Carnegie Mellon University

Lecture 10: Distillation, Consistency Models & Flow Maps

Yutong (Kelly) He

10-799 Diffusion & Flow Matching, Feb 5th, 2026

Modal aws

Quiz time!

10 minutes

Closed-book

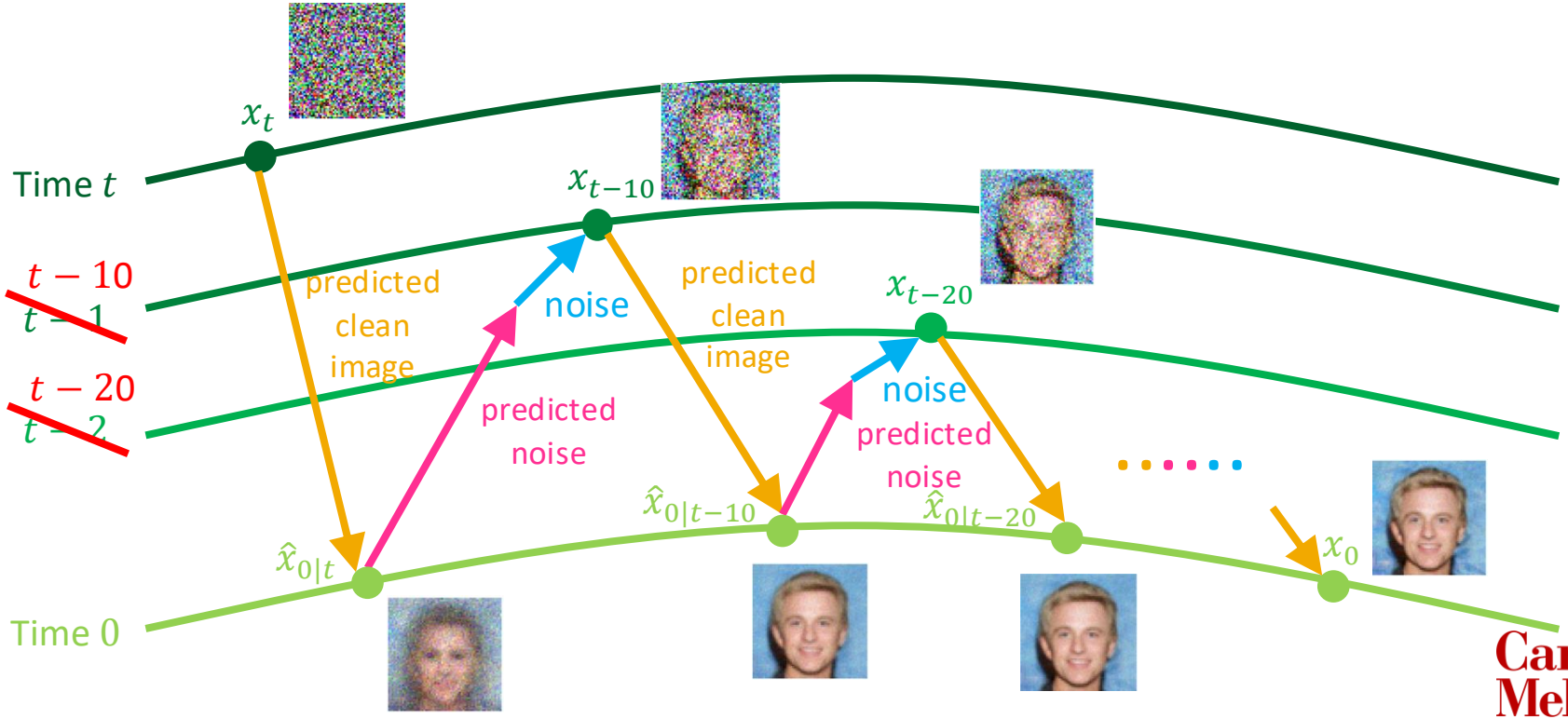
Pen & Paper

If you don’t want to stay for the lecture, feel free to leave after submitting your quiz!

Housekeeping Announcements

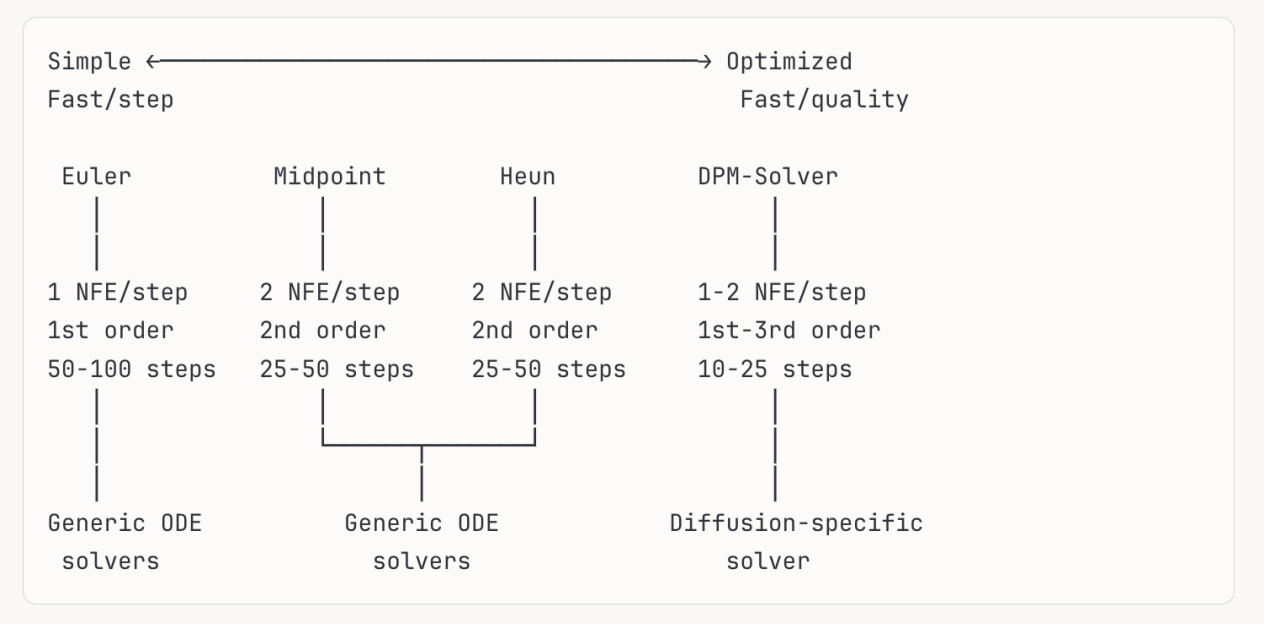
- Next class we will have Linqi (Alex) Zhou from Luma AI to give us a guest lecture
    - Will have pizza for in-person
- Homework 3 is out! https://kellyyutonghe.github.io/10799S26/homework/
    - Due date: 2/15 Sun, Late Due date: 2/17 Mon
    - Start early and finish early if possible! This way you'll have more time for HW 4
    - Poster PDF submission 2/25 Wed, Poster Session 2/26 Thur
    - No class on 2/24 Tue
- AWS credit allocation poll in Discord

Previously we have learned a bunch of sampling methods to make inference faster



Carnegie Mellon University

Previously we have learned a bunch of sampling methods to make inference faster



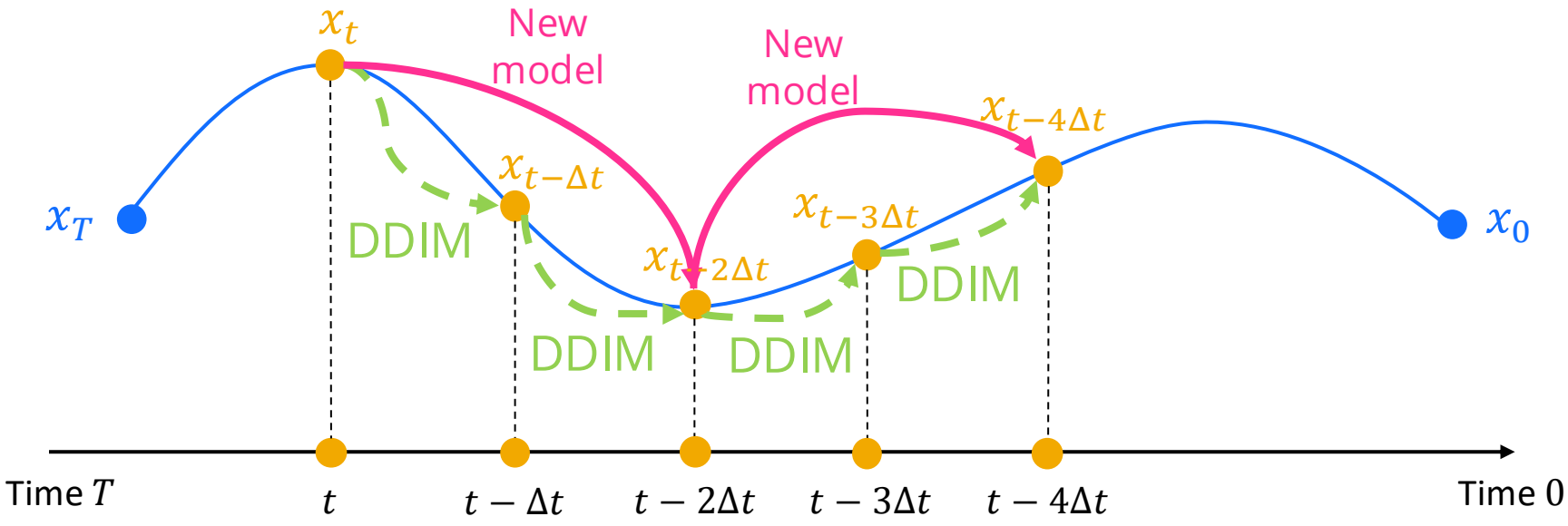
Carnegie Mellon University

What else can we do to accelerate inference?

Is it possible to **train** a model such that it also take less time for inference?

Suppose we already have a pre-trained diffusion model, what is the simplest way to train another model for faster sampling?

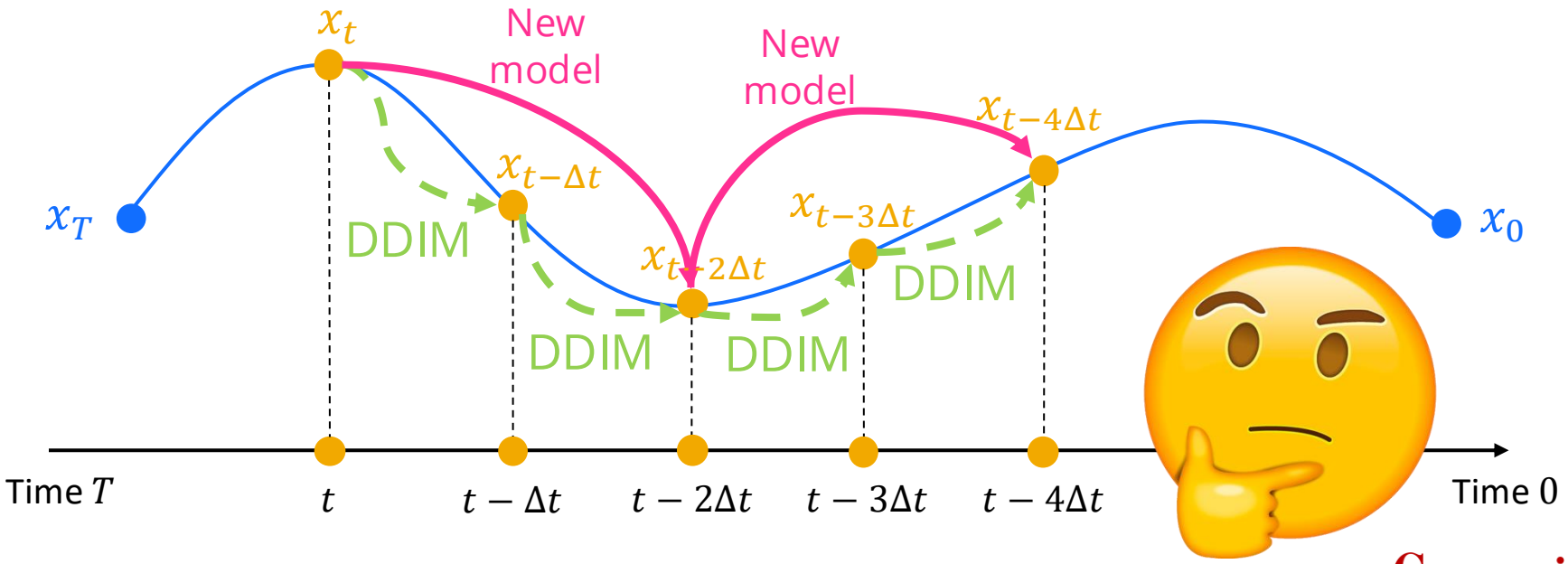
What if we just try to have 1 step in the new model to match with 2 steps in the original diffusion model



2X speedup from DDIM!

