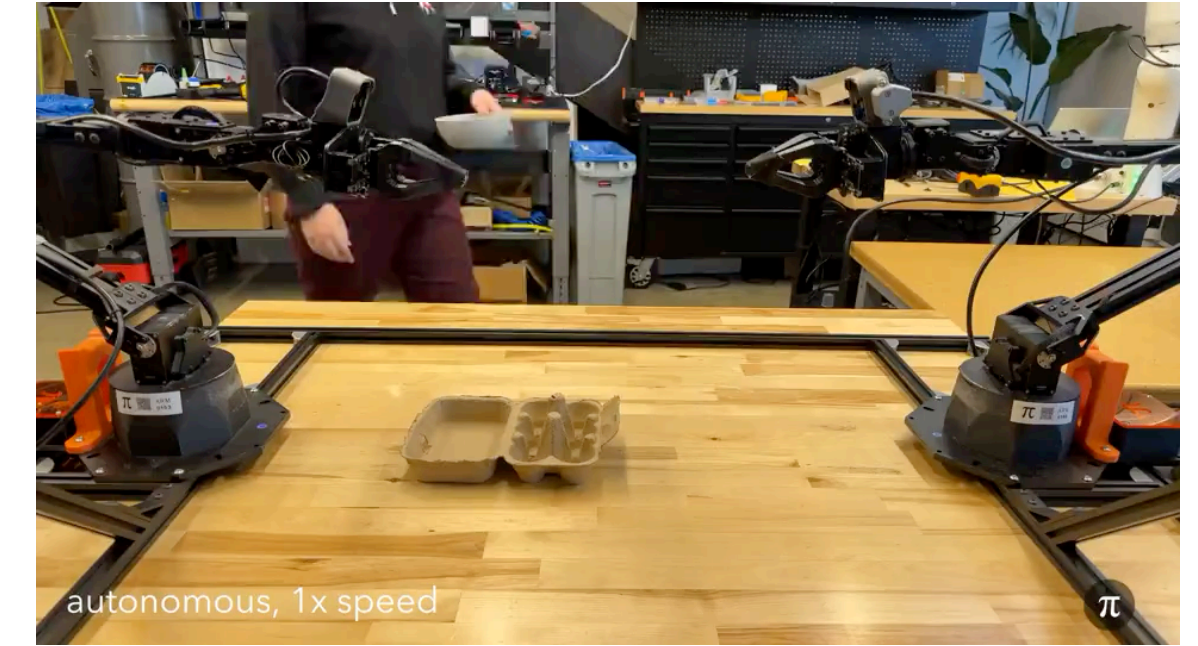
Stable Diffusion,
DALL·E,
Midjourney



OpenAI Sora,
Google Veo,
Meta MovieGen



Google
AlphaFold 3,
Boltz 2, Chai-1



π_0 , Isaac
GROOT

**Most SOTA generative models for images/videos/proteins/
robotics: Diffusion and Flow Models**

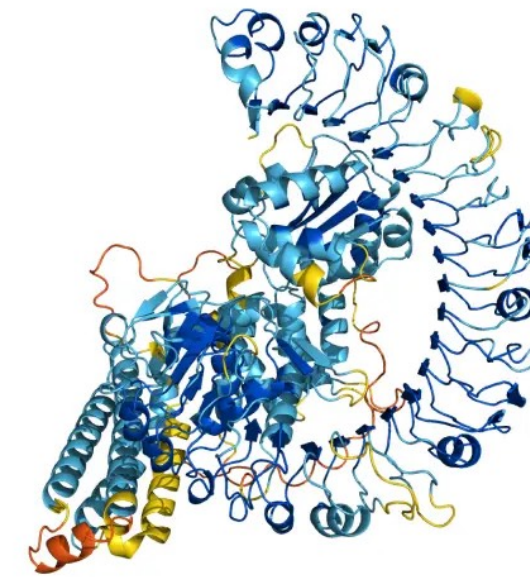
Flow and Diffusion Models generate structured continuous outputs: images, videos, molecular structures, robot actions



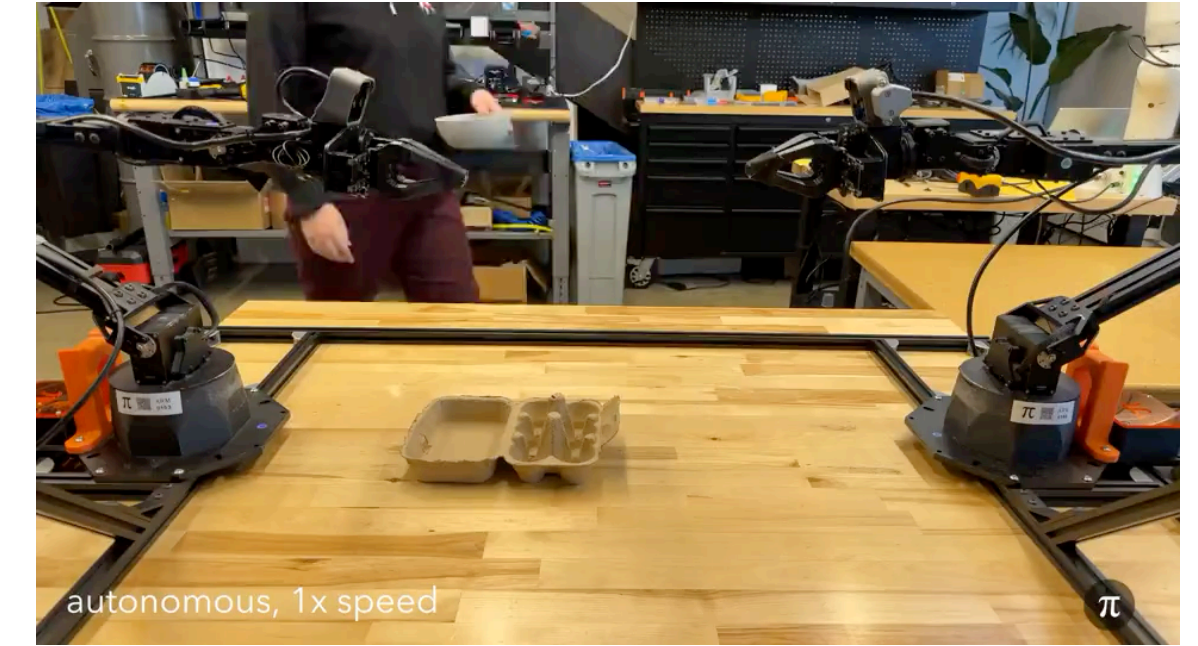
Stable Diffusion,
DALL·E,
Midjourney



OpenAI Sora,
Google Veo,
Meta MovieGen



Google
AlphaFold 3,
Boltz 2, Chai-1



π_0 , Isaac
GROOT

**Most SOTA generative models for images/videos/proteins/
robotics: Diffusion and Flow Models**

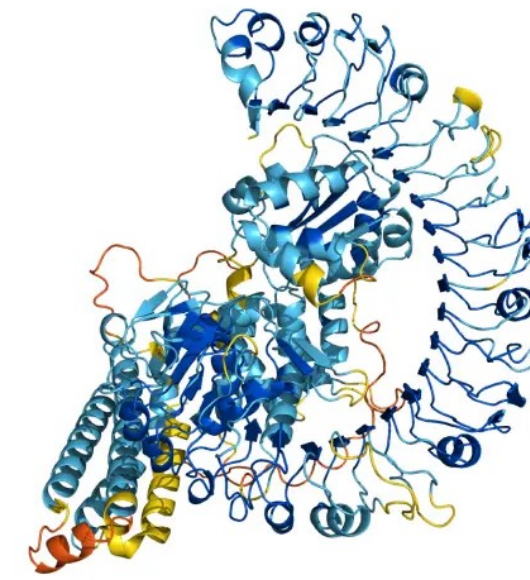
Flow and Diffusion Models generate structured continuous outputs: images, videos, molecular structures, robot actions



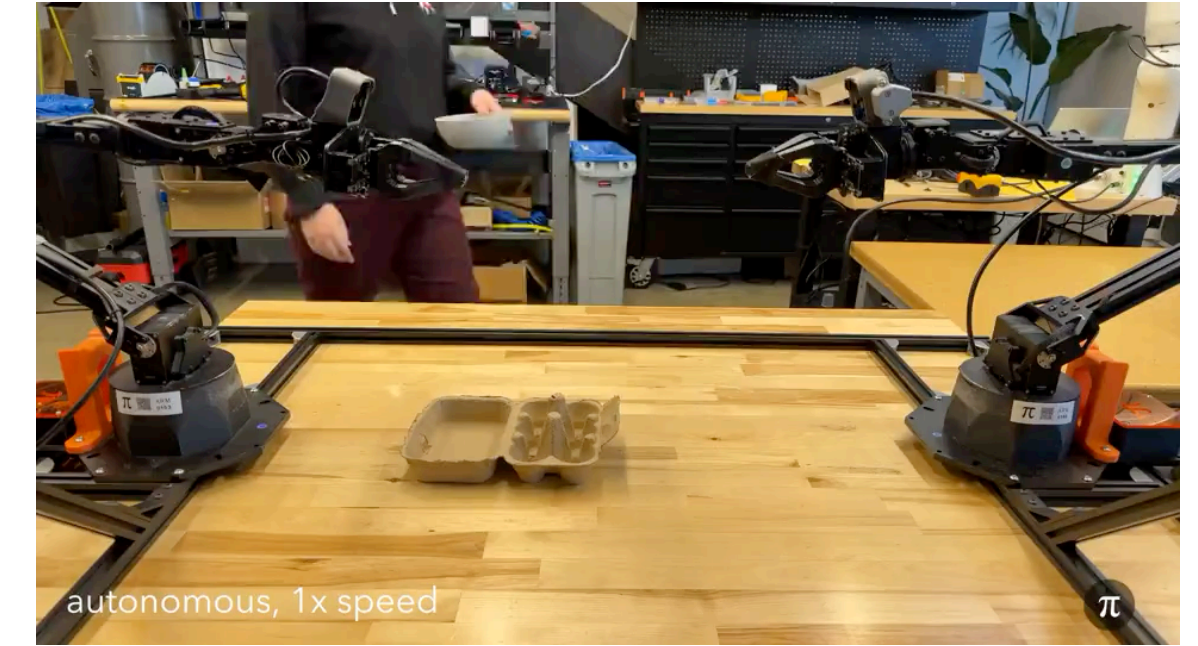
Stable Diffusion,
DALL·E,
Midjourney



OpenAI Sora,
Google Veo,
Meta MovieGen



Google
AlphaFold 3,
Boltz 2, Chai-1



π_0 , Isaac
GROOT

**Most SOTA generative models for images/videos/proteins/
robotics: Diffusion and Flow Models**

Agenda

This lecture: Theory and Practice of Flow / Diffusion Models

This lecture is a light introduction to flow and diffusion models from first principles. By the end:

1. You should be able to explain how to train a generator using ordinary regression
2. How to sample from it using numerical integration
3. How diffusion fits into the picture

Section 1

From Generation to Sampling

What does it mean to “generate” something?

We represent images/videos/robotic actions as vectors

Images:

- Height H and Width W
- 3 color channels (RGB)

$$z \in \mathbb{R}^{H \times W \times 3}$$

Videos:

- T time frames
- Each frame is an image

$$z \in \mathbb{R}^{T \times H \times W \times 3}$$

Action Chunk (in robotics):

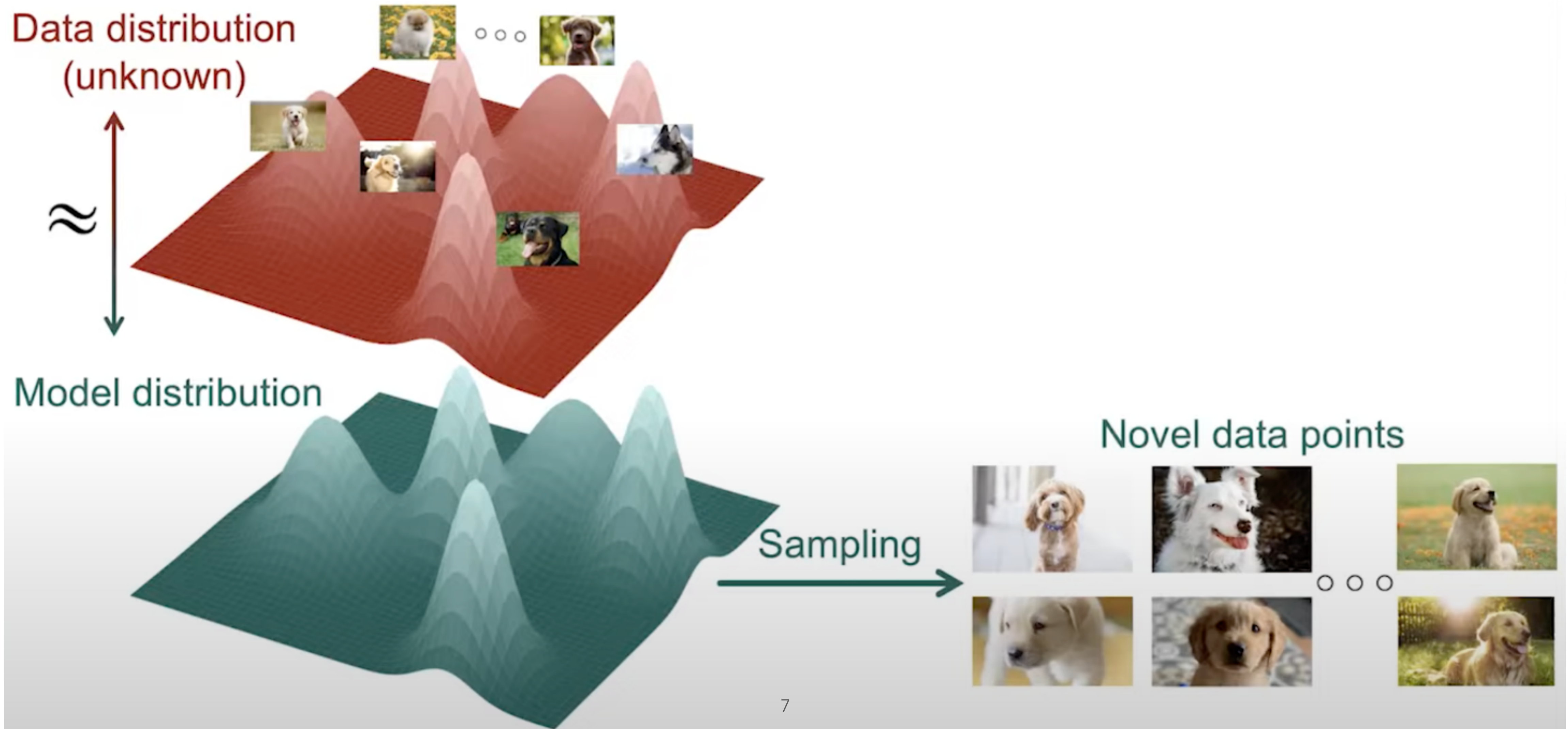
- Horizon H
- Each action has dimension D

$$z \in \mathbb{R}^{H \times D}$$

We represent the objects we want to generate as vectors:

$$z \in \mathbb{R}^d$$

What does it mean to generate something?



Generation as sampling from the data distribution

Data distribution: Distribution of objects that we want to generate:

p_{data}

Probability density:

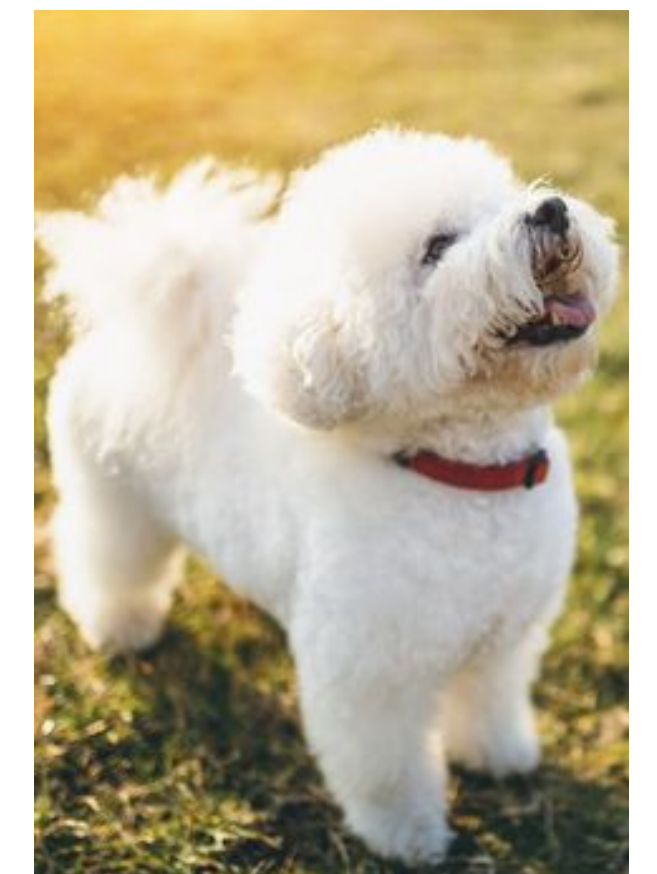
$$p_{\text{data}} : \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$$
$$z \mapsto p_{\text{data}}(z)$$

Generation means sampling the data distribution:

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



$z =$



A Dataset consists of samples from the data distribution

Training requires datasets: To train our algorithms, we need a dataset

Examples:

- Images: available images from the internet
- Videos: YouTube

A dataset consists of a finite number of samples from the data distribution:

$$z_1, z_2, \dots, z_N \sim p_{\text{data}}$$

Conditional Generation allows us to condition on prompts

Data distribution

p_{data}

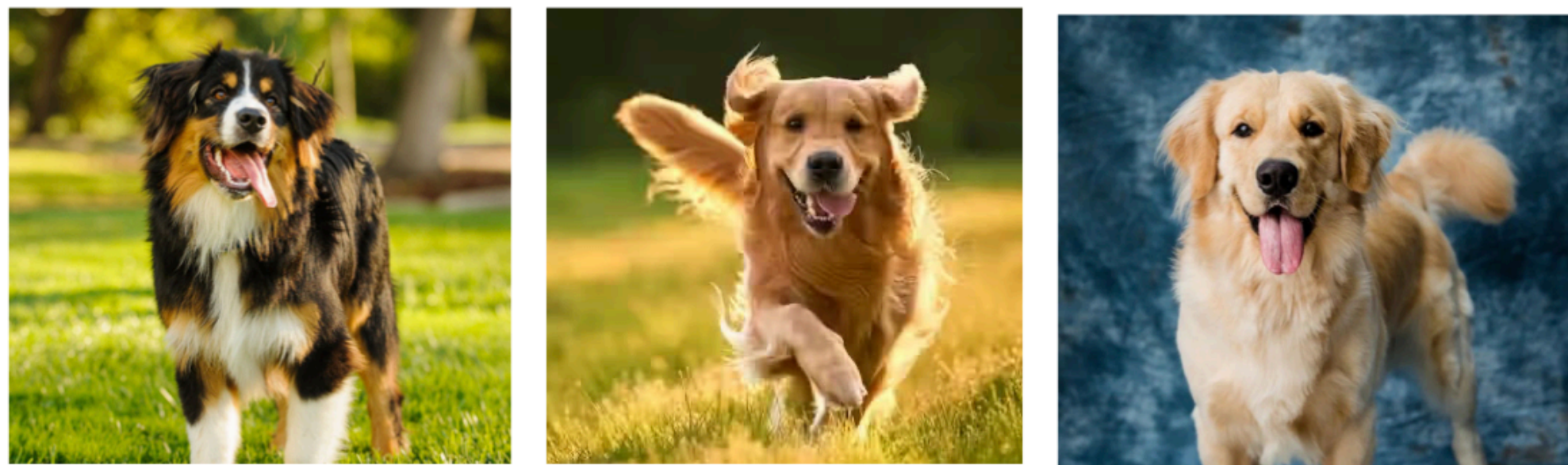
Conditional data distribution

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

Fixed prompt

Conditional variable: y

"Dog"



$y = \text{"Dog"}$

$y = \text{"Cat"}$

$y = \text{"Landscape"}$



Conditional generation means sampling the conditional data distribution:

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

In this lecture, we will build an unconditional generation then briefly add prompts and classifier-free guidance.

Generative Models generate samples from data distribution

Initial distribution:

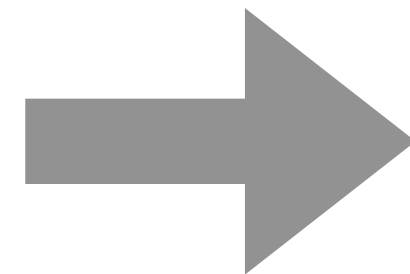
$$p_{\text{init}}$$

Default:

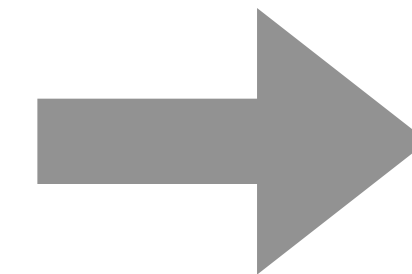
$$p_{\text{init}} = \mathcal{N}(0, I_d)$$

A generative model converts samples from an initial distribution (e.g. Gaussian) into samples from the data distribution:

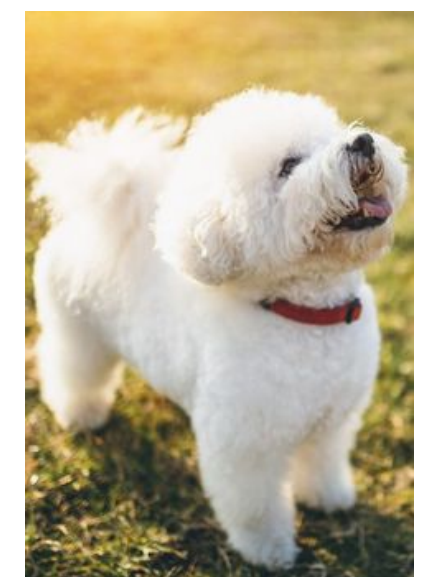
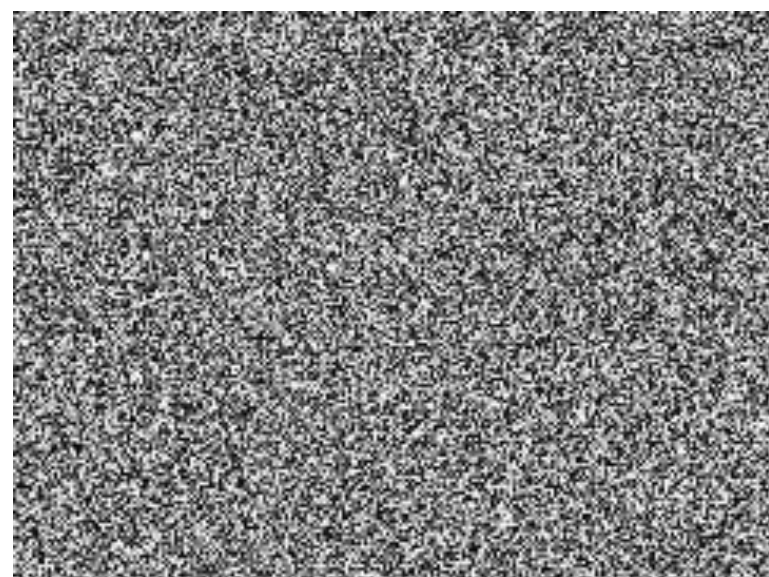
$$X_0 \sim p_{\text{init}}$$



