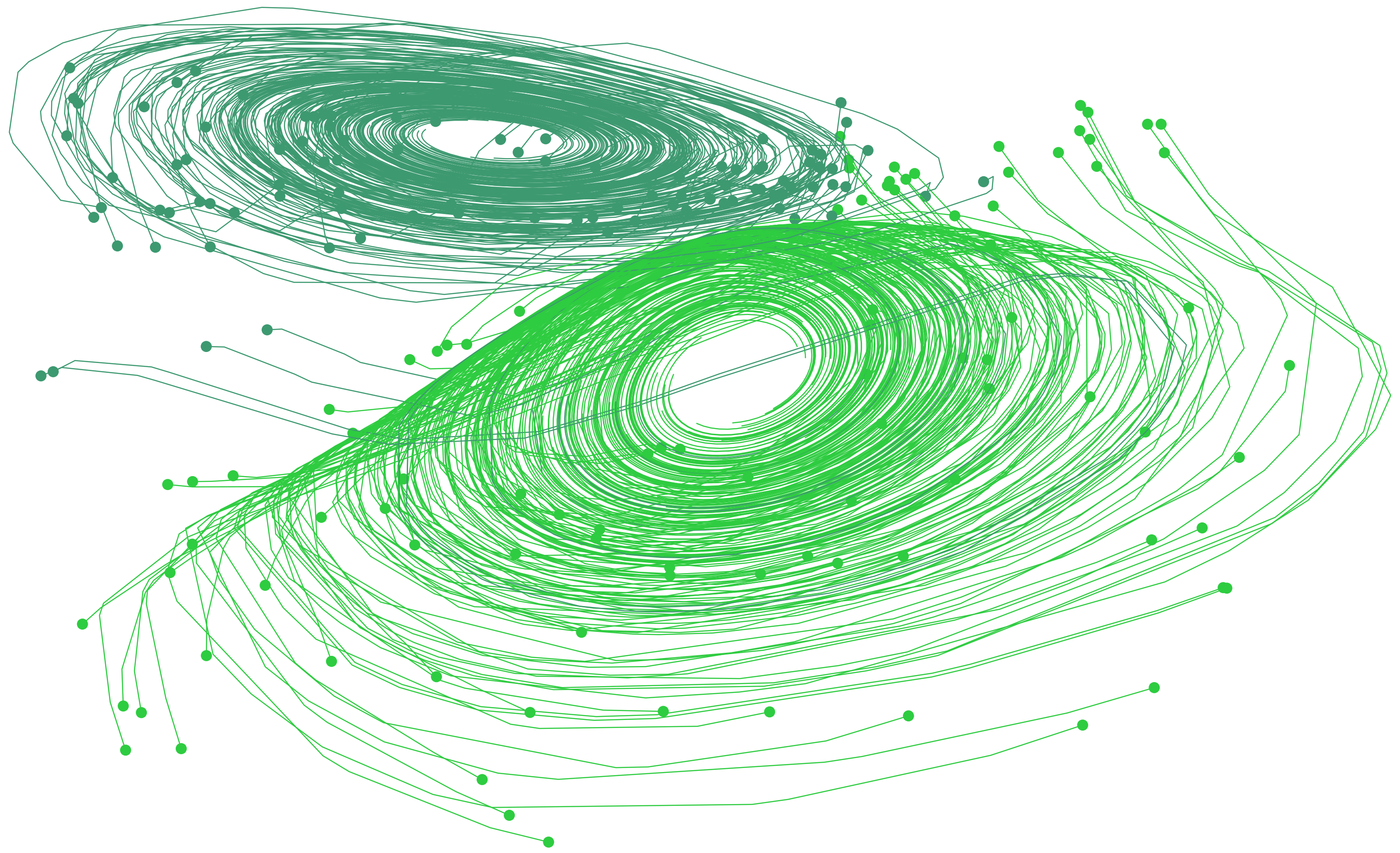
Neural Ordinary Differential Equations

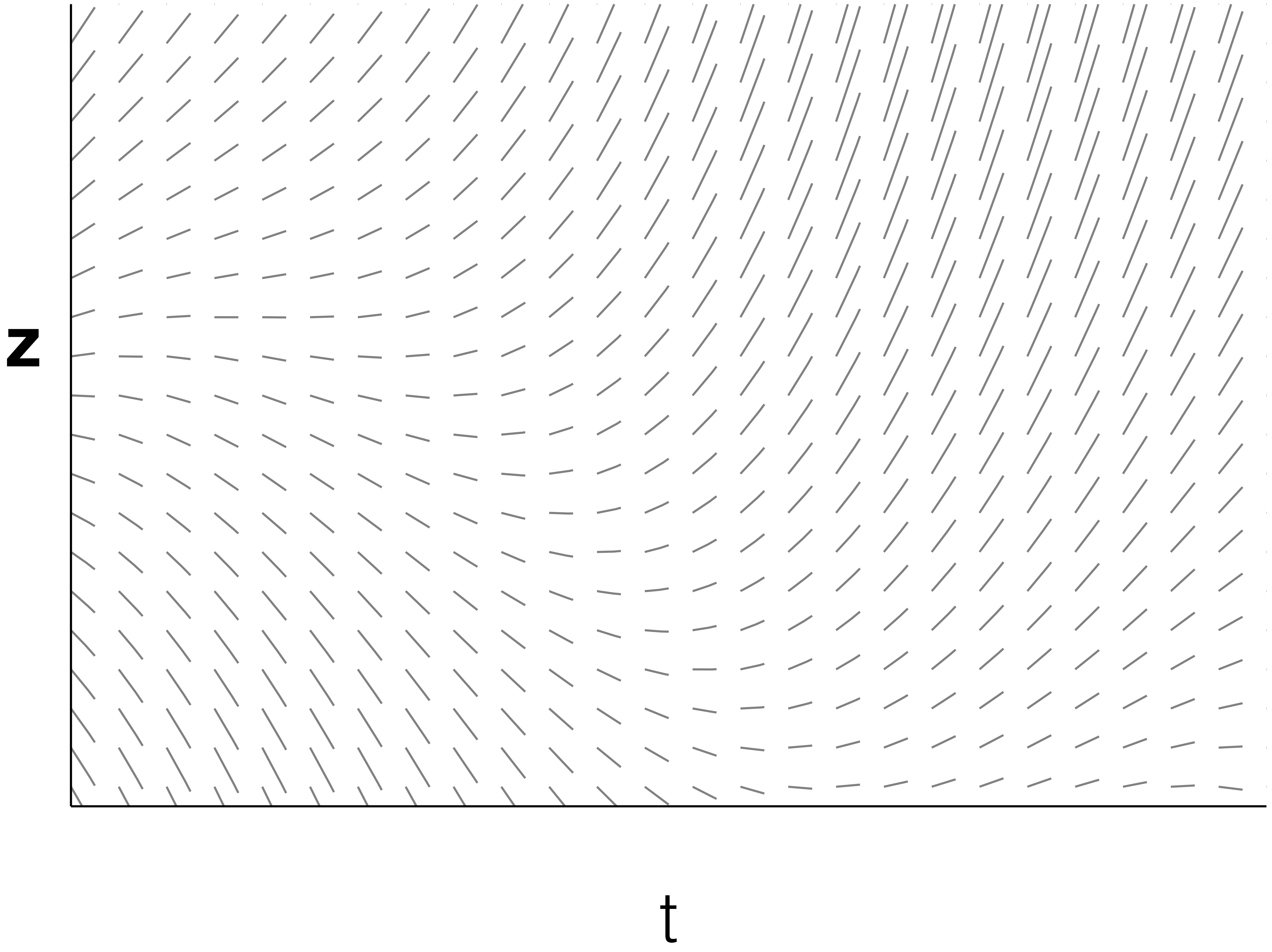


Ricky Chen, Yulia Rubanova, Jesse Bettencourt, David Duvenaud
University of Toronto, Vector Institute

VELUT ARBOR ÆVO

VECTOR INSTITUTE

Background: ODE Solvers



- Vector-valued **z** changes in time
- Time-derivative: $\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$
- Initial-value problem: given $\mathbf{z}(t_0)$, find:

$$\mathbf{z}(t_1) = \mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt$$

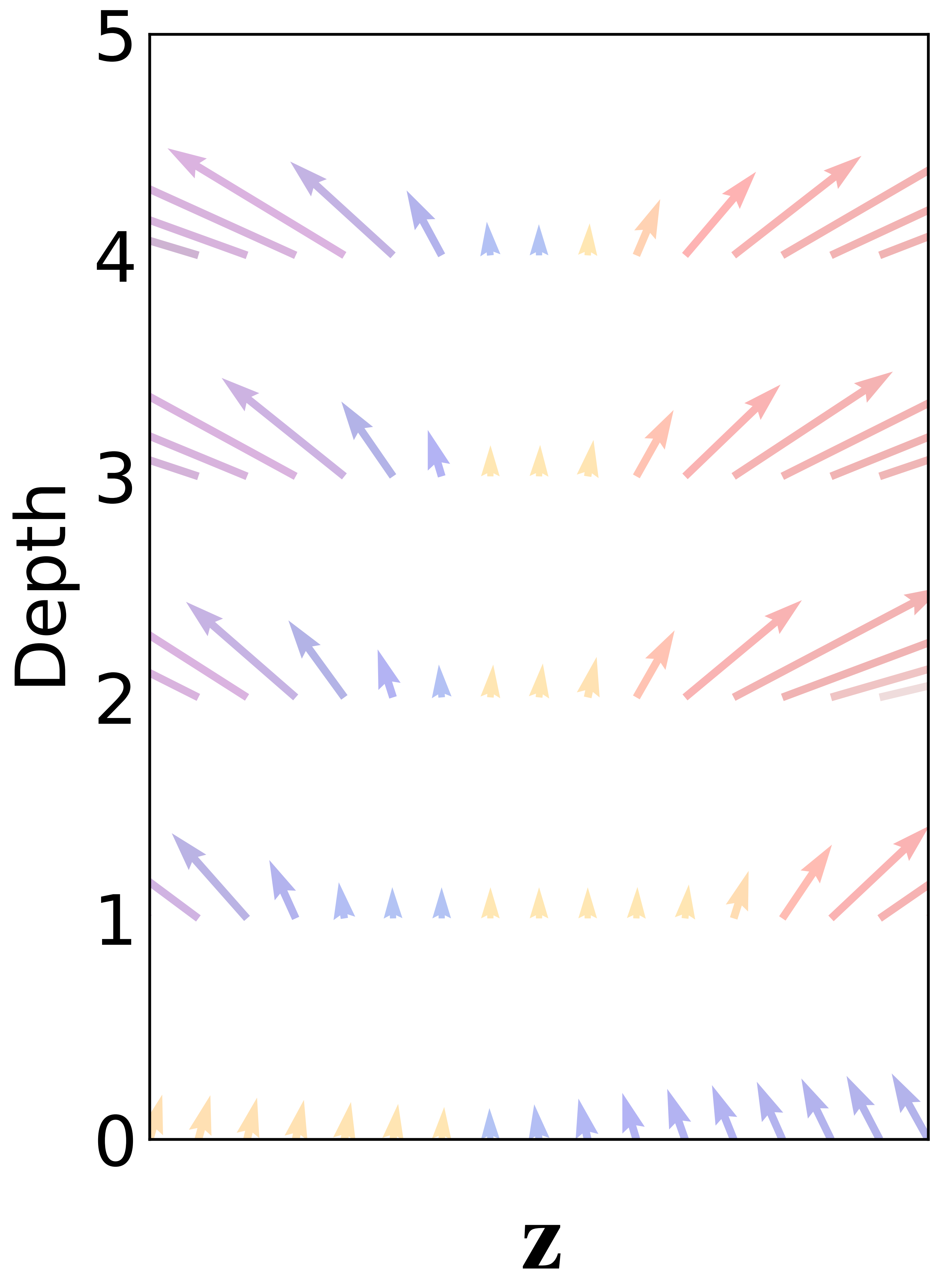
- Euler approximates with small steps:

$$\mathbf{z}(t + h) = \mathbf{z}(t) + hf(\mathbf{z}, t)$$

Resnets as Euler integrators

```
def f(z, t, θ):
  return nnet(z, θ[t])

def resnet(z):
  for t in [1:T]:
    z = z + f(z, t, θ)
  return z
```



Related Work

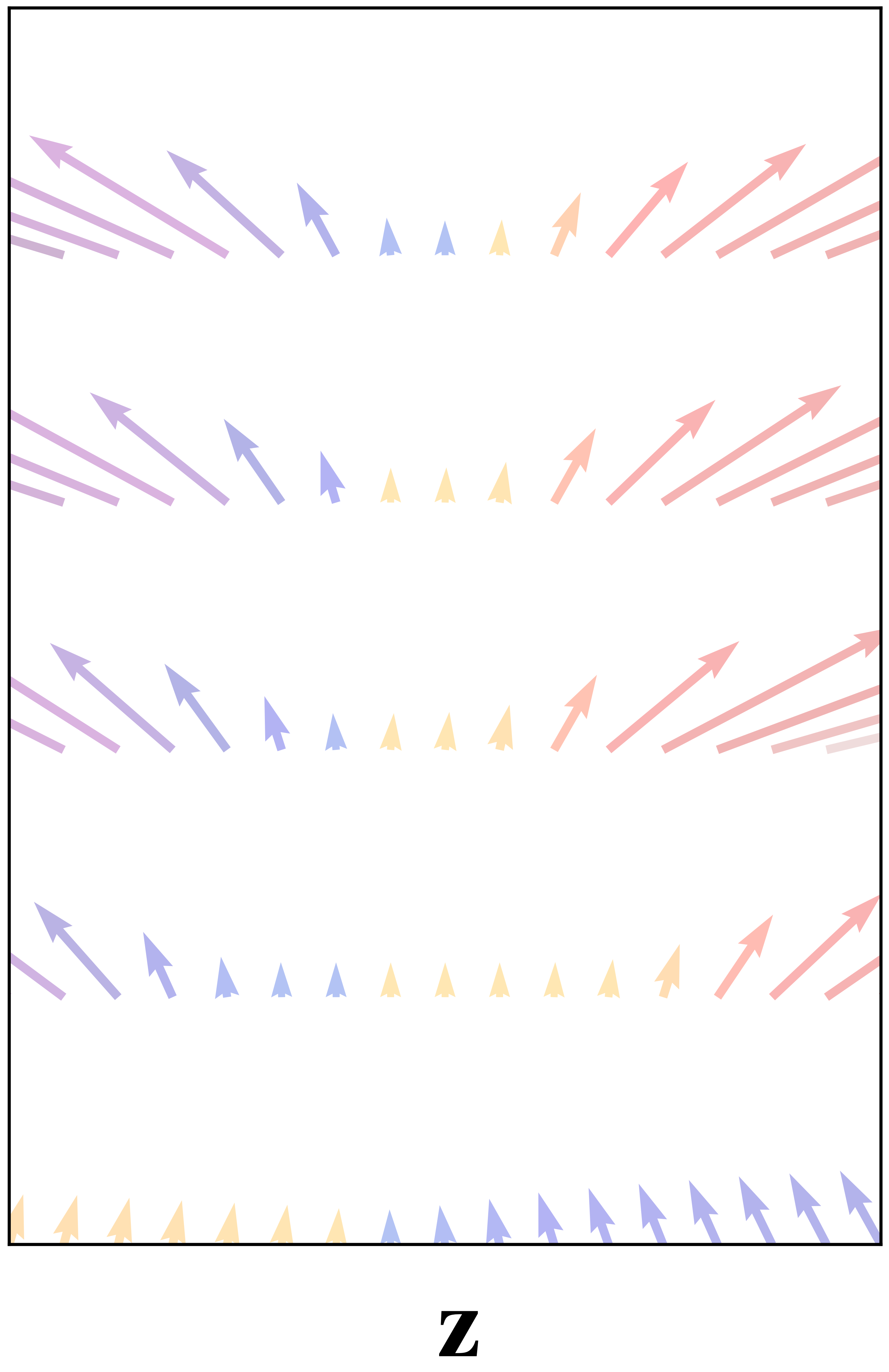
- Continuous-time nets once seemed natural
  LeCun (1988), Pearlmutter (1995)