Carnegie Mellon University

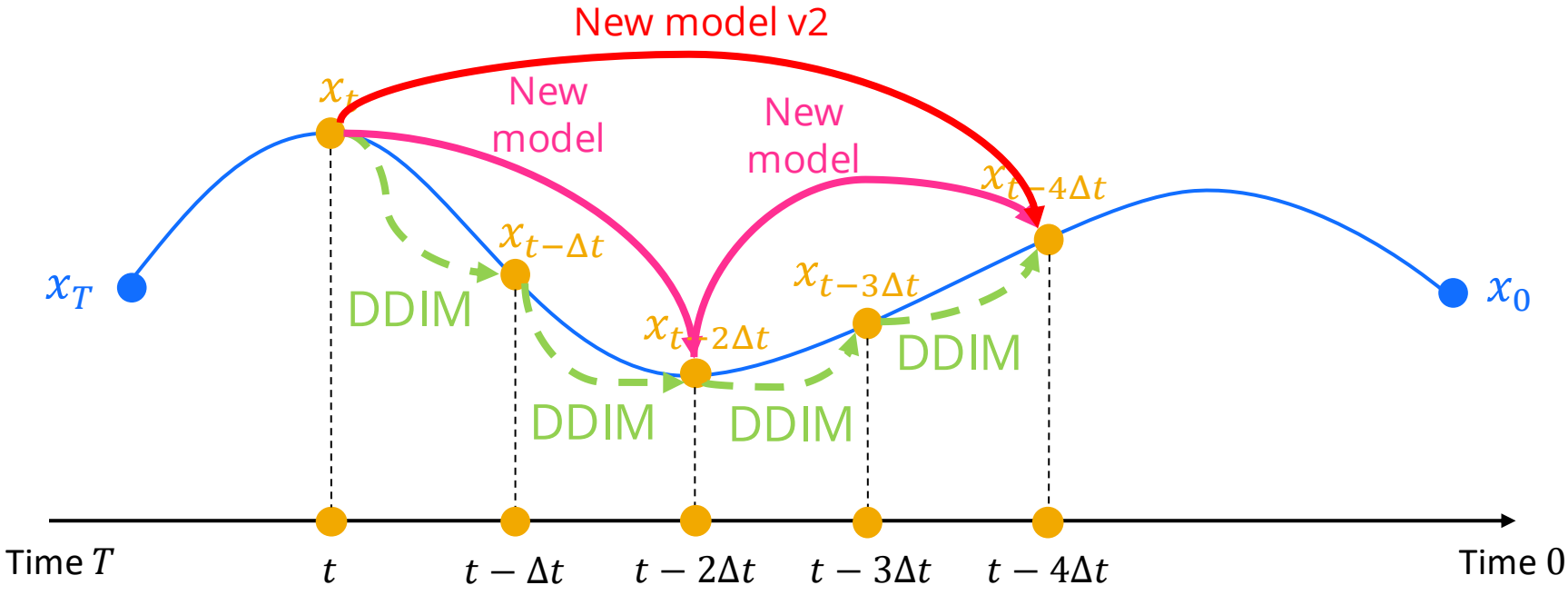
What if we just try to have 1 step in the new model to match with 2 steps in the original diffusion model



More speed up from this new model?

Carnegie Mellon University

What if we just try to have 1 step in another new model to match with 2 steps in the first new model



Now you have progressive distillation!

Carnegie Mellon University

Progressive distillation

Algorithm 1 Standard diffusion training

Require: Model $\hat{\mathbf{x}}_\theta(\mathbf{z}_t)$ to be trained
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$

while not converged **do**
- $\mathbf{x} \sim \mathcal{D}$ ▷ Sample data
- $t \sim U[0, 1]$ ▷ Sample time
- $\epsilon \sim N(0, I)$ ▷ Sample noise
- $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$ ▷ Add noise to data
- $\tilde{\mathbf{x}} = \mathbf{x}$ ▷ Clean data is target for $\hat{\mathbf{x}}$
- $\lambda_t = \log[\alpha_t^2/\sigma_t^2]$ ▷ log-SNR
- $L_\theta = w(\lambda_t)\|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$ ▷ Loss
- $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$ ▷ Optimization

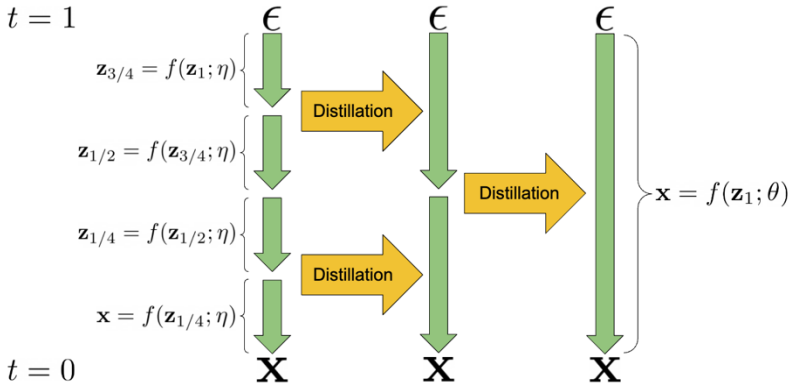
end while

Algorithm 2 Progressive distillation

Require: Trained teacher model $\hat{\mathbf{x}}_\eta(\mathbf{z}_t)$
Require: Data set $\mathcal{D}$
Require: Loss weight function $w()$
Require: Student sampling steps N

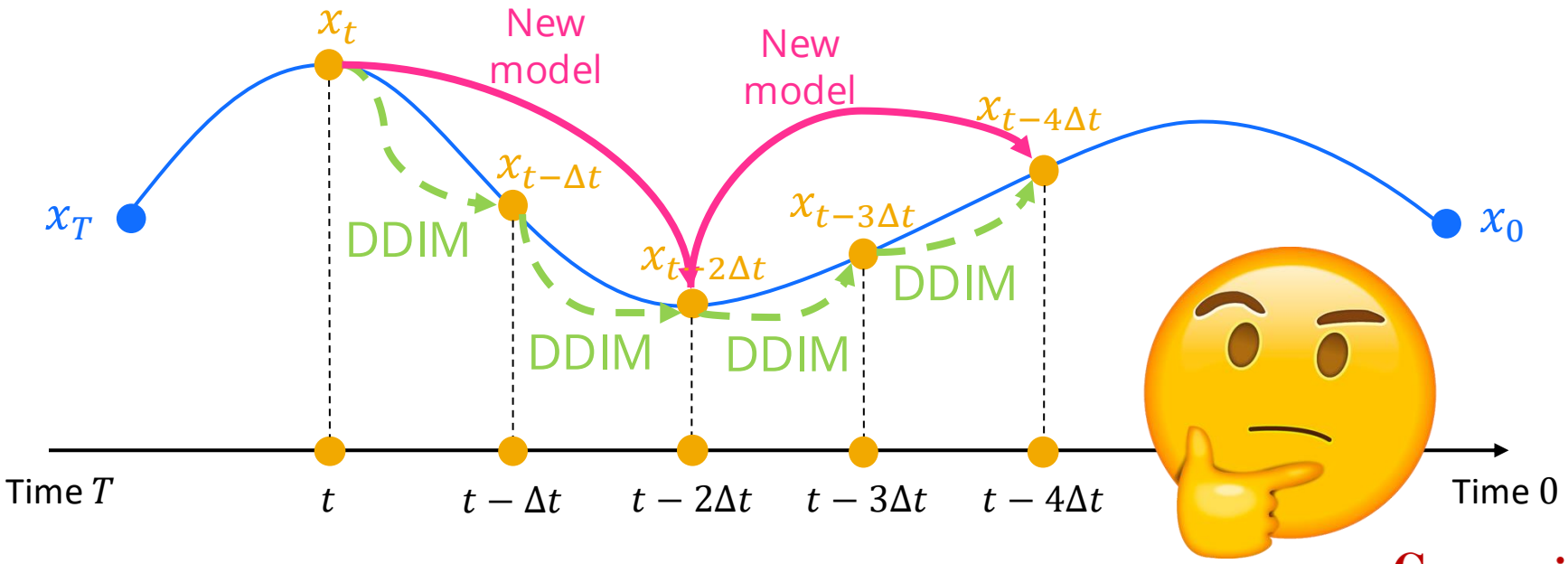
for K iterations **do**
- $\theta \leftarrow \eta$ ▷ Init student from teacher
- **while** not converged **do**
  - $\mathbf{x} \sim \mathcal{D}$
  - $t = i/N, \;\; i \sim Cat[1, 2, \ldots, N]$
  - $\epsilon \sim N(0, I)$
  - $\mathbf{z}_t = \alpha_t \mathbf{x} + \sigma_t \epsilon$
  - `# 2 steps of DDIM with teacher`
  - $t' = t - 0.5/N, \;\; t'' = t - 1/N$
  - $\mathbf{z}_{t'} = \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_t) + \frac{\sigma_{t'}}{\sigma_t}(\mathbf{z}_t - \alpha_t \hat{\mathbf{x}}_\eta(\mathbf{z}_t))$
  - $\mathbf{z}_{t''} = \alpha_{t''} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}) + \frac{\sigma_{t''}}{\sigma_{t'}}(\mathbf{z}_{t'} - \alpha_{t'} \hat{\mathbf{x}}_\eta(\mathbf{z}_{t'}))$
  - $\tilde{\mathbf{x}} = \frac{\mathbf{z}_{t''} - (\sigma_{t''}/\sigma_t)\mathbf{z}_t}{\alpha_{t''} - (\sigma_{t''}/\sigma_t)\alpha_t}$ ▷ Teacher $\hat{\mathbf{x}}$ target
  - $\lambda_t = \log[\alpha_t^2/\sigma_t^2]$
  - $L_\theta = w(\lambda_t)\|\tilde{\mathbf{x}} - \hat{\mathbf{x}}_\theta(\mathbf{z}_t)\|_2^2$
  - $\theta \leftarrow \theta - \gamma \nabla_\theta L_\theta$
- **end while**
- $\eta \leftarrow \theta$ ▷ Student becomes next teacher
- $N \leftarrow N/2$ ▷ Halve number of sampling steps

end for



Carnegie Mellon University

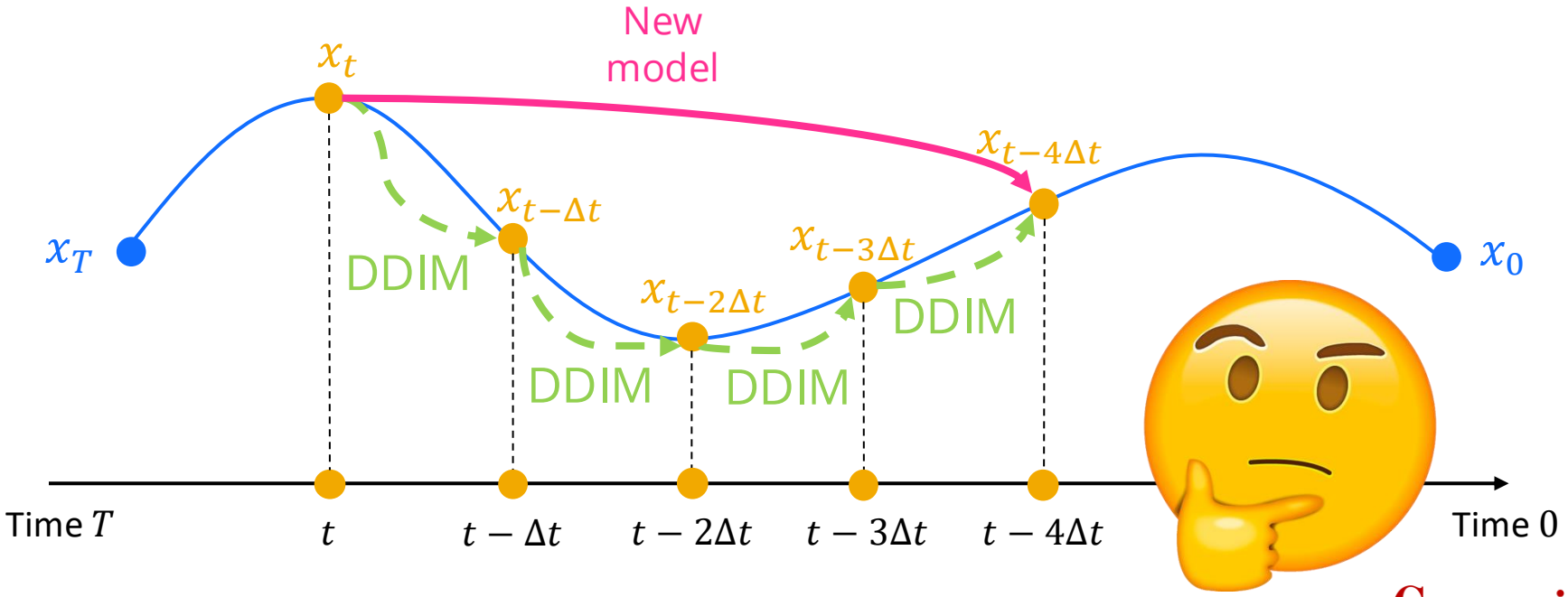
Progressive distillation



Why are we only matching 2 steps?