Generative
Model

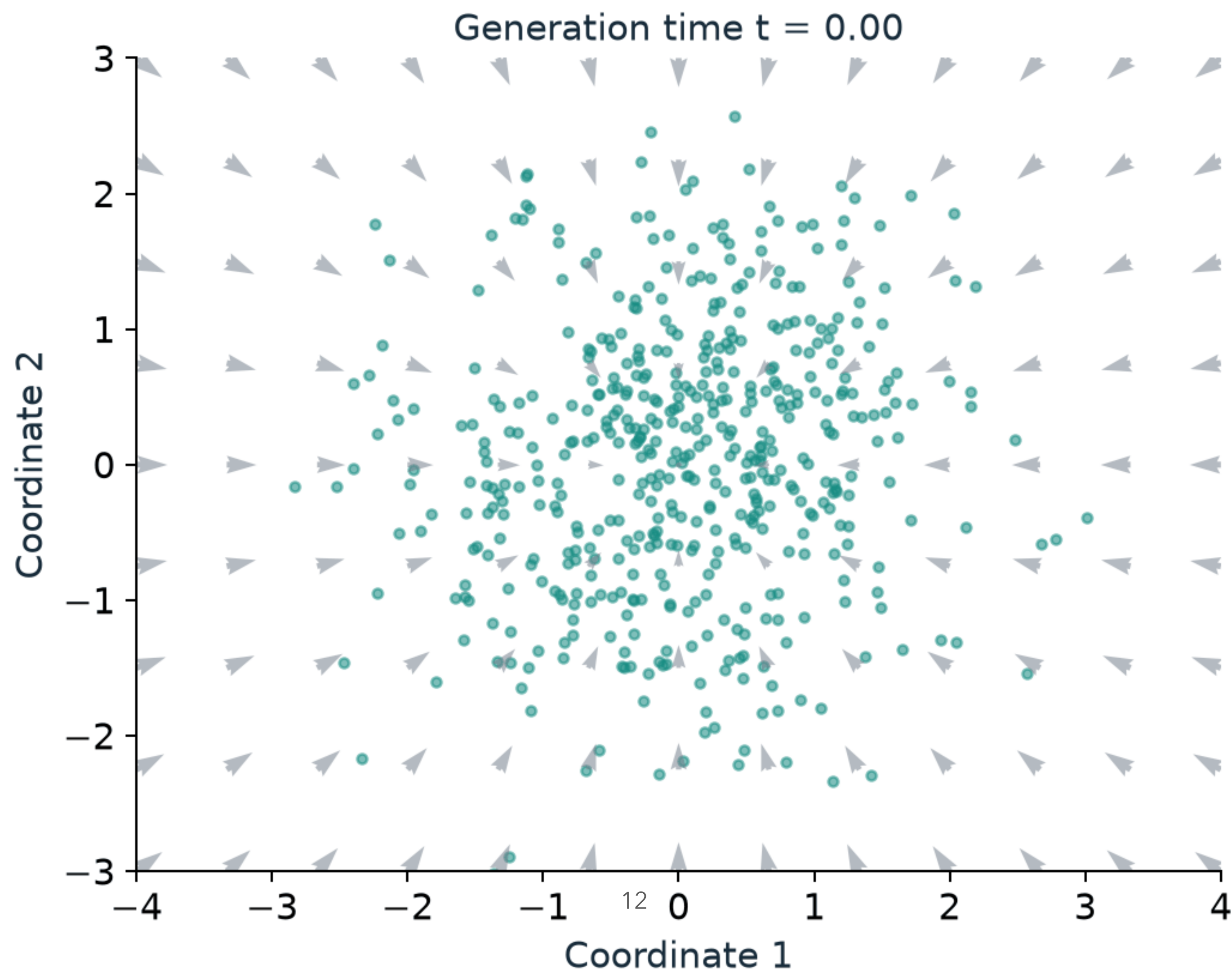


$$z \sim p_{\text{data}}$$



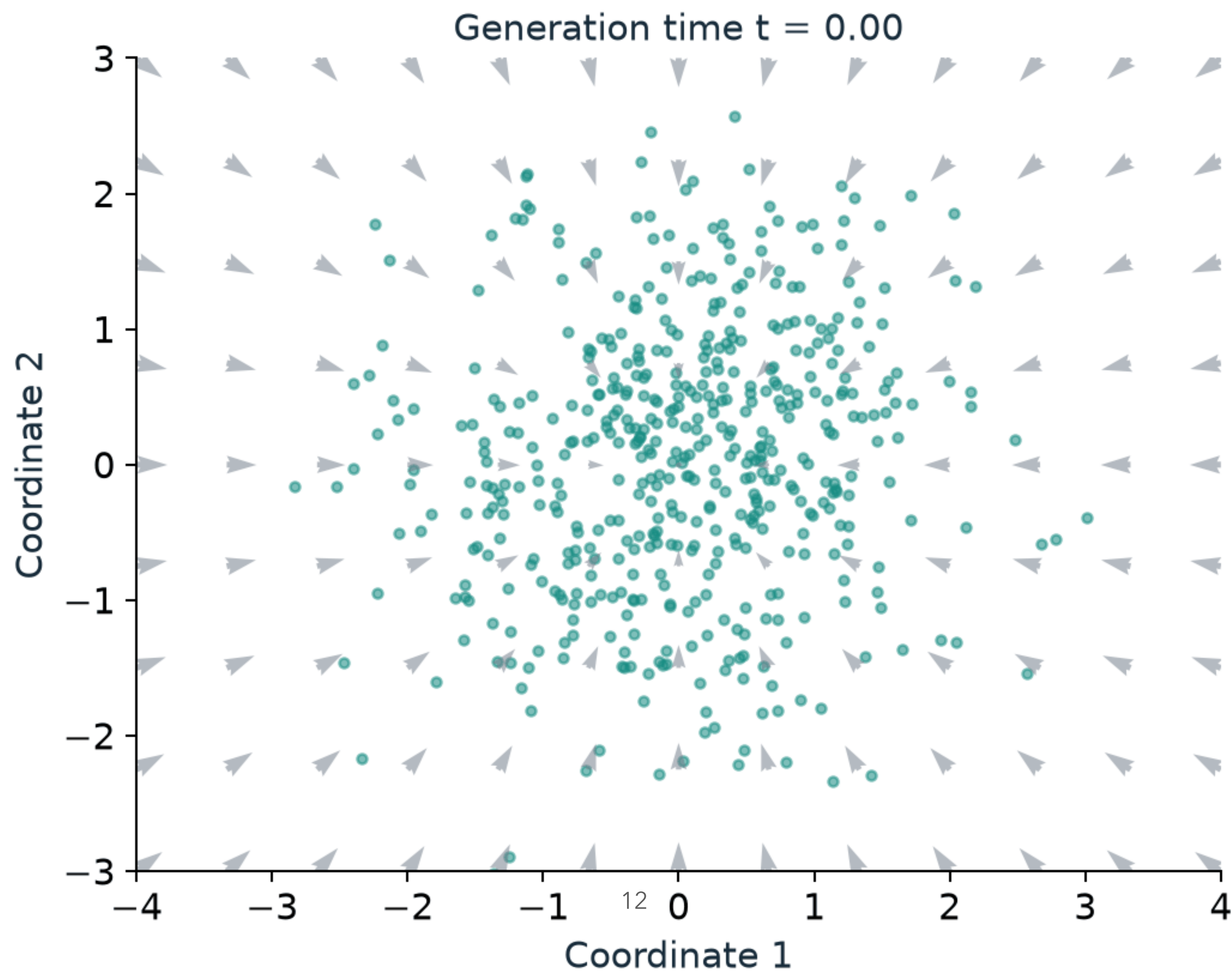
Generative Models generate samples from data distribution

Gaussian noise to a 3-mode distribution



Generative Models generate samples from data distribution

Gaussian noise to a 3-mode distribution



Section 2

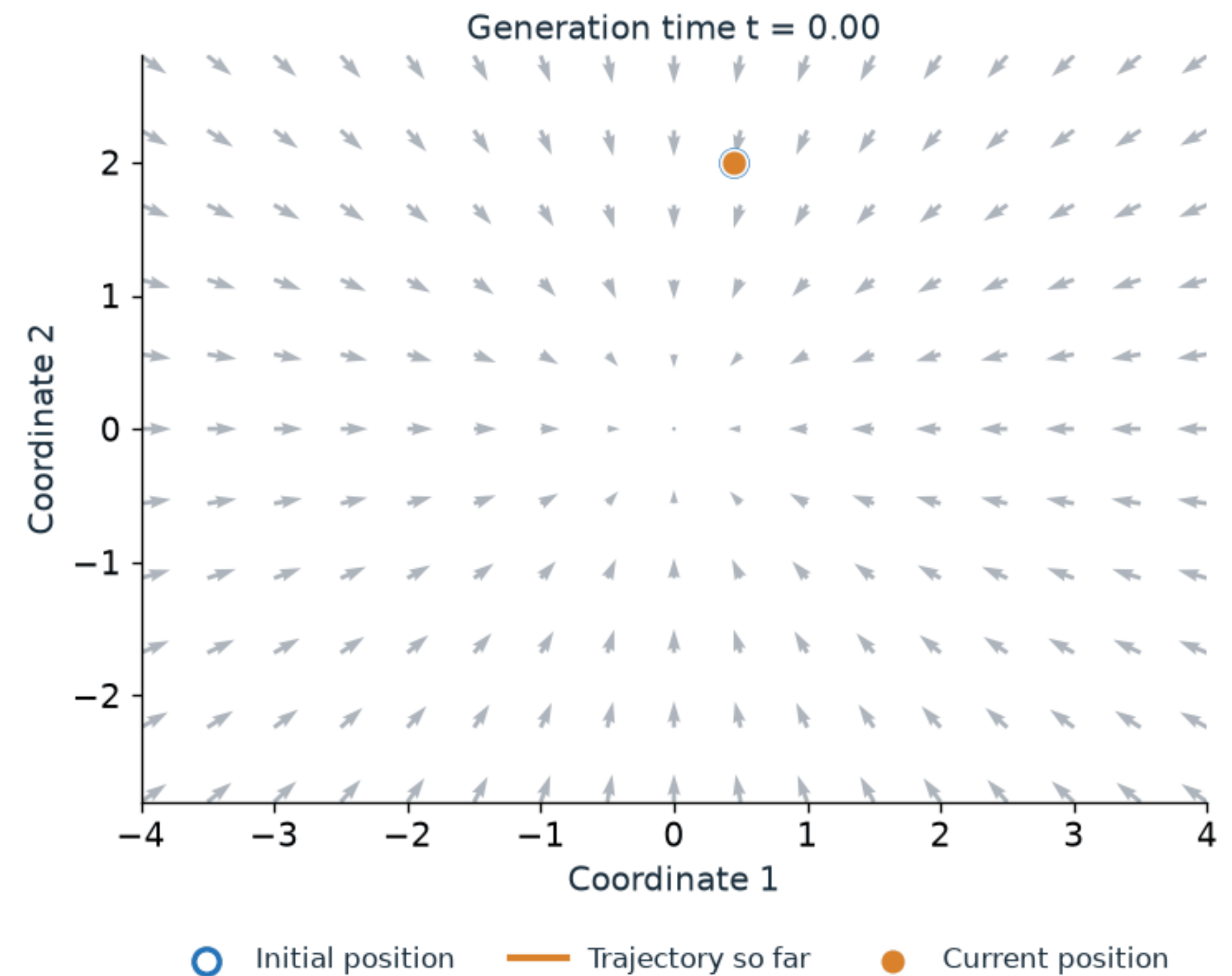
Flow and Diffusion Models

Differential equations and how we can build generative models from them.

Ordinary Differential Equations (ODEs)

An ODE trajectory follows a velocity field

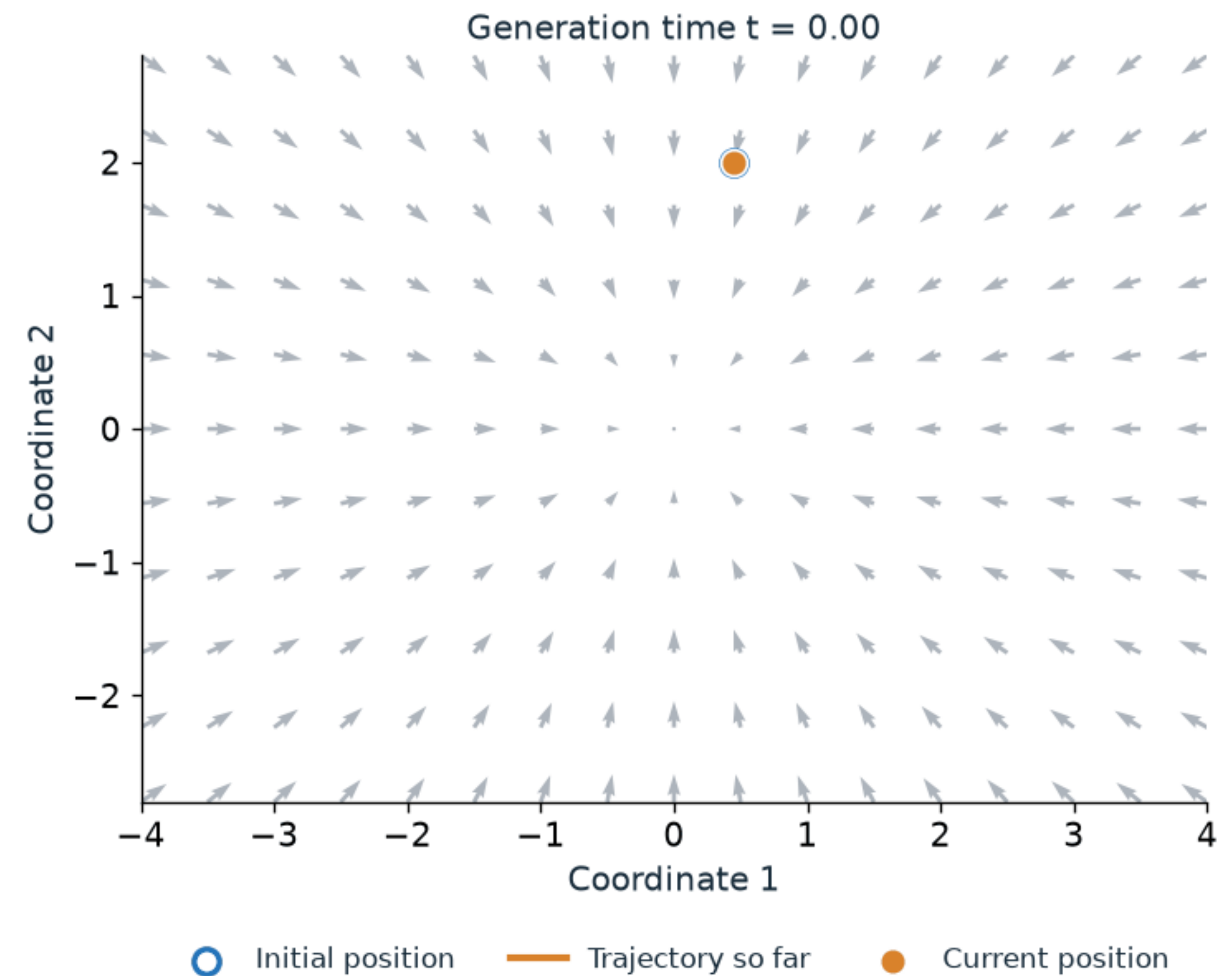
ODE: $\frac{dX_t}{dt} = u_t(X_t)$



Ordinary Differential Equations (ODEs)

An ODE trajectory follows a velocity field

ODE: $\frac{dX_t}{dt} = u_t(X_t)$



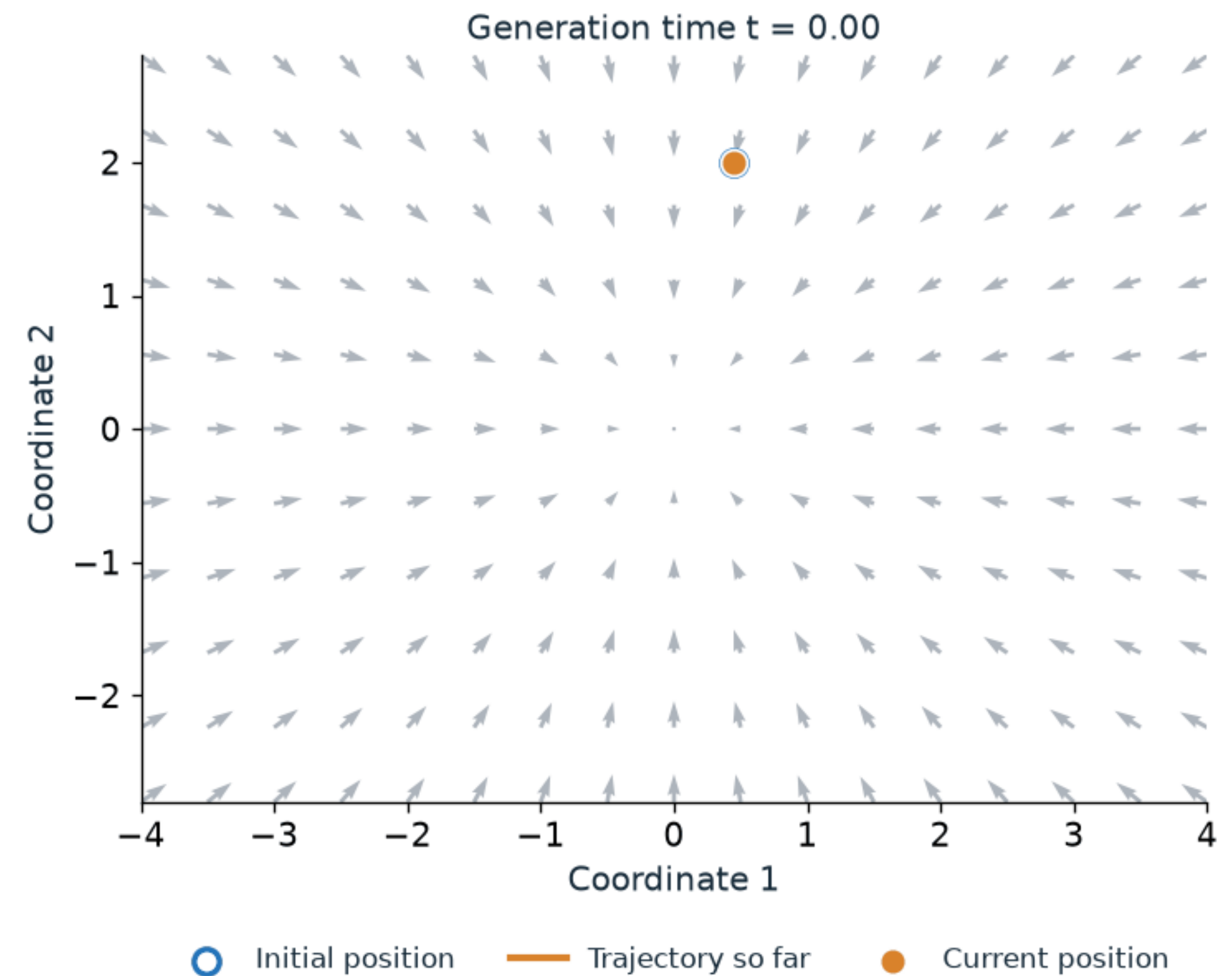
Ordinary Differential Equations (ODEs)

An ODE trajectory follows a velocity field

ODE: $\frac{dX_t}{dt} = u_t(X_t)$

Example: Linear ODE

$$\dot{x} = u_t(x) = -ax \quad (a > 0)$$

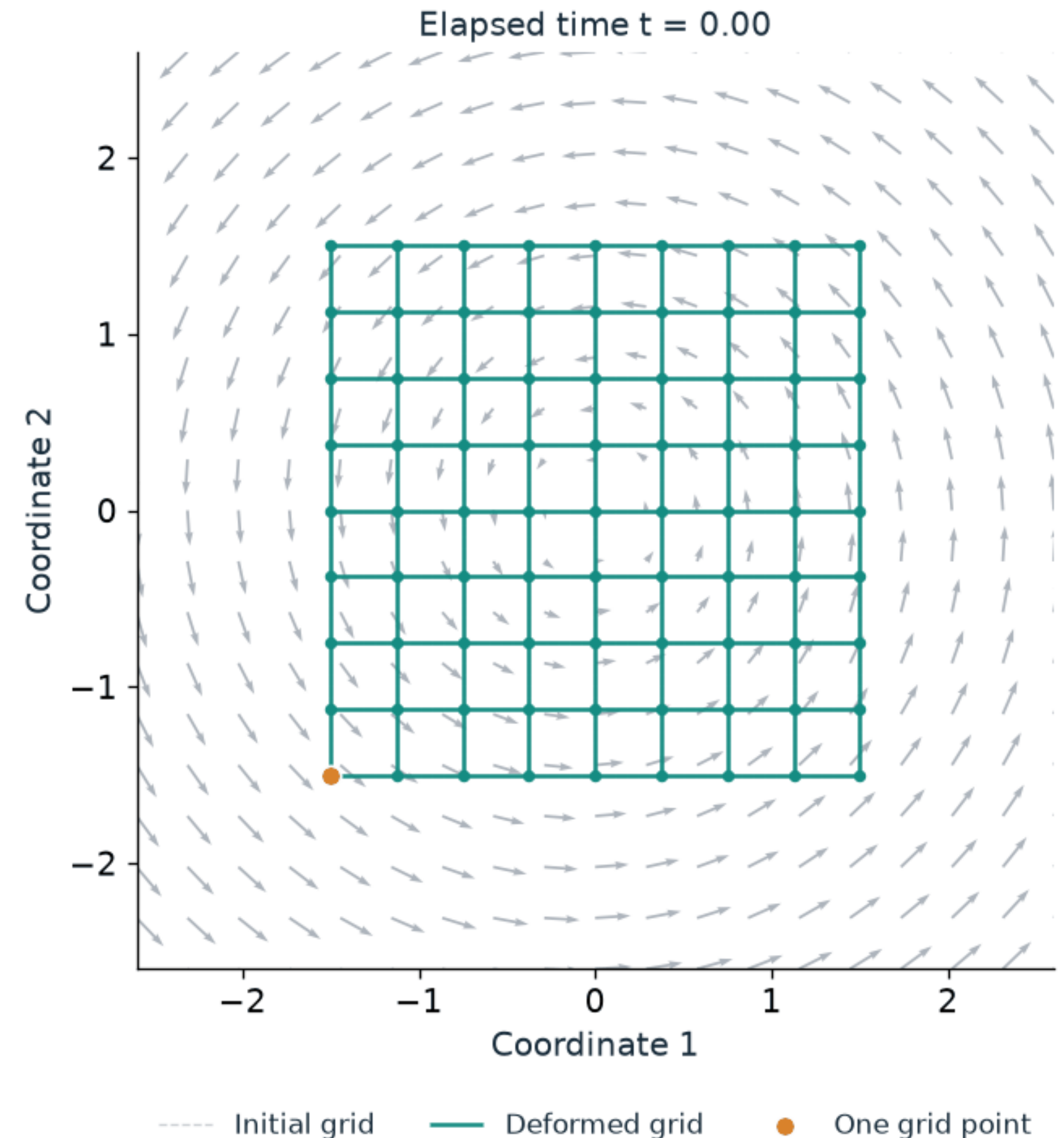


Ordinary Differential Equations (ODEs)

A Flow: solution to an ODE
for all initial conditions

ODE: $\frac{dX_t}{dt} = u_t(X_t)$

Flow: $\Phi_t : x_0 \mapsto \Phi_t(x_0)$

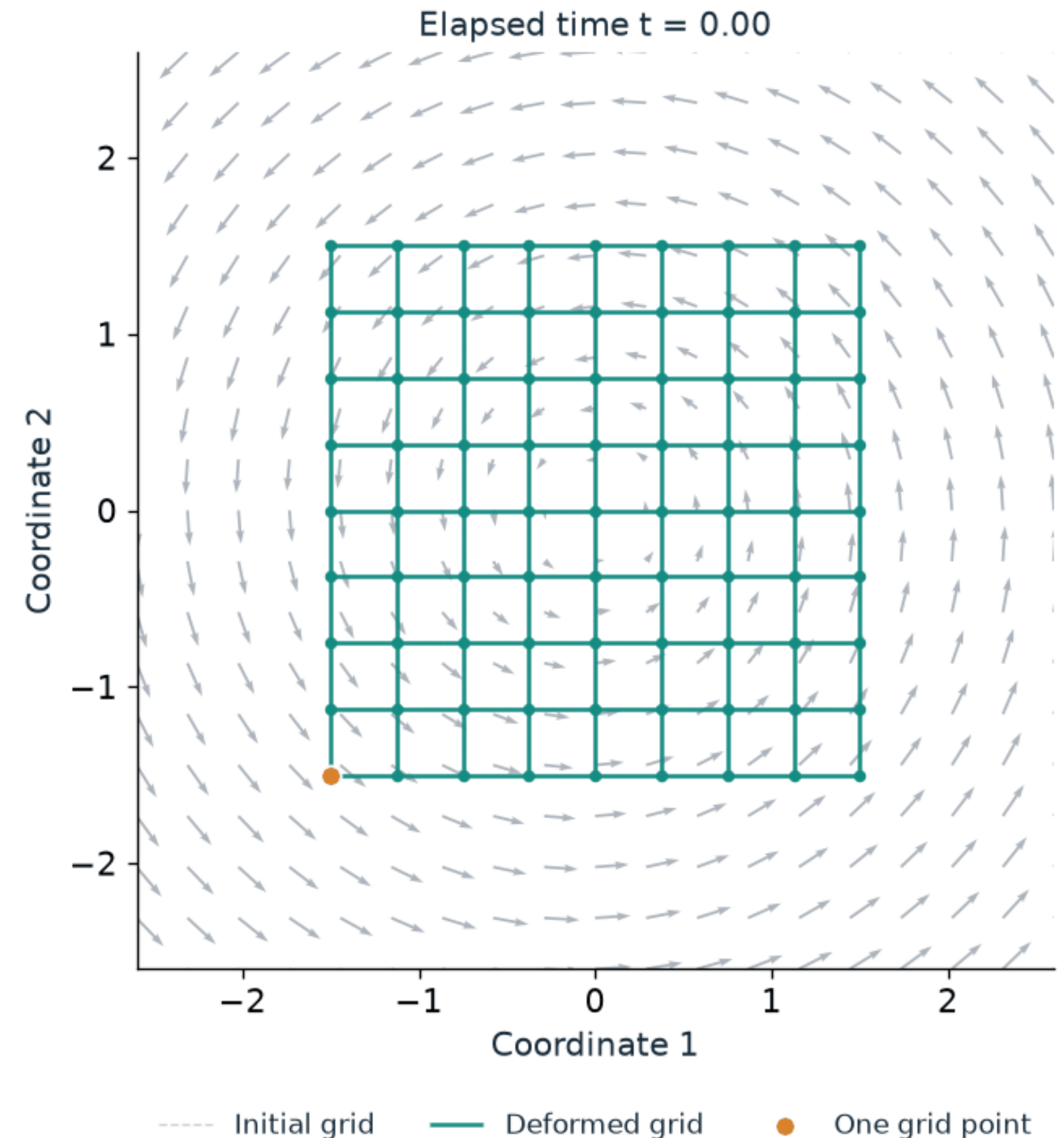


Ordinary Differential Equations (ODEs)

A Flow: solution to an ODE
for all initial conditions

ODE: $\frac{dX_t}{dt} = u_t(X_t)$

Flow: $\Phi_t : x_0 \mapsto \Phi_t(x_0)$



Existence and Uniqueness Theorem for ODEs

Theorem (Picard-Lindelöf theorem): If the vector field $u_t(x)$ is continuously differentiable with bounded derivatives, then a unique solution to the ODE

$$X_0 = x_0, \quad \frac{dX_t}{dt} = u_t(X_t)$$

exists. Therefore, a flow map exists. Generally, this is true if the vector field is [Lipschitz](#).