- Solver-inspired architectures:
  Lu et al. (2017), Haber & Ruthotto (2017), Ruthotto & Haber (2018)

- ODE-inspired training methods:
  Chang et al. (2017, 2018)

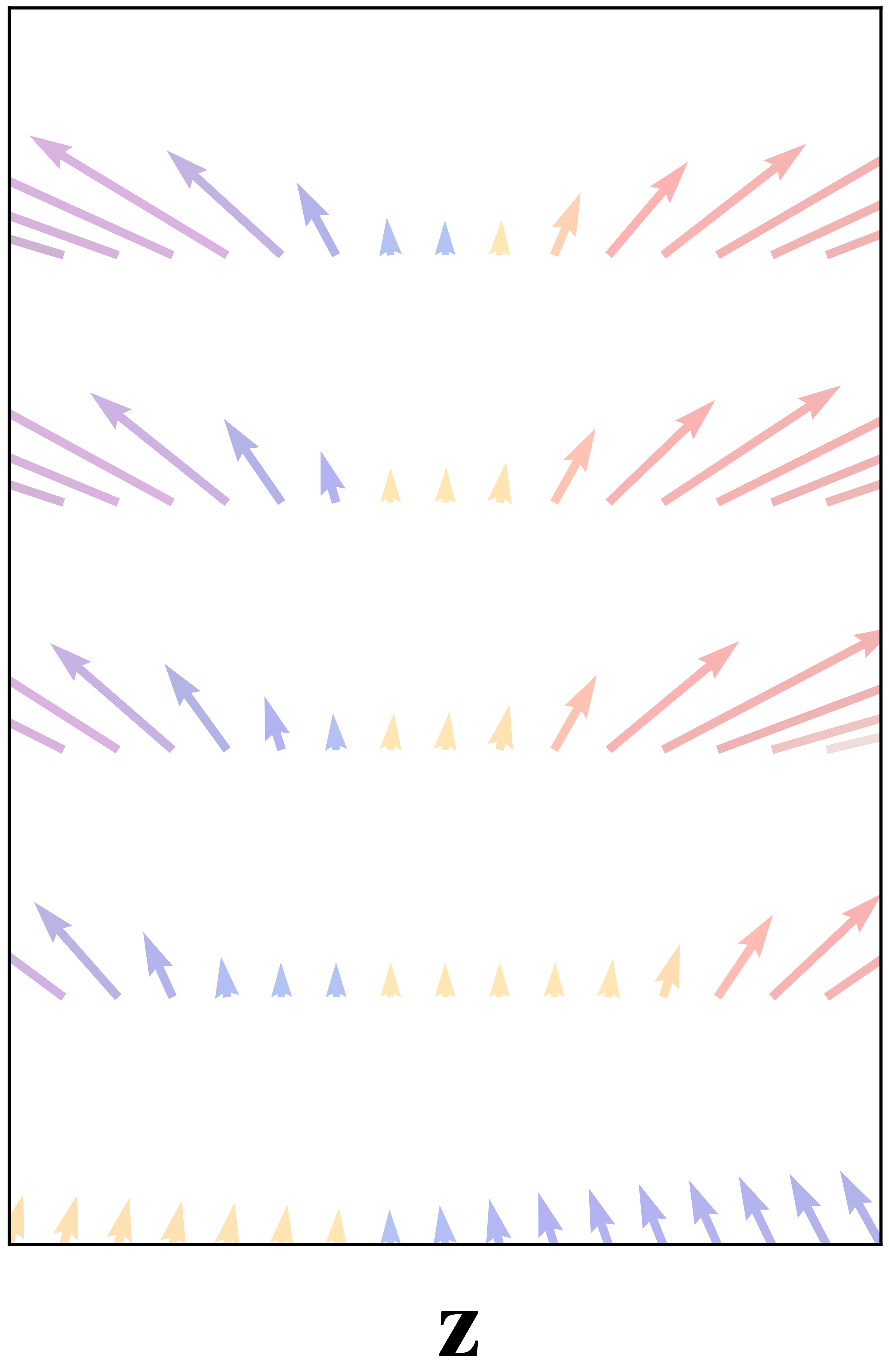
```
def f(z, t, θ):
    return nnet(z, θ[t])

def resnet(z, θ):
    for t in [1:T]:
        z = z + f(z, t, θ)
    return z
```



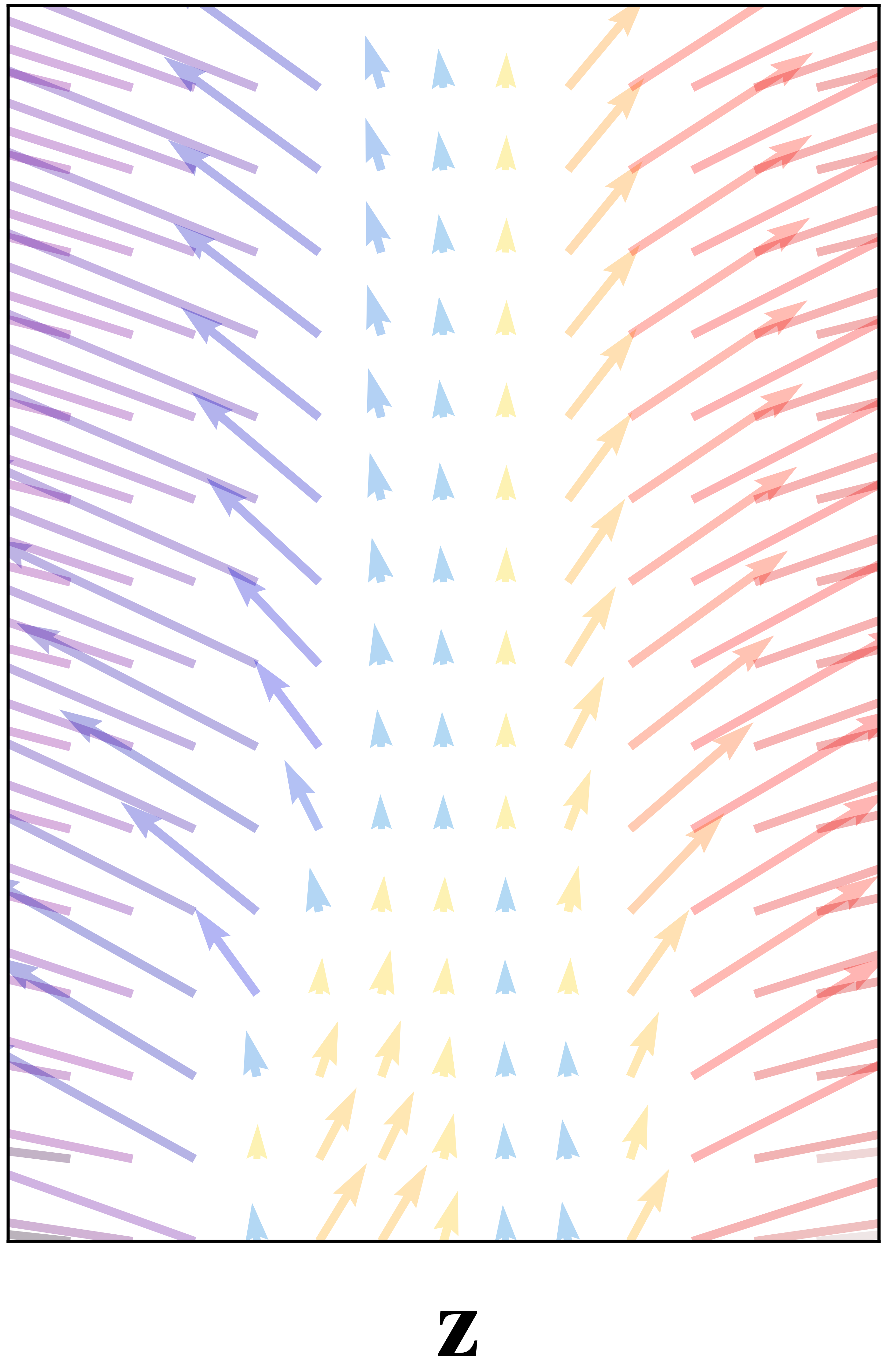
```
def f(z, t, θ):
  return nnet([z, t], θ)

def resnet(z, θ):
  for t in [1:T]:
    z = z + f(z, t, θ)
  return z
```



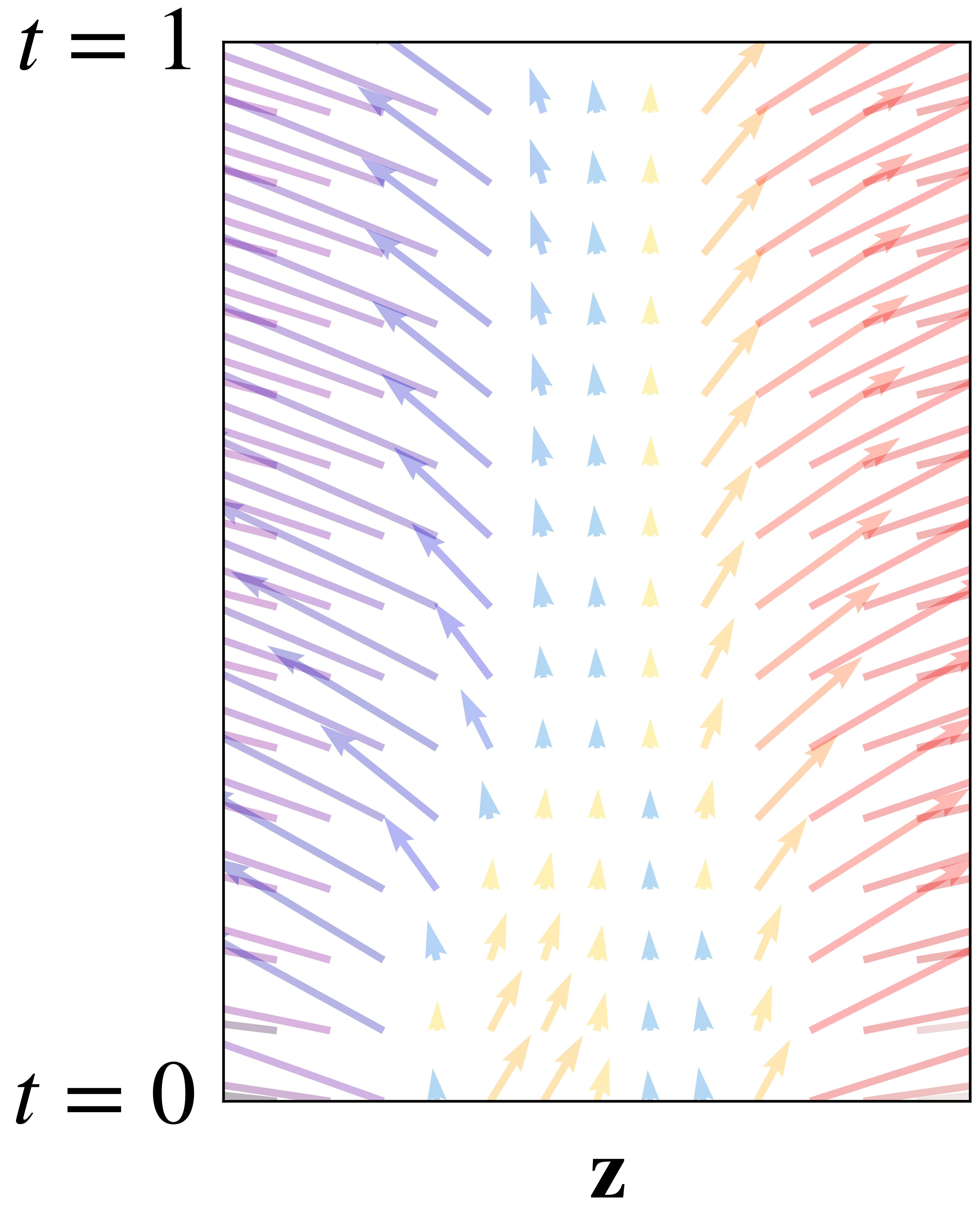
```
def f(z, t, θ):
  return nnet([z, t], θ)

def resnet(z, θ):
  for t in [1:T]:
    z = z + f(z, t, θ)
  return z
```



```
def f(z, t, θ):
    return nnet([z, t], θ)

def ODEnet(z, θ):
    return ODESolve(f, z, 0, 1, θ)
```



How to train an ODE net?

$$L(\theta) = L\left(\int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right)$$

$$\frac{\partial L}{\partial \theta} = ?$$

- Don't backprop through solver: High memory cost, extra numerical error
- Approximate the derivative, don't differentiate the approximation!