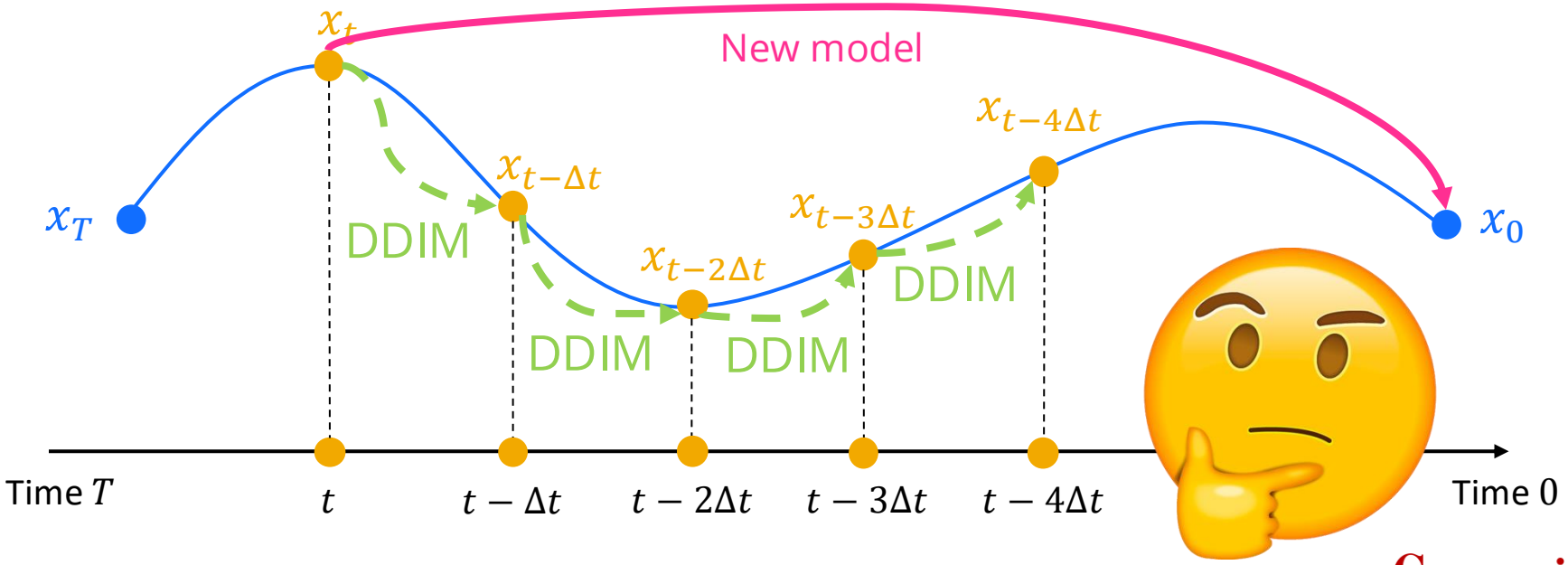
Carnegie Mellon University

Why can't we directly match 4 steps with the student model?



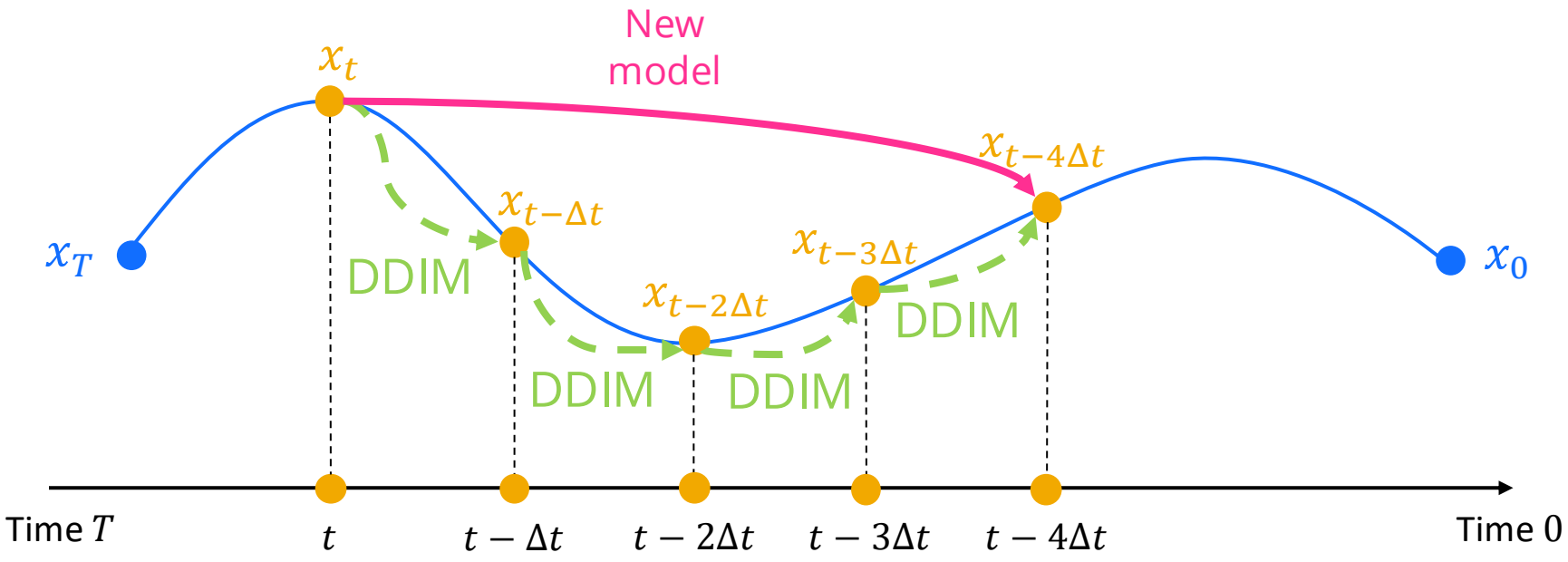
Carnegie Mellon University

Actually, why can't we directly match t steps so that we can directly get to the clean image from time t?



Carnegie Mellon University

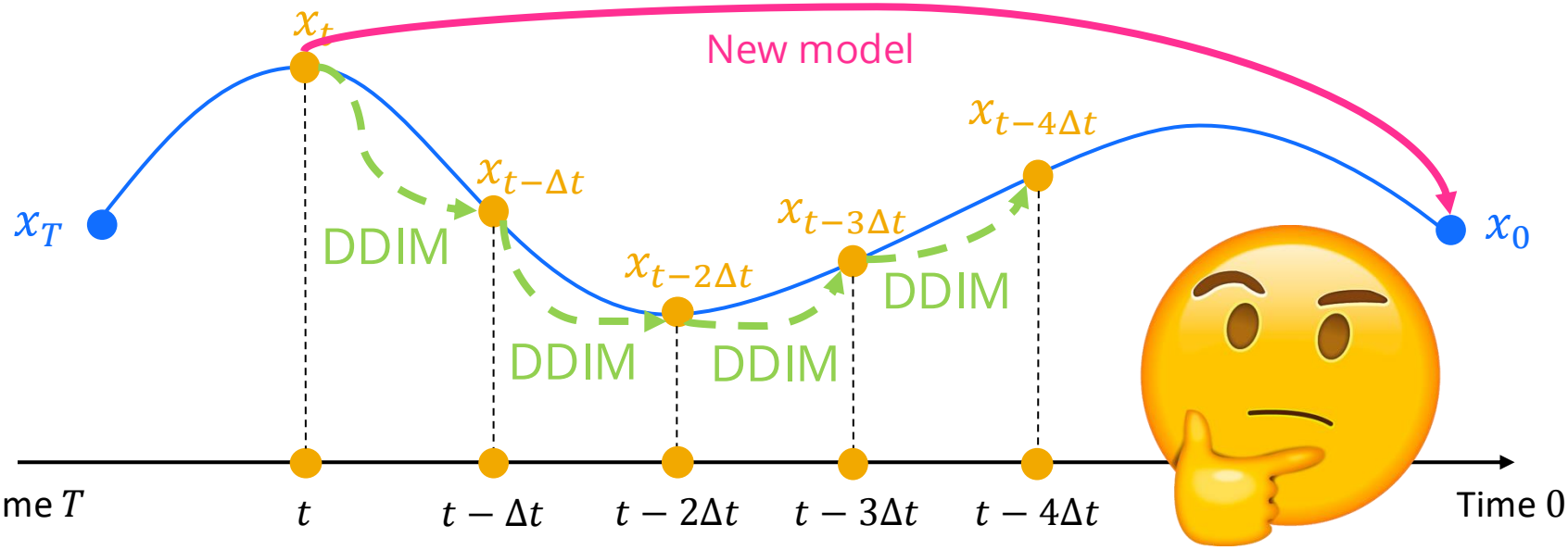
Training a student model to match 4 steps in teacher



You'll need to take 4 forward passes at each training step!

Carnegie Mellon University

Training a student to match arbitrary t steps in teacher

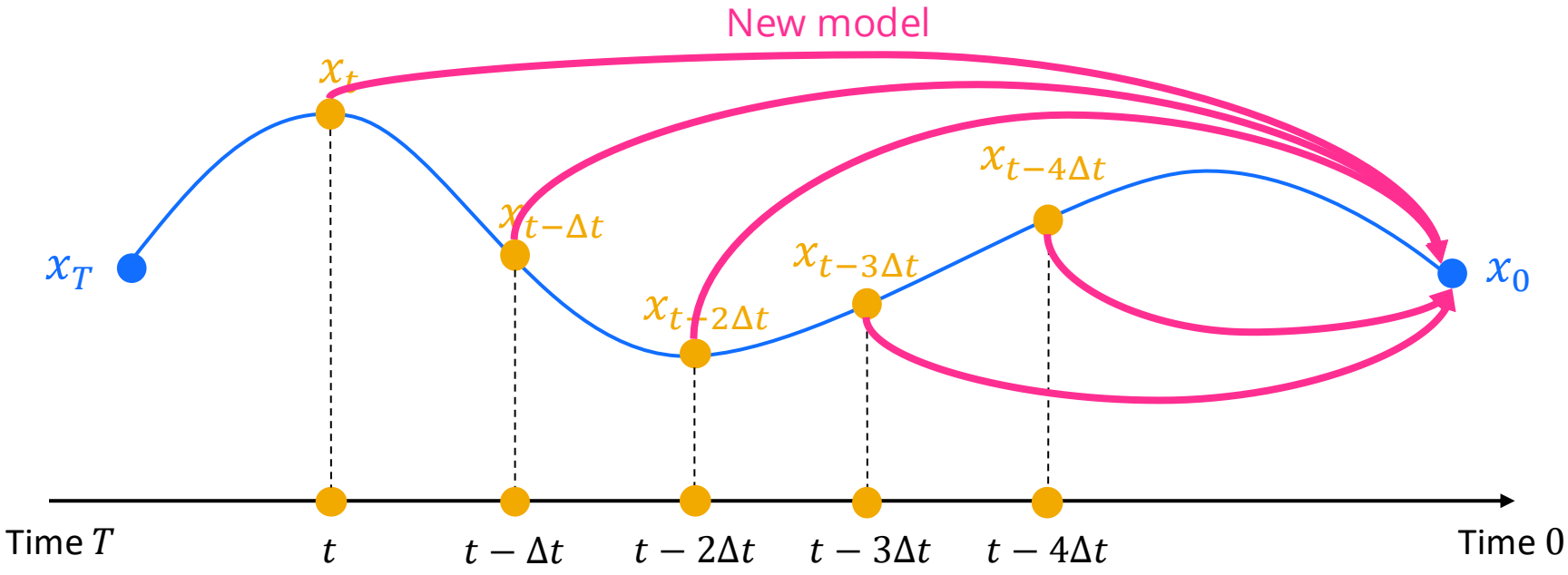


You'll need to take t forward passes at each training step

=> expensive and hard to parallelize

Carnegie Mellon University

Let's take another look at this problem



Carnegie Mellon University

We can rephrase what we want from this angle

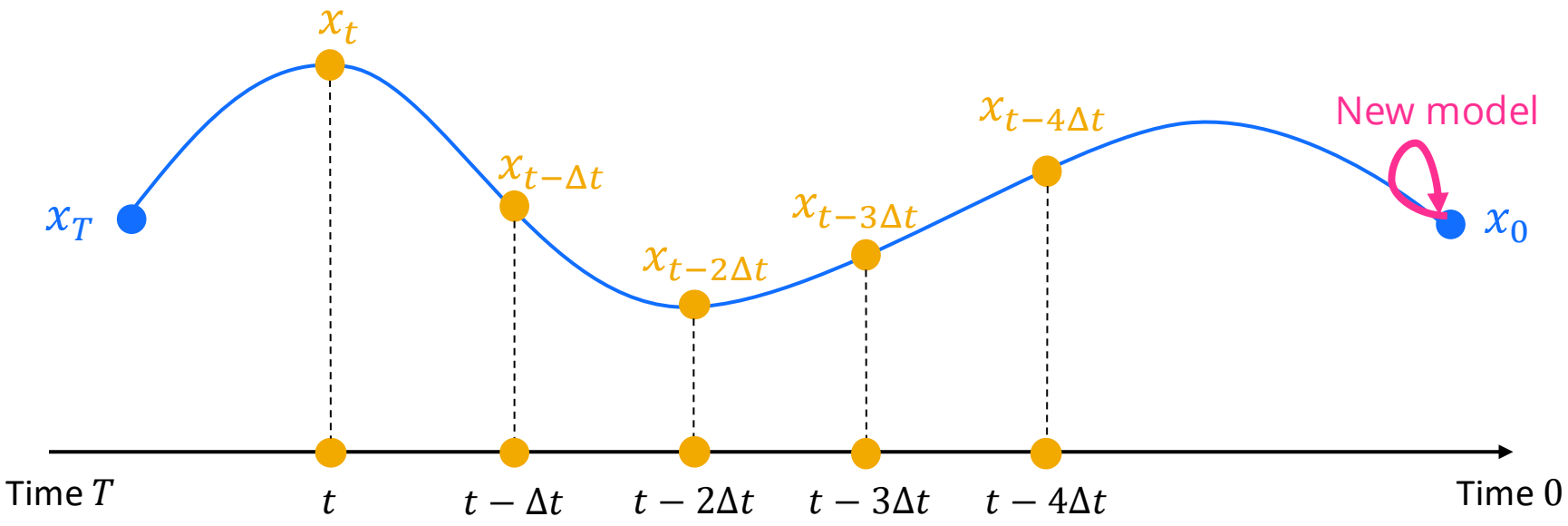
So what we really want is a new model that is **self-consistent:**