In cases of interest for machine learning, **unique solutions for ODE/flows exist.**

Numerical ODE simulation - Euler method

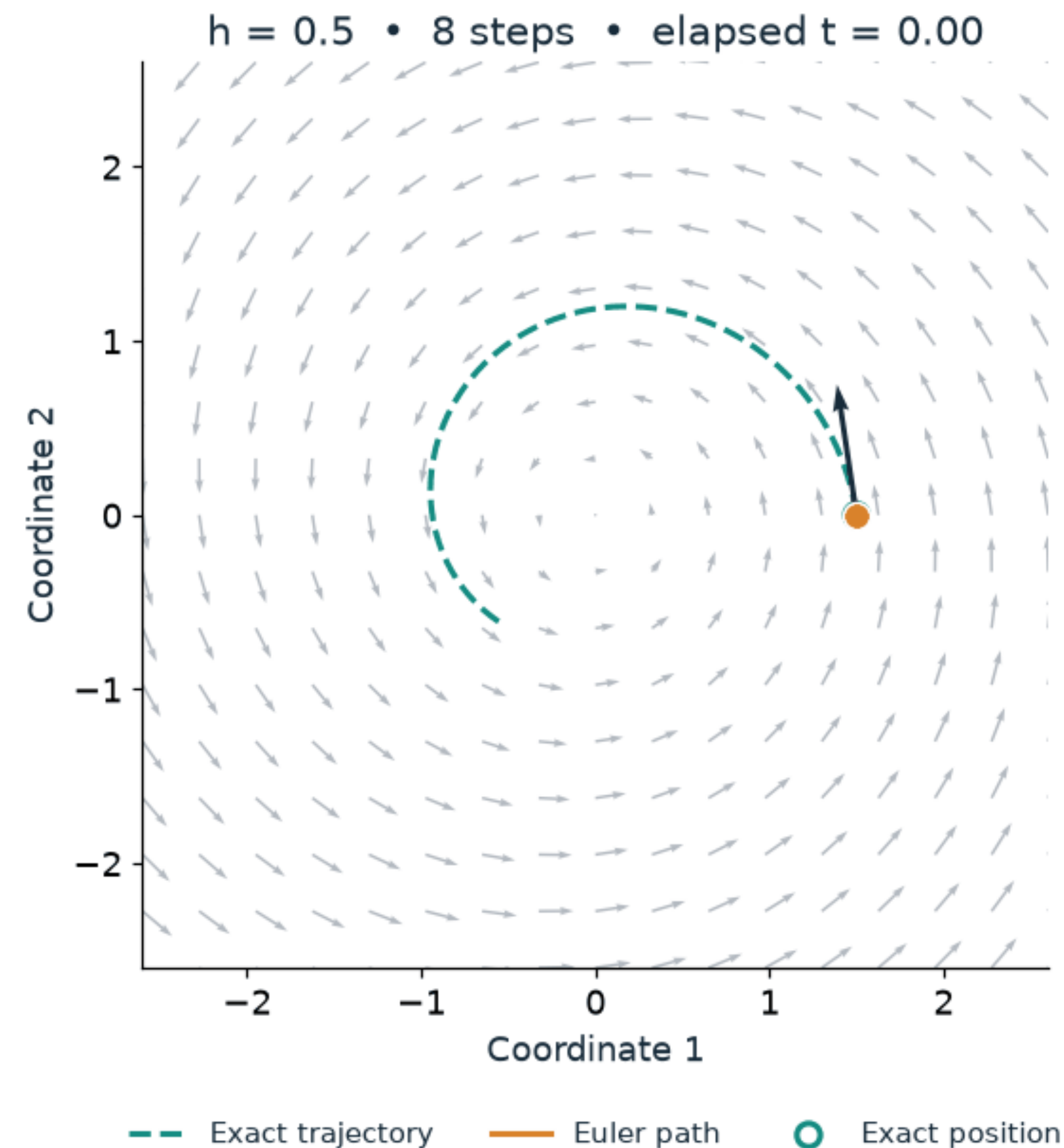
Algorithm 1: Simulating an ODE with the Euler method

Requires: Vector field u_t , initial condition x_0 , number of steps n

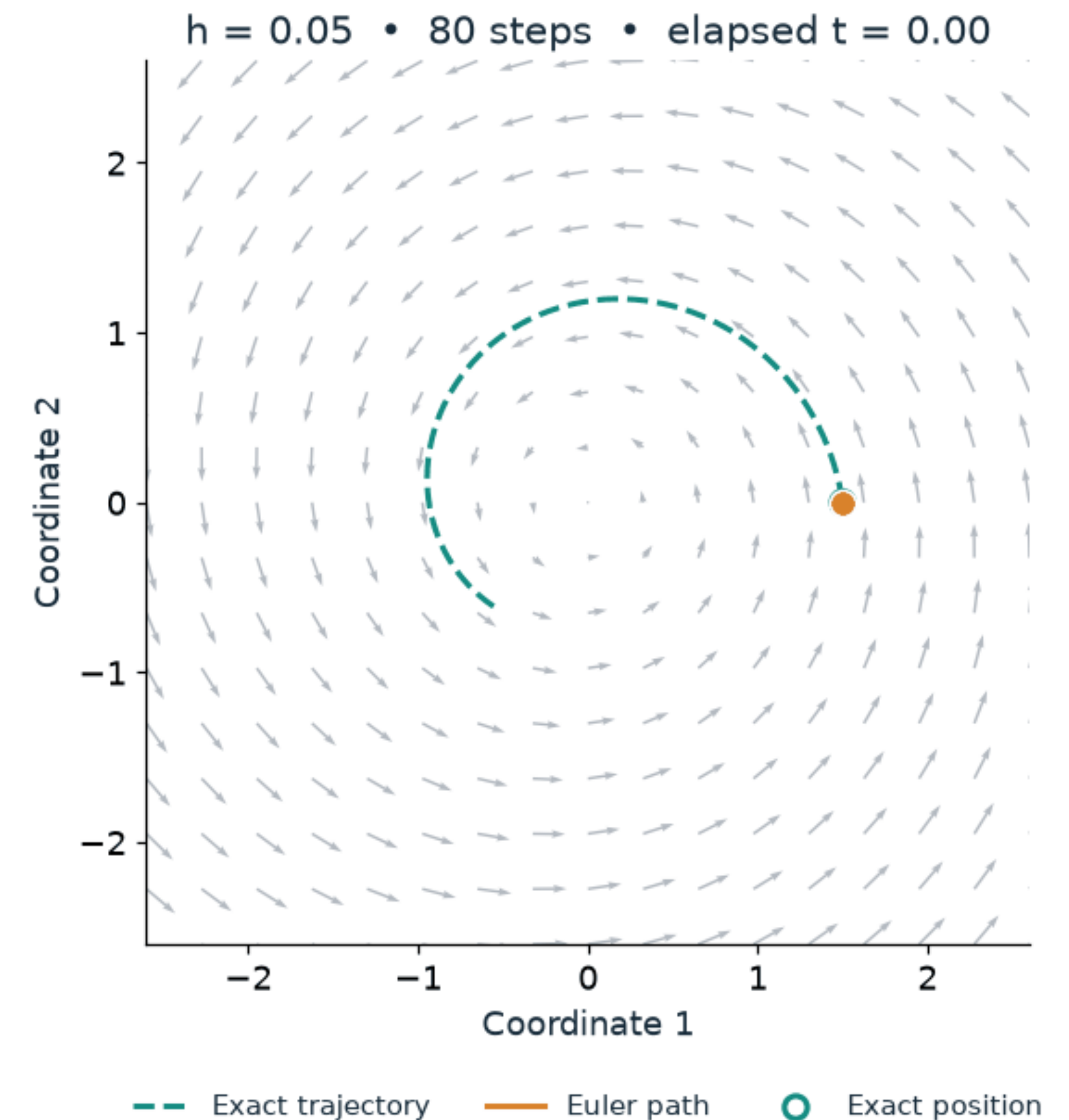
1. Set $t = 0$
2. Set step size $h = \frac{1}{n}$
3. Set $X_0 = x_0$
4. **for** $i = 1, \dots, n$ **do**
5. $X_{t+h} = X_t + hu_t(X_t)$
6. Update $t \rightarrow t + h$
7. **end for**
8. **return** $X_0, X_h, X_{2h}, \dots, X_1$

Small steps into the direction of the vector field

Numerical ODE simulation - Euler method

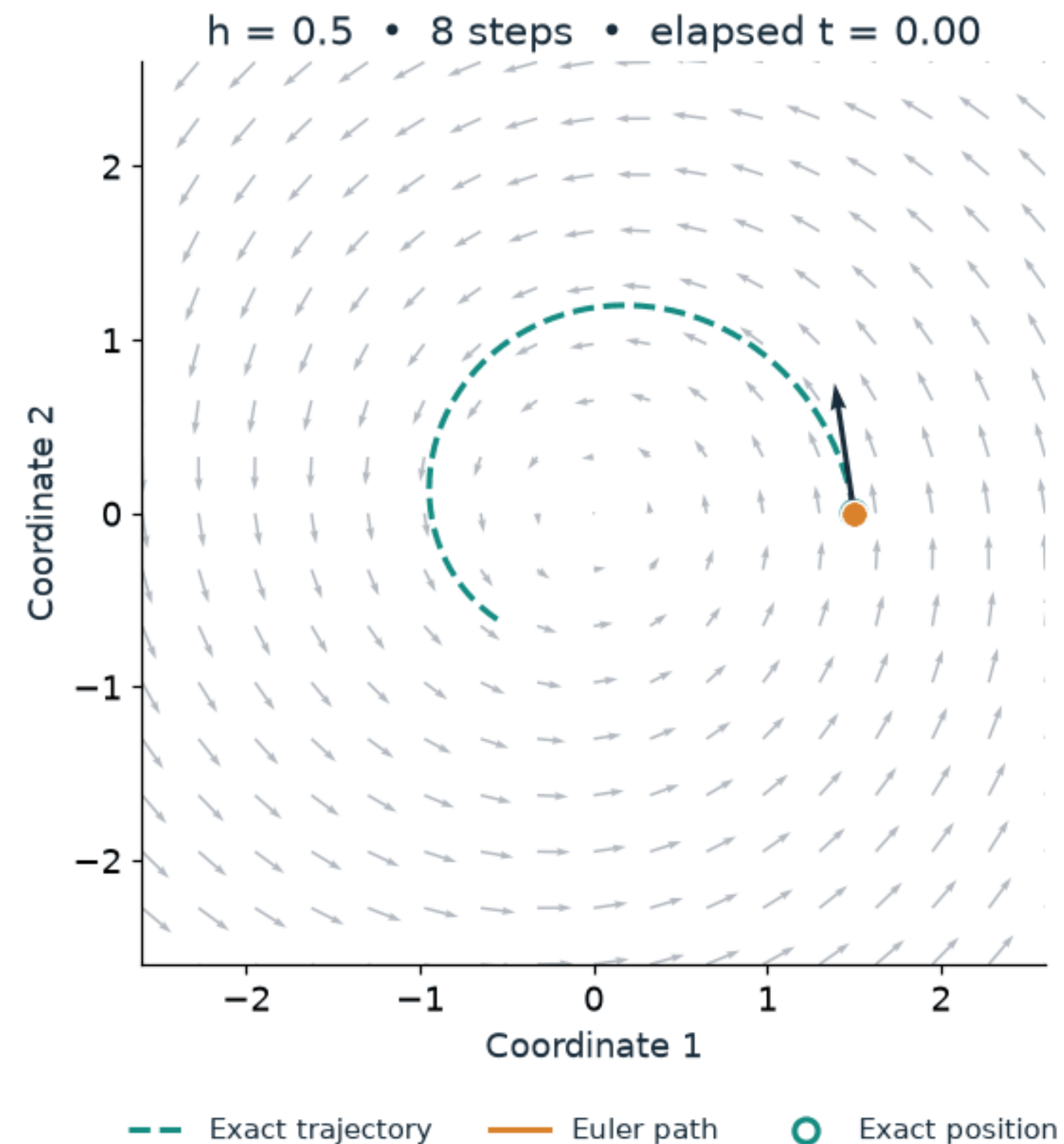


Large step size: **more efficient** but **higher error**

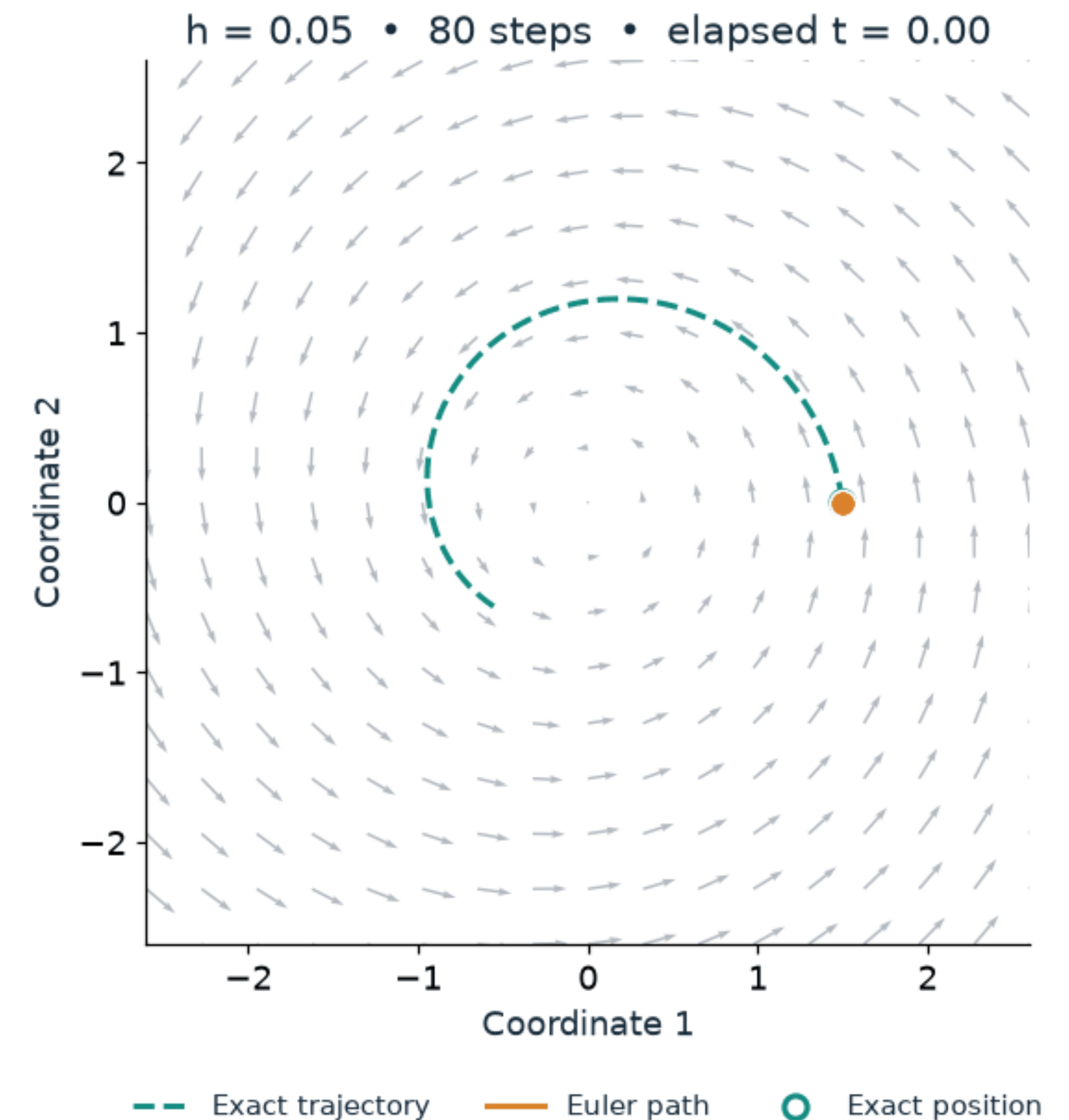


Small step size: **lower error** but **less efficient**

Numerical ODE simulation - Euler method

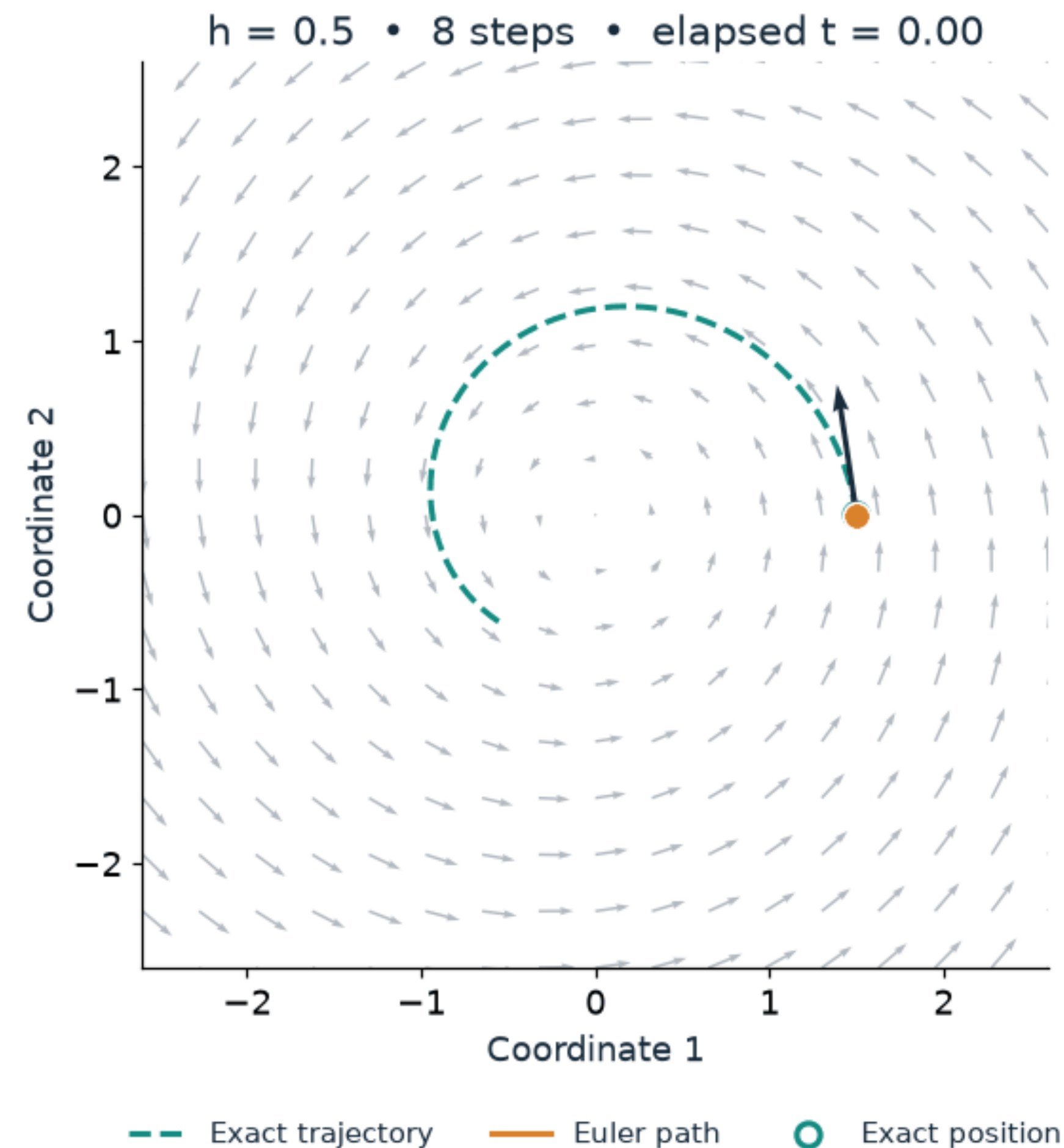


Large step size: **more efficient** but **higher error**

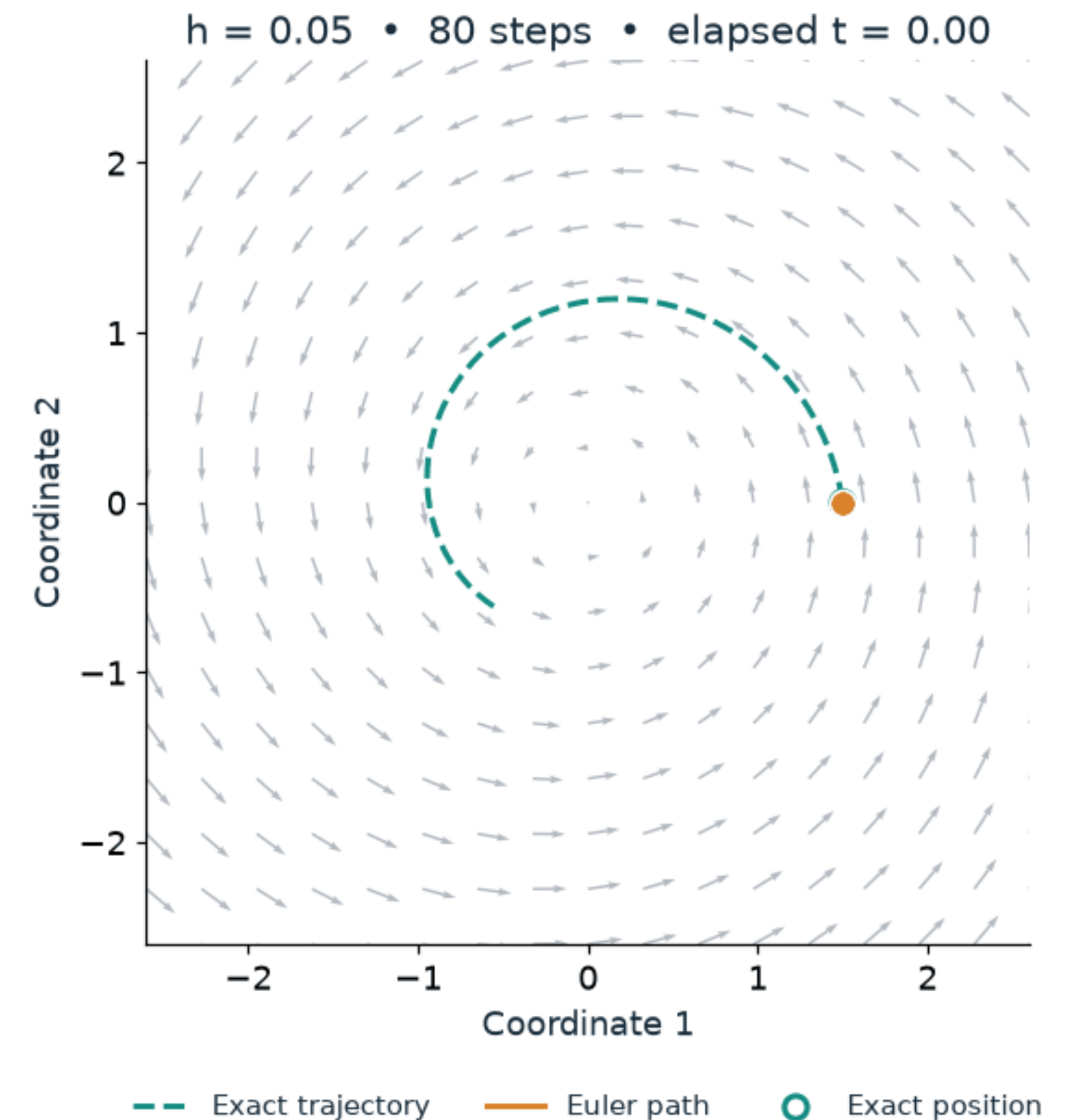


Small step size: **lower error** but **less efficient**

Numerical ODE simulation - Euler method



Large step size: **more efficient** but **higher error**



Small step size: **lower error** but **less efficient**

How to generate objects with a Flow Model

Algorithm 2: Sampling from a Flow Model with Euler method

Requires: 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 $X_0 \sim p_{\text{init}} = \mathcal{N}(0, I_d)$ Random initialization!
4. **for** $i = 1, \dots, n$ **do**
5. $X_{t+h} = X_t + hu_t^\theta(X_t)$
6. Update $t \rightarrow t + h$
7. **end for**
8. **return** X_1