Continuous-time Backpropagation

Standard Backprop:

$$\frac{\partial L}{\partial \mathbf{z}_t} = \frac{\partial L}{\partial \mathbf{z}_{t+1}} \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \mathbf{z}_t}$$

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial \mathbf{z}_t} \frac{\partial f(\mathbf{z}_t, \theta)}{\partial \theta_t}$$

- Can build adjoint dynamics with autodiff, compute all gradients with another ODE solve:

```
def f_and_a([z, a, d], t):
  return [f, -a*df/da, -a*df/dθ)

[z0, dL/dx, dL/dθ] =
   ODESolve(f_and_a,
   [z(t1), dL/dz(t), 0], t1, t0)
```

Adjoint sensitivities:
(Pontryagin et al., 1962):

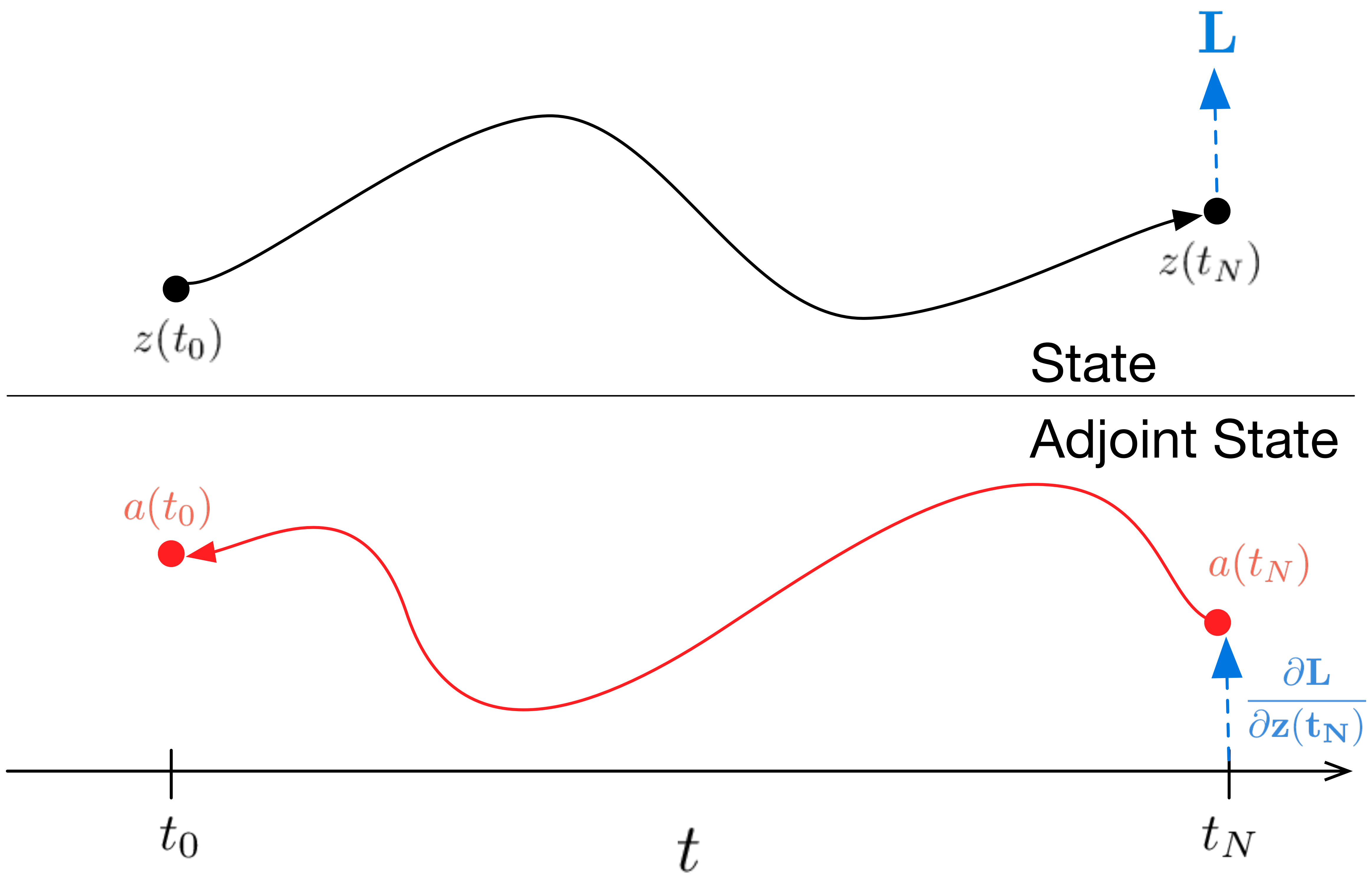
$$\mathbf{a}(t) = \frac{\partial L}{\partial \mathbf{z}(t)}$$

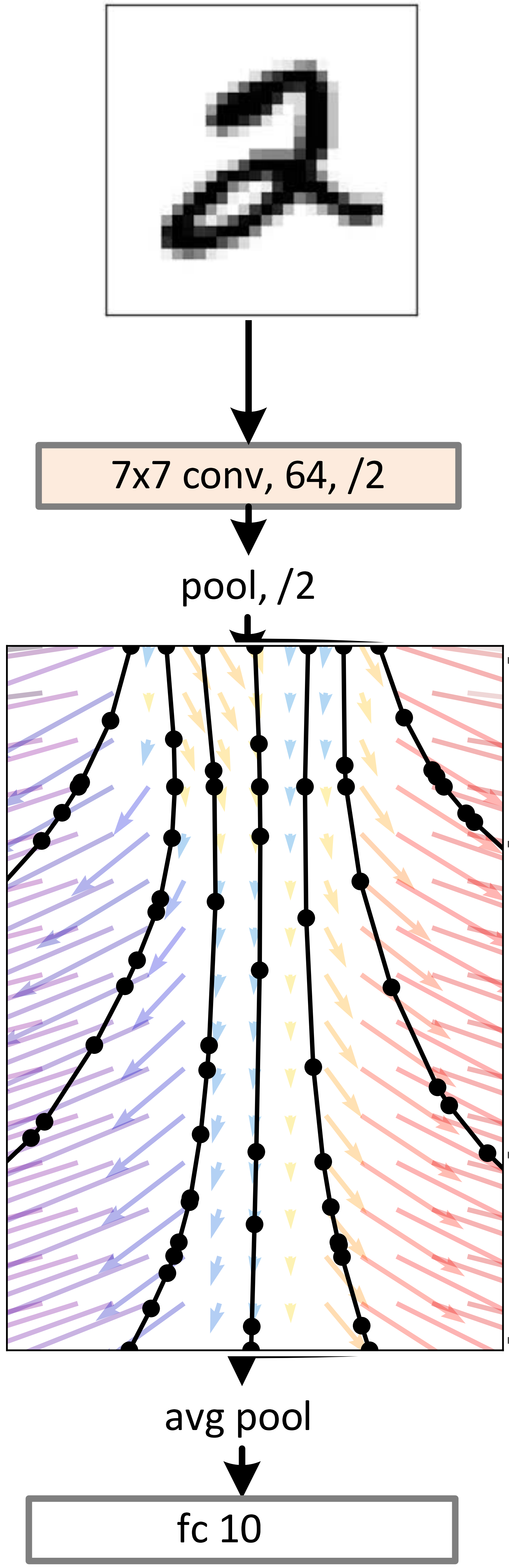
$$\frac{\partial \mathbf{a}(t)}{\partial t} = \mathbf{a}(t) \frac{\partial f(\mathbf{z}_t, t, \theta)}{\partial \mathbf{z}}$$

$$\frac{\partial L}{\partial \theta} = \int_{t_1}^{t_0} \mathbf{a}(t) \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$

O(1) Memory Gradients

- No need to store activations, just run dynamics backwards from output.
- Reversible ResNets (Gomez et al., 2018) must partition dimensions.





Drop-in replacement for Resnets

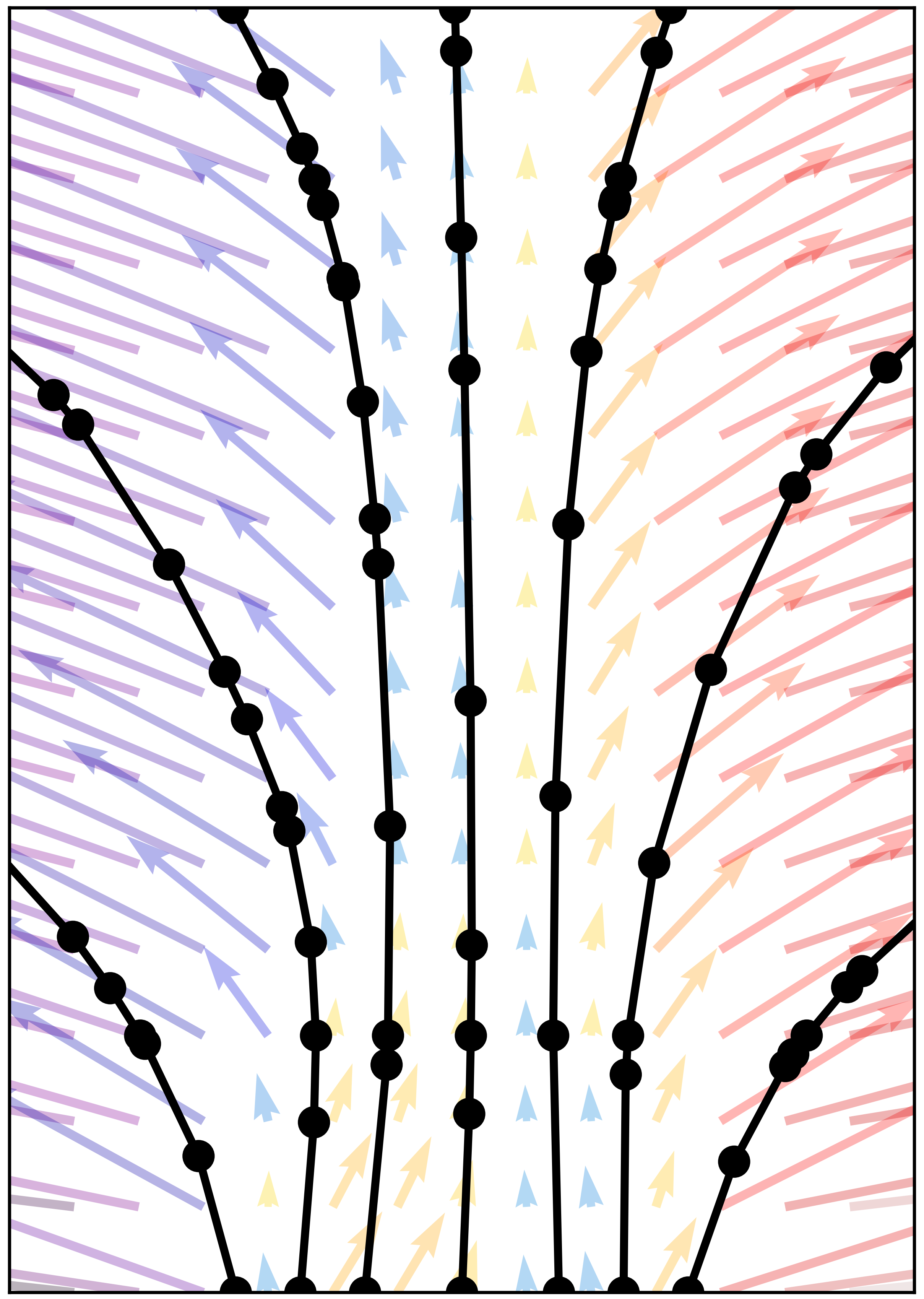
- Same performance with fewer parameters.

	Test Error	# Params
1-Layer MLP	1.60%	0.24 M
ResNet	0.41%	0.60 M
ODE-Net	0.42%	0.22 M

How deep are ODE-nets?

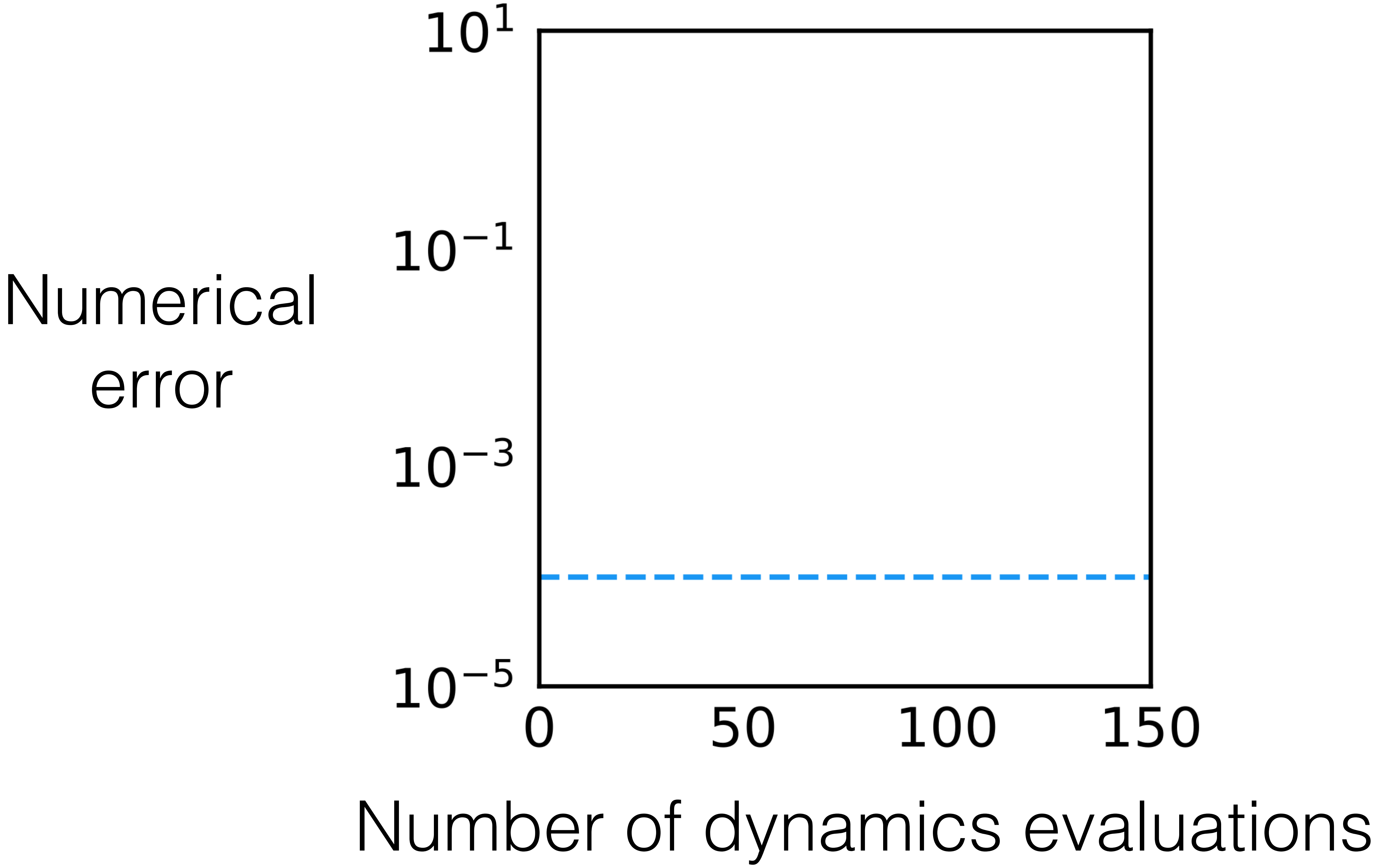
- ‘Depth’ is left to ODE solver.
- Dynamics become more demanding during training
- 2-4x the depth of resnet architectures
- Chang et al. (2018) build such a schedule by hand