For any two points on the same PF-ODE trajectory produced by the pre-trained diffusion models, the new model should predict the same final clean data output

$$f_\theta(x_t, t) = f_\theta(x_s, s) \text{ for all } t, s \in [0,1]$$

But how do we make sure that the final outputs look like real images

What happens when t is close to 0



We can rephrase what we want from this angle

So what we really want is a new model that is **self-consistent:**

For any two points on the same PF-ODE trajectory produced by the pre-trained diffusion models, the new model should predict the same final clean data output

$$f_\theta(x_t, t) = f_\theta(x_s, s) \text{ for all } t, s \in [0,1]$$

In addition, the new model also need to satisfy the **boundary condition**:

$$f_\theta(x_0, 0) = x_0$$

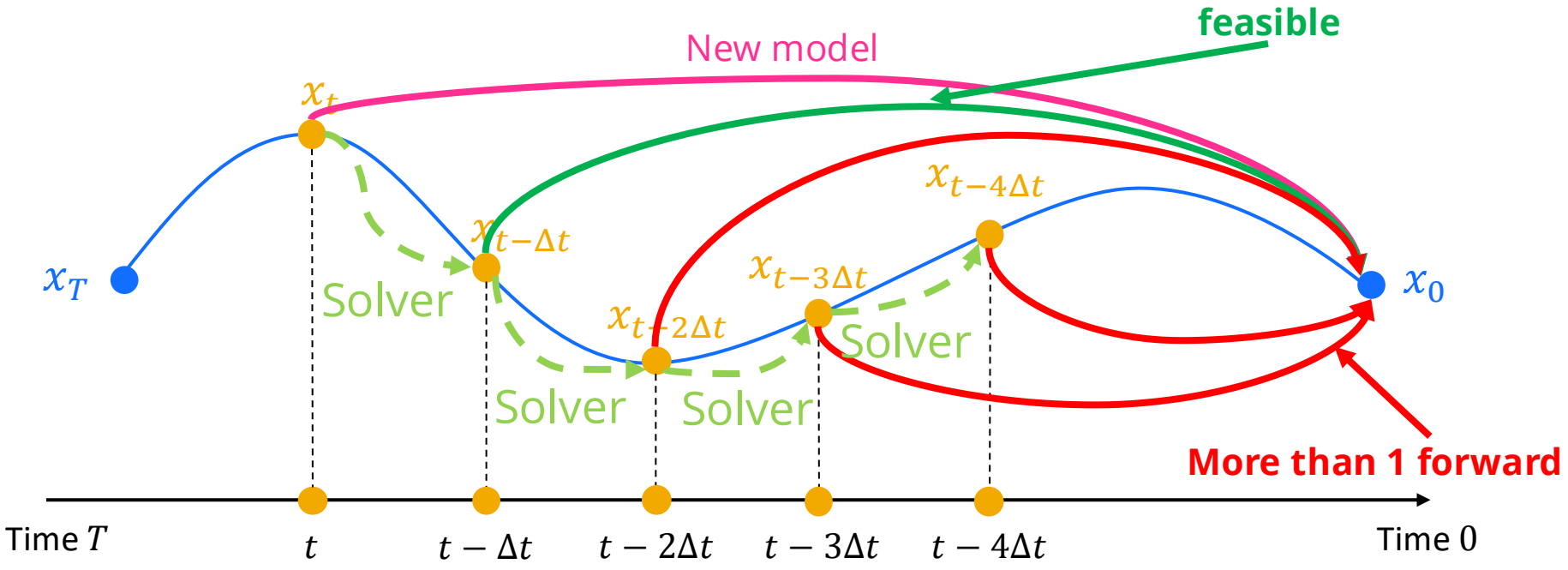
(In practice, we do $f_\theta(x_\delta, 0) = x_\delta$ for small enough δ to avoid singularity)

Now we have consistency models!

Carnegie Mellon University

Song et al. "Consistency Models". ICML 2023. https://arxiv.org/pdf/2303.01469

How to train a consistency model



Carnegie Mellon University

How to train a consistency model

So what we really want is a new model that is **self-consistent:**

For any two points on the same PF-ODE trajectory produced by the pre-trained diffusion models, the new model should predict the same final clean data output

$$f_\theta(x_t, t) = f_\theta(x_s, s) \text{ for all } t, s \in [0,1]$$

Let's denote the pre-trained diffusion teacher as ϕ and randomly sample a t

From x_t, we first solve one step with teacher to get $\hat{x}^\phi_{t-\Delta t} = x_t + \Delta t\ \text{Solver}(x_t, t; \phi)$

Then we match the f_θ prediction from x_t and $\hat{x}^\phi_{t-\Delta t}$

$$L(\theta; \phi) = E[\lambda(t) d(f_\theta(x_t, t), \text{Stopgrad}(f_\theta\left(\hat{x}^\phi_{t-\Delta t}, t - \Delta t\right)))]$$

Loss weighting

Distance, can be L2 or perceptual distance

Carnegie Mellon University

How to train a consistency model

In addition, the new model also need to satisfy the **boundary condition**:

$$f_\theta(x_0, 0) = f_\theta(x_0, 0)$$

(In practice, we do $f_\theta(x_\delta, 0) = x_\delta$ for small enough δ to avoid singularity)

From EDM, we can have preconditioning

$$D_\theta(\boldsymbol{x}; \sigma) = c_{\text{skip}}(\sigma)\, \boldsymbol{x} + c_{\text{out}}(\sigma)\, F_\theta\big(c_{\text{in}}(\sigma)\, \boldsymbol{x};\ c_{\text{noise}}(\sigma)\big),$$

Predicts clean data x0 | Skip connection weight | Network output weight | Trained network

So as long as we have $c_{\text{skip}}(\delta) = 1$, $c_{\text{out}}(\delta) = 0$, we satisfy boundary condition

Can we train a consistency model without a pre-trained diffusion model?

How to train a consistency model from scratch

So what we really want is a new model that is **self-consistent:**

For any two points on the same ~~PF-~~ODE trajectory ~~produced by the pre-trained diffusion models~~, the new model should predict the same final clean data output

$$f_\theta(x_t, t) = f_\theta(x_s, s) \text{ for all } t, s \in [0,1]$$

Let's first produce a pair of data x_t and $x_{t-\Delta t}$ from the same ODE trajectory

The easiest way is to fix the same noise z, and have $x_t = x_0 + tz$, $x_{t-\Delta t} = x_0 + (t - \Delta t)z$

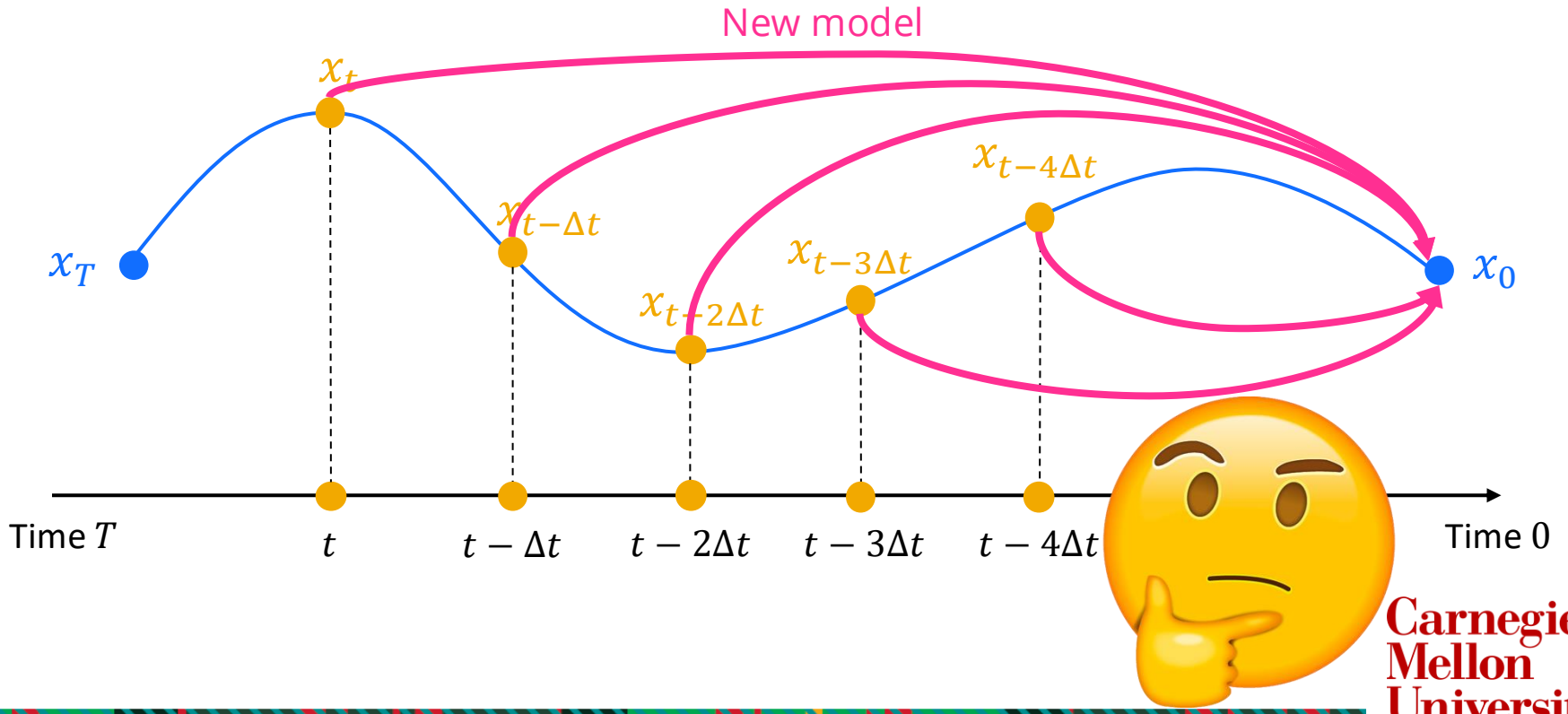
Then we match the f_θ prediction from x_t and $x_{t-\Delta t}$

$$L(\theta; \phi) = E[\lambda(t) d(f_\theta(x_t, t), \text{Stopgrad}(f_\theta(x_{t-\Delta t}, t - \Delta t)))]$$