How to generate objects with a **Flow** Model

Example of videos generated via Flow Model sampling



How to generate objects with a **Flow** Model

Example of videos generated via Flow Model sampling



How to generate objects with a **Flow** Model

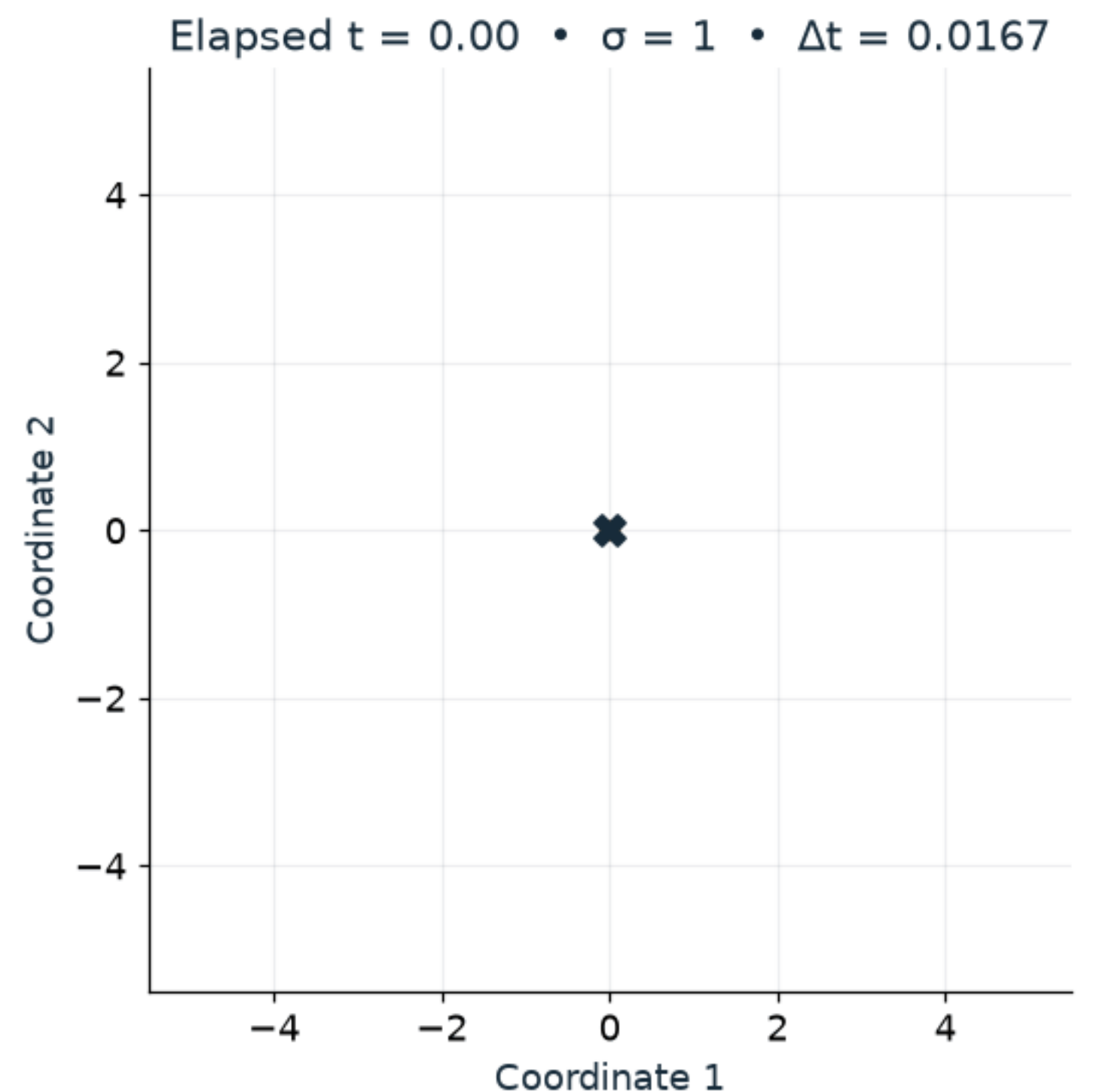
Example of videos generated via Flow Model sampling



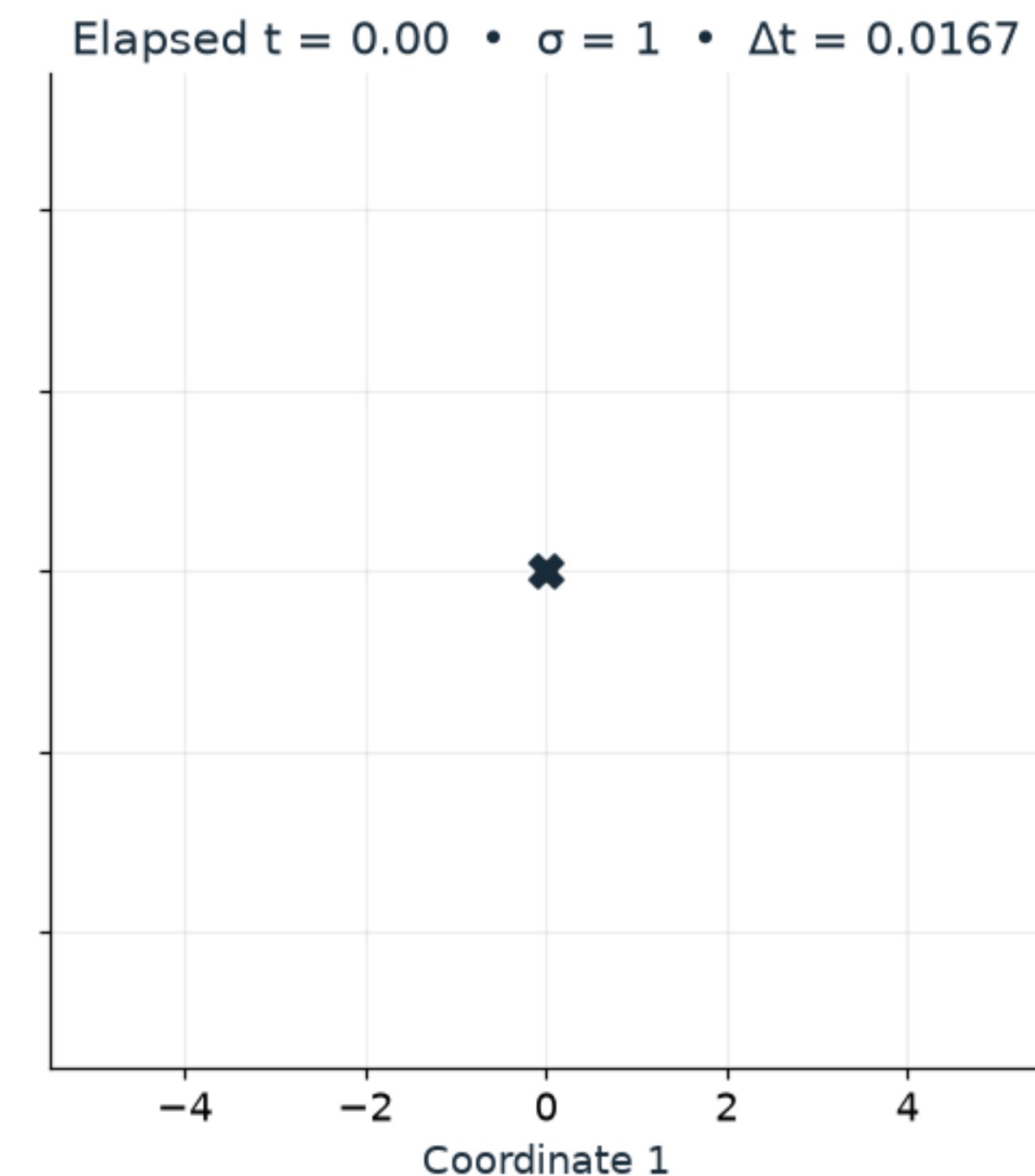
Stochastic Differential Equations (SDEs)

Stochastic Process: Random Trajectories - Brownian Motion

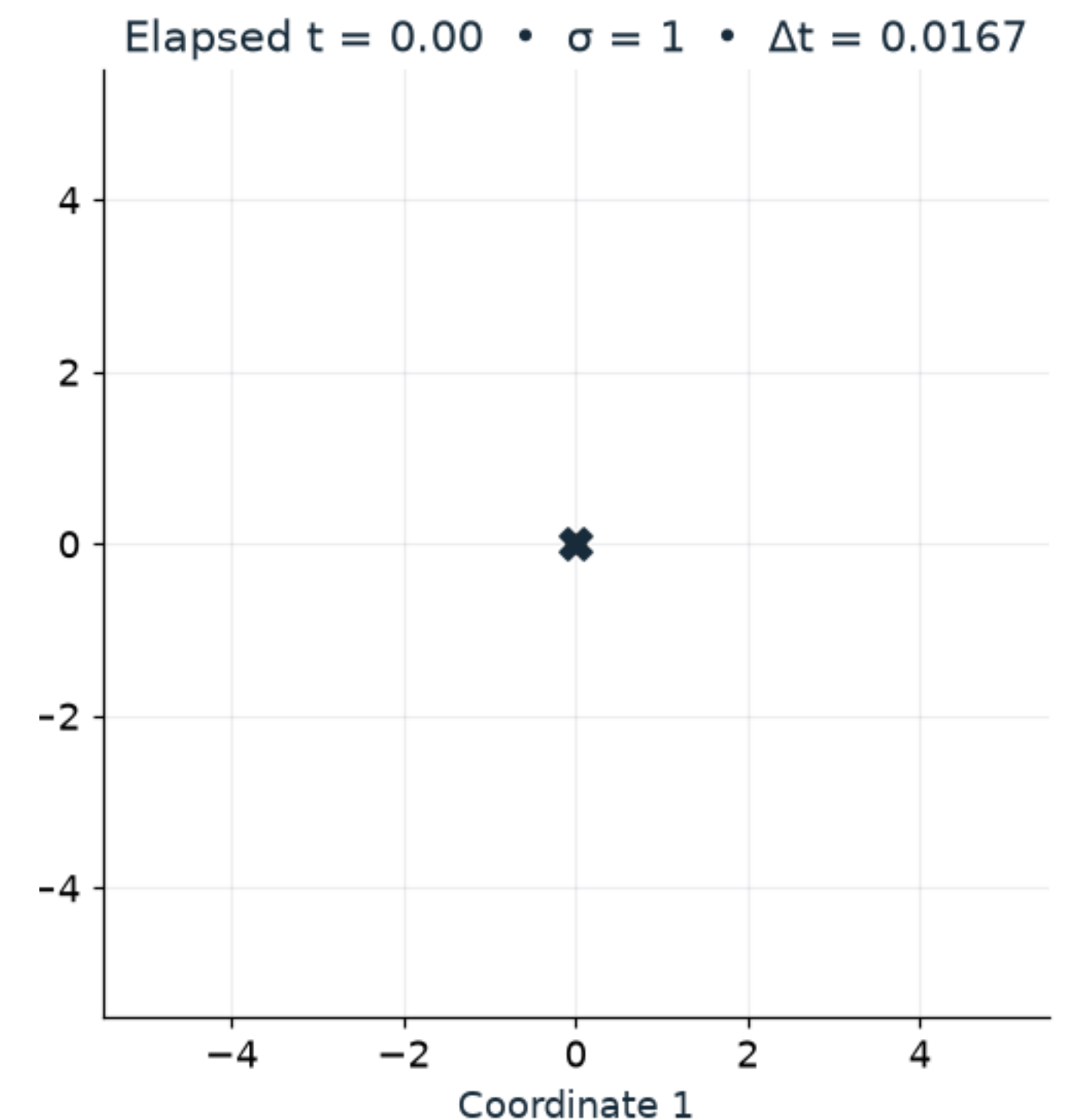
$$dX_t = u_t(X_t)dt + \sigma_t dW_t$$



✕ Same initial point — Random path ● Current position



✕ Same initial point — Random path ● Current position



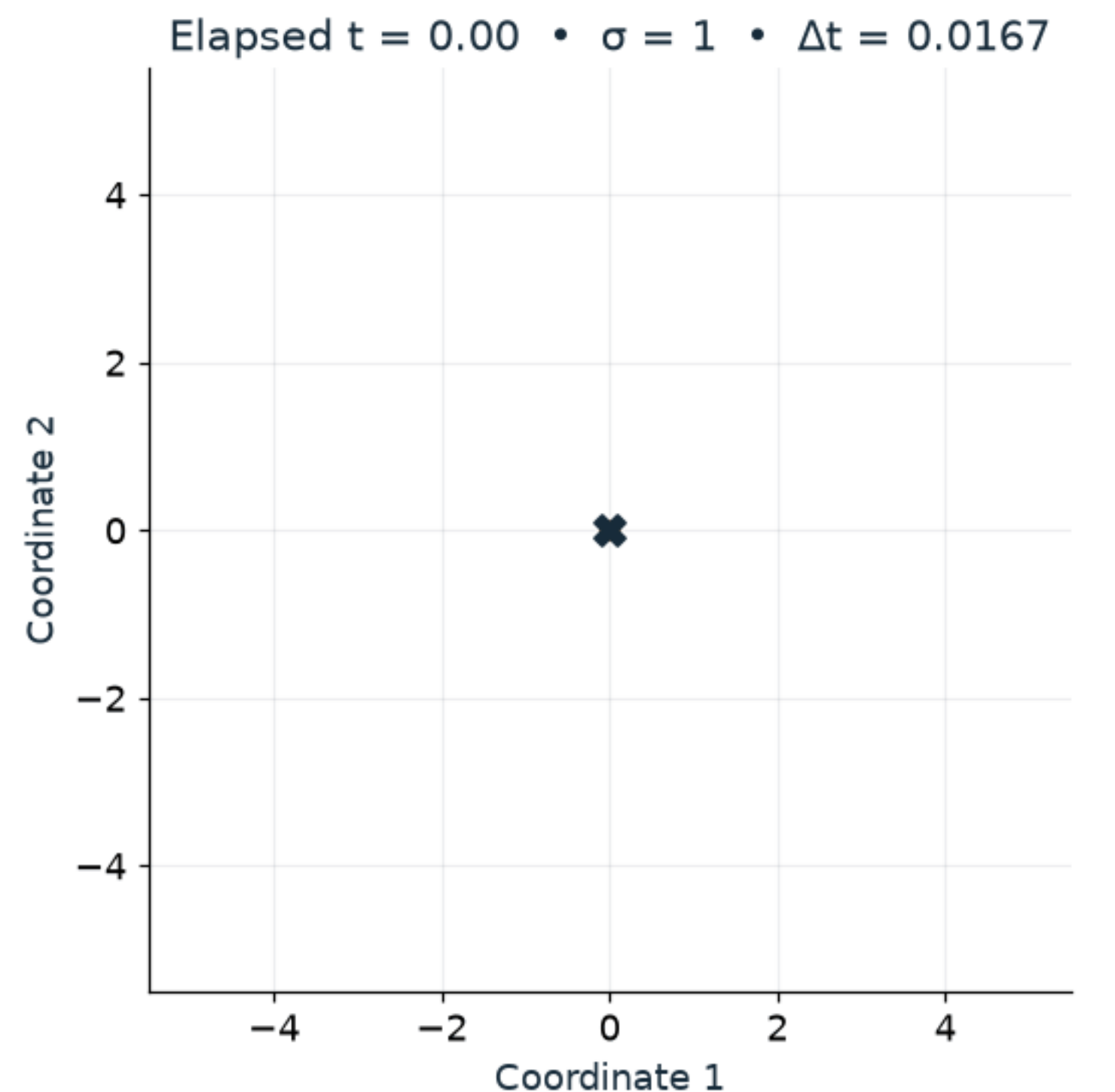
✕ Same initial point — Random path ● Current position

With the same initial state, fresh random increments produce different trajectories

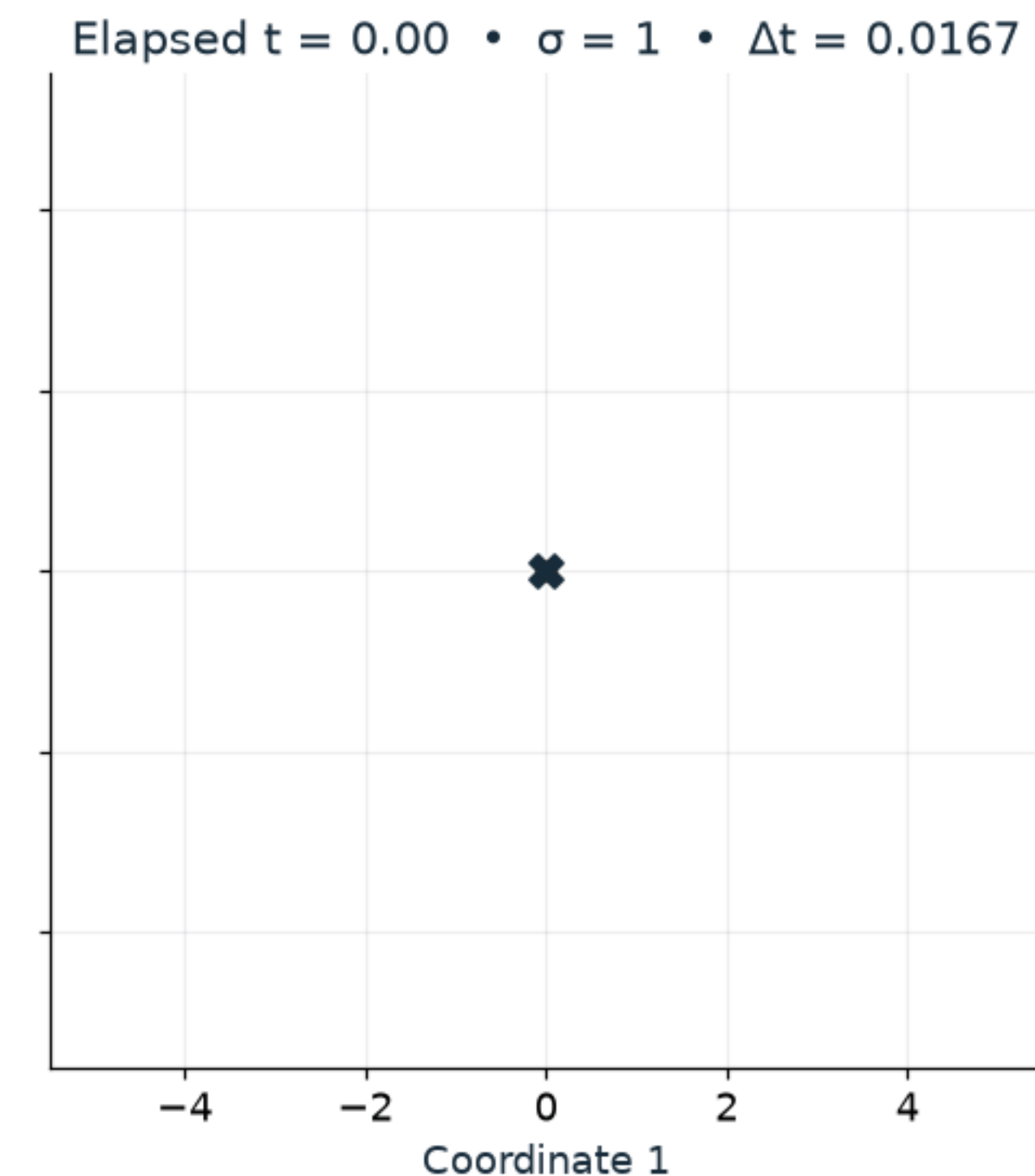
Stochastic Differential Equations (SDEs)

Stochastic Process: Random Trajectories - Brownian Motion

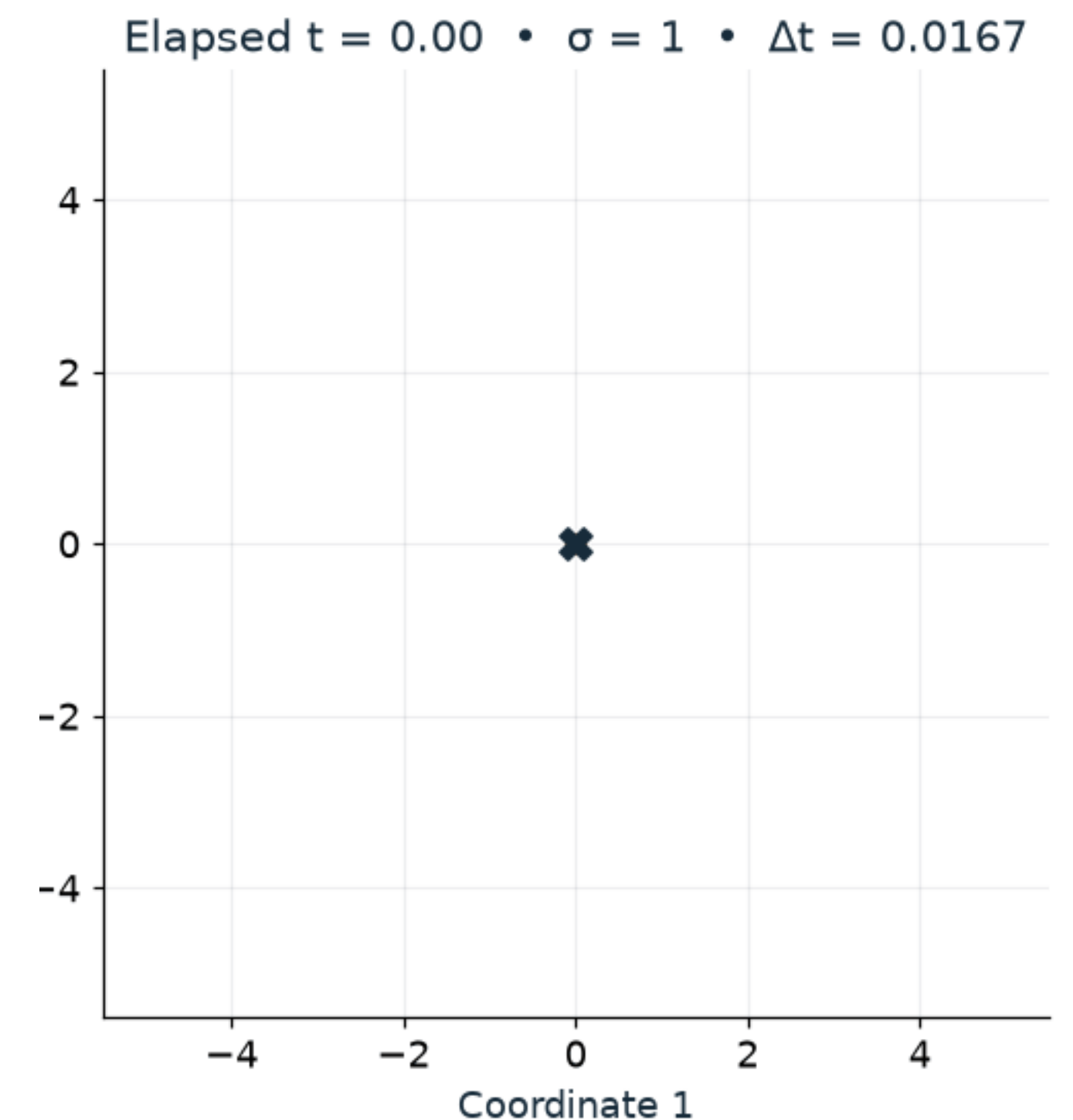
$$dX_t = u_t(X_t)dt + \sigma_t dW_t$$



✕ Same initial point — Random path ● Current position



✕ Same initial point — Random path ● Current position



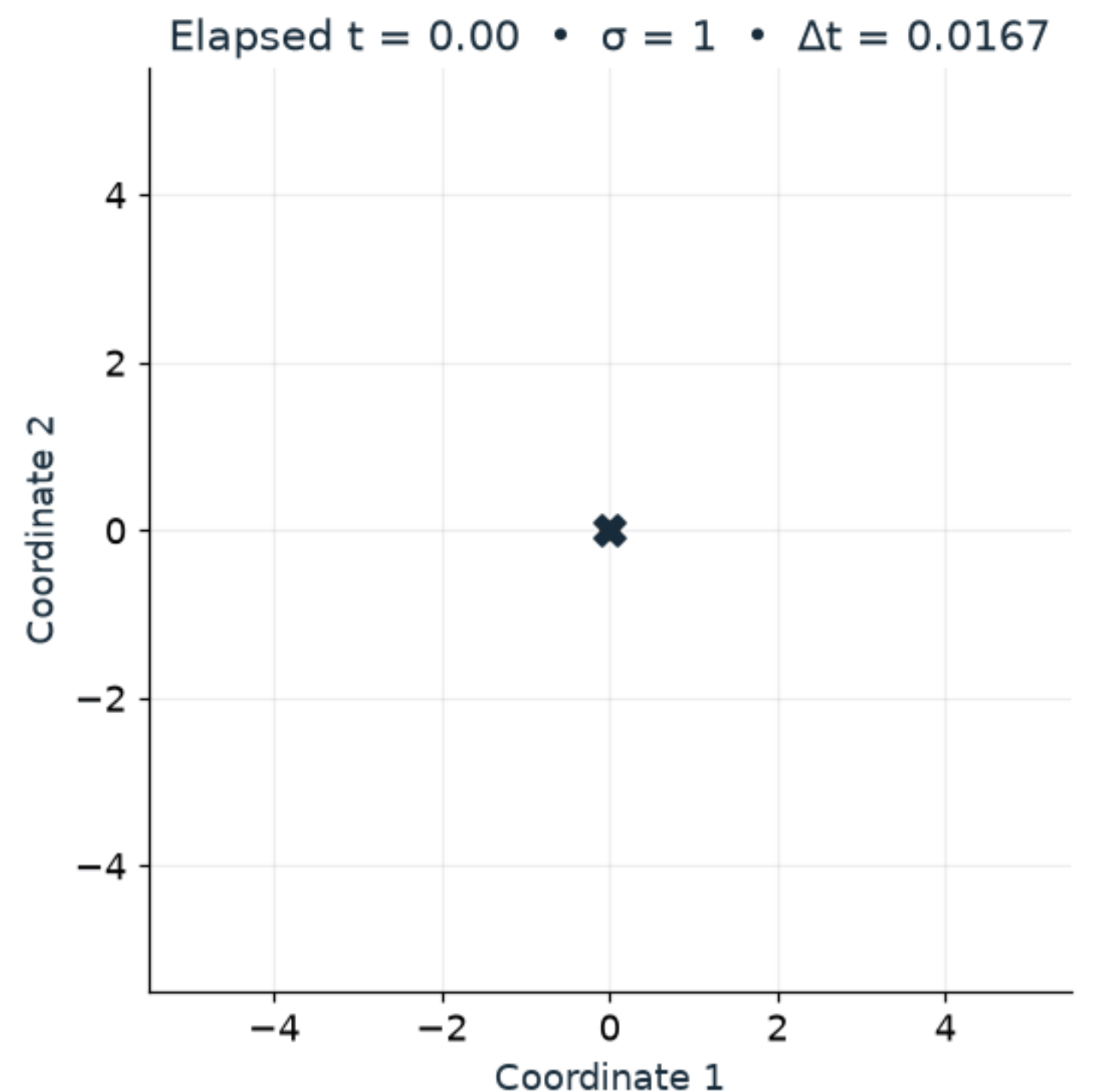
✕ Same initial point — Random path ● Current position

With the same initial state, fresh random increments produce different trajectories