Num
evals



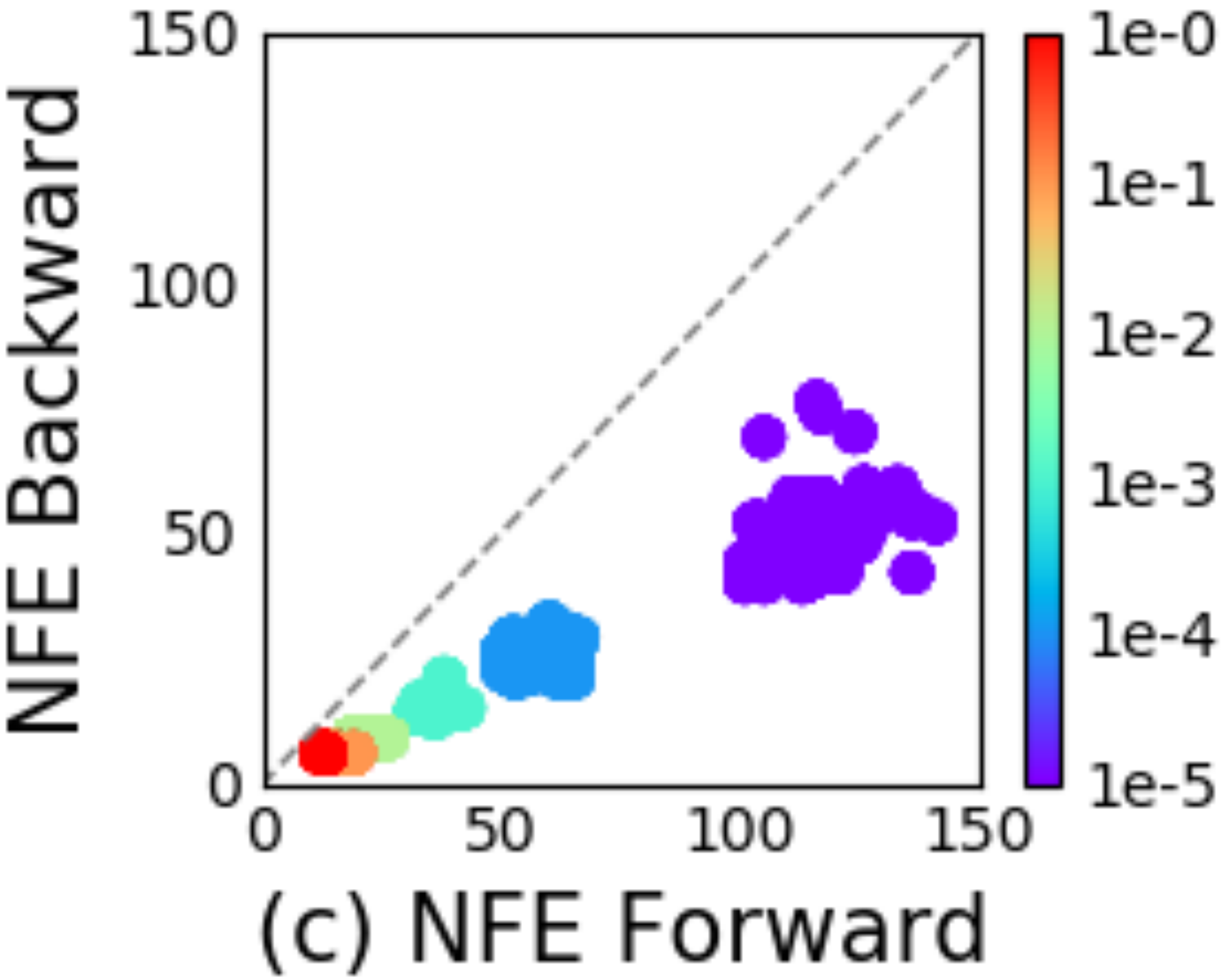
Explicit Error Control

```
ODESolve(f, x, t0, t1, θ, tolerance)
```



Reverse vs Forward Cost

- Empirically, reverse pass roughly half as expensive as forward pass
- Again, adapts to instance difficulty
- Num evaluations comparable to number of layers in modern nets

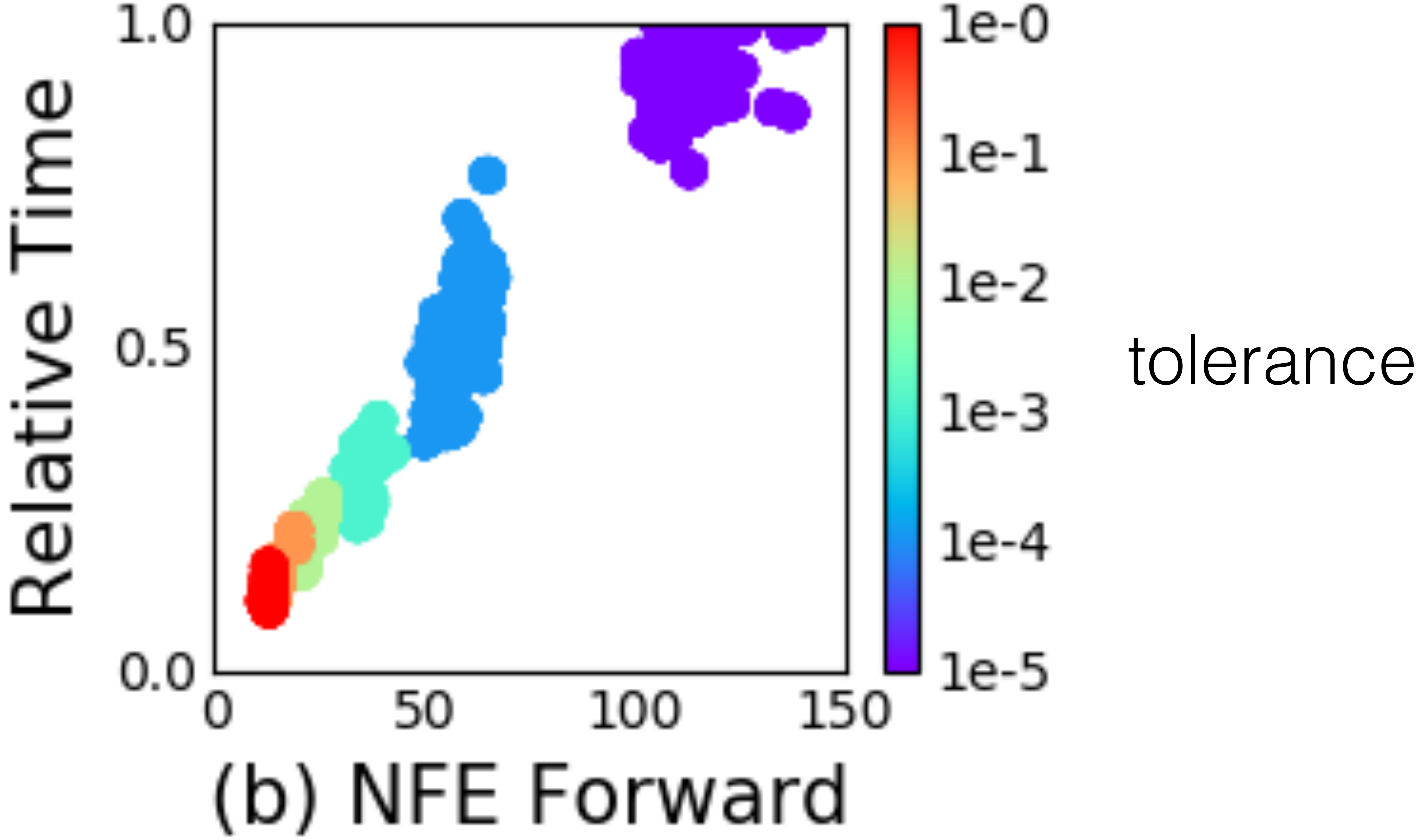


Speed-Accuracy Tradeoff

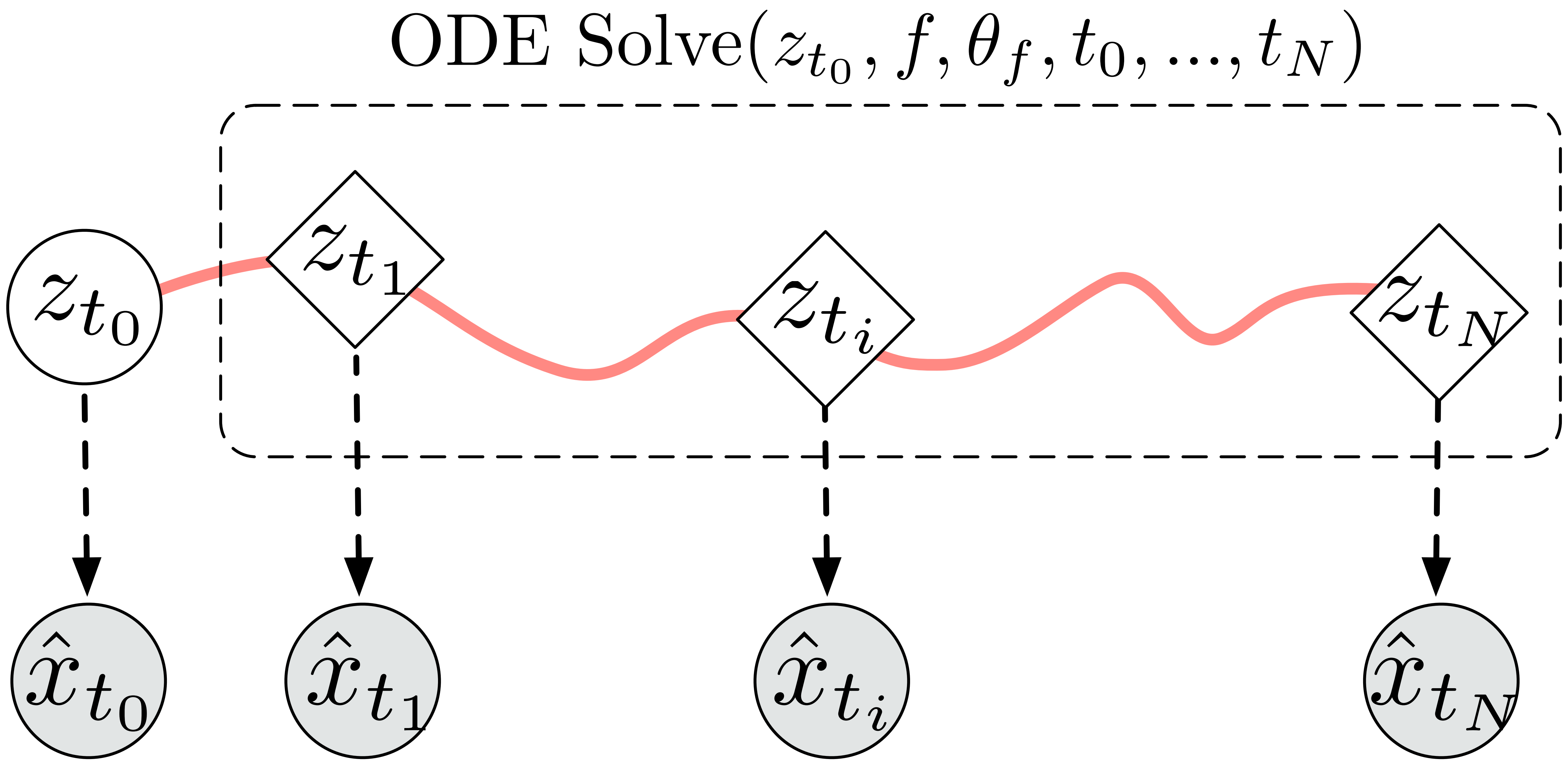
output = ODESolve(f, z0, t0, t1, theta, tolerance)

- Time cost is dominated by evaluation of dynamics

- Roughly linear with number of forward evaluations



Continuous-time models

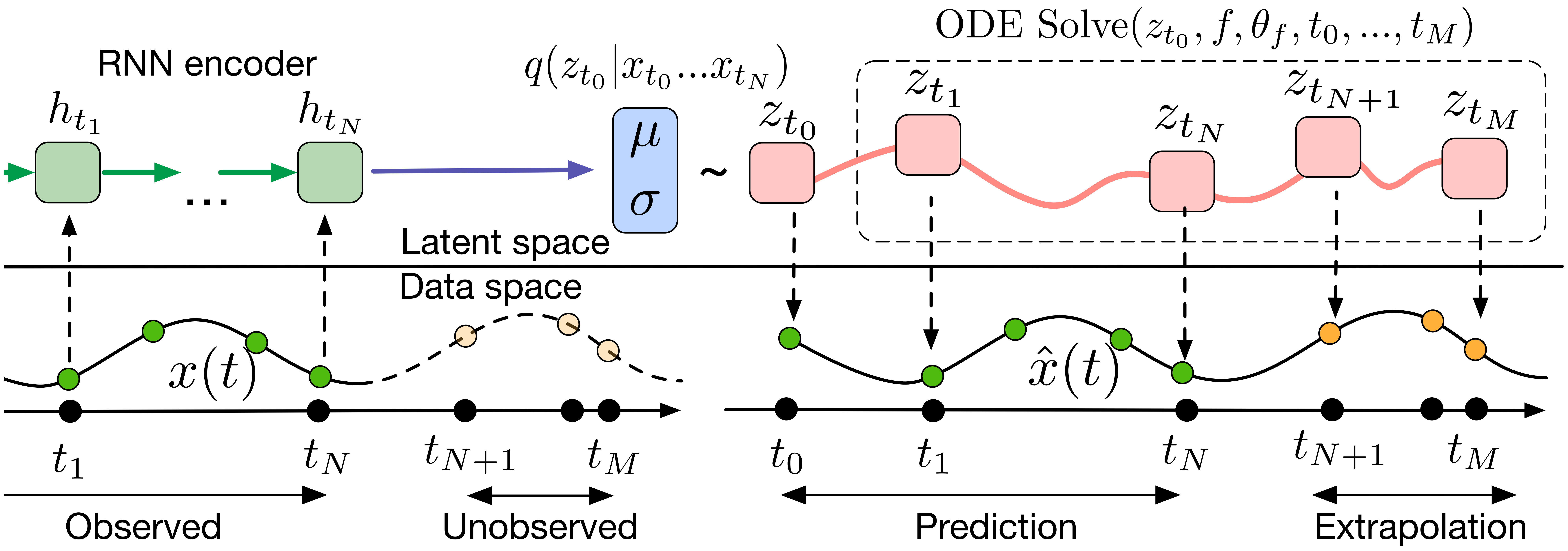


- Well-defined state at all times
- Dynamics separate from inference
- Irregularly-timed observations.