Carnegie Mellon University

Song et al. "Consistency Models". ICML 2023. https://arxiv.org/pdf/2303.01469

What problems does consistency model have

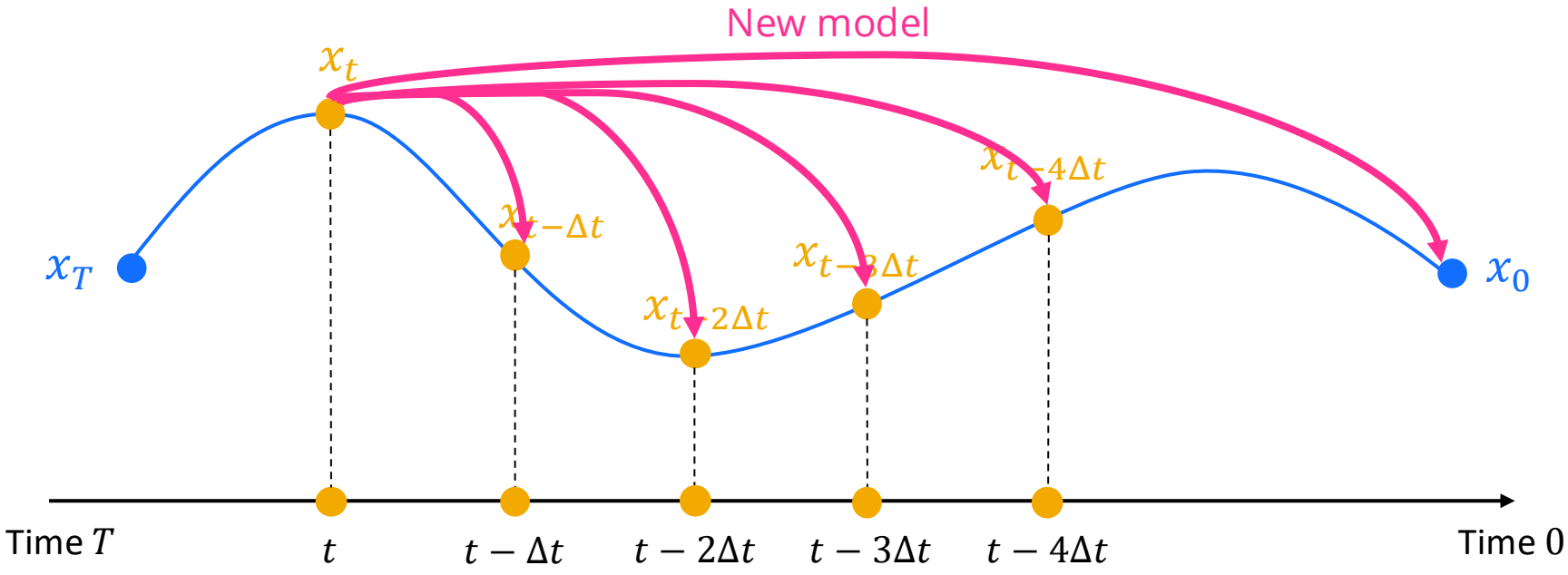


Carnegie Mellon University

What problems does consistency model have

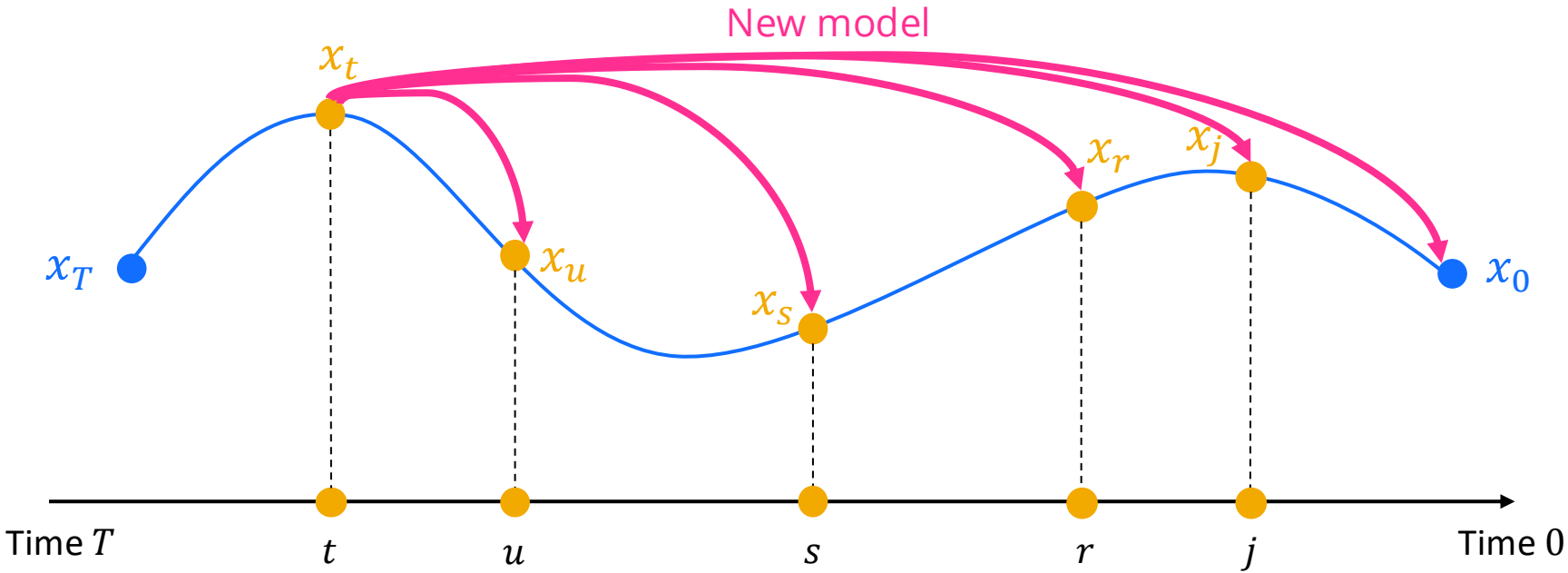
- The model is only trained to predict the clean data x_0
    - => the model doesn't naturally do well with multi-step sampling
    - => difficult to tradeoff quality v.s. speed (i.e. can't scale at inference time)
- Completely lost the ability to calculate exact loglikelihood

How about instead of learning to jump to the end, we learn to jump from anywhere to anywhere



Carnegie Mellon University

How about instead of learning to jump to the end, we learn to jump from anywhere to anywhere

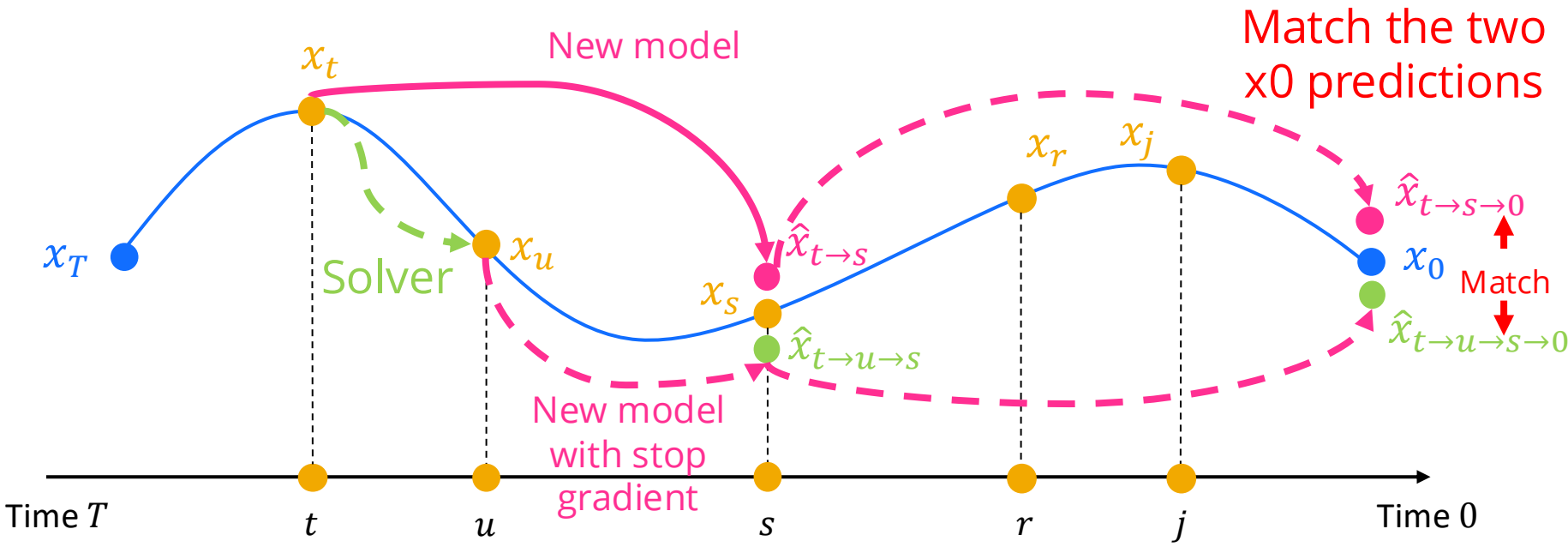


This is a consistency trajectory model!

Carnegie Mellon University

Kim et al. “Consistency Trajectory Models: Learning Probability Flow ODE Trajectory of Diffusion”. ICLR 2024. https://arxiv.org/pdf/2310.02279

How to train a consistency trajectory model



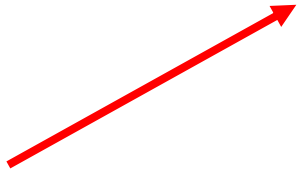
Carnegie Mellon University

New ”boundary” condition

Because we don’t always match to the end point now, we need to change the boundary condition into a “tangent” condition

In other words, when we are trying to jump from t to t, we should match the score function

In other words, we should just add a normal diffusion/score matching loss



This also gives us exact loglikelihood back!

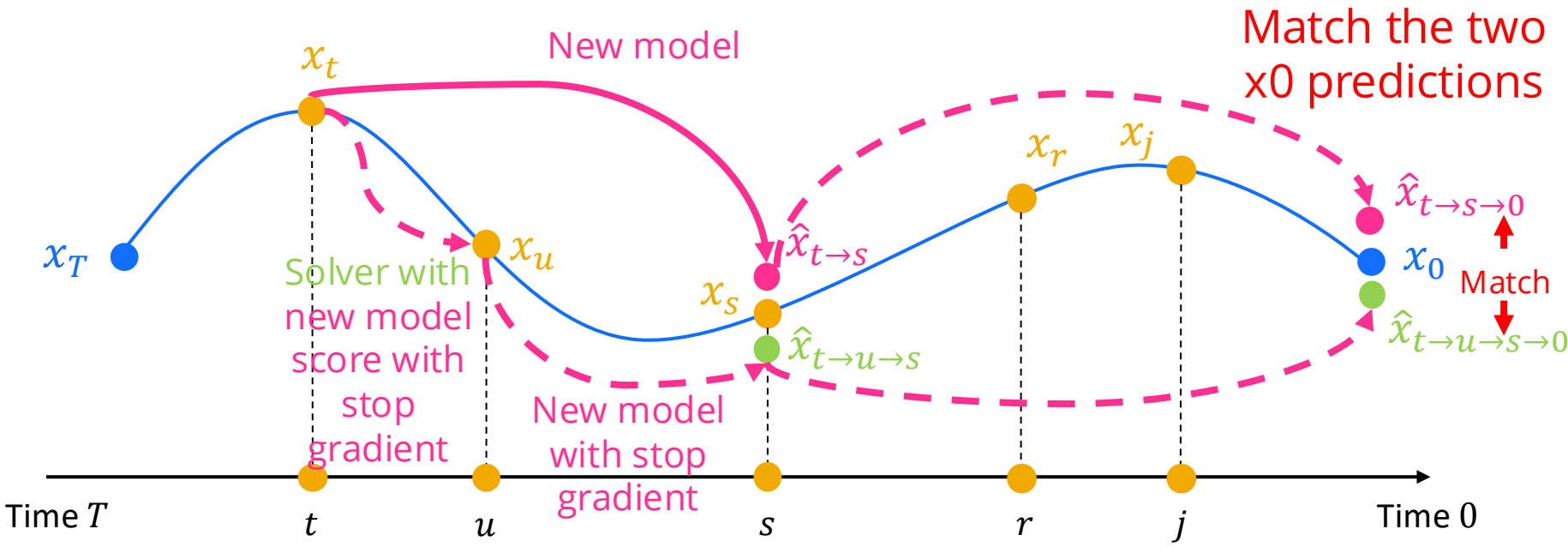
How to train a consistency trajectory model

The loss function is basically

Loss = CTM loss + Score matching loss + GAN loss

Adding GAN loss gives a lot of boost in FID!

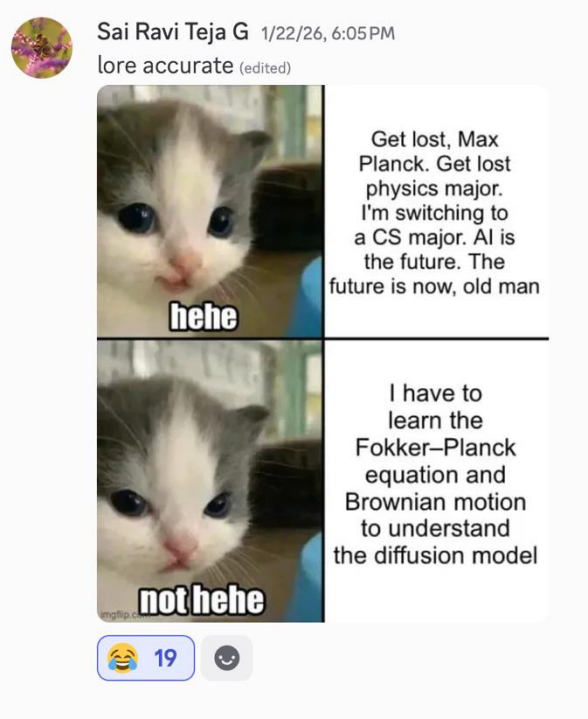
How to train a consistency trajectory model from scratch



Carnegie Mellon University

Why does this work?

Turns out as per usual, physics people already have answers for us long time ago



Screenshot from our Discord server

Carnegie Mellon University

There is this notion in math/physics called flow map

Chapter 5

Flow Maps and Dynamical Systems

Main concepts: In this chapter we introduce the concepts of continuous and discrete dynamical systems on phase space. Keywords are: classical mechanics, phase space, vector field, linear systems, flow maps, dynamical systems

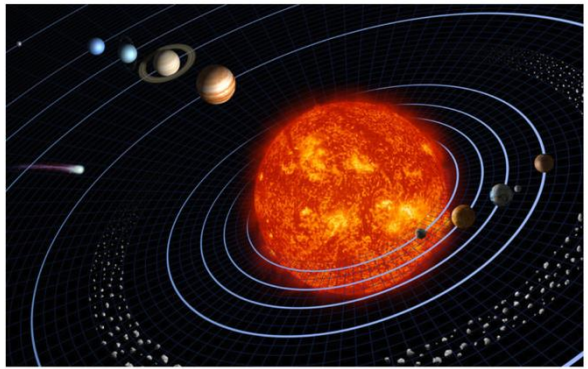
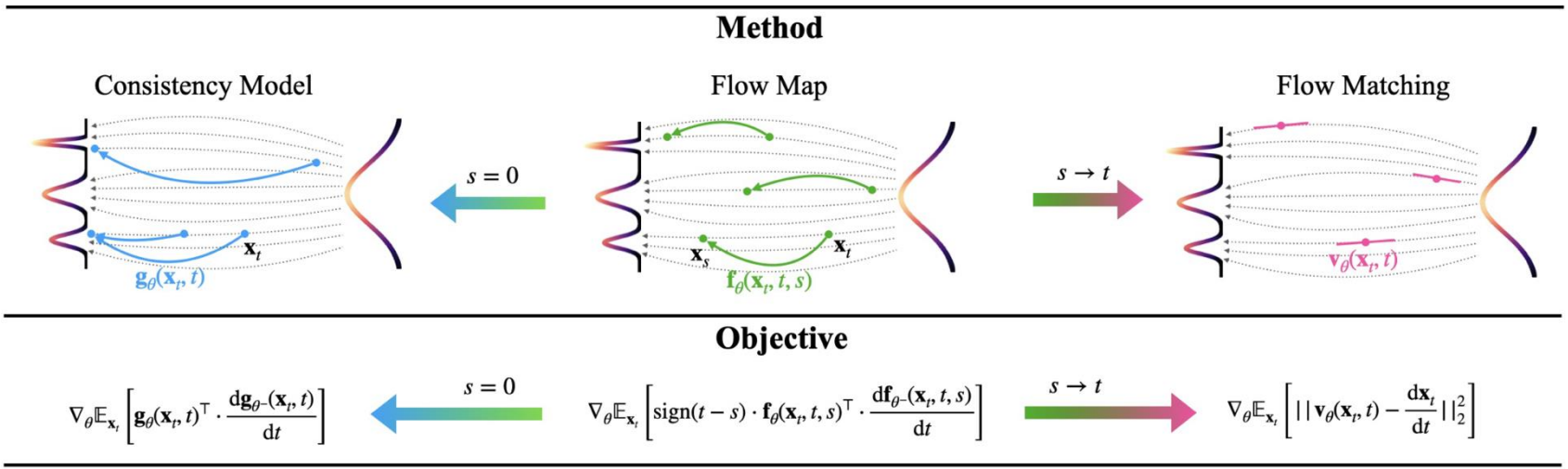