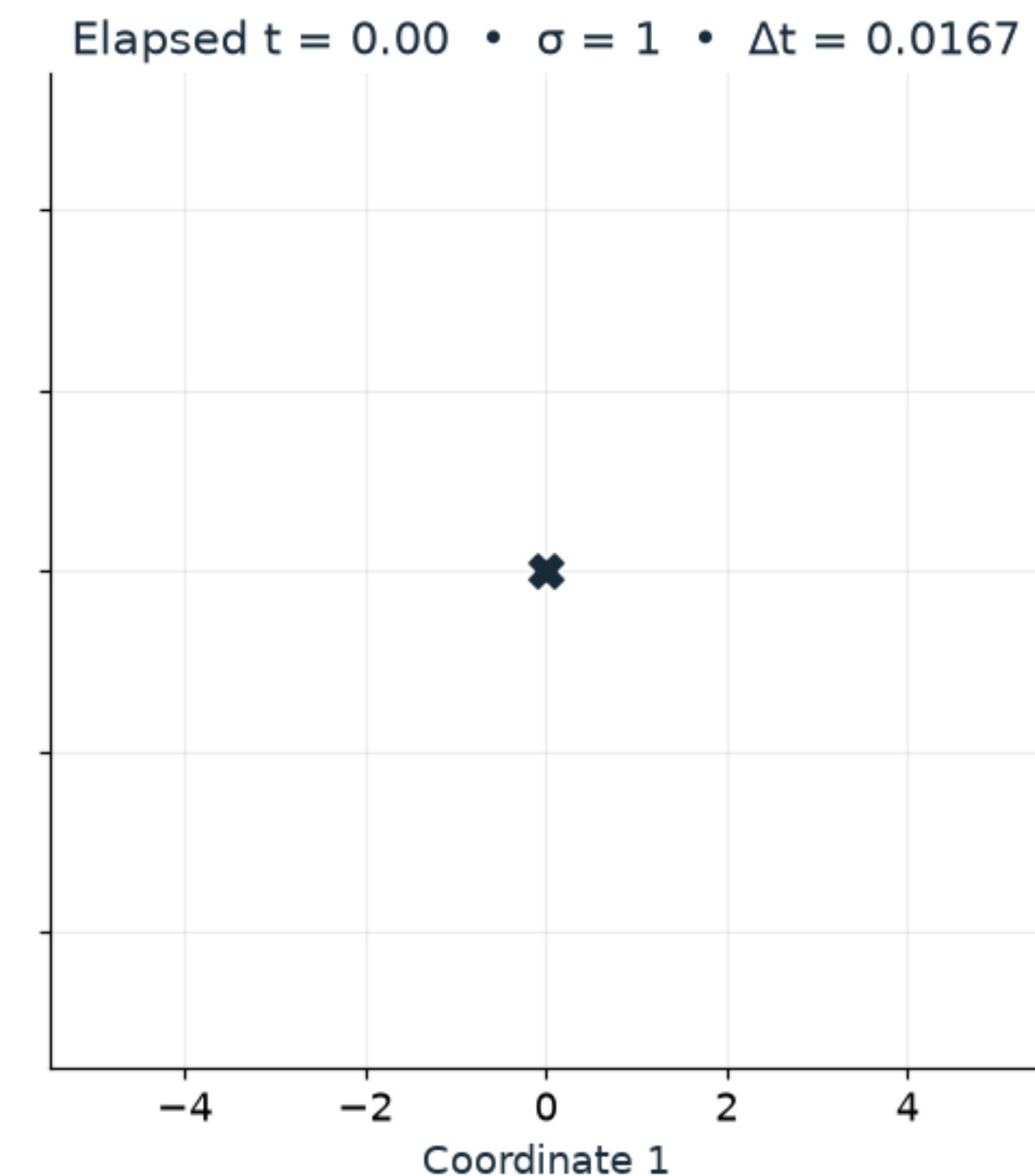
Stochastic Differential Equations (SDEs)

Stochastic Process: Random Trajectories - Brownian Motion

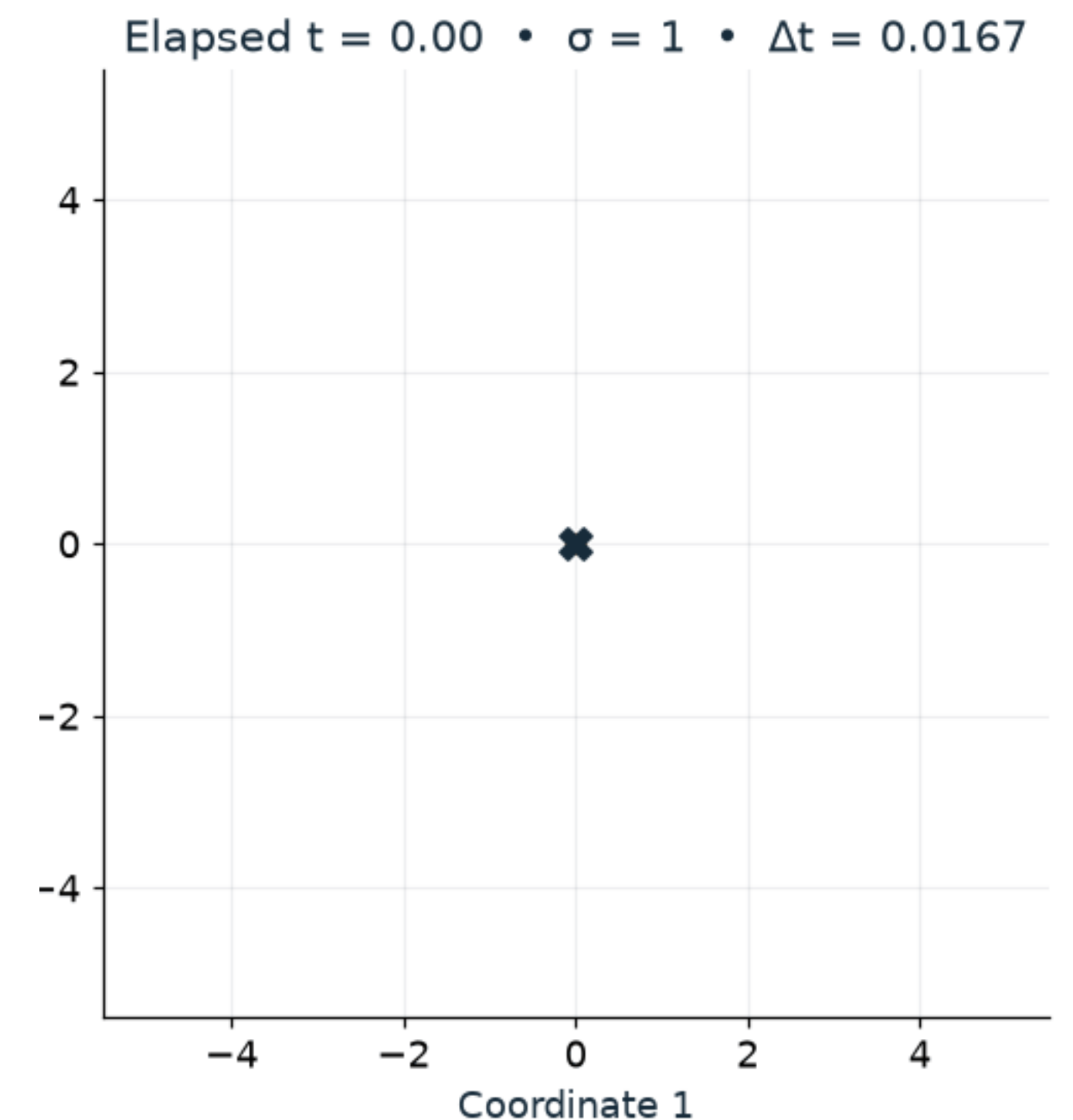
$$dX_t = u_t(X_t)dt + \sigma_t dW_t$$



✕ Same initial point — Random path ● Current position



✕ Same initial point — Random path ● Current position



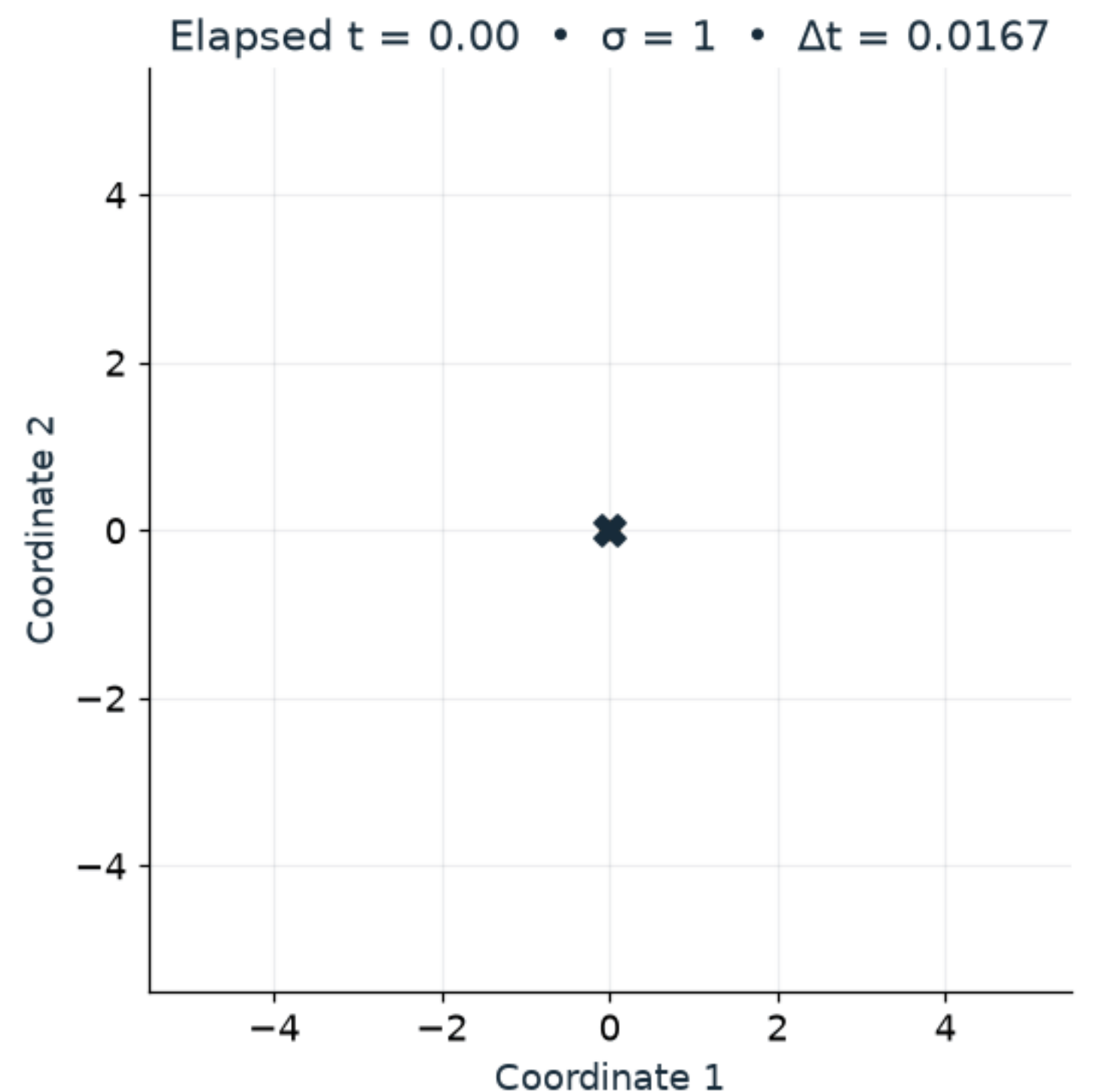
✕ Same initial point — Random path ● Current position

With the same initial state, fresh random increments produce different trajectories

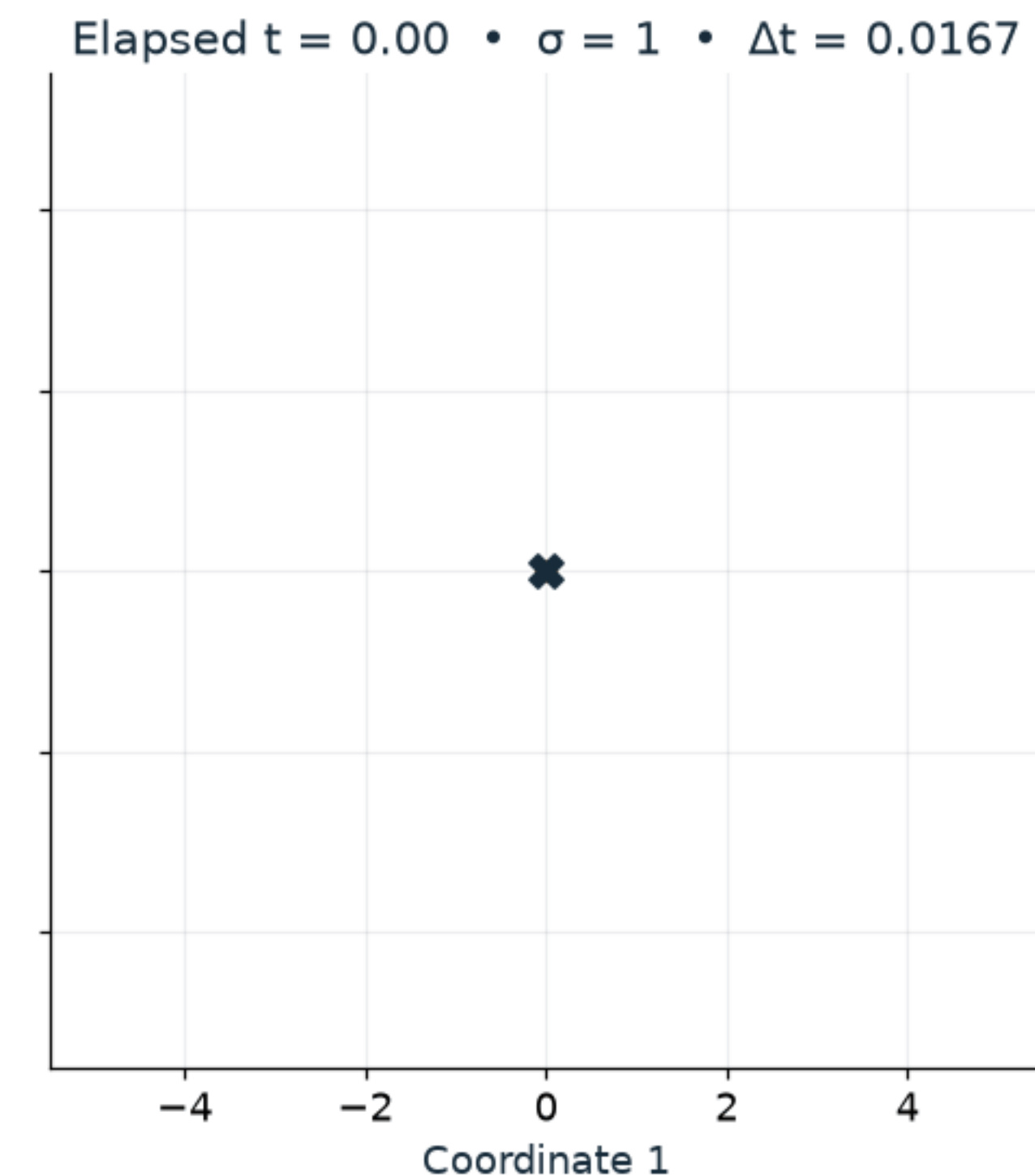
Stochastic Differential Equations (SDEs)

Stochastic Process: Random Trajectories - Brownian Motion

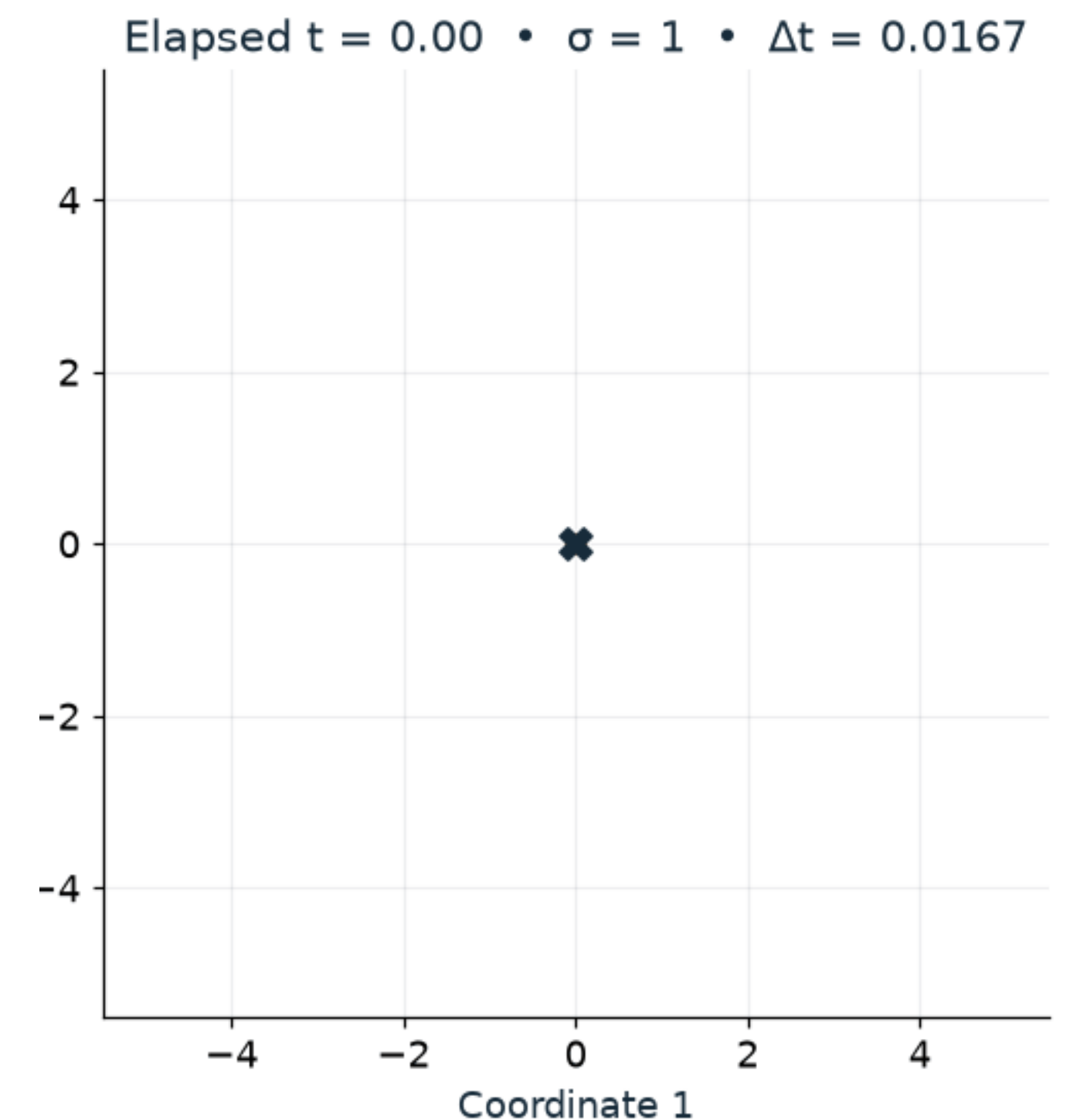
$$dX_t = u_t(X_t)dt + \sigma_t dW_t$$



x Same initial point — Random path • Current position



x Same initial point — Random path • Current position



x Same initial point — Random path • Current position

With the same initial state, fresh random increments produce different trajectories

Stochastic Differential Equations (SDEs)

Stochastic Process: Random Trajectories - Brownian Motion

$$dX_t = u_t(X_t)dt + \sigma_t dW_t$$

A standard Brownian motion W_t is defined by the following properties:

- Start at zero: $W_0 = 0$
- Stationary Gaussian increments: $W_{t+h} - W_t \sim \mathcal{N}(0, h)$
- Continuous paths

Existence and Uniqueness Theorem for SDEs

Theorem: If the vector field $u_t(x)$ is continuously differentiable with bounded derivatives and the diffusion coefficient σ_t is continuous, then a unique solution (in distribution) to the SDE

$$X_0 = x_0, \quad dX_t = u_t(X_t)dt + \sigma_t dW_t$$

exists. Generally, this is true if the vector field is [Lipschitz](#).

In cases of interest for machine learning, **unique solutions for SDEs exist.**

More details in stochastic calculus class

Numerical SDE simulation (Euler-Maruyama) method

Algorithm 3: Sampling from an SDE with the Euler-Maruyama method

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

1. Set $t = 0$
2. Set step size $h = \frac{1}{n}$
3. Set $X_0 = x_0$
4. **for** $i = 1, \dots, n$ **do**
5. Draw a sample $\epsilon \sim \mathcal{N}(0, I_d)$
6. $X_{t+h} = X_t + hu_t(X_t) + \sigma_t \sqrt{h} \epsilon$
7. Update $t \rightarrow t + h$
8. **end for**
9. **return** $X_0, X_h, X_{2h}, \dots, X_1$

Add additional noise with variance h scaled by the diffusion coefficient σ_t

How to generate objects with a Diffusion Model

Algorithm 3: Sampling from an SDE with the Euler-Maruyama method

Requires: Neural network u_t^θ , number of steps n , and diffusion coefficient σ_t

1. Set $t = 0$
2. Set step size $h = \frac{1}{n}$
3. Draw a sample $X_0 \sim p_{\text{init}} = \mathcal{N}(0, I_d)$
4. **for** $i = 1, \dots, n$ **do**
5. Draw a sample $\epsilon \sim \mathcal{N}(0, I_d)$
6. $X_{t+h} = X_t + hu_t^\theta(X_t) + \sigma_t \sqrt{h} \epsilon$
7. Update $t \rightarrow t + h$
8. **end for**
9. **return** X_1

Flow vs Diffusion Models

**Flow
Model**

Initialize:

$$X_0 \sim p_{\text{init}},$$

e.g. Gaussian

ODE:

$$dX_t = u_t^\theta(X_t)dt$$

Neural network
vector field

Diffusion coefficient

**Diffusion
Model**

Initialize:

$$X_0 \sim p_{\text{init}},$$

SDE:

$$dX_t = u_t^\theta(X_t)dt + \sigma_t dW_t$$

To get samples, simulate ODE/SDE from $t=0$ to $t=1$ and return X_1

Section 3

Training for Flow and Diffusion Models

Derive training algorithm for flow and diffusion models

Subsection 1

Flow Matching

The Flow Matching Matrix

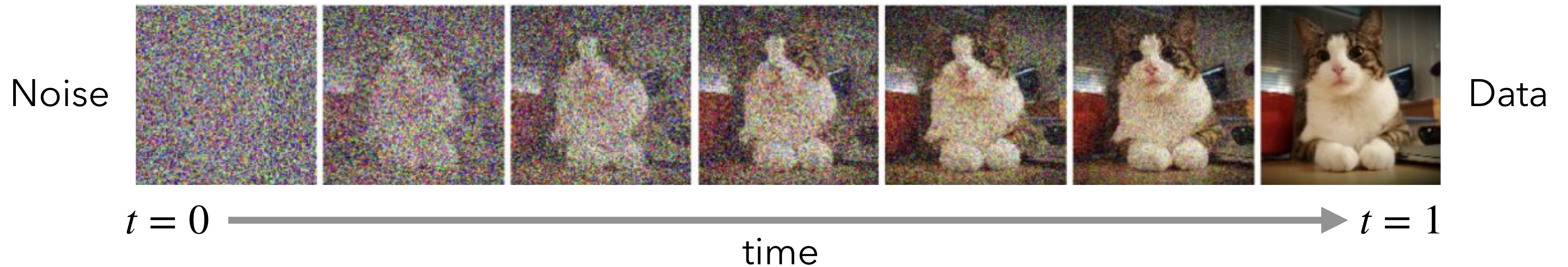


"Conditional" = "per single data points"

"Marginal" = "across distribution of data points"

Probability Paths

Choose a path whose point we can manufacture



Conditional Probability Paths $p_t(\cdot | z)$

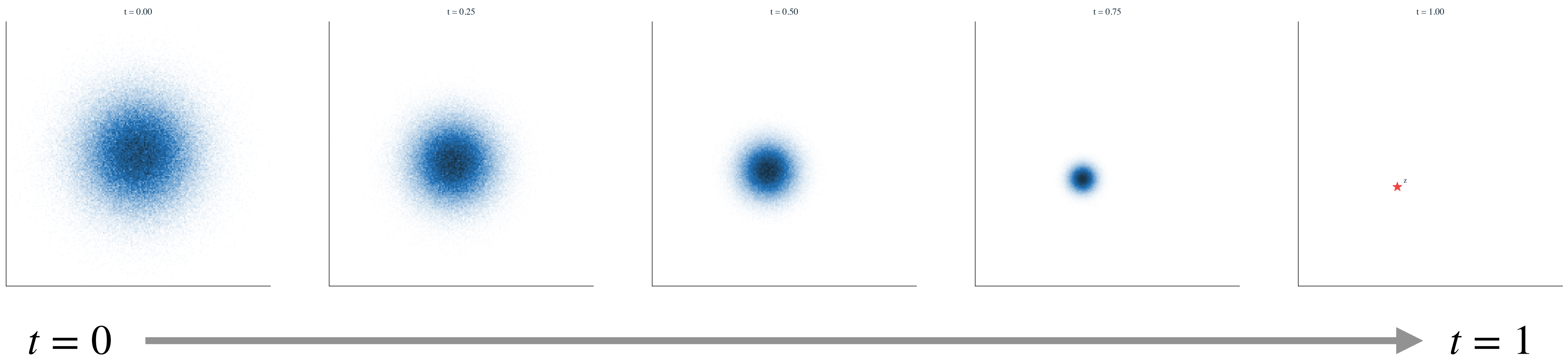
Interpolates p_{init} and a data point z