$$
\begin{aligned}
\mathbf{z}_{t_0} &\sim p(\mathbf{z}_{t_0}) \\
\mathbf{z}_{t_1}, \mathbf{z}_{t_2}, \ldots, \mathbf{z}_{t_N} &= \mathrm{ODESolve}(\mathbf{z}_{t_0}, f, \theta_f, t_0, \ldots, t_N) \\
\text{each} \quad \mathbf{x}_{t_i} &\sim p(\mathbf{x}|\mathbf{z}_{t_i}, \theta_{\mathbf{x}})
\end{aligned}
$$

Continuous-time RNNs

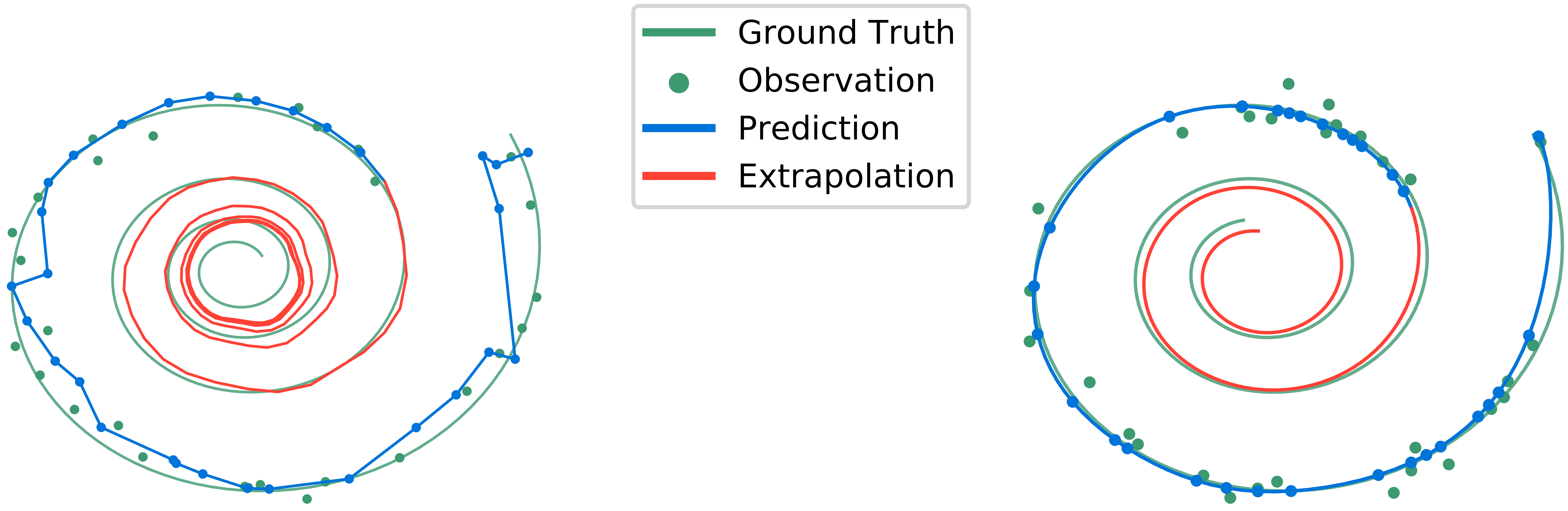
- Can do VAE-style inference with an RNN encoder
- Actually, more like a Deep Kalman Filter



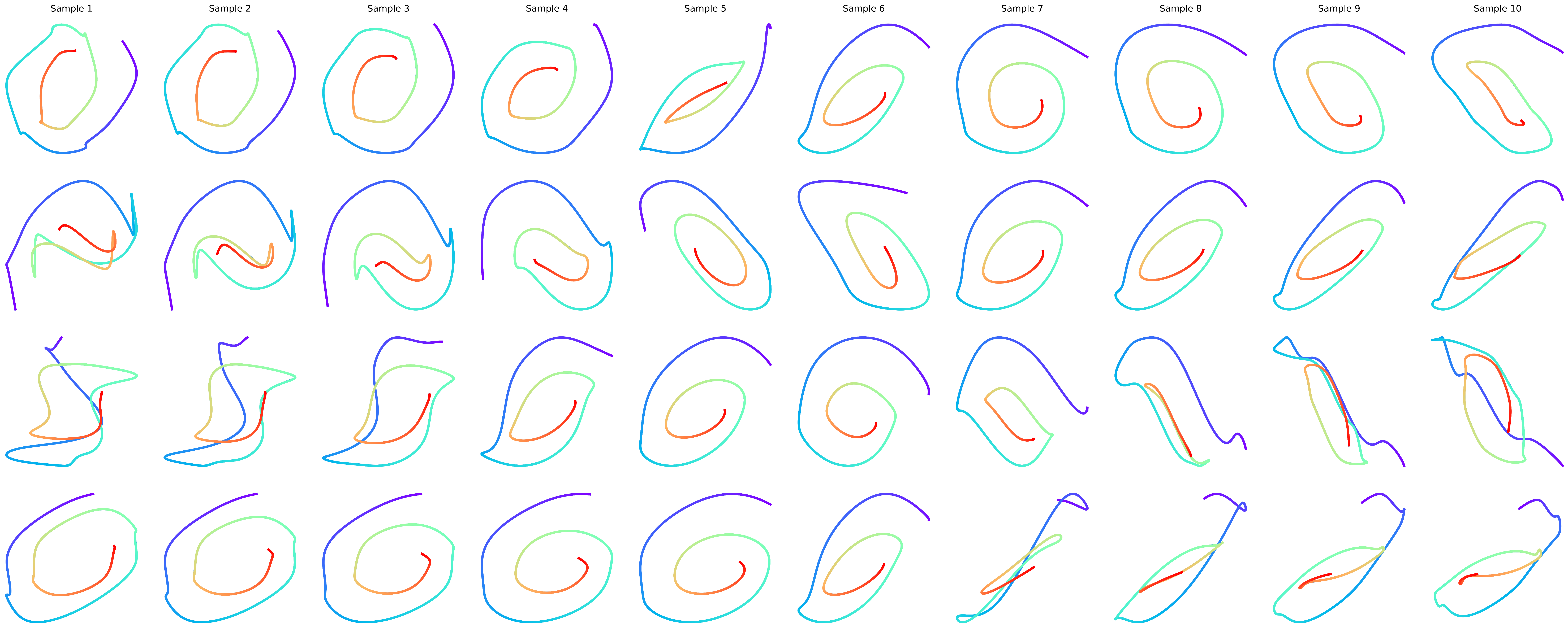
Continuous-time models

Recurrent Neural Net

Latent ODE



Latent space interpolation

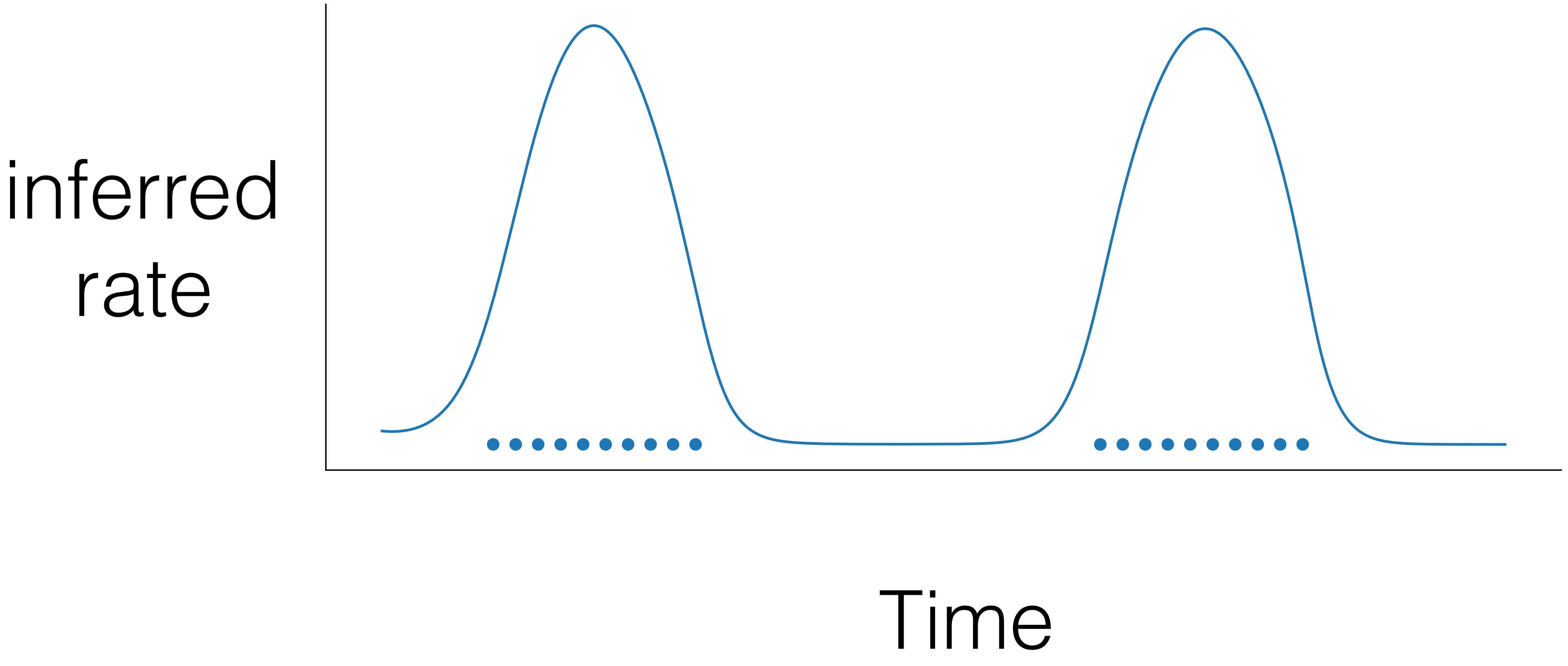


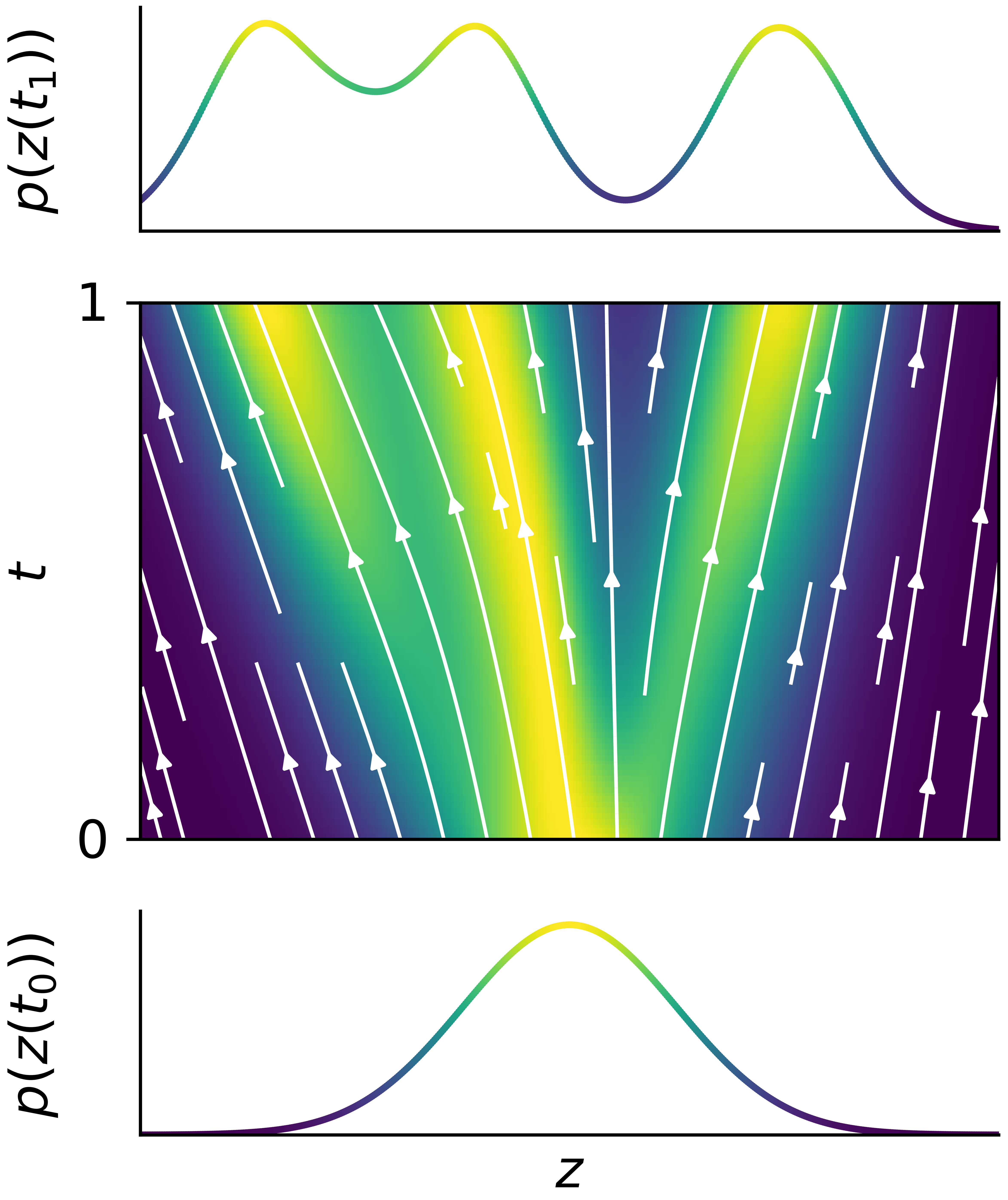
Each latent point corresponds to a trajectory