Figure 5.1: A solar system is well modelled by classical mechanics. (source: Wikimedia Commons)

Carnegie Mellon University

Screenshot from https://webspace.science.uu.nl/~frank011/Classes/numwisk/ch5.pdf

Flow maps (the flow version of CTM)



Definition 2.1 (Flow Map). Given an ODE $\mathrm{d}x_t = v(x_t, t)\mathrm{d}t$, the flow map $\Phi : \mathbb{R}^d \times [0,1]^2 \to \mathbb{R}^d$ is the solution operator that maps any state at time t to its corresponding state at time s:

$$\Phi(x_t, t, s) = x_t + \int_t^s v(x_\tau, \tau)d\tau = x_s \tag{2.5}$$

Sabour et al. Align Your Flow: Scaling Continuous-Time Flow Map Distillation. NeurIPS 2025. https://arxiv.org/pdf/2506.14603

Ai et al. Joint Distillation for Fast Likelihood Evaluation and Sampling in Flow-based Models. ICLR 2026. https://arxiv.org/pdf/2512.02636

Carnegie Mellon University

How to build a valid flow map

Let $u(x,t,s)$ denotes the flow map displacement from x at t to s, $v(x,t)$ is the instantaneous velocity, $\Phi(x,t,s) = x + (t-s)u(x,t,s) \approx x_s$ is the flow map jump

1. You need to satisfy the **tangent condition:** $u(x,t,t) = v(x,t)$
2. You need to also satisfy one of the following three conditions:
    1) **Lagrangian condition:** $\partial_s \Phi(x,t,s) = u(\Phi(x,t,s),s,s)$
    2) **Eulerian condition:** $\partial_t \Phi(x,t,s) + \nabla_x \Phi(x,t,s)u(x,t,t) = 0$
    3) **Semigroup condition:** $\Phi(x,t,s) = \Phi(\Phi(x,t,r),r,s)$

Carnegie Mellon University

Boffi et al. "How to build a consistency model: Learning flow maps via self-distillation". NeurIPS 2025. https://arxiv.org/pdf/2505.18825

Analogy: a rubber duck flowing down the river

Suppose I am a magician that can teleport a rubber duck in a particular river, in other words, for any duck flowing down this particular river, I can teleport the duck so that the duck's location in the river will be the same as if the duck take the normal river flow down

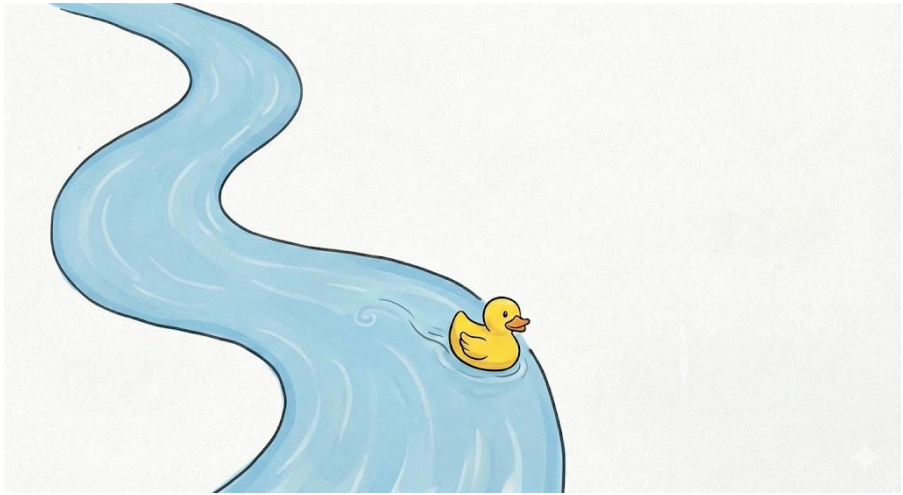
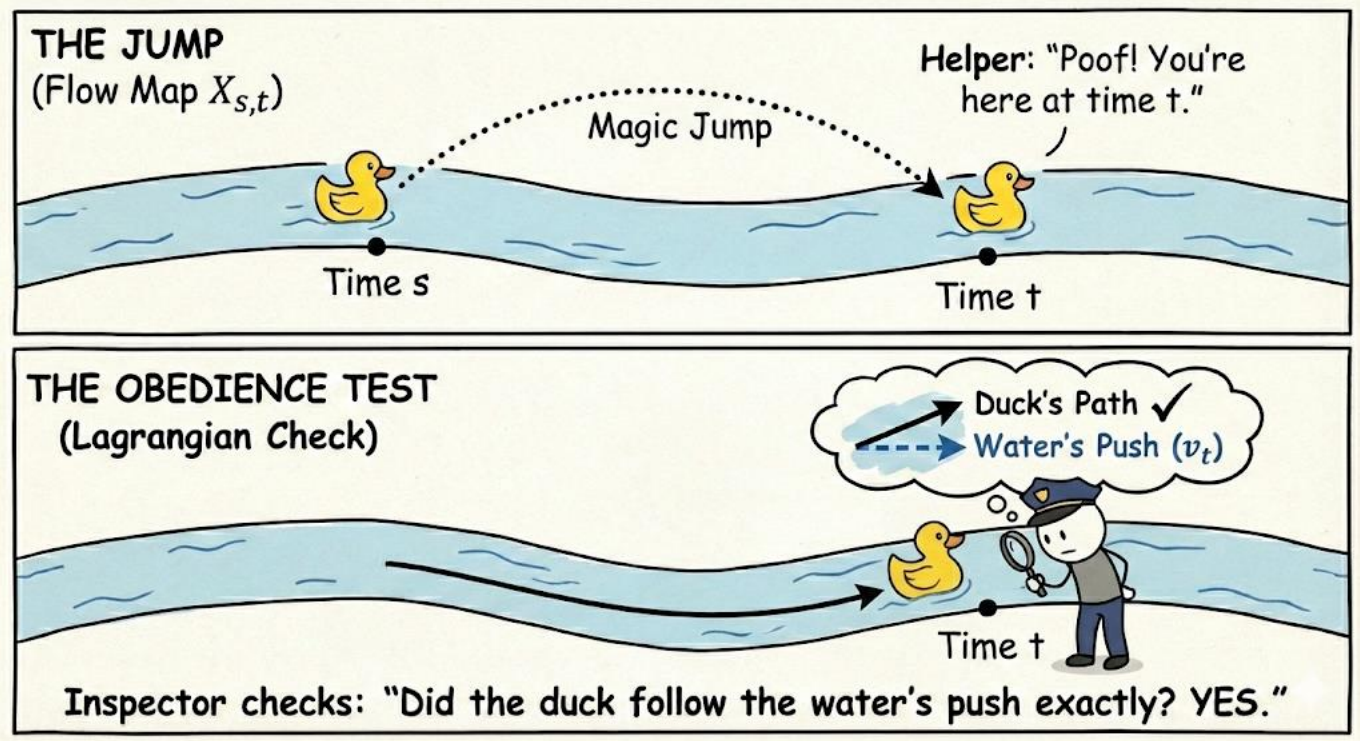


Image generated by Nano Banana

Lagrangian condition: follow a rubber duck in the flow

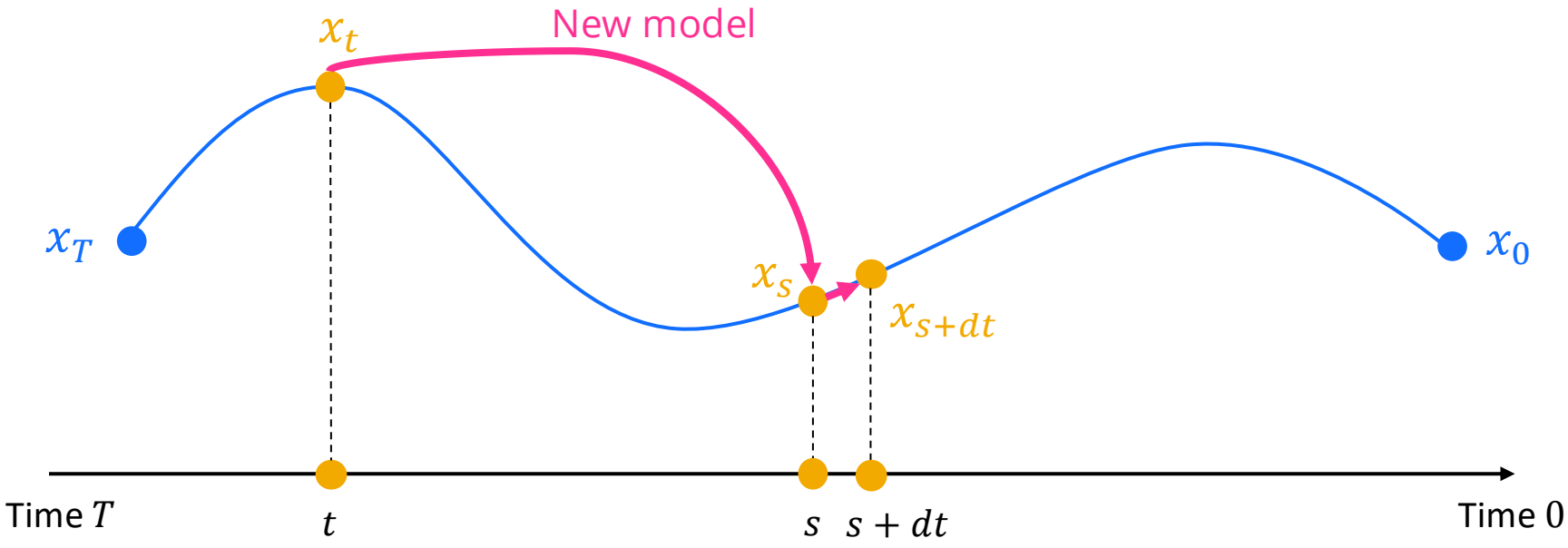


The **instantaneous flow** == The **infinitesimally small flow map jump** for this duck right here

Carnegie Mellon University

Image generated by Nano Banana

Lagrangian condition: follow a rubber duck in the flow



Carnegie Mellon University

Eulerian: no matter what part of the rive you start jumping the duck, it will always end up at the same place if you make the jump

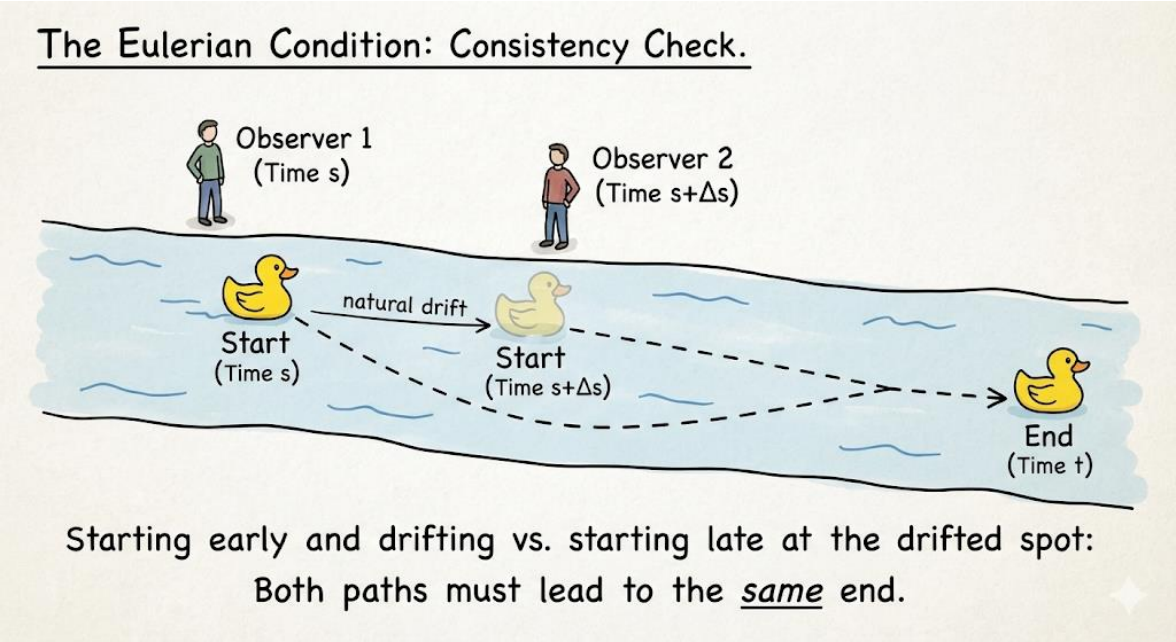
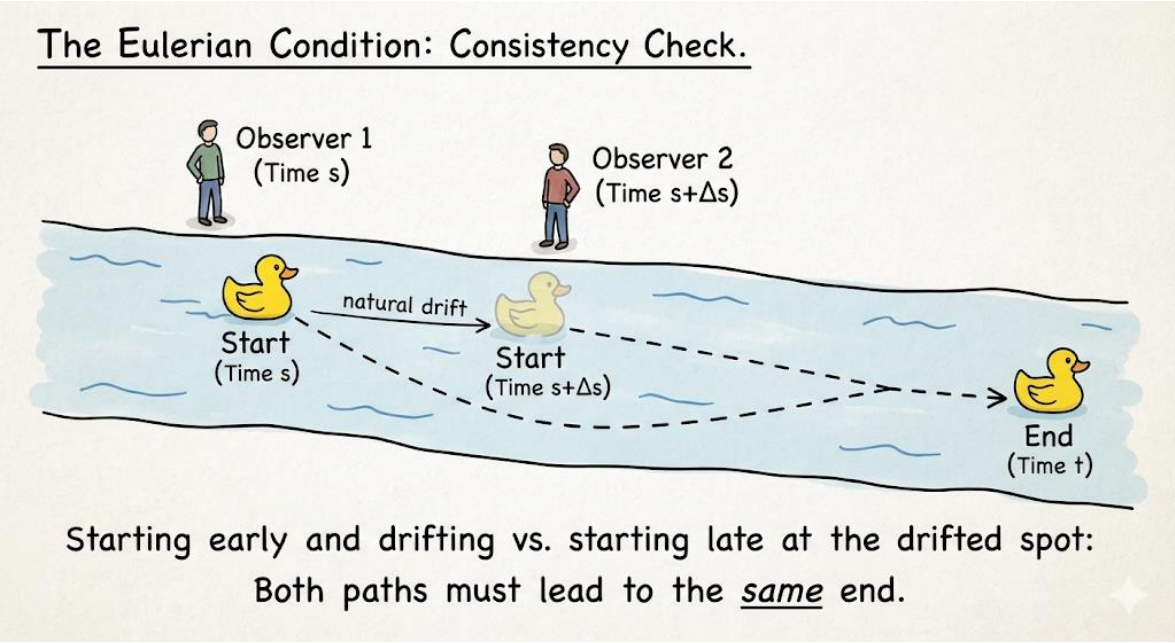