Conditional Probability Paths $p_t(\cdot | z)$

Interpolates p_{init} and a data point z

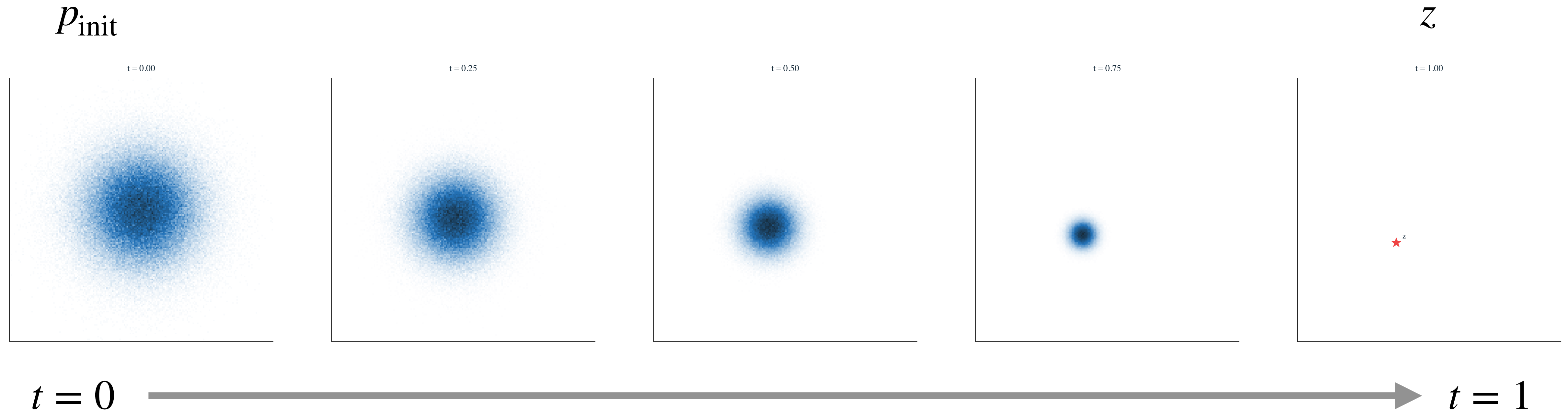
p_{init}

z



Conditional Probability Paths $p_t(\cdot | z)$

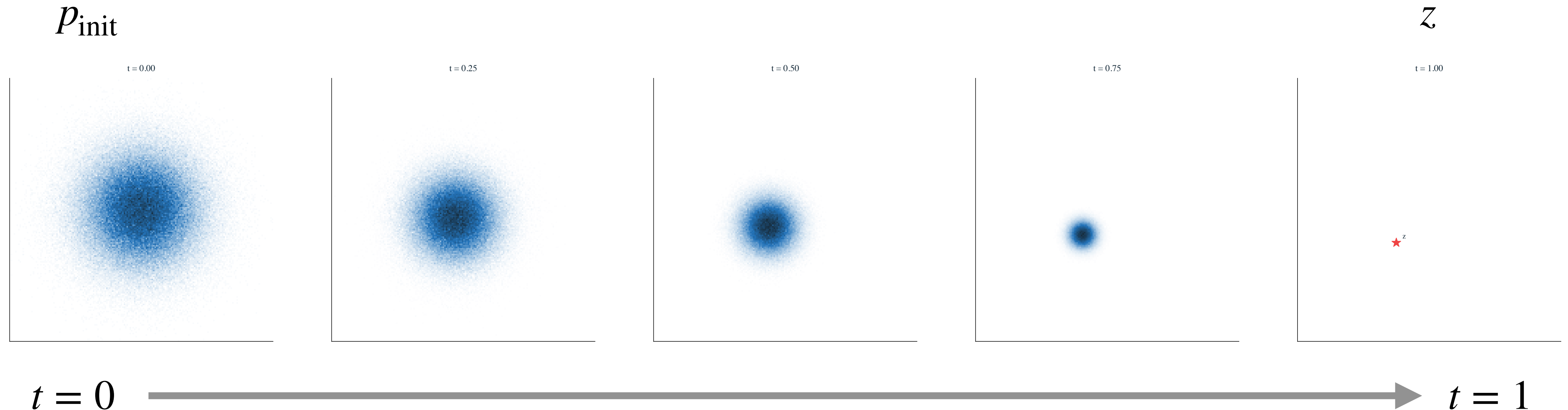
Interpolates p_{init} and a data point z



We may choose a Gaussian probability path: $p_t(\cdot | z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$

Conditional Probability Paths $p_t(\cdot | z)$

Interpolates p_{init} and a data point z



We may choose a Gaussian probability path: $p_t(\cdot | z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$

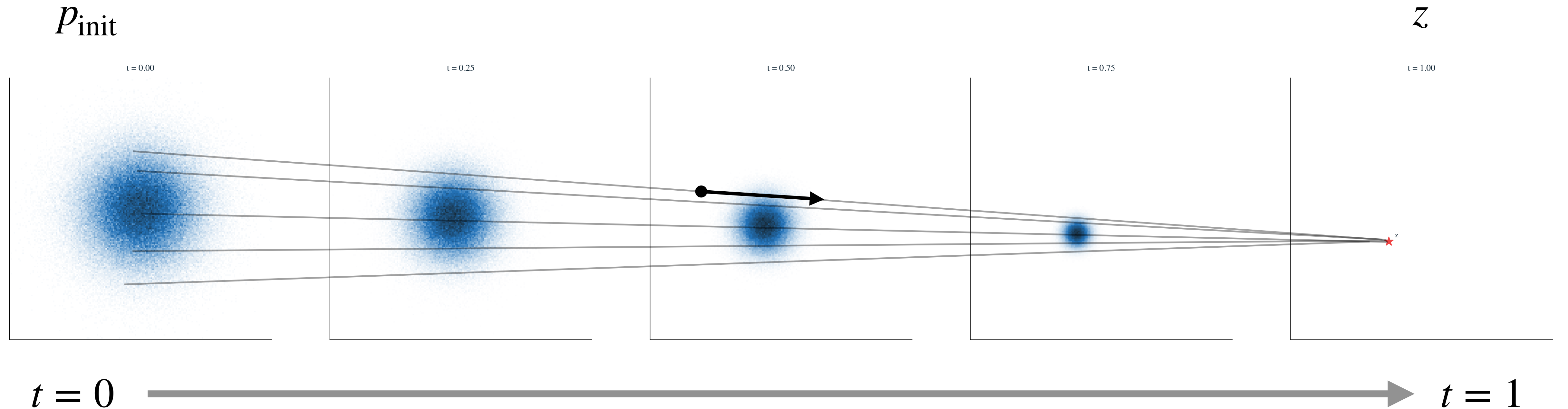
Example: $z \sim p_{\text{data}}, \quad \epsilon \sim \mathcal{N}(0, I_d), \quad t \sim \text{Uniform}(0, 1)$

$x_t = t z + (1 - t) \epsilon$

{

$\alpha_t = t$
 $\beta_t = 1 - t$

Conditional Vector Field $u_t^{\text{target}}(x | z)$



For each data point z , we denote $u_t^{\text{target}}(x | z)$ as the conditional vector field

$$X_0 \sim p_{\text{init}}, \quad \frac{dX_t}{dt} = u_t^{\text{target}}(X_t | z) \Rightarrow X_t \sim p_t(\cdot | z) \quad 0 \leq t \leq 1$$

Conditional Vector Field for Gaussian

A path with a velocity we already know

Conditional Vector Field for Gaussian

A path with a velocity we already know

Example:

$$z \sim p_{\text{data}}, \quad \epsilon \sim \mathcal{N}(0, I_d), \quad t \sim \text{Uniform}(0, 1), \text{ such that } x_t = tz + (1 - t)\epsilon.$$

Conditional Vector Field for Gaussian

A path with a velocity we already know

Example:

$z \sim p_{\text{data}}, \quad \epsilon \sim \mathcal{N}(0, I_d), \quad t \sim \text{Uniform}(0, 1),$ such that $x_t = tz + (1 - t)\epsilon$.

At the constructed point x_t , $u_t^{\text{target}}(x_t \mid z) = z - \epsilon$.

Conditional Vector Field for Gaussian

A path with a velocity we already know

Example:

$z \sim p_{\text{data}}, \quad \epsilon \sim \mathcal{N}(0, I_d), \quad t \sim \text{Uniform}(0, 1),$ such that $x_t = tz + (1 - t)\epsilon$.

At the constructed point x_t , $u_t^{\text{target}}(x_t \mid z) = z - \epsilon$.

More generally, at an arbitrary point x , $u_t^{\text{target}}(x \mid z) = \frac{z - x}{1 - t}$.

Conditional Vector Field for Gaussian

A path with a velocity we already know

Recall a Gaussian probability path: $p_t(\cdot | z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$

We can show that:

$$u_t^{\text{target}}(x | z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t} \alpha_t \right) z + \frac{\dot{\beta}_t}{\beta_t} x.$$

Example:

$z \sim p_{\text{data}}, \quad \epsilon \sim \mathcal{N}(0, I_d), \quad t \sim \text{Uniform}(0, 1)$, such that $x_t = tz + (1 - t)\epsilon$.

At the constructed point x_t , $u_t^{\text{target}}(x_t | z) = z - \epsilon$.

More generally, at an arbitrary point x , $u_t^{\text{target}}(x | z) = \frac{z - x}{1 - t}$.

Marginal Probability Paths $p_t(\cdot)$ and vector fields $u_t^{\text{target}}(x)$

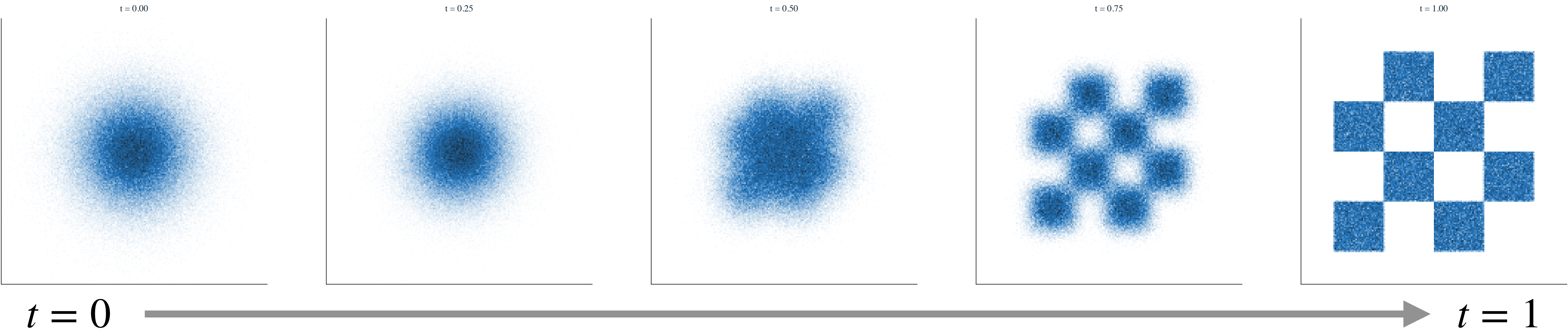
Interpolates p_{init} and p_{data}

Marginal Probability Paths $p_t(\cdot)$ and vector fields $u_t^{\text{target}}(x)$

Interpolates p_{init} and p_{data}

p_{init}

p_{data}

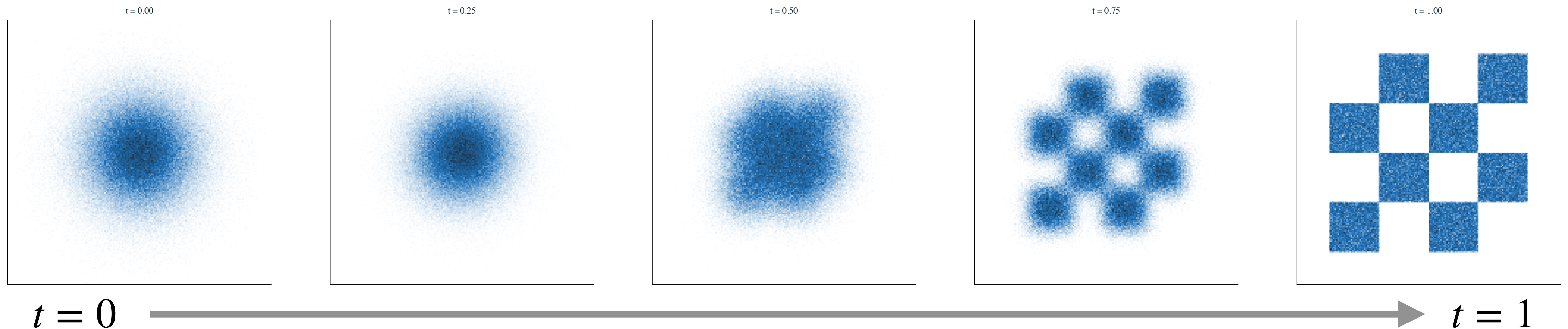


Marginal Probability Paths $p_t(\cdot)$ and vector fields $u_t^{\text{target}}(x)$

Interpolates p_{init} and p_{data}

p_{init}

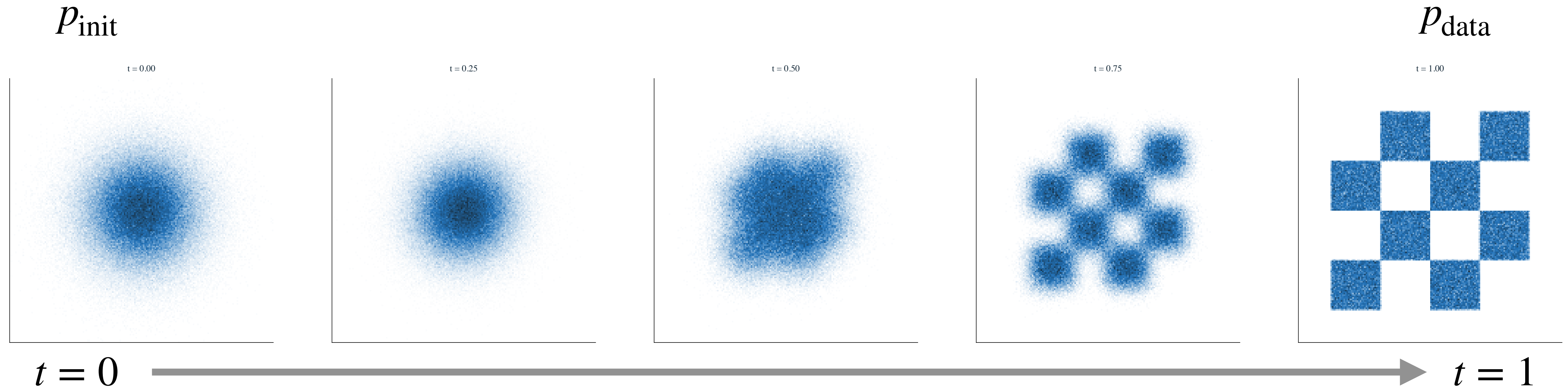
p_{data}



Marginal probability path:
$$p_t(x) = \int p_t(x | z) p_{\text{data}}(z) dz$$

Marginal Probability Paths $p_t(\cdot)$ and vector fields $u_t^{\text{target}}(x)$

Interpolates p_{init} and p_{data}

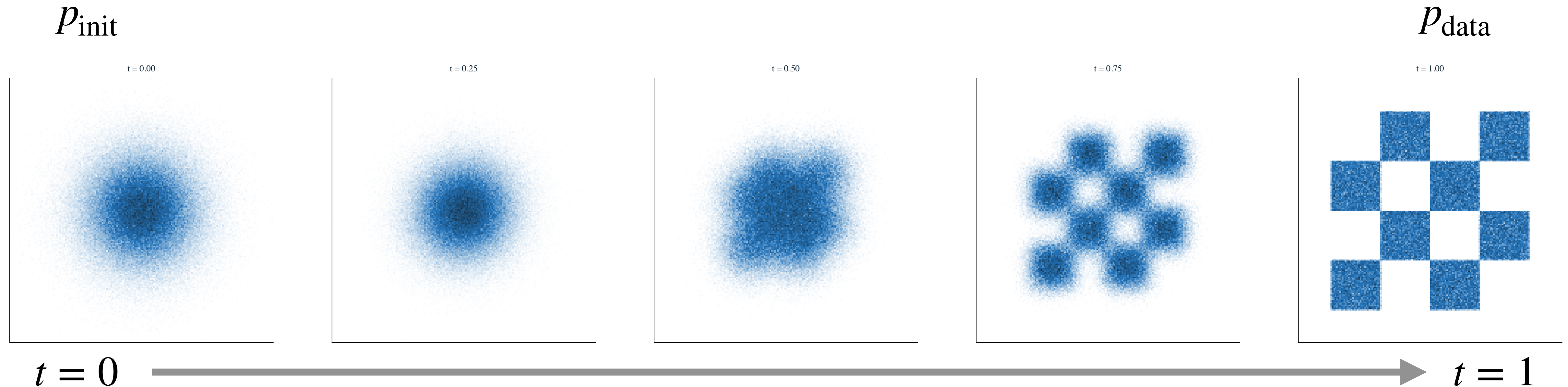


Marginal probability path:
$$p_t(x) = \int p_t(x | z) p_{\text{data}}(z) dz$$

Marginal vector field:
$$u_t^{\text{target}}(x) = \mathbb{E}[u_t(x_t | z) | x_t = x, t] = \int u_t^{\text{target}}(x | z) \frac{p_t(x | z) p_{\text{data}}(z)}{p_t(x)} dz$$

Marginal Probability Paths $p_t(\cdot)$ and vector fields $u_t^{\text{target}}(x)$

Interpolates p_{init} and p_{data}

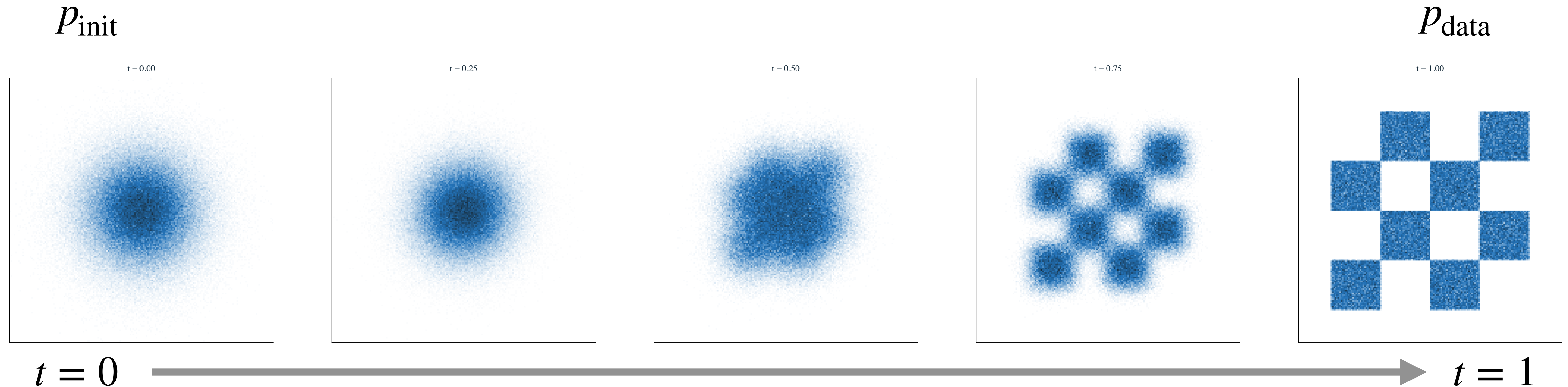


Marginal probability path: $p_t(x) = \int p_t(x | z) p_{\text{data}}(z) dz$

Marginal vector field: $u_t^{\text{target}}(x) = \mathbb{E}[u_t(x_t | z) | x_t = x, t] = \int u_t^{\text{target}}(x | z) \frac{p_t(x | z) p_{\text{data}}(z)}{p_t(x)} dz$

Marginal Probability Paths $p_t(\cdot)$ and vector fields $u_t^{\text{target}}(x)$

Interpolates p_{init} and p_{data}



Marginal probability path: $p_t(x) = \int p_t(x | z) p_{\text{data}}(z) dz$

Marginal vector field: $u_t^{\text{target}}(x) = \mathbb{E}[u_t(x_t | z) | x_t = x, t] = \int u_t^{\text{target}}(x | z) \frac{p_t(x | z) p_{\text{data}}(z)}{p_t(x)} dz$

GOAL! But intractable

Learning the marginal vector field $u_t^{\text{target}}(x)$

We want $u_t^\theta(x) \approx u_t^{\text{target}}(x)$

Learning the marginal vector field $u_t^{\text{target}}(x)$

We want $u_t^\theta(x) \approx u_t^{\text{target}}(x)$

Flow Matching loss: $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x) \right\|^2 \right]$

Learning the marginal vector field $u_t^{\text{target}}(x)$

We want $u_t^\theta(x) \approx u_t^{\text{target}}(x)$

Flow Matching loss: $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x) \right\|^2 \right]$

Conditional Flow Matching loss: $\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x | z) \right\|^2 \right]$

Learning the marginal vector field $u_t^{\text{target}}(x)$

We want $u_t^\theta(x) \approx u_t^{\text{target}}(x)$

Flow Matching loss: $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x) \right\|^2 \right]$

Conditional Flow Matching loss: $\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x | z) \right\|^2 \right]$

We can show that: $\mathcal{L}_{\text{CFM}}(\theta) = \mathcal{L}_{\text{FM}}(\theta) + C \quad \Rightarrow \quad \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{FM}}(\theta)$

Learning the marginal vector field $u_t^{\text{target}}(x)$

We want $u_t^\theta(x) \approx u_t^{\text{target}}(x)$

Flow Matching loss: $\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x) \right\|^2 \right]$

Conditional Flow Matching loss: $\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t \sim \text{Unif}, z \sim p_{\text{data}}, x \sim p_t(\cdot|z)} \left[\left\| u_t^\theta(x) - u_t^{\text{target}}(x | z) \right\|^2 \right]$

We can show that: $\mathcal{L}_{\text{CFM}}(\theta) = \mathcal{L}_{\text{FM}}(\theta) + C \quad \Rightarrow \quad \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{FM}}(\theta)$

Minimizing $\mathcal{L}_{\text{CFM}}(\theta)$ is equivalent to minimizing $\mathcal{L}_{\text{FM}}(\theta)$!!

Training Procedure (General)

Algorithm 4: Flow Matching Training Procedure

Requires: A dataset of samples $z \sim p_{\text{data}}$, a neural network u_t^θ

1. **for** each mini-batch of data **do**
2. Sample a data example z from the dataset
3. Sample a random time $t \sim \text{Unif}_{[0,1]}$
4. Sample $x \sim p_t(\cdot | z)$
5. Compute loss $\mathcal{L}(\theta) = \left\| u_t^\theta(x) - u_t^{\text{target}}(x | z) \right\|^2$
6. Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
7. **end for**

Training Procedure (Gaussian)

Algorithm 4: Flow Matching Training for Gaussian with straight path conditional paths

Requires: A dataset of samples $z \sim p_{\text{data}}$, a neural network u_t^θ

1. **for** each mini-batch of data **do**
2. Sample a data example z from the dataset
3. Sample a random time $t \sim \text{Unif}_{[0,1]}$
4. Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5. Set $x = tz + (1 - t)\epsilon$ ← Straight line schedule
6. Compute loss $\mathcal{L}(\theta) = \left\| u_t^\theta(x) - (z - \epsilon) \right\|^2$
7. Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
8. **end for**

Training uses sampled points along known paths. It does not require simulating the learned ODE!