Poisson Process Likelihoods

$$\log p(t_1, \ldots, t_N | t_{\text{start}}, t_{\text{end}}) = \sum_{i=1}^{N} \log \lambda(\mathbf{z}(t_i)) - \int_{t_{\text{start}}}^{t_{\text{end}}} \lambda(\mathbf{z}(t)dt$$

- Can condition on arrival times to inform latent state





Instantaneous Change of Variables

$$\frac{d\mathbf{z}}{dt} = f(\mathbf{z}(t), t)$$

$$\Downarrow$$

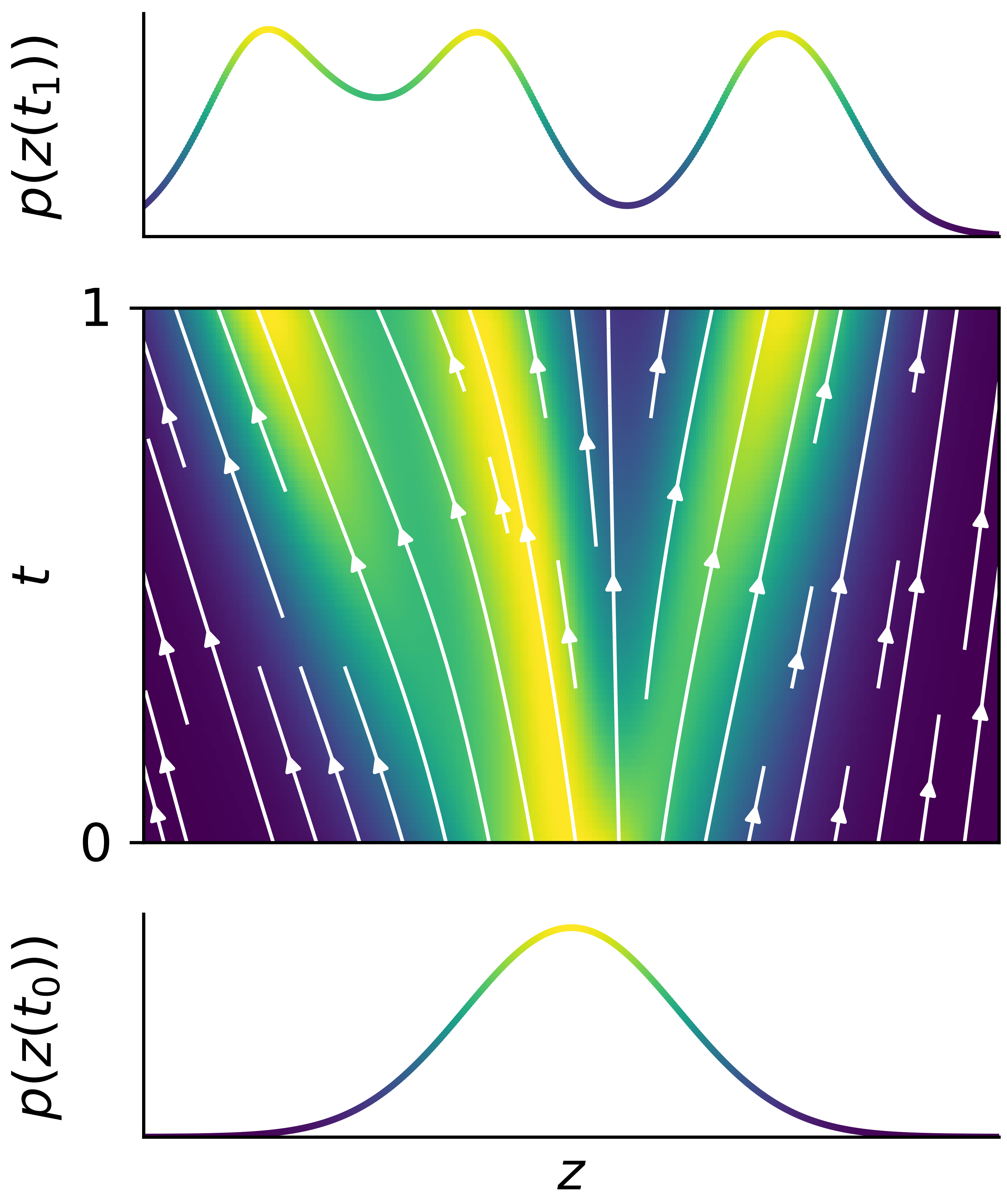
$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\mathbf{tr}\left(\frac{df}{d\mathbf{z}(t)}\right)$$

- Worst-case cost O(D^2).
- Only need continuously differentiable f

Continuous Normalizing Flows

$$\log p(\mathbf{z}(t_1)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \operatorname{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt$$

- Reversible dynamics, so can train from data by maximum likelihood
- No discriminator or recognition network, train by SGD
- No need to partition dimensions



Trading Depth for Width

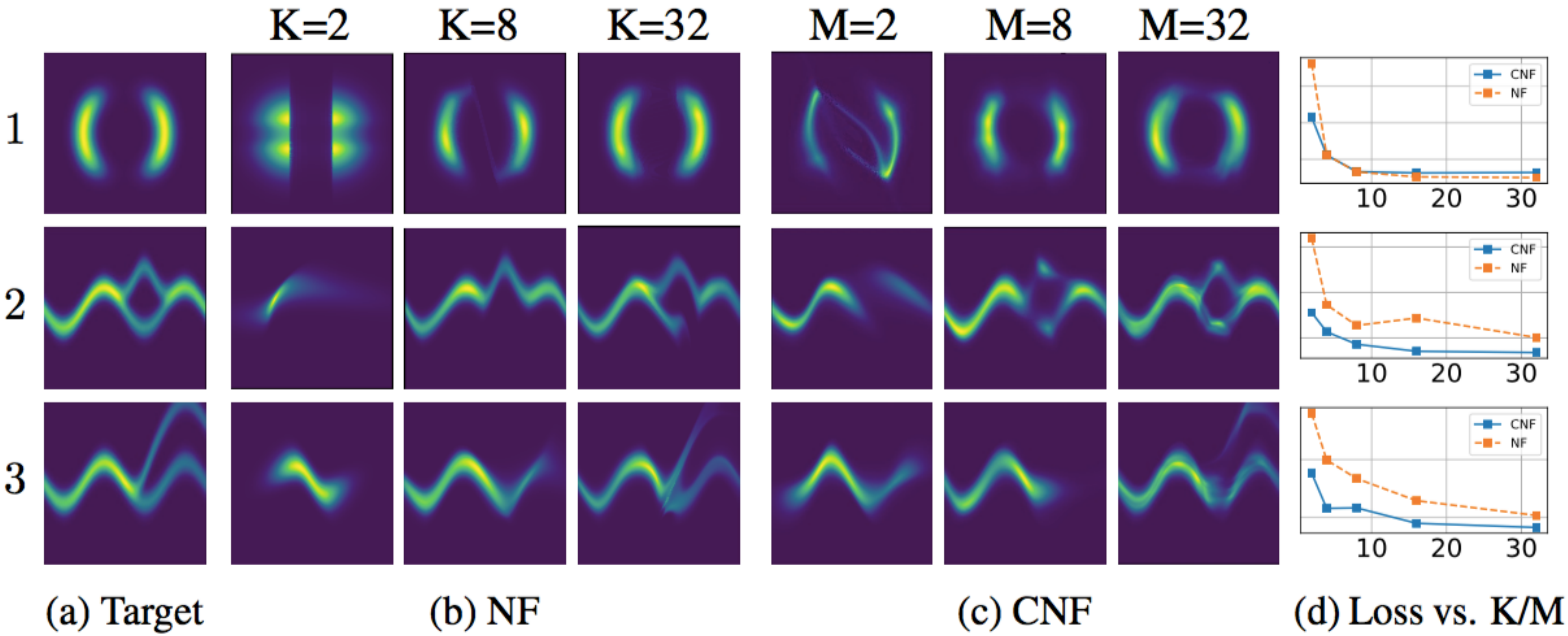


Figure 5: Comparison of NF and CNFs on learning generative models (noise → data) trained to minimize the reverse KL.

Thinking about scalability

log-probability of the data under the discrete model:

$$\log p(x) = \log p(z_T) + \sum_{t=0}^{T} \log |\partial f^t / \partial z_t|$$

log-probability of the data under the continuous model:

$$\log p(x) = \log p(z_T) + \int_0^1 \nabla f(z_t, t) dt$$

log-dets vs divergence

For a general $f : \mathcal{R}^N \to \mathcal{R}^N$, $\partial f / \partial x$ takes time $O(N^2)$

Given $\partial f / \partial x$ computing $\log |\partial f / \partial x|$ takes time $O(N^3)$

Given $\partial f / \partial x$, $\nabla f(z_t, t)$ can be computed in $O(N)$ thus we are constrained by the $O(N^2)$ cost of computing the Jacobian

but, using two tricks we can produce an unbiased estimator for this quantity with $O(N)$ computation

Stochastic divergence estimation

$\partial f / \partial x$ requires $O(N^2)$ to compute, but can compute $e^T(\partial f / \partial x)$ using automatic differentiation in time $O(N)$ for any vector e.

For any matrix A, if $\mathbb{E}[e] = 0, \mathrm{Cov}(e) = I$, we have:

$$\mathrm{Tr}(A) = \mathbb{E}_{p(e)}[e^T A e]$$

(Hutchinson's estimator)

given that $\nabla f(z) = \mathrm{Tr}(\partial f / \partial z)$, we have:

$$\nabla f(z) = \mathbb{E}_{p(e)}[e^T (\partial f / \partial z) e]$$

which means we can use simple Monte Carlo to get an unbiased estimate in $O(N)$

Unbiased log-likelihood estimation

Stochastic divergence estimates can be incorporated into the instantaneous change of variables with:

$$
\begin{aligned}
\log p(x) &= \log p(z_T) + \int_0^1 \nabla f(z_t) dt \\
&= \log p(z_T) + \int_0^1 \mathbb{E}_{p(e)} \left[e^T \frac{\partial f}{\partial z_t} e \right] dt \\
&= \log p(z_T) + \mathbb{E}_{p(e)} \left[\int_0^1 e^T \frac{\partial f}{\partial z_t} e dt \right]
\end{aligned}
$$

Can sample a single e and integrate the divergence estimates to obtain an unbiased estimate of log p(x) for any differentiable f!

3-line tf implementation

```
dfdz = f(z, t)
e = tf.random_normal(tf.shape(z))
div = tf.reduce_sum(
  tf.gradients(dfdz, z, grad_ys=e) * e)
```