Image generated by Nano Banana

Carnegie Mellon University

Eulerian: the total derivative w.r.t. t is 0

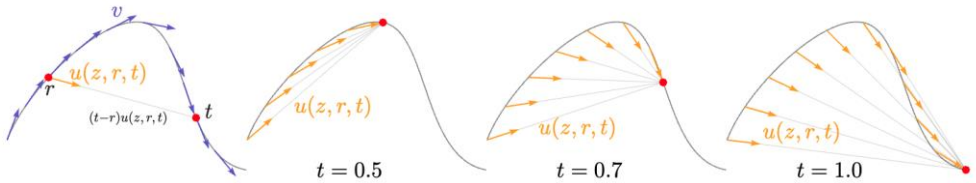
$$\partial_t \Phi(x,t,s) + \nabla_x \Phi(x,t,s) u(x,t,t) = \frac{d}{dt} \Phi(x,t,s) = 0$$



In other words, changing the starting time doesn't matter as long as the duck is following the same flow

Carnegie Mellon University

Eulerian Instantiation: MeanFlow



The MeanFlow Identity. To have a formulation amenable to training, we rewrite Eq. (3) as:

$$(t-r)u(z_t,r,t) = \int_r^t v(z_\tau,\tau)d\tau. \tag{4}$$

Now we differentiate both sides with respect to t, treating r as independent of t. This leads to:

$$\frac{d}{dt}(t-r)u(z_t,r,t) = \frac{d}{dt}\int_r^t v(z_\tau,\tau)d\tau \implies u(z_t,r,t) + (t-r)\frac{d}{dt}u(z_t,r,t) = v(z_t,t), \tag{5}$$

where the manipulation of the left hand side employs the product rule and the right hand side uses the fundamental theorem of calculus[2]. Rearranging terms, we obtain the identity:

$$\underbrace{u(z_t,r,t)}_{\text{average vel.}} = \underbrace{v(z_t,t)}_{\text{instant. vel.}} - (t-r)\underbrace{\frac{d}{dt}u(z_t,r,t)}_{\text{time derivative}} \tag{6}$$

Carnegie Mellon University

MeanFlow is an Eulerian flow map

Eulerian condition: $\partial_t \Phi(x,t,s) + \nabla_x \Phi(x,t,s) u(x,t,t) = 0$

$$(s-t)\frac{d}{dt}u(x_t,t,s) + v(x_t,t) = u(x_t,t,s)$$

$$(s-t)\left(\partial_t u(x_t,t,s) + \nabla_{x_t} u(x_t,t,s) v(x_t,t)\right) + v(x_t,t) = u(x_t,t,s)$$

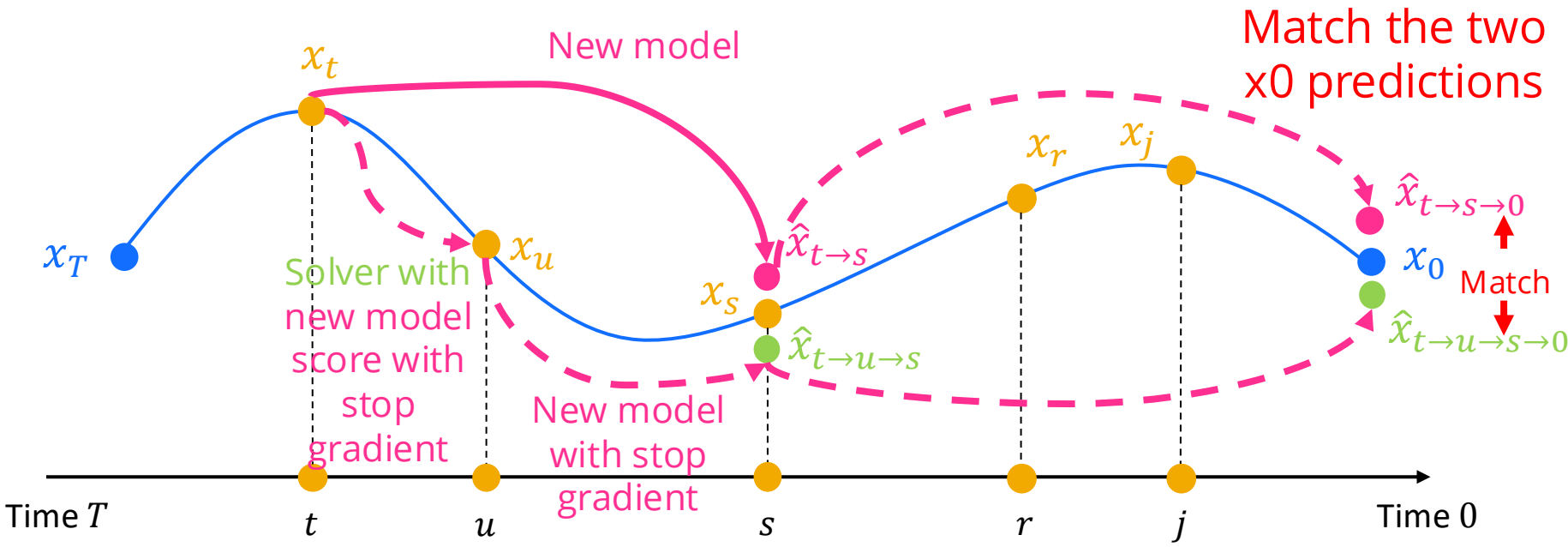
$$-u(x_t,t,s) + (s-t)\partial_t u(x_t,t,s) + (I + (s-t)\nabla_{x_t} u(x_t,t,s))v(x_t,t) = 0$$

$$\partial_t(x_t + (s-t)u(x_t,t,s)) + \nabla_{x_t}(x_t + (s-t)u(x_t,t,s))v(x_t,t) = 0$$

$$\partial_t \Phi(x_t,t,s) + \nabla_{x_t} \Phi(x_t,t,s) u(x_t,t,t) = 0$$

$$\lim_{t \to s} u(x_t,t,t) = v(x_t,t)$$

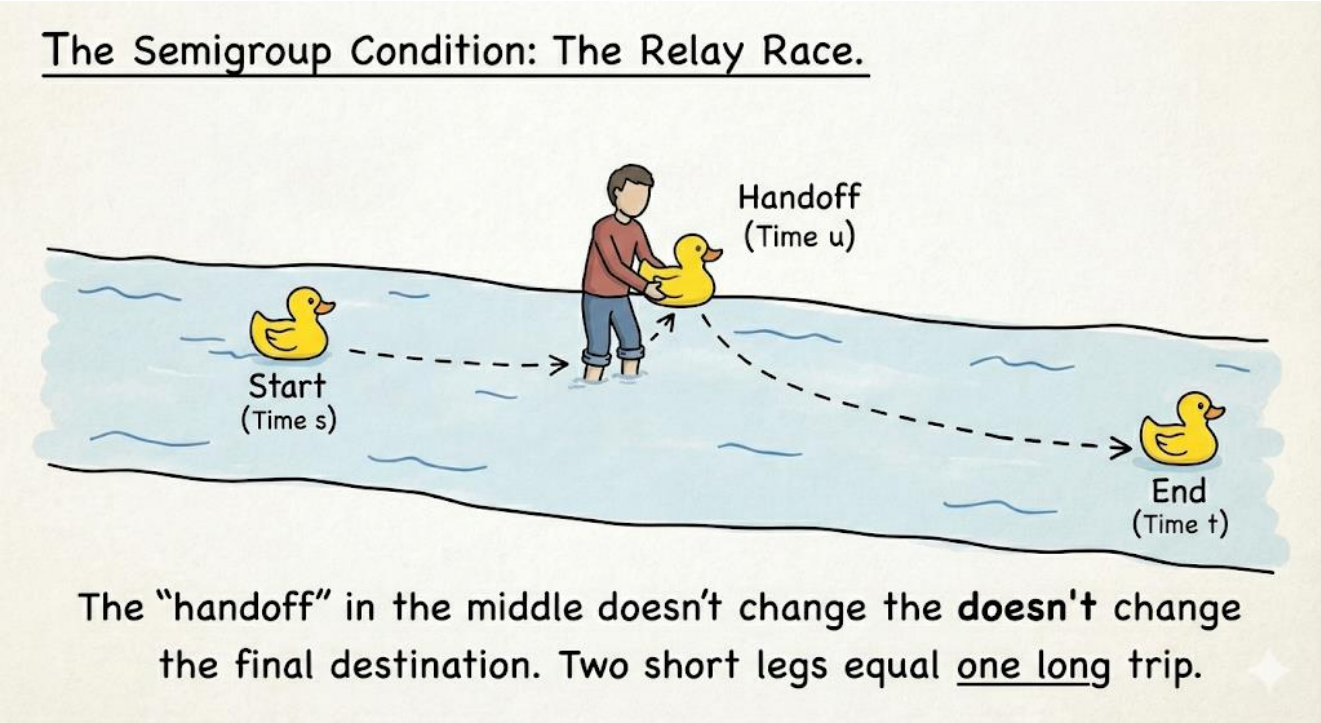
CTM is Eulerian too



Score matching loss guarantees the tangent condition

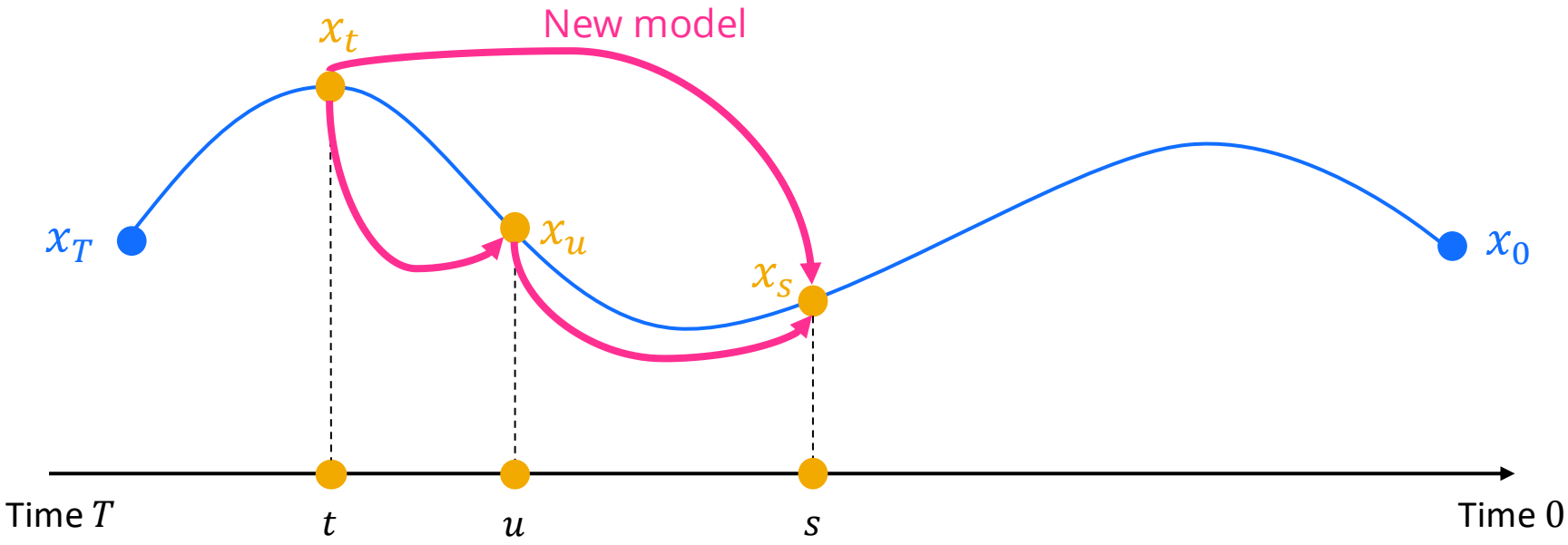
Carnegie Mellon University

Semigroup: making two jumps is the same as making one large jump

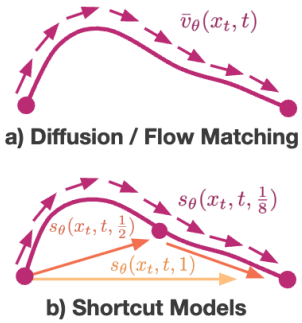


Carnegie Mellon University

Semigroup: making two jumps is the same as making one large jump



Semigroup Instantiation: Shortcut models



Algorithm 1 Shortcut Model Training

while not converged **do**

$x_0 \sim \mathcal{N}(0, I),\ x_1 \sim D,\ (d, t) \sim p(d, t)$

$x_t \leftarrow (1-t)\,x_0 + t\,x_1$ — Noise data point

for first k batch elements **do**

$s_{\text{target}} \leftarrow x_1 - x_0$ — Flow-matching target

$d \leftarrow 0$

for other batch elements **do**

$s_t \leftarrow s_\theta(x_t, t, d)$ — First small step

$x_{t+d} \leftarrow x_t + s_t\,d$ — Follow ODE

$s_{t+d} \leftarrow s_\theta(x_{t+d}, t+d, d)$ — Second small step

$s_{\text{target}} \leftarrow \text{stopgrad}(s_t + s_{t+d})/2$ — Self-consistency target