The Flow Matching Matrix



Defines
distributions from
noise to data

Defines training
target that we want
to learn

Loss function that
we want to
minimize during
training

The Continuity Equation Holds

Given $X_0 \sim p_{\text{init}}$, $\frac{dX_t}{dt} = u_t(X_t)$

Follow probability path:

$$X_t \sim p_t \quad (0 \leq t \leq 1)$$



Continuity equation holds:

$$\partial_t p_t(x) = -\operatorname{div}(p_t u_t)(x)$$

Net
probability
flux at x

Outflow - inflow
of probability
mass from u

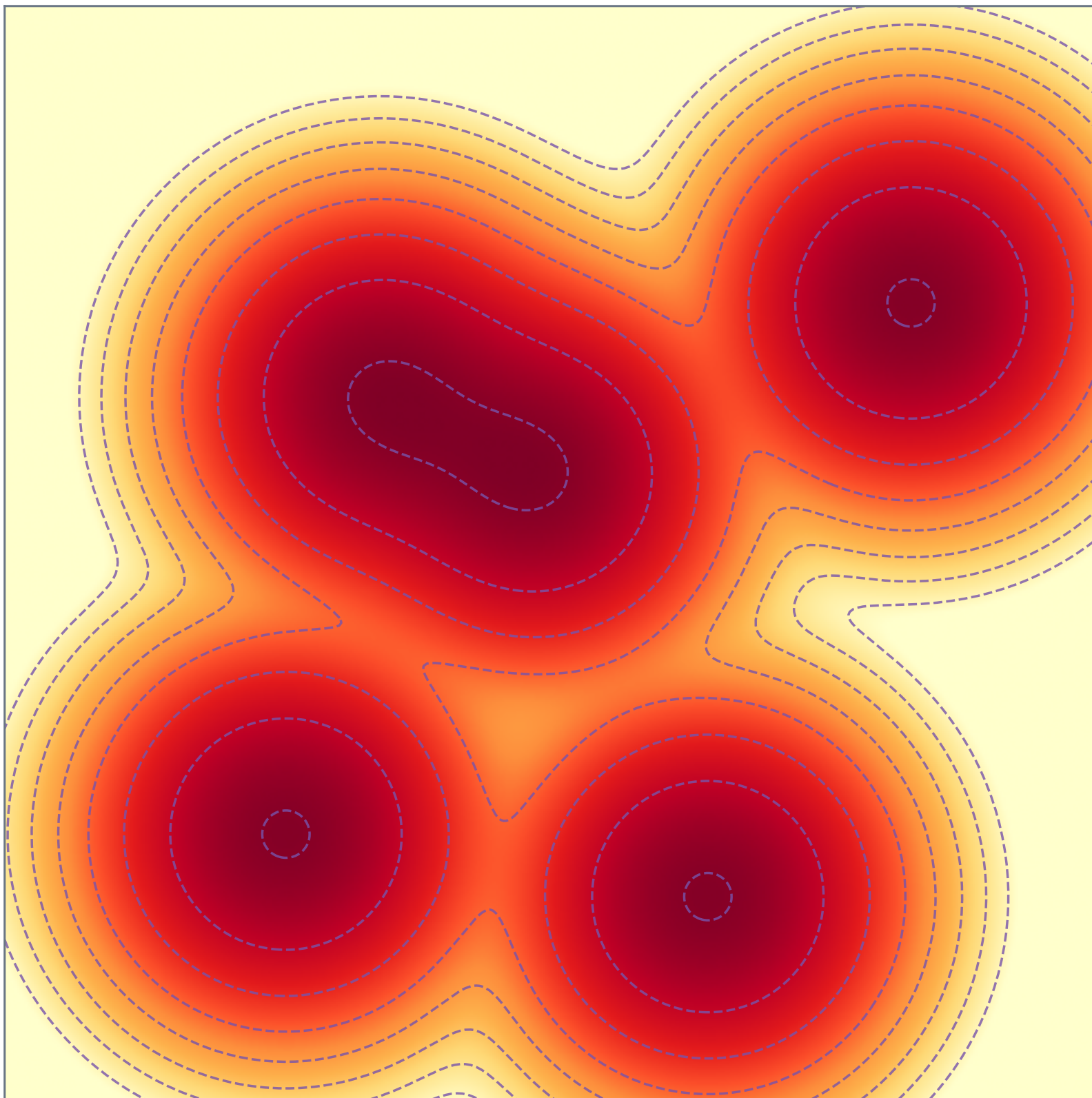
Subsection 2

Score Matching

Score functions = Spatial gradients of the log-density

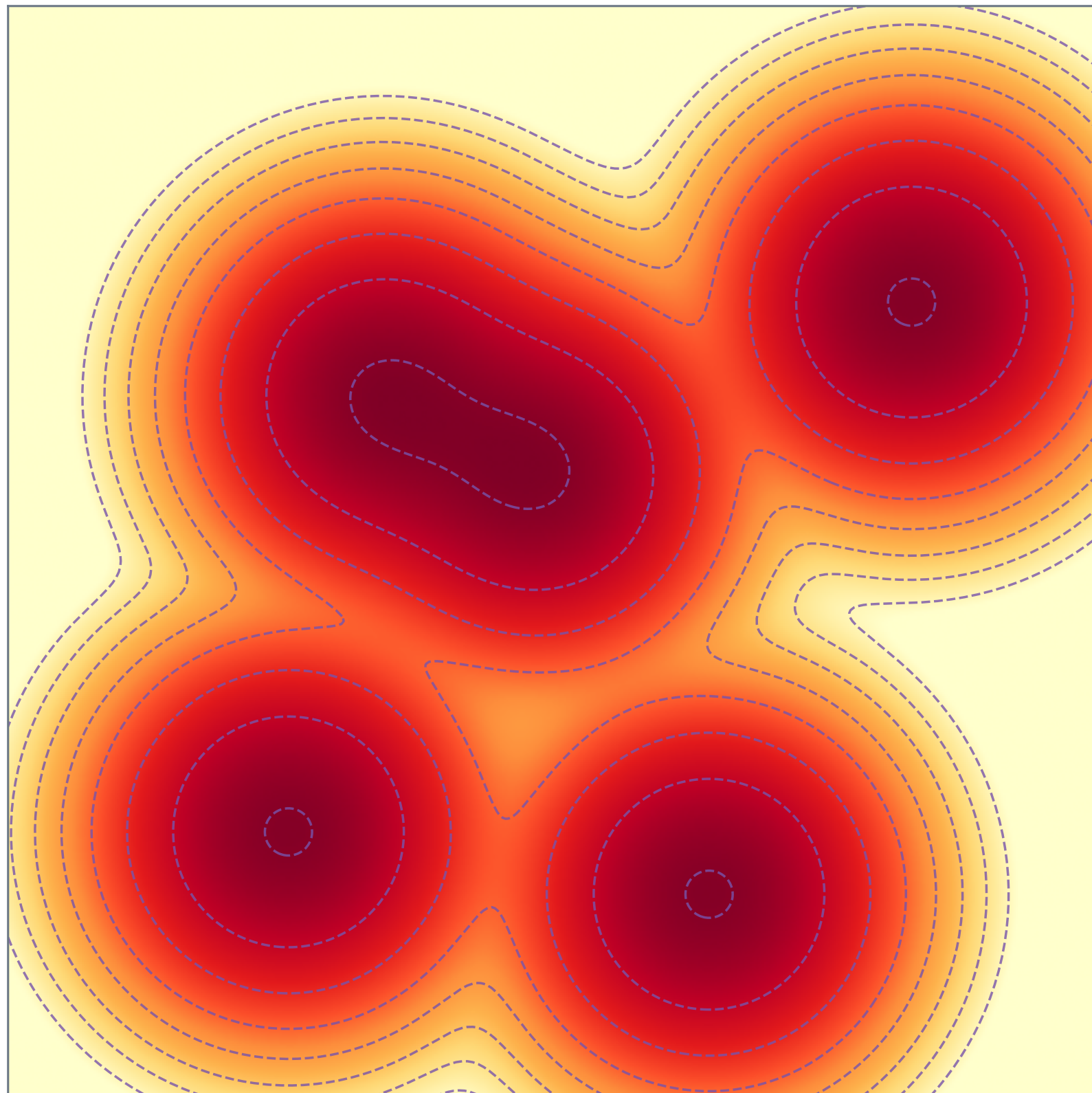
Score functions = Spatial gradients of the log-density

Log-density: $\log \mathbf{q}(\mathbf{x})$



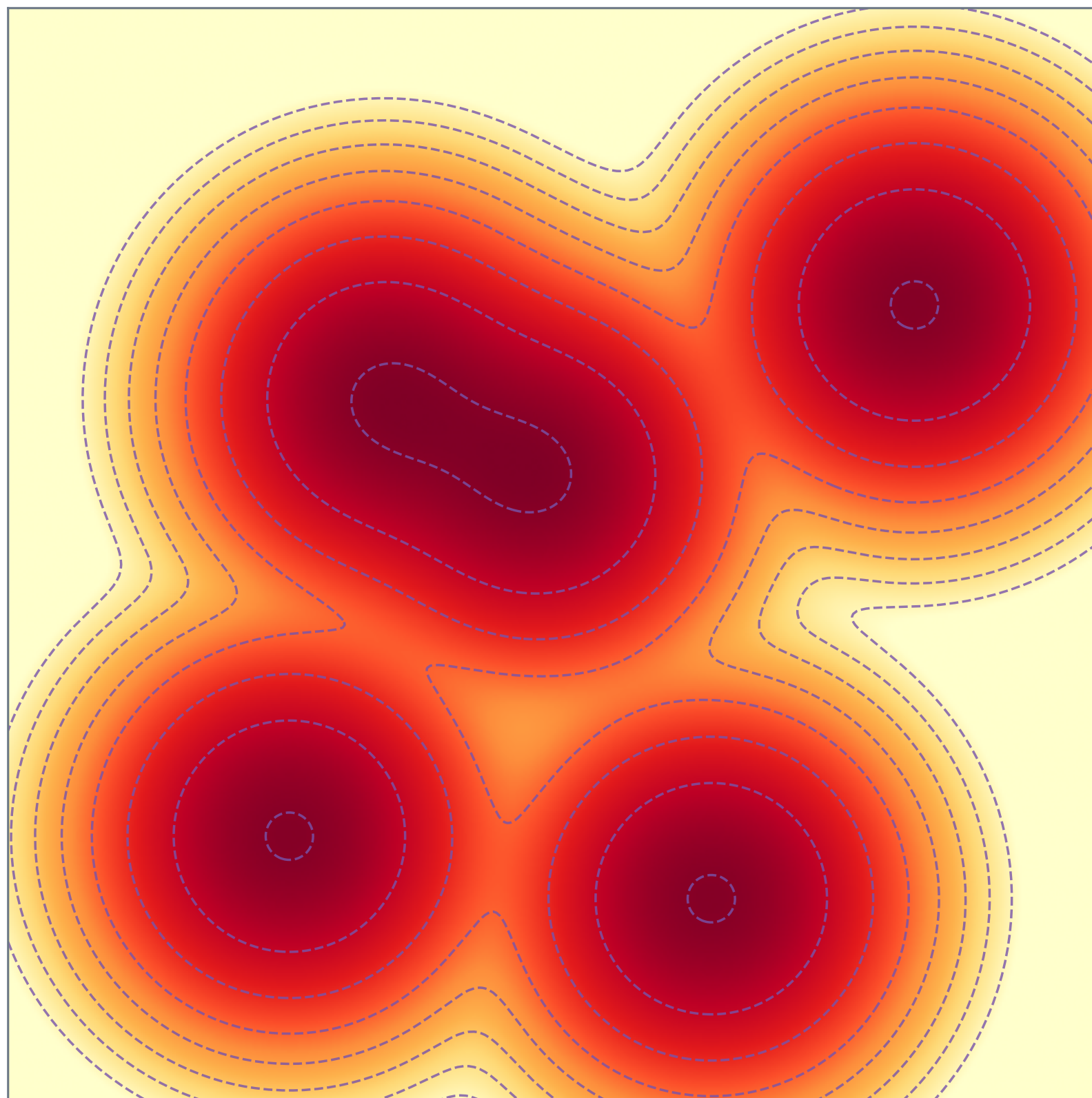
Score functions = Spatial gradients of the log-density

Log-density: $\log \mathbf{q}(\mathbf{x})$

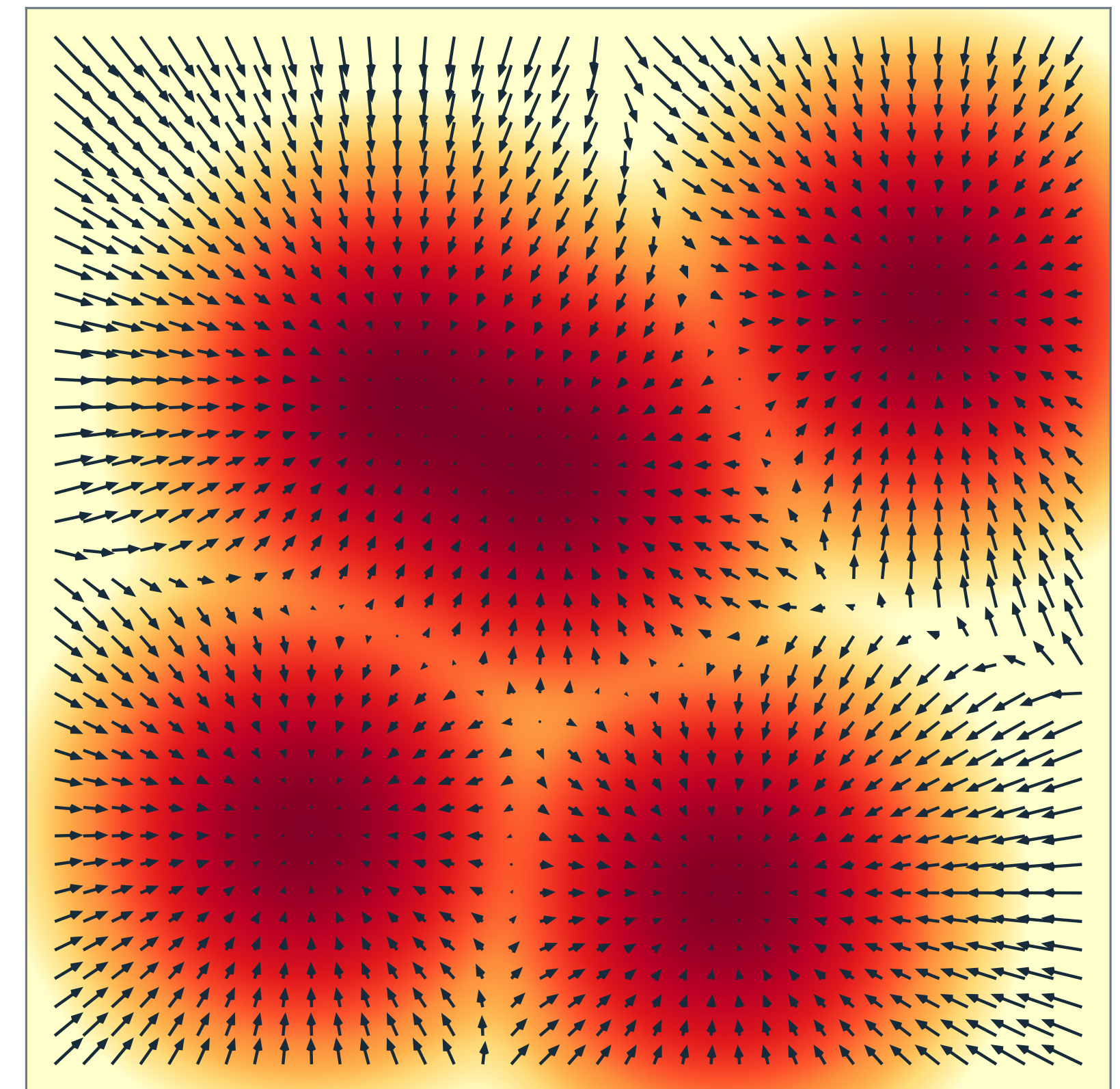


Score functions = Spatial gradients of the log-density

Log-density: $\log \mathbf{q}(\mathbf{x})$



Score function: $s_t(\mathbf{x}) = \nabla_{\mathbf{x}} \log \mathbf{q}(\mathbf{x})$



Example - Score of Gaussian Probability Path

$$s_t(x \mid z) = \nabla_x \log p_t(x \mid z) = -\frac{1}{\beta_t^2}x + \frac{\alpha_t}{\beta_t^2}z$$

Proof sketch:

Start with $p_t(x \mid z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d)$

$$\begin{aligned}\log p_t(x \mid z) &= C(t) - \frac{\|x - \alpha_t z\|^2}{2\beta_t^2} \\ \nabla_x \log p_t(x \mid z) &= -\frac{x - \alpha_t z}{\beta_t^2}\end{aligned}$$

Example - Score of Gaussian Probability Path

$$s_t(x \mid z) = \nabla_x \log p_t(x \mid z) = -\frac{1}{\beta_t^2}x + \frac{\alpha_t}{\beta_t^2}z$$

For the constructed point $x_t = \alpha_t z + \beta_t \epsilon$, $s_t(x \mid z) = \frac{-\epsilon}{\beta_t}$, $\beta_t > 0$

Proof sketch:

Start with $p_t(x \mid z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d)$

$$\begin{aligned}\log p_t(x \mid z) &= C(t) - \frac{\|x - \alpha_t z\|^2}{2\beta_t^2} \\ \nabla_x \log p_t(x \mid z) &= -\frac{x - \alpha_t z}{\beta_t^2}\end{aligned}$$

Recap on Gaussian Flow Matching

Conditional Probability Path, Vector Field, and Score

	Notation	Key Property	Gaussian Example
Conditional Probability Path	$p_t(\cdot z)$	Interpolates p_{init} and a data point z	$\mathcal{N}(\alpha_t z, \beta_t^2 I_d)$
Conditional Vector Field	$u_t^{\text{target}}(x z)$	ODE follows conditional path	$\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t} \alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x$
Conditional Score Function	$\nabla_x \log p_t(x z)$	Gradient of log-density	$\frac{\alpha_t}{\beta_t^2} z - \frac{1}{\beta_t^2} x$

Note: **For a fixed example z and time t , both Cond. Vector Field and Score function are affine in x !**

Velocity Field to Score Function

$$a_t = \left(\beta_t^2 \frac{\dot{\alpha}_t}{\alpha_t} - \dot{\beta}_t \beta_t \right), \quad b_t = \frac{\dot{\alpha}_t}{\alpha_t}, \quad \alpha_t \neq 0.$$

$$u_t^{\text{target}}(x \mid z) = a_t \nabla_x \log p_t(x \mid z) + b_t x.$$

$$u_t^{\text{target}}(x) = a_t \nabla_x \log p_t(x) + b_t x.$$

Early Diffusion Models learned the score function instead and then just transformed it into the vector field.

For a fixed Gaussian path, velocity, score, and noise are connected prediction targets.

Training Procedure (General)

Algorithm 5: Score Matching Training Procedure

Requires: A dataset of samples $z \sim p_{\text{data}}$, a score network s_t^θ

1. **for** each mini-batch of data **do**
2. Sample a data example z from the dataset
3. Sample a random time $t \sim \text{Unif}_{[0,1]}$
4. Sample $x \sim p_t(\cdot | z)$
5. Compute loss $\mathcal{L}(\theta) = \left\| s_t^\theta(x) - \nabla \log p_t(x | z) \right\|^2$
6. Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
7. **end for**

Denoising: Learn from noise we added ourselves

Algorithm 6: Score Matching Training Procedure

Requires: A dataset of samples $z \sim p_{\text{data}}$, a score network s_t^θ

1. **for** each mini-batch of data **do**
2. Sample a data example z from the dataset
3. Sample a random time $t \sim \text{Unif}_{[0,1]}$
4. Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5. Set $x = \alpha_t z + \beta_t \epsilon$
6. Compute loss $\mathcal{L}(\theta) = \left\| s_t^\theta(x) + \frac{\epsilon}{\beta_t} \right\|^2$ 
7. Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
8. **end for**

$$\log p_t(x | z) = C(t) - \frac{\|x - \alpha_t z\|^2}{2\beta_t^2}$$
$$\nabla_x \log p_t(x | z) = -\frac{x - \alpha_t z}{\beta_t^2}$$

The network predicts the noise that was used to corrupt the data point!
(**DENOISING** diffusion models)

Fokker-Planck Equation Holds

$$\text{Given } X_0 \sim p_{\text{init}}, \quad dX_t = u_t(X_t)dt + \sigma_t dW_t$$

Follow probability path:

$$X_t \sim p_t \quad (0 \leq t \leq 1)$$



Fokker Planck equation holds:

$$\partial_t p_t(x) = -\operatorname{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x)$$

Continuity eq.

Heat eq.

(Dispersion)

Stochastic Sampling of diffusion models

Choose noise level σ_t . By “SDE extension trick” we can sample from:

$$dX_t = \left[u_t^{\text{target}}(X_t) + \frac{\sigma_t^2}{2} \nabla_X \log p_t(X_t) \right] dt + \sigma_t dW_t$$

For Gaussian probability paths, we can express this solely in terms of the score:

$$dX_t = \left[\left(a_t + \frac{\sigma_t^2}{2} \right) \nabla_x \log p_t(X_t) + b_t X_t \right] dt + \sigma_t dW_t$$

Plugin the score network:

$$dX_t = \left[\left(a_t + \frac{\sigma_t^2}{2} \right) s_t^\theta(X_t) + b_t X_t \right] dt + \sigma_t dW_t$$

We can add stochasticity while preserving the intended probability path, provided we also correct the drift.

Why would we want stochastic/SDE dynamics

In practice:

- ▶ **Training error:** The neural network has not perfectly learned the marginal vector field/score.
- ▶ **Simulation error:** We need to simulate the SDE/ODE, leading to discretization error.

Downstream applications: Fine-tuning, inference-time optimization, etc. might require stochastic evolution.

Why would we want stochastic/SDE dynamics

In practice:

- ▶ **Training error:** The neural network has not perfectly learned the marginal vector field/score.
- ▶ **Simulation error:** We need to simulate the SDE/ODE, leading to discretization error.

Downstream applications: Fine-tuning, inference-time optimization, etc. might require stochastic evolution.

ODE sampling often leads to the best results. Therefore, SDE sampling is an option, not a must!

Section 3

Classifier-Free Guidance

Enforcing coherence to additional context or prompts

Conditioning vs Unconditional Generation



A swamp ogre with a pearl earring by Johannes Vermeer



A car made out of vegetables.



heat death of the universe,
line art

Image source: Scaling Rectified Flow Transformer for High-Resolution Image Synthesis

Unguided: "Generate an image."

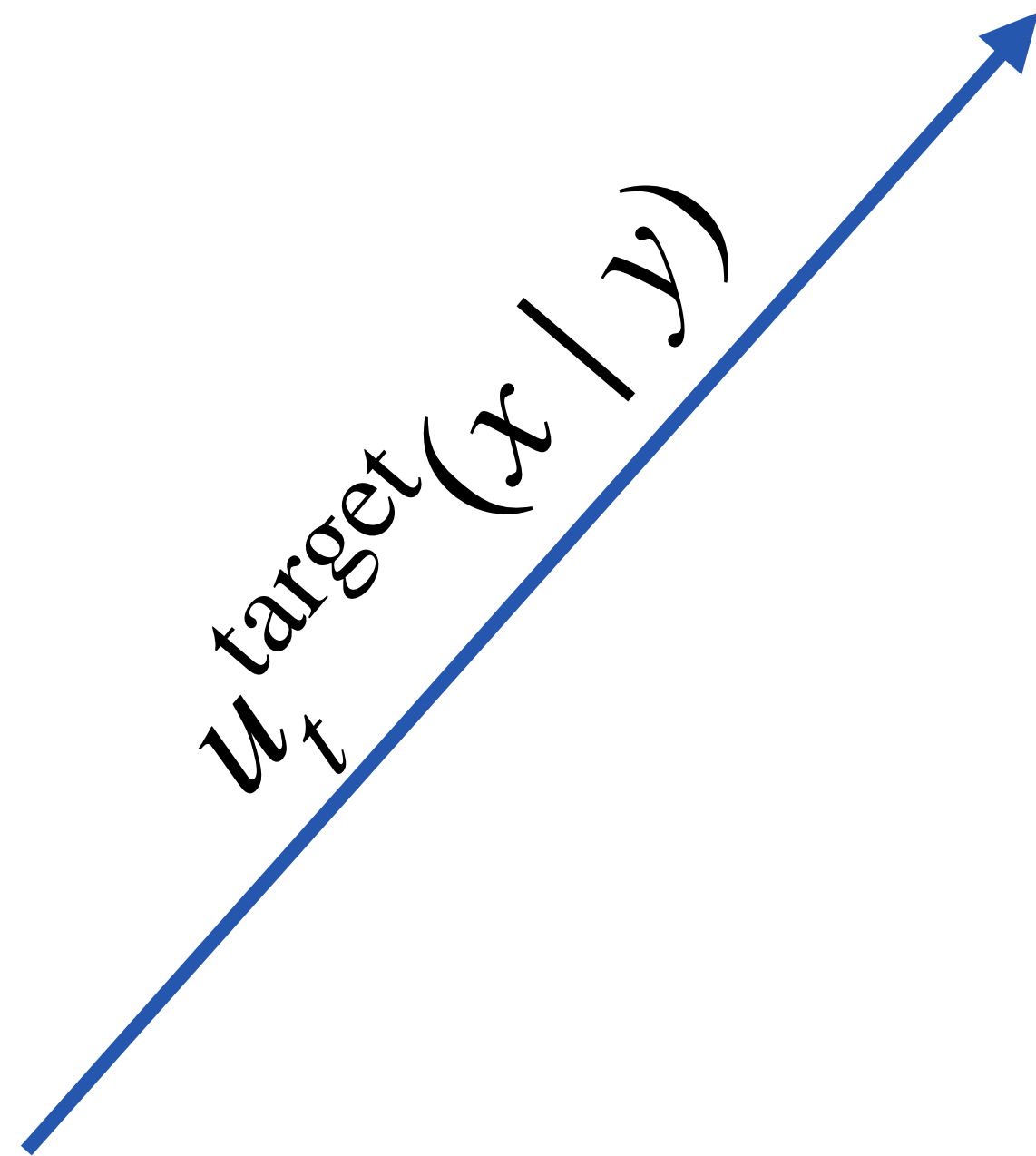
Guided : "Generate an image of a dog posing as an astronaut on the moon."