Putting it all together

We define a generative model for data

$$z_0 \sim p(z_0)$$

$$\frac{\partial z(t)}{\partial t} = f(z(t), t, \theta)$$

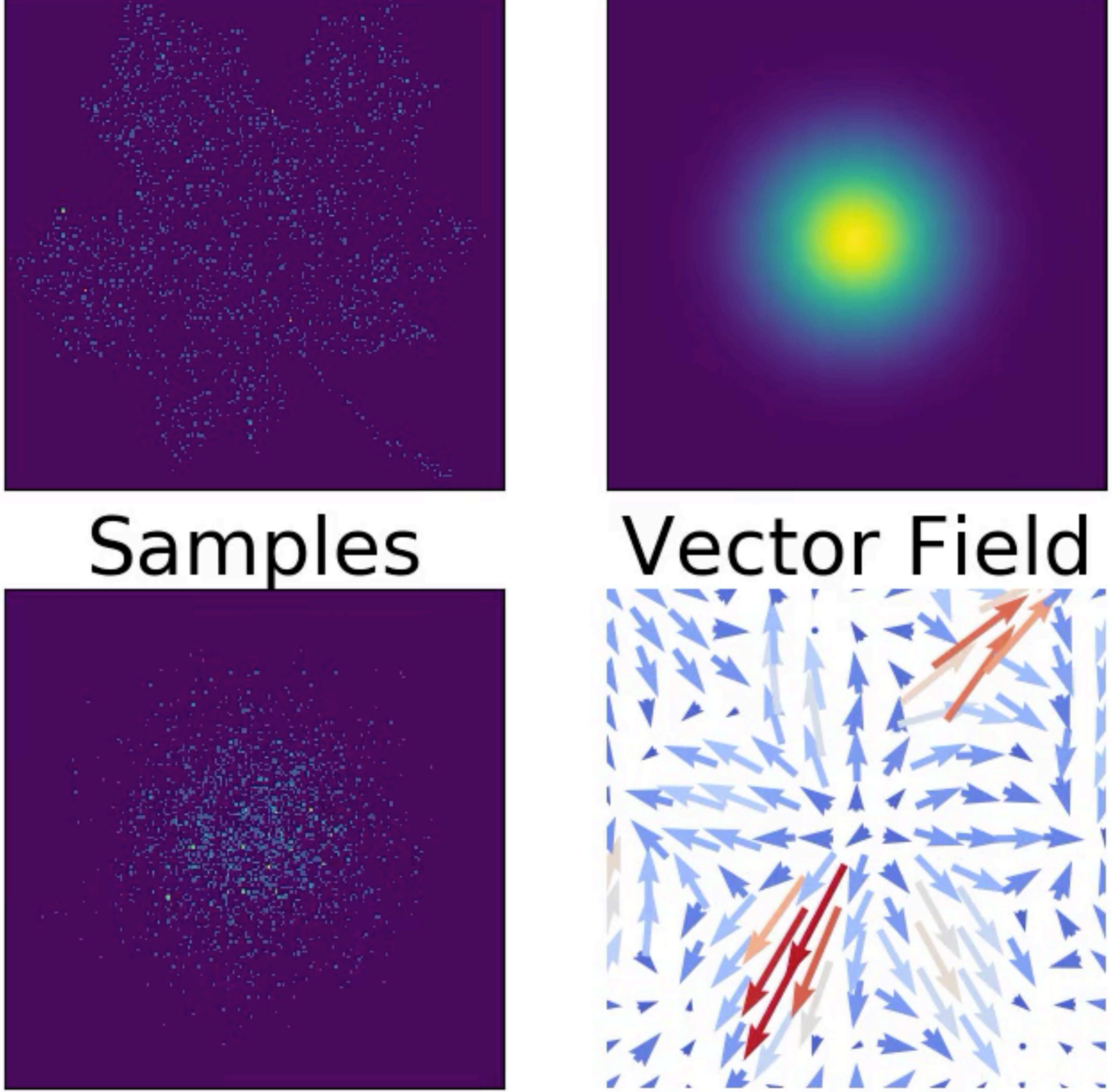
$$x = z_1$$

Where θ is trained using stochastic gradient ascent to maximize

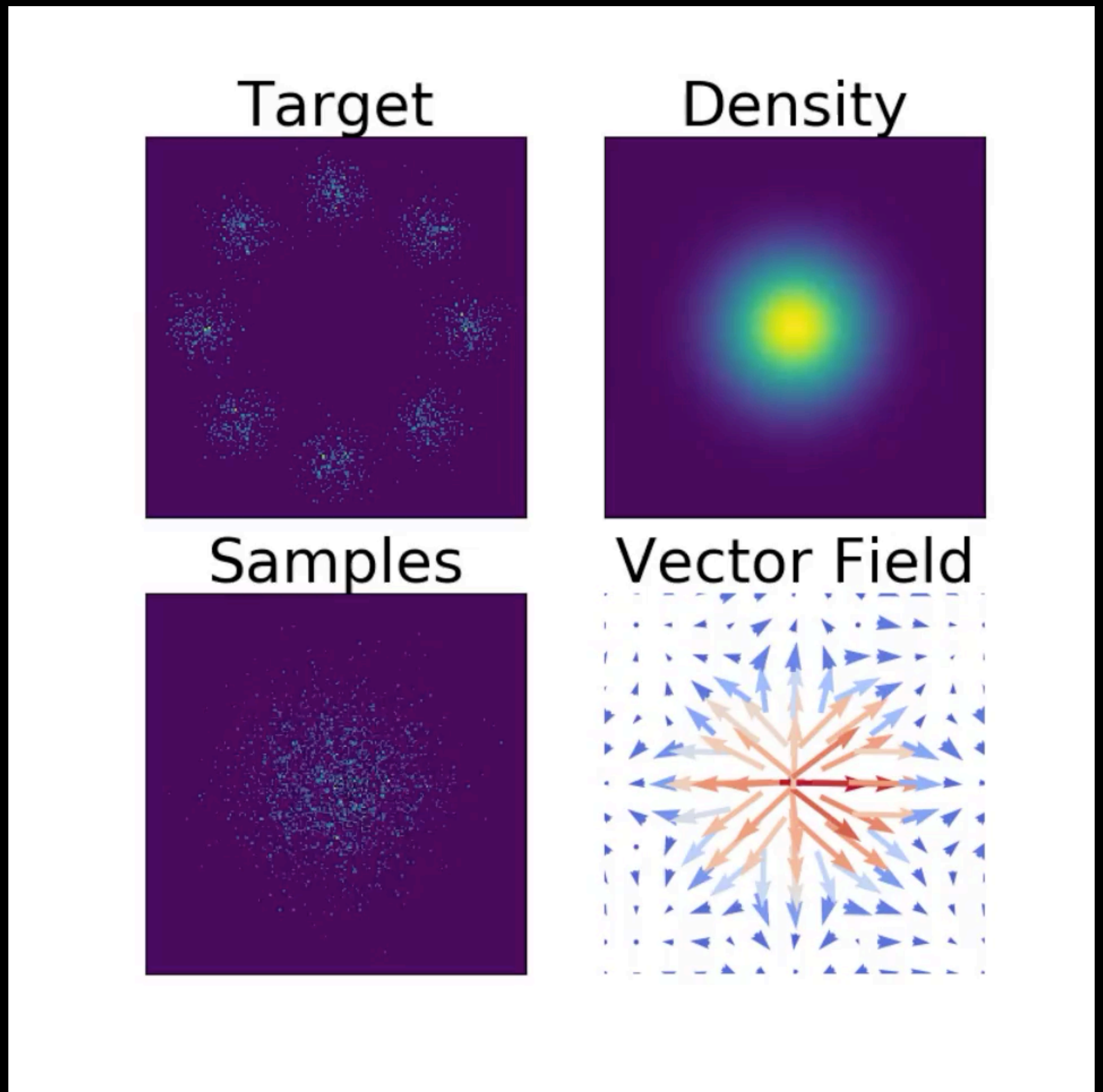
$$\log p(x) = \log p(z_0) + \mathbb{E}_{p(e)} \left[\int_0^1 e^T \frac{\partial f}{\partial z(t)} e dt \right]$$

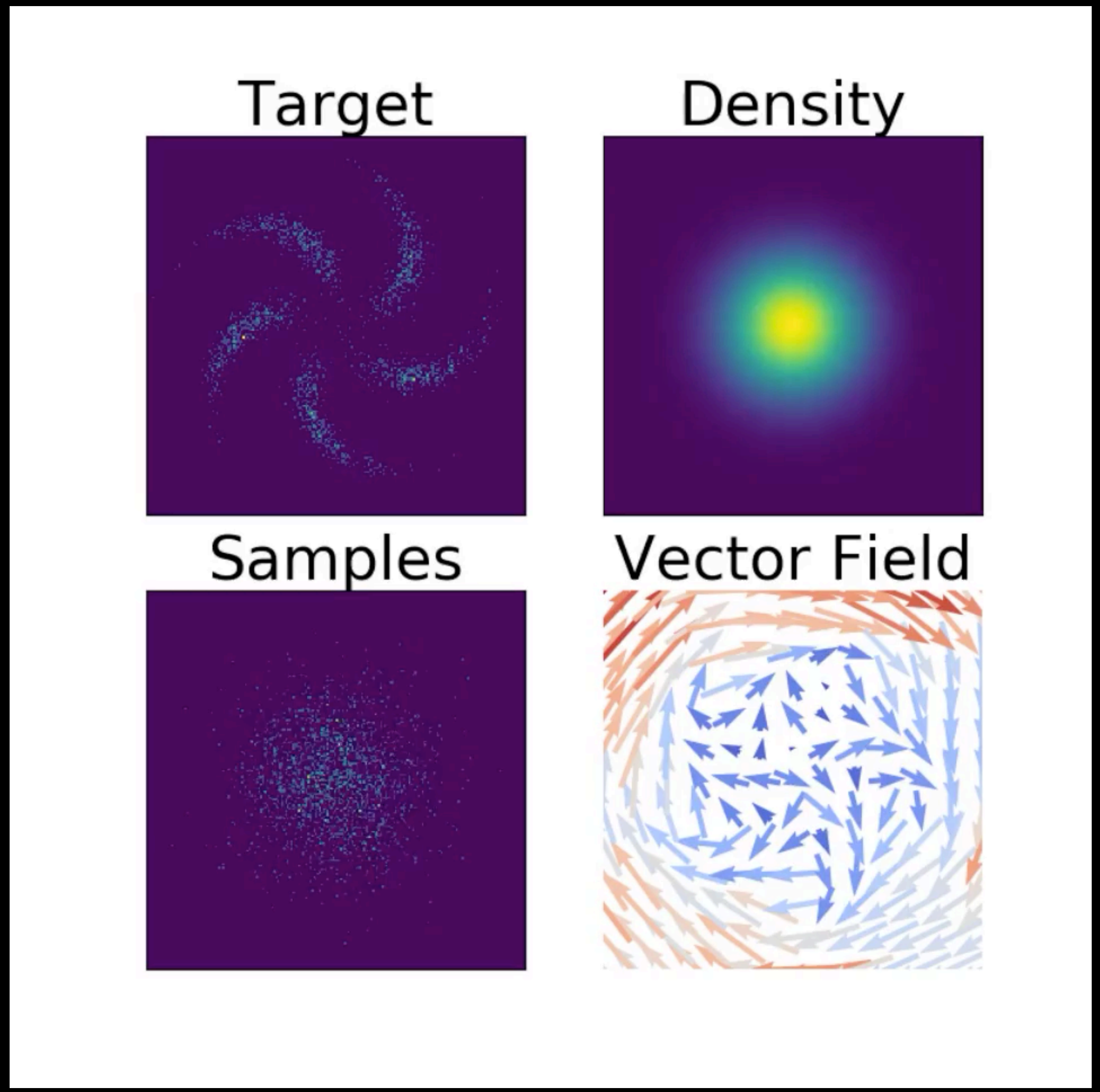
giving the first scalable invertible generative model which allows unrestricted architectures to specify the dynamics!

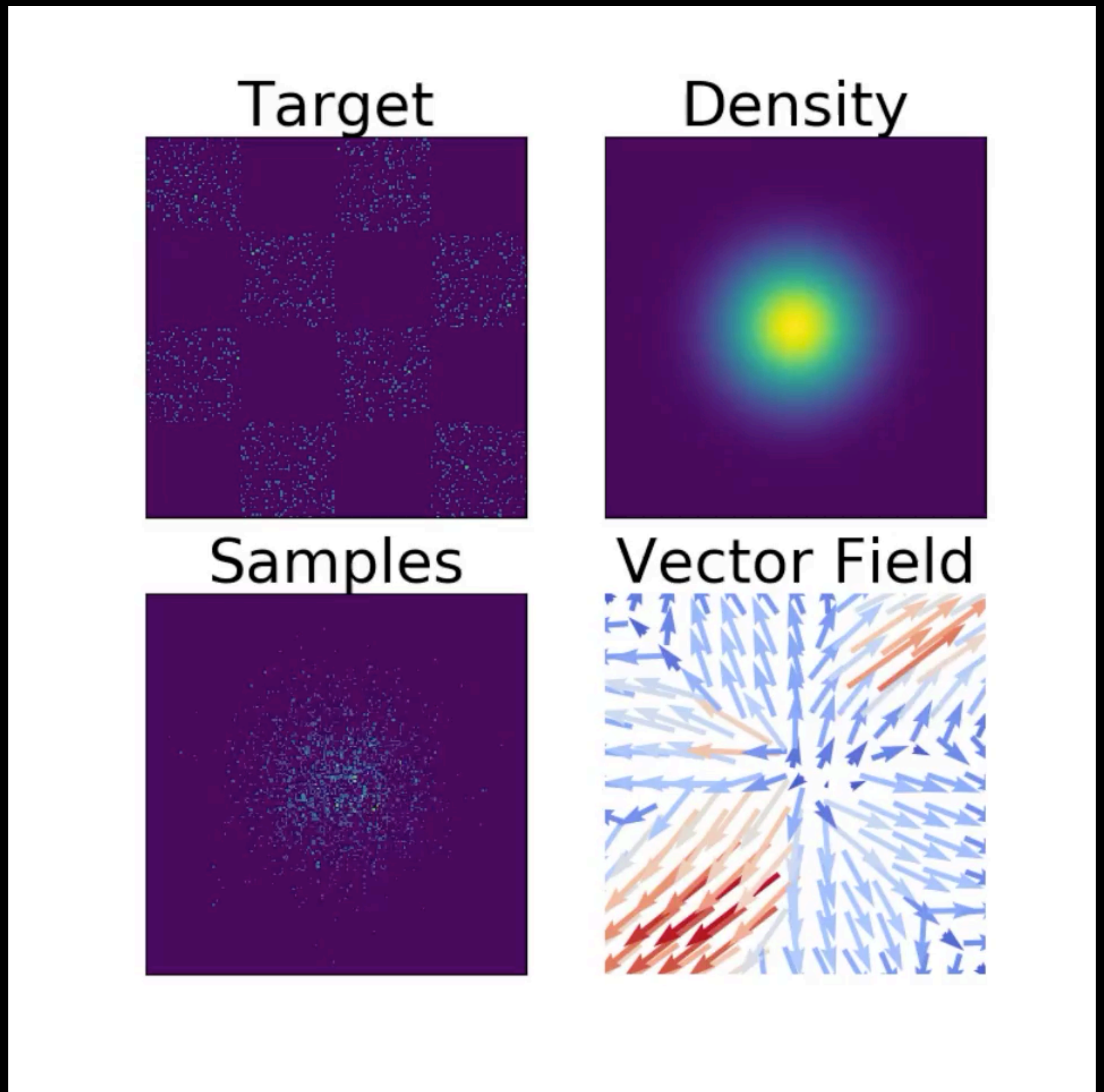
We call it Free-Form Jacobian of Reversible Dynamics (FFJORD)