$\theta \leftarrow \nabla_\theta ||s_\theta(x_t, t, 2d) - s_{\text{target}}||^2$

Make sure the tangent condition holds for sampling

Make sure the semigroup property holds for likelihood

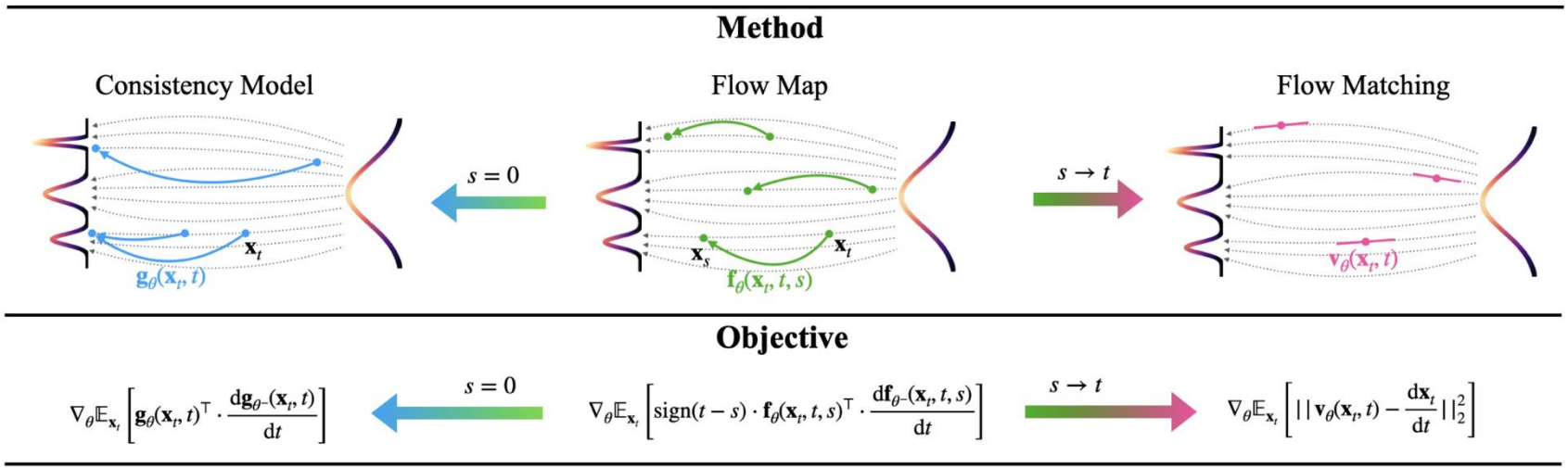
Tangent condition: $u(x, t, t) = v(x, t)$

Semigroup condition: $\Phi(x, t, s) = \Phi(\Phi(x, t, r), r, s)$

Carnegie Mellon University

Are flow maps perfect now?

Flow maps only accelerate the sampling part! Likelihood evaluation is still very very slow!!



Definition 2.1 (Flow Map). Given an ODE $\mathrm{d}x_t = v(x_t, t)\mathrm{d}t$, the flow map $\Phi : \mathbb{R}^d \times [0,1]^2 \to \mathbb{R}^d$ is the solution operator that maps any state at time t to its corresponding state at time s:

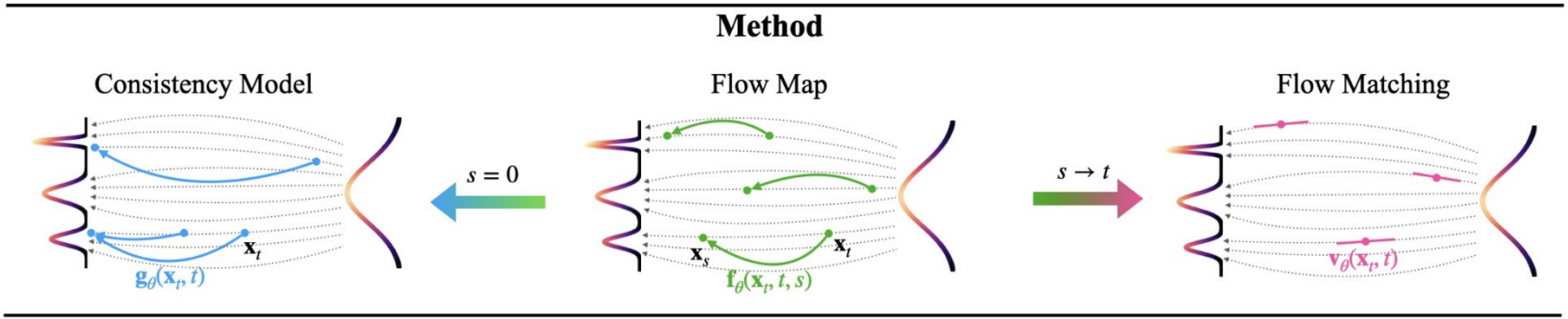
$$\Phi(x_t, t, s) = x_t + \int_t^s v(x_\tau, \tau)d\tau = x_s \tag{2.5}$$

Sabour et al. Align Your Flow: Scaling Continuous-Time Flow Map Distillation. NeurIPS 2025. https://arxiv.org/pdf/2506.14603

Ai et al. Joint Distillation for Fast Likelihood Evaluation and Sampling in Flow-based Models. ICLR 2026. https://arxiv.org/pdf/2512.02636

Carnegie Mellon University

Why flow maps fail to calculate likelihood fast



Definition 2.1 (Flow Map). Given an ODE $\mathrm{d}x_t = v(x_t, t)\mathrm{d}t$, the flow map $\Phi : \mathbb{R}^d \times [0,1]^2 \to \mathbb{R}^d$ is the solution operator that maps any state at time t to its corresponding state at time s:

$$\Phi(x_t, t, s) = x_t + \int_t^s v(x_\tau, \tau)d\tau = x_s \tag{2.5}$$

It only learns to jump in the sampling space and forgets about the likelihood!

Calculating the likelihood still needs solving the instantaneous divergence ODE (numerical integration with 100-1000 steps)!

Carnegie Mellon University

Sabour et al. Align Your Flow: Scaling Continuous-Time Flow Map Distillation. NeurIPS 2025. https://arxiv.org/pdf/2506.14603

~~Why flow maps fail to calculate likelihood fast~~ Actually …

Definition 2.1 (Flow Map). Given an ODE $\mathrm{d}x_t = v(x_t, t)\mathrm{d}t$, the flow map $\Phi : \mathbb{R}^d \times [0,1]^2 \to \mathbb{R}^d$ is the solution operator that maps any state at time t to its corresponding state at time s:

$$\Phi(x_t, t, s) = x_t + \int_t^s v(x_\tau, \tau)d\tau = x_s \tag{2.5}$$

It only learns to jump in the sampling space and forgets about the likelihood!

Calculating the likelihood still needs **solving** the instantaneous divergence **ODE** (numerical **integration** with 100-1000 steps)!

Sampling => integrating some ODE => can learn jumps via flow maps

Likelihood calculation => integrating some ODE => can learn jumps via flow maps as well???

Carnegie Mellon University

Ai et al. Joint Distillation for Fast Likelihood Evaluation and Sampling in Flow-based Models. ICLR 2026. https://arxiv.org/pdf/2512.02636

~~Why flow maps fail to calculate likelihood fast~~ Actually …

Definition 2.1 (Flow Map). Given an ODE $\mathrm{d}x_t = v(x_t, t)\mathrm{d}t$, the flow map $\Phi : \mathbb{R}^d \times [0,1]^2 \to \mathbb{R}^d$ is the solution operator that maps any state at time t to its corresponding state at time s:

$$\Phi(x_t, t, s) = x_t + \int_t^s v(x_\tau, \tau)d\tau = x_s \tag{2.5}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix}\hat{x}_t \\ \log p_{t;\theta}(\hat{x}_t)\end{bmatrix} = \begin{bmatrix}v_\theta(\hat{x}_t, t) \\ -\operatorname{div}(v_\theta(\hat{x}_t, t)).\end{bmatrix}$$

Sampling => integrating some ODE => can learn jumps via flow maps

Likelihood => integrating some ODE => can learn jumps via flow maps

A coupled system of ODE => can learn to jump together ???

Carnegie Mellon University

Ai et al. Joint Distillation for Fast Likelihood Evaluation and Sampling in Flow-based Models. ICLR 2026. https://arxiv.org/pdf/2512.02636

F2D2: joint flow maps for both sampling and likelihood

Let $z_t = \log p_t(x_t) \in \mathbb{R}$ and $\hat{z}_t = \log p_{t;\theta}(\hat{x}_t) \in \mathbb{R}$ denote the log likelihood, we can then separately parametrize the flow maps for the two subsystems in Eq. (2.3) as

$$\begin{aligned} \Phi_{X;\theta_X}(\hat{x}_t, t, s) &= \hat{x}_t + (s-t)u_{\theta_X}(\hat{x}_t, t, s), \\ \Phi_{Z;\theta_Z}(\hat{x}_t, \hat{z}_t, t, s) &= \hat{z}_t + (s-t)D_{\theta_Z}(\hat{x}_t, t, s) \end{aligned} \tag{3.2}$$

Here $u_{\theta_X}(\hat{x}_t, t, s)$ still estimates the average velocity, and $D_{\theta_Z}(x_t, t, s)$ approximates the average divergence $D_{\theta_Z}(x_t, t, s) \approx -\frac{1}{s-t}\int_t^s \operatorname{div}(v(x_\tau, \tau))d\tau$ along the true trajectory between t and s.

Therefore, denoting the joint state at time t as $y_t = (x_t, z_t)^\top$, we can then parametrize the joint flow map using shared parameter θ as

$$\begin{aligned} \Phi_{Y;\theta}(\hat{y}_t, t, s) &= \begin{bmatrix} \Phi_X(\hat{x}_t, t, s) \\ \Phi_Z(\hat{x}_t, \hat{z}_t, t, s) \end{bmatrix} = \hat{y}_t + (s-t)f_\theta(\hat{x}_t, t, s), \\ f_\theta(\hat{x}_t, t, s) &= \begin{bmatrix} u_\theta(\hat{x}_t, t, s) \\ D_\theta(\hat{x}_t, t, s) \end{bmatrix} \end{aligned} \tag{3.3}$$

Carnegie Mellon University

Ai et al. Joint Distillation for Fast Likelihood Evaluation and Sampling in Flow-based Models. ICLR 2026. https://arxiv.org/pdf/2512.02636

Training an F2D2 model

$$\mathcal{L}_{\text{F2D2}}(\theta) := \mathcal{L}_{\text{VM}}(\theta) + \mathcal{L}_{\text{u}}(\theta) + \mathcal{L}_{\text{div}}(\theta) + \mathcal{L}_{\text{D}}(\theta)$$

Make sure the tangent condition holds for sampling

Make sure the flow map condition holds for sampling

Make sure the tangent condition holds for likelihood

Make sure the flow map condition holds for likelihood

Make sure it's a flow map for sampling

Make sure it's a flow map for likelihood

Carnegie Mellon University

We are now in the realm of the state-of-the-arts

- **2/3 (Tue):** How to use diffusion/flow models for robotics, control & decision making (**Max Simchowitz**, MLD Prof.) ✅
- **2/5 (Thur):** Text-to-image models and SOTA techniques ✅
- **2/10 (Tue):** How to sample from diffusion/flow models with a single step (Quiz 5) ✅
    - Distillation
    - Consistency models
    - Flow maps
- **2/12 (Thur):** Real-time generation techniques for video (**Linqi (Alex) Zhou**, Luma AI) (Pizza for people who come in person)

Talk from Linqi (Alex) Zhou

Linqi (Alex) Zhou is a research scientist at Luma AI. Previously he was a PhD student at Stanford University advised by Prof. Stefano Ermon. He cofounded start-up Apparate Labs which was later acquired by Luma AI.

Talk from Linqi (Alex) Zhou

Title: Towards Efficient Inference-Time Scaling without Distillation

Abstract: Most generative modeling methods for continuous domains struggle to simultaneously deliver high sample quality, stable training, and efficient inference. Diffusion models achieve state-of-the-art fidelity, but their iterative sampling is computationally expensive. A common strategy to accelerate inference is to distill a pretrained diffusion model into a faster sampler. However, this adds an additional training stage and introduces further instability and operational complexity. In contrast, we introduce a series of efforts towards designing principled pre-training algorithms that directly enable one- or few-step generation, while achieving higher sample quality than diffusion models.