Classifier-free guidance

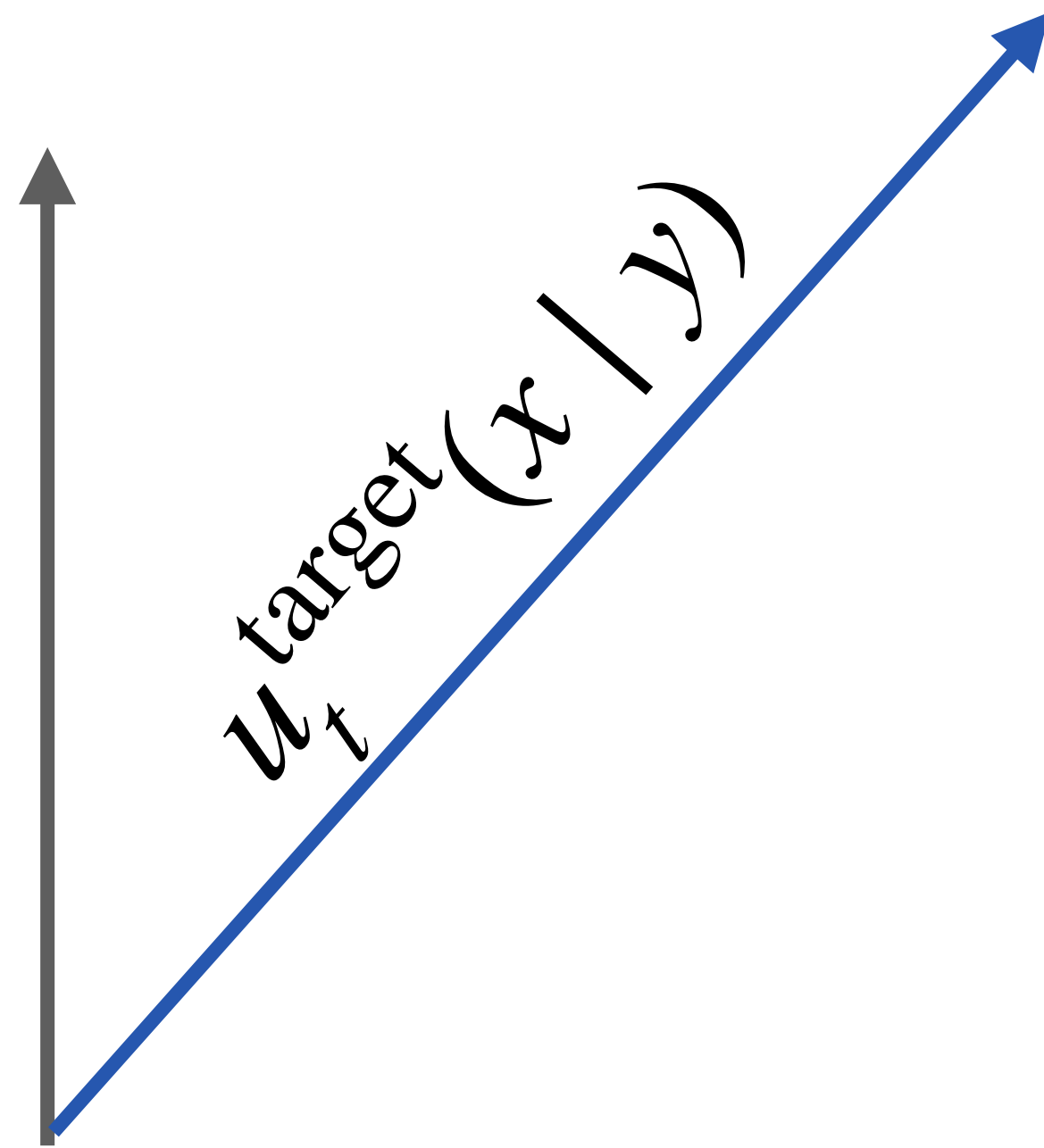
Classifier-free guidance

$$u_t^{\text{target}}(x / y)$$

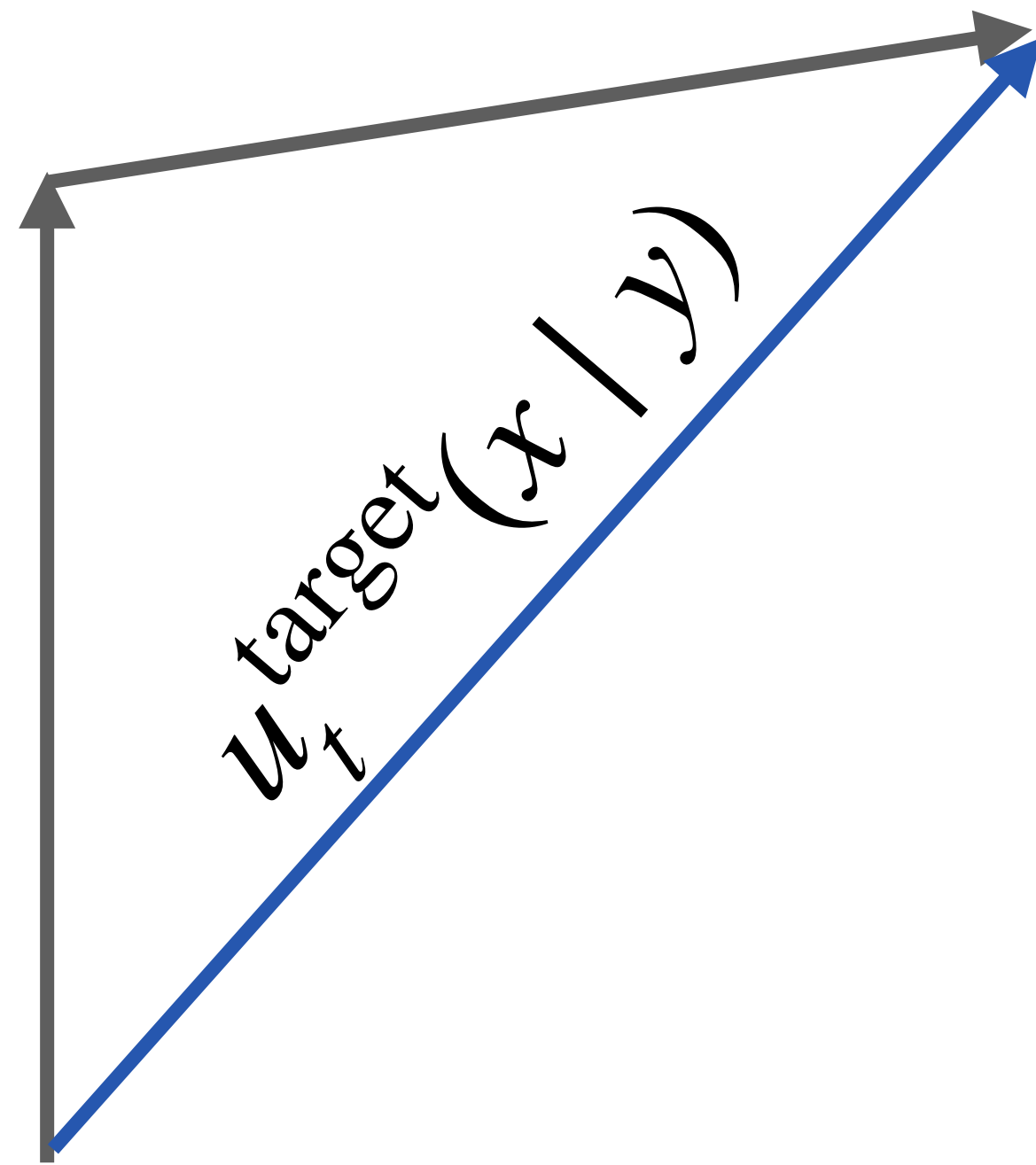
Classifier-free guidance



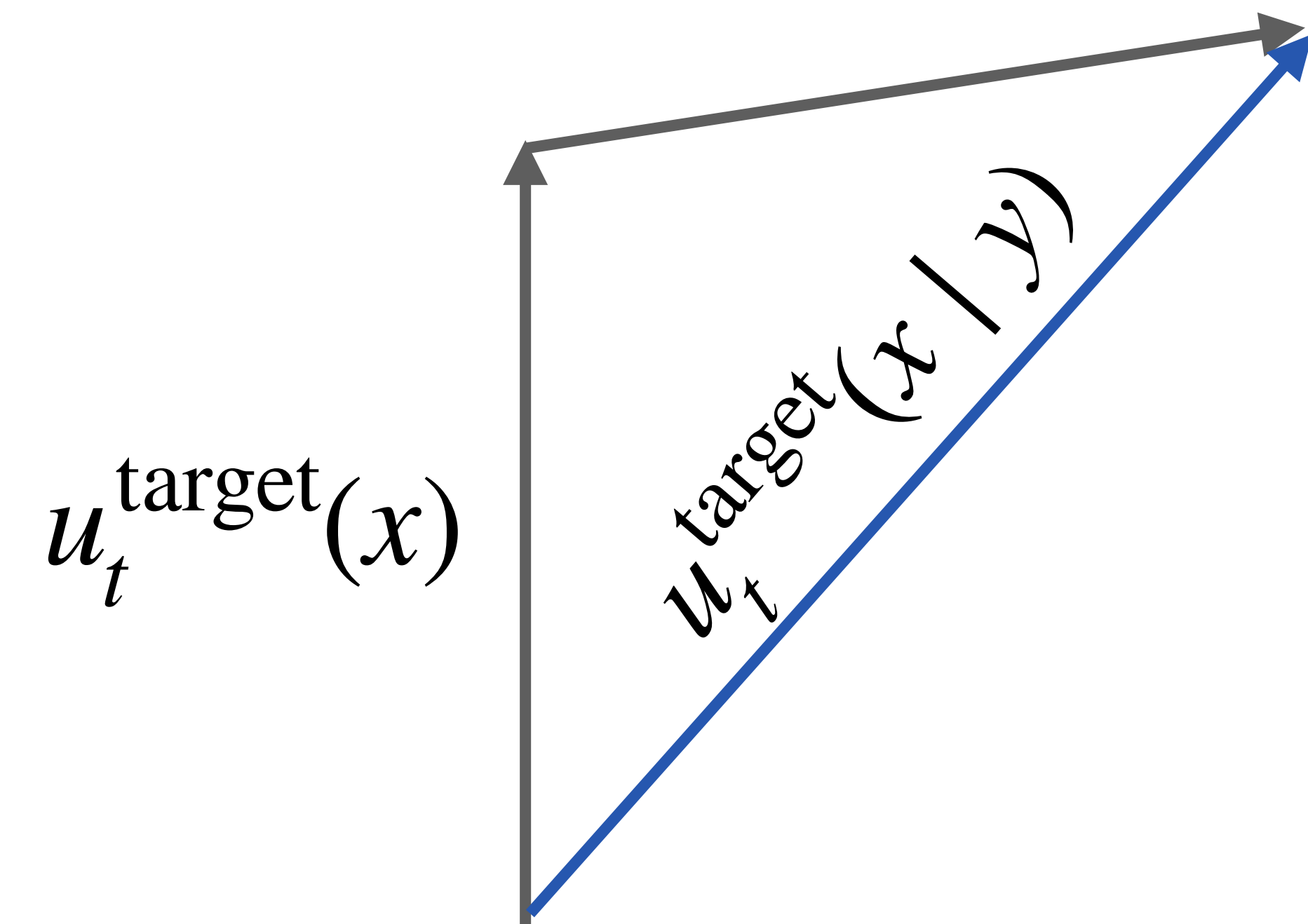
Classifier-free guidance



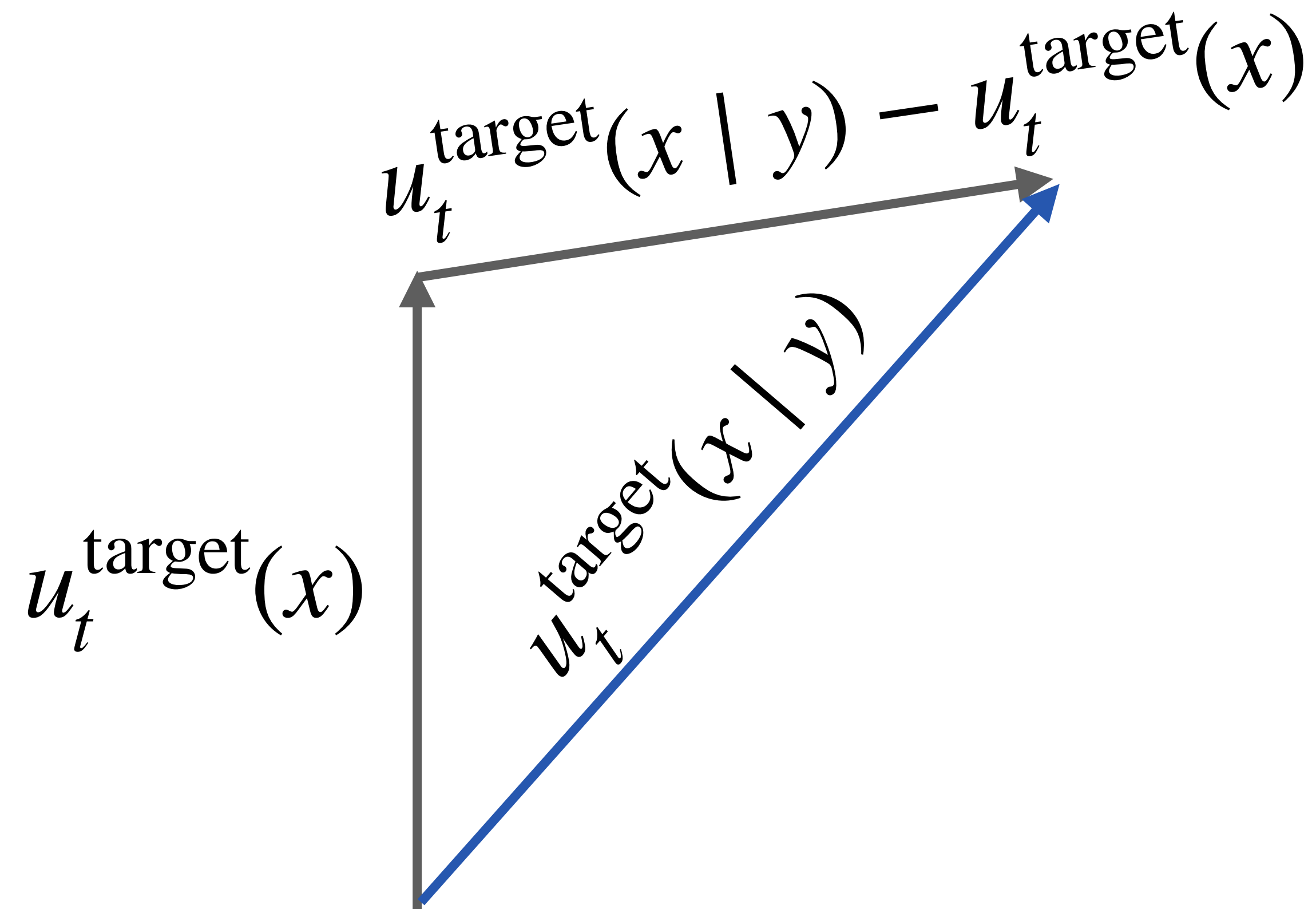
Classifier-free guidance



Classifier-free guidance

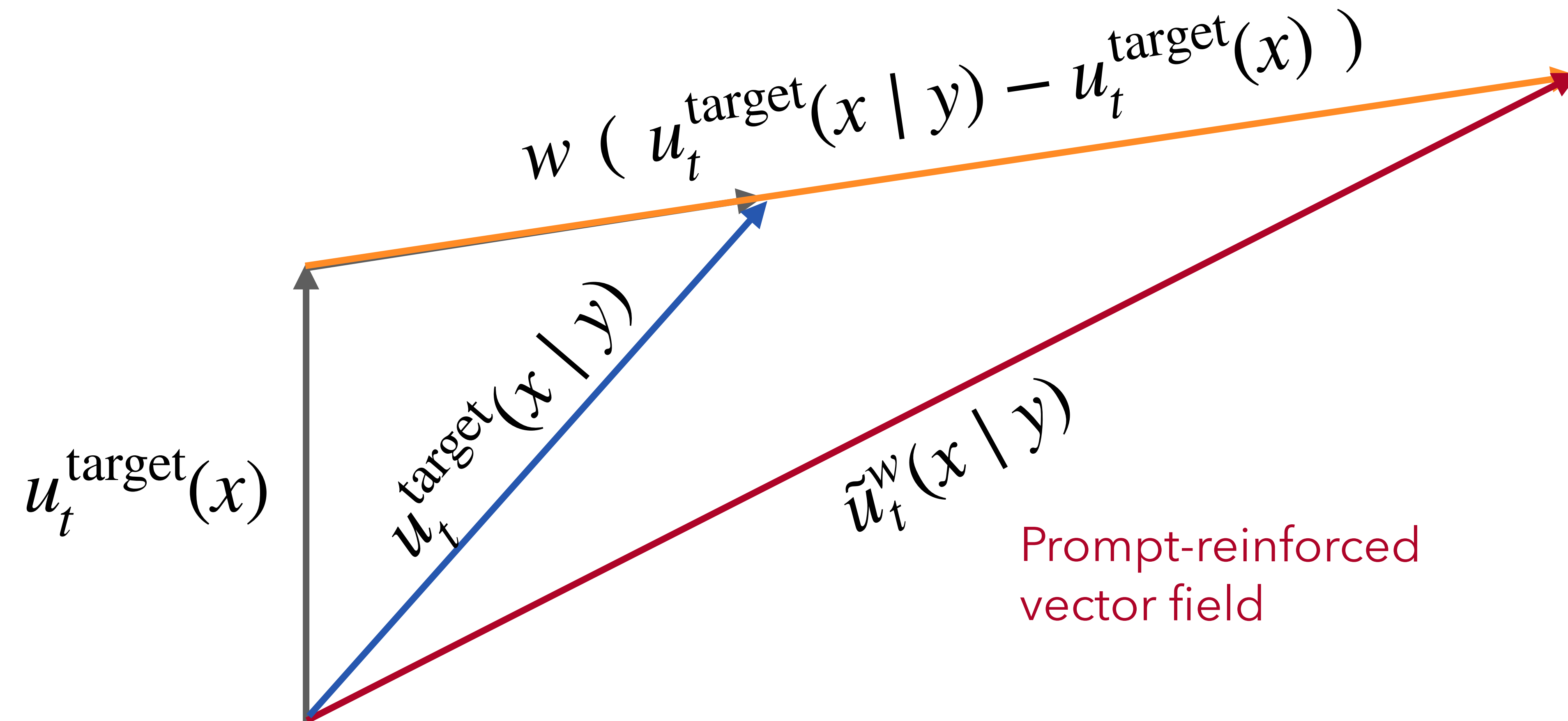


Classifier-free guidance



Classifier-free guidance

w is a guidance scale.



Classifier-free guidance Training

Algorithm 7: Classifier-free guidance training

Requires: Paired dataset $(z, y) \sim p_{\text{data}}$, a neural network u_t^θ

1. **for** each mini-batch of data **do**
2. Sample a data example (z, y) from the dataset
3. Sample a random time $t \sim \text{Unif}_{[0,1]}$
4. Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5. Set $x = \alpha_t z + \beta_t \epsilon$
6. With probability p drop label: $y \leftarrow \emptyset$ Drop label with a certain probability!
7. Compute loss $\mathcal{L}(\theta) = \left\| u_t^\theta(x \mid y) - u_t^{\text{target}}(x \mid z) \right\|^2$
8. Update the model parameters θ via gradient descent on $\mathcal{L}(\theta)$
9. **end for**

Sampling with Classifier-Free Guidance

Same as before but we use the weighed vector field:

$$u_t^{\theta,w}(X_t | y) = (1 - w)u_t^{\theta}(X_t | \emptyset) + wu_t^{\theta}(X_t | y)$$

Algorithm 8: Classifier-Free Guidance Sampling Procedure

Requires: A guided vector field $u_t^{\theta}(x | y)$, trained with conditional dropout

1. Select a prompt $y \in \mathcal{Y}$, or take $y = \emptyset$ for unguided sampling.
2. Select a **guidance scale** $w > 1$.
3. Initialize $X_0 \sim p_{\text{init}}$.
4. Simulate $dX_t = [(1 - w)u_t^{\theta}(X_t | \emptyset) + wu_t^{\theta}(X_t | y)]dt$ from $t = 0$ to $t = 1$.
5. Return X_1

Guidance trades condition adherence against diversity

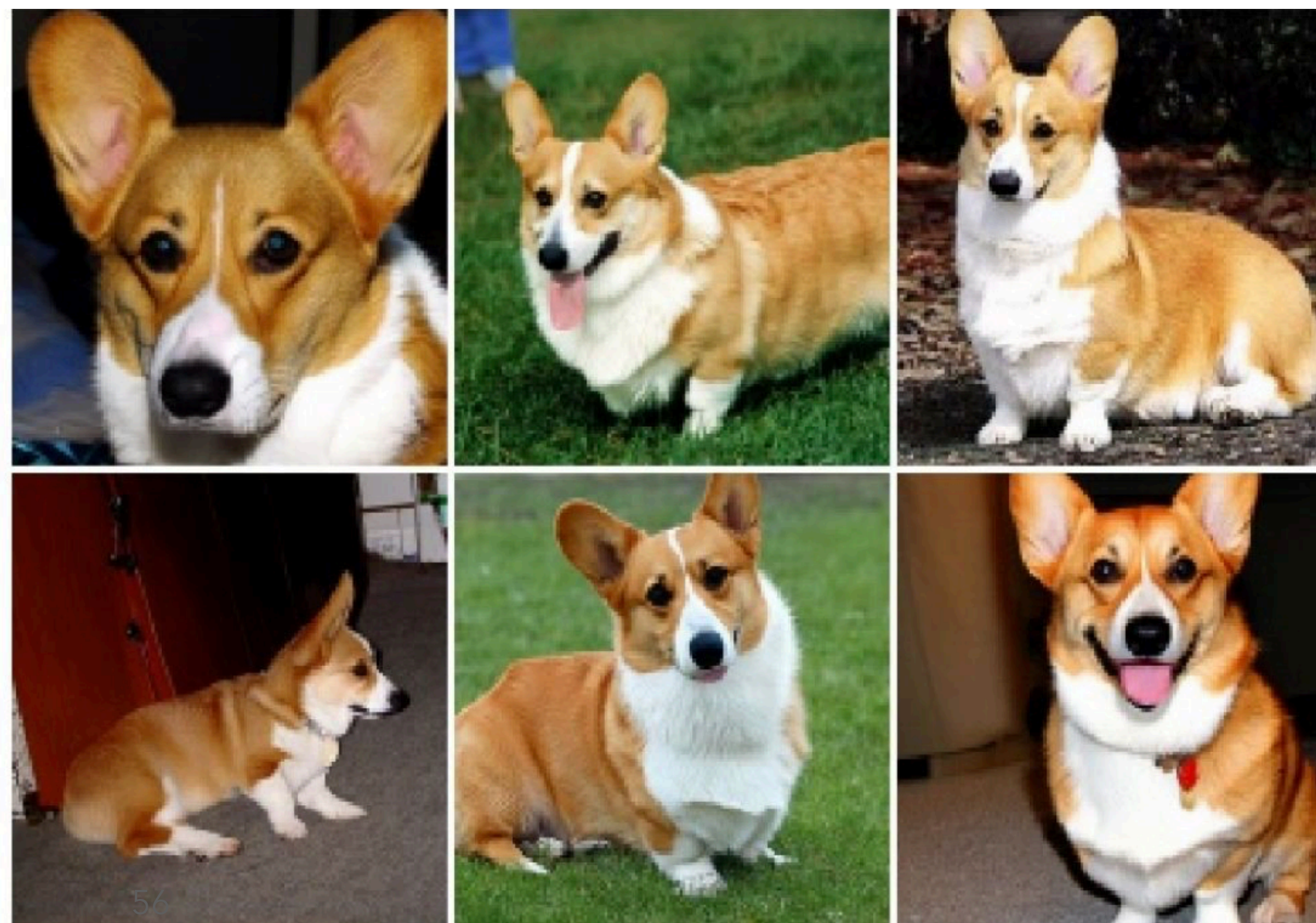
Without classifier-free guidance, "almost" nothing would work

Image source: Classifier-free
diffusion guidance

$w = 1$



$w = 4$



Section 4

Guide to the diffusion literature

Different interpretations for diffusion models

Time conventions

“Flow time convention”:

- ▶ Noise: $t = 0$
- ▶ Data: $t = 1$

“Diffusion time convention”:

- ▶ Noise: $t \rightarrow \infty$
- ▶ Data: $t = 0$

“Discrete time”:

- ▶ Use discrete time steps instead of continuous time steps
- ▶ No ODE or SDE but a Markov chain
- ▶ DDIM $\approx$ Probability flow ODE

Flow matching, rectified flows, stochastic interpolants

Score-based diffusion models with SDEs

DDPM (Denoising Diffusion Probabilistic Model)

DDIM (Denoising Diffusion Implicit Model)

Noising Procedure

Probability path:

$$p_t(x \mid z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$$

Interpolant function:

$$I_t(\epsilon, z) = \alpha_t z + \beta_t \epsilon$$

“Forward” diffusion process:

$$dX_t = a_t(X_t)dt + \sigma_t dW_t$$

Flow matching, rectified flows

stochastic interpolants

Denoising diffusion models

Note: For Gaussian probability paths, the above procedures are equivalent.

Constructing noising procedures via forward noising processes

Forward SDE (data $\rightarrow$ noise)

$$\textcircled{\mathbf{x}(0)} \longrightarrow d\mathbf{x} = f(\mathbf{x}, t)dt + g(t)d\mathbf{w} \longrightarrow \textcircled{\mathbf{x}(T)}$$



$$\textcircled{\mathbf{x}(0)} \longleftarrow d\mathbf{x} = \left[f(\mathbf{x}, t) - g^2(t) \overset{\text{score function}}{\boxed{\nabla_{\mathbf{x}} \log p_t(\mathbf{x})}} \right] dt + g(t)d\bar{\mathbf{w}} \longleftarrow \textcircled{\mathbf{x}(T)}$$

Reverse SDE (noise $\rightarrow$ data)

The time-reversed SDE is a specific solution to the SDE extension trick discussed earlier for a specific noise level. Empirically, this is often not the best solution in practice.

Flow Matching

- ✓ Probably the “simplest” flow and diffusion algorithms
- ✓ Allows you to restrict yourself to flows
- ✓ Allows you to go from arbitrary p_{init} to arbitrary p_{data}

Note: The method presented here allows to convert arbitrary distributions into arbitrary distributions!

Takeaways

- ✓ Choose intermediate distributions connecting noise to data
- ✓ Learn a field using targets we can construct from data and noise
- ✓ Generate by integrating that field, deterministically or stochastically

When reading a paper, ask: What is the path? What does the network predict?
How is it sampled?

References

1. An Introduction to Flow Matching and Diffusion Models. <https://diffusion.csail.mit.edu>
2. Flow Matching for Generative Modeling. <https://arxiv.org/abs/2210.02747>
3. Score-Based Generative Modeling through Stochastic Differential Equations. <https://arxiv.org/abs/2011.13456>
4. Scaling Rectified Flow Transformers for High-Resolution Image Synthesis. <https://arxiv.org/abs/2403.03206>
5. An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale. <https://arxiv.org/abs/2010.11929>
6. High-Resolution Image Synthesis with Latent Diffusion Models. <https://arxiv.org/abs/2112.10752>
7. Classifier-Free Diffusion Guidance. <https://arxiv.org/abs/2207.12598>

This lecture adapts the flow-first exposition and selected illustrations from Peter Holderrieth and Ezra Erives, MIT 6.S184 (2026), with additional examples and annotations.