Samples
Vector Field

FFJORD in action







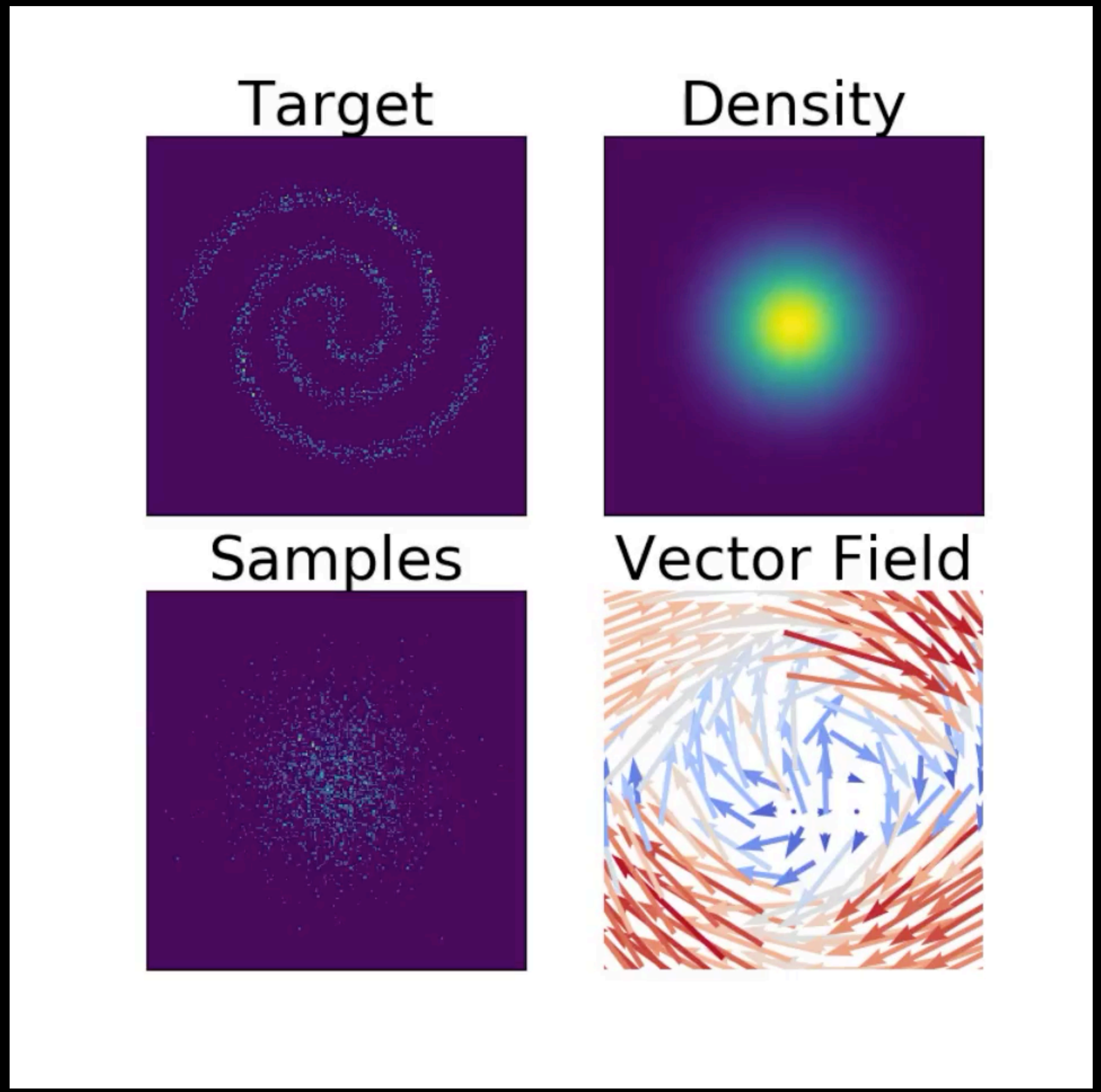


Image Models

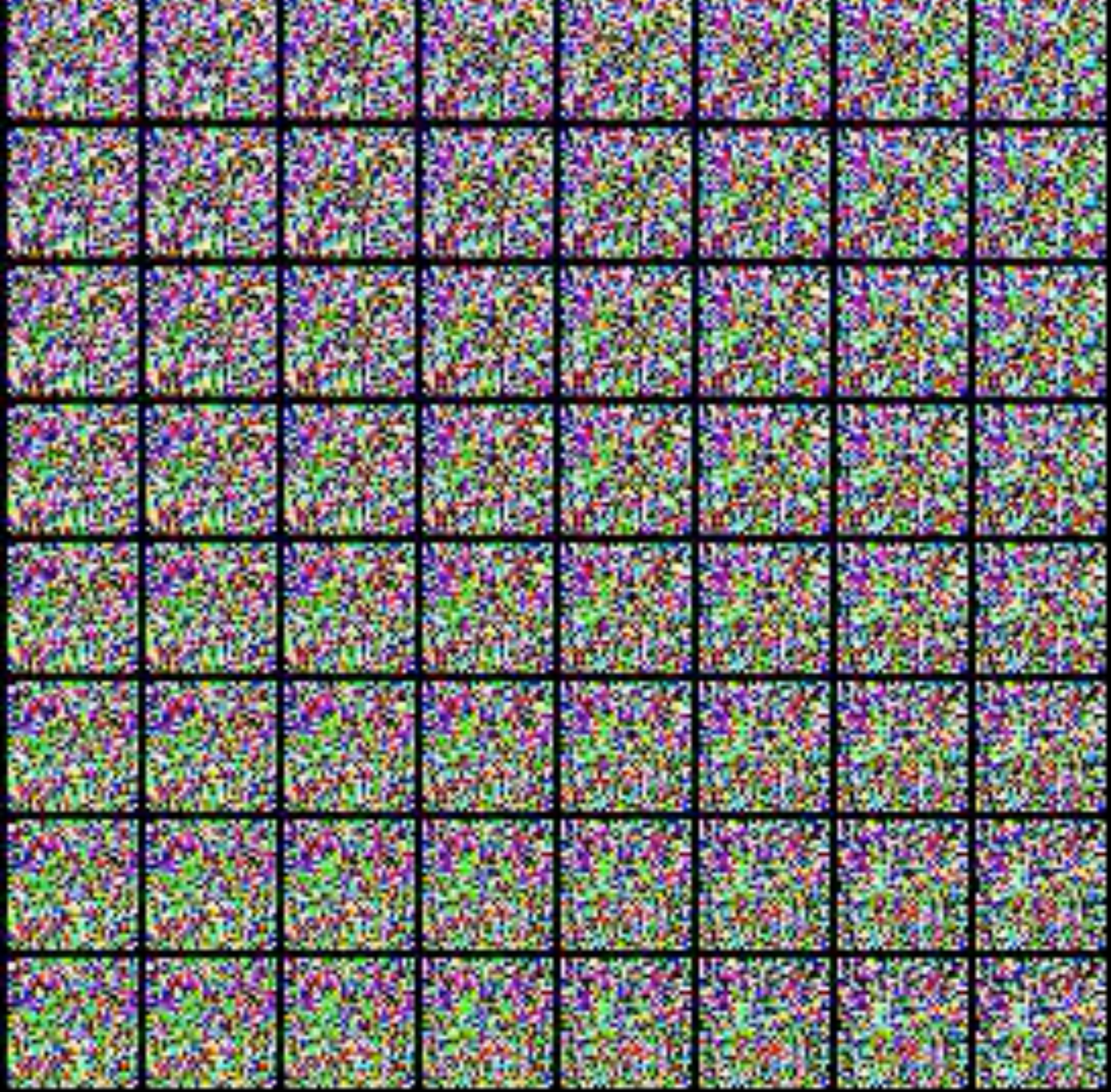


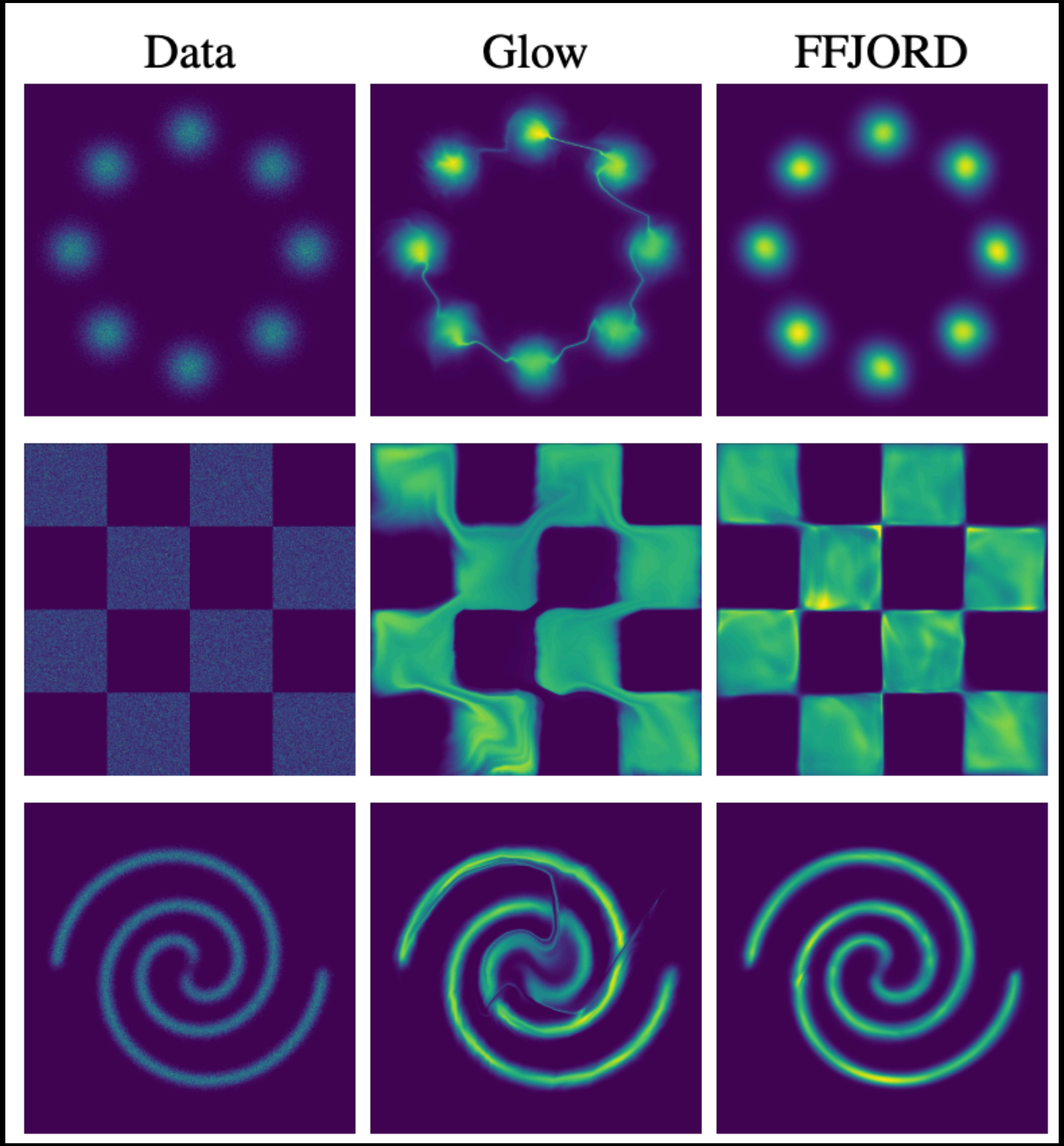
Image samples

- FFJORD outperforms both real-nvp and Glow on MNIST and can match their performance using a single flow step

- performs comparably to Glow on CIFAR10 while using 2% as many parameters

- Real win is on non-image domains where we don’t know how to partition dimensions

Image samples



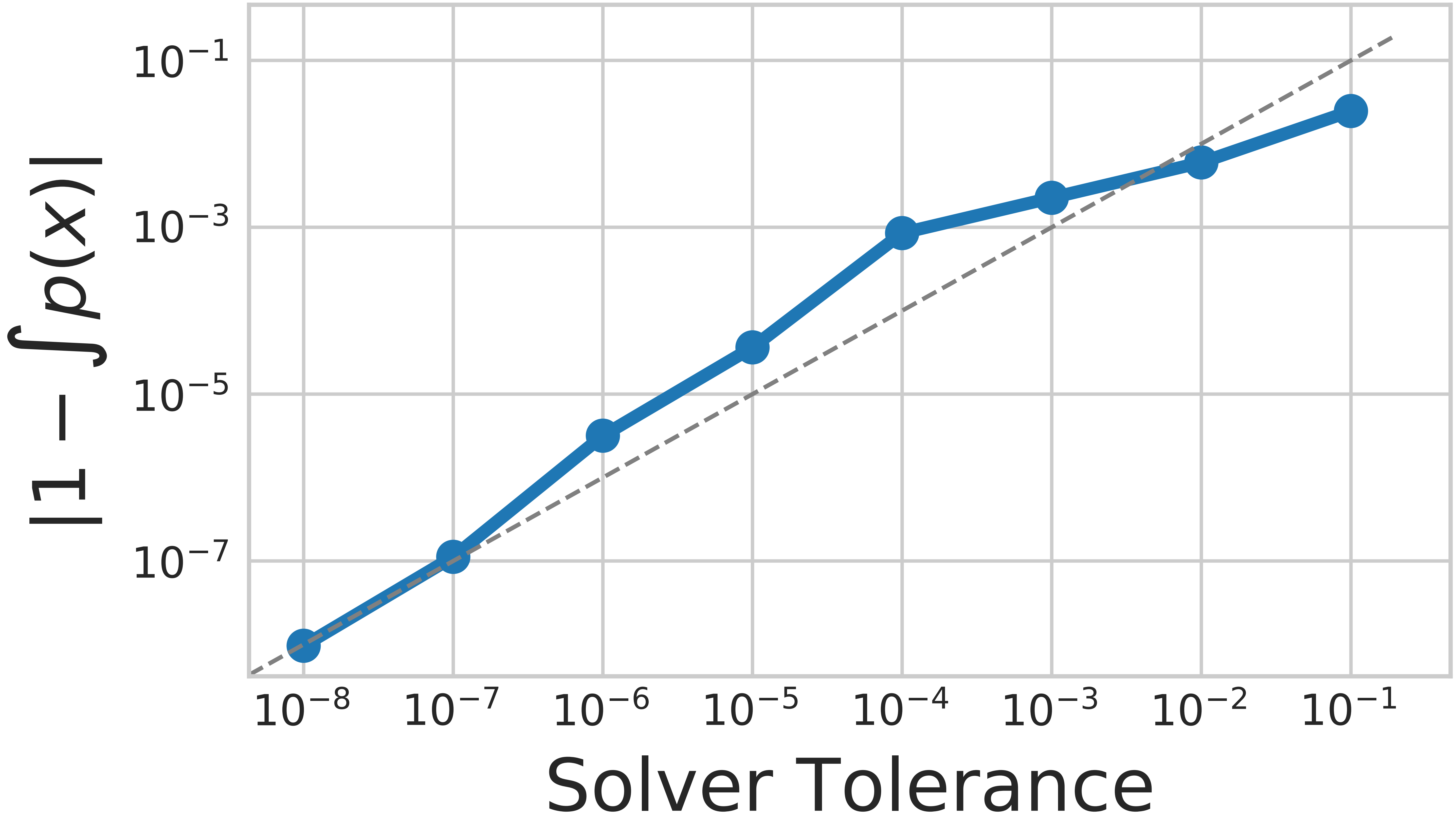
	Method	Train on data	One-pass Sampling	Exact log-likelihood	Free-form Jacobian
	Variational Autoencoders	✓	✓	✗	✓
	Generative Adversarial Nets	✓	✓	✗	✓
	Likelihood-based Autoregressive	✓	✗	✓	✗
Change of Variables	Normalizing Flows	✗	✓	✓	✗
Change of Variables	Reverse-NF, MAF, TAN	✓	✗	✓	✗
Change of Variables	NICE, Real NVP, Glow, Planar CNF	✓	✓	✓	✗
Change of Variables	**FFJORD**	✓	✓	✓	✓

Density Modeling

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	CIFAR10
Real NVP	-0.17	-8.33	18.71	13.55	-153.28	1.06*	3.49*
Glow	-0.17	-8.15	18.92	11.35	-155.07	1.05*	**3.35***
FFJORD	**-0.46**	**-8.59**	**14.92**	**10.43**	**-157.40**	**0.99*** ($1.05^{\dagger}$)	3.40*
MADE	3.08	-3.56	20.98	15.59	-148.85	2.04	5.67
MAF	-0.24	-10.08	17.70	11.75	-155.69	1.89	4.31
TAN	-0.48	-11.19	15.12	11.01	-157.03	-	-
MAF-DDSF	-0.62	-11.96	15.09	8.86	-157.73	-	-

What about numerical error?

- Is density accurate?
- Can choose precision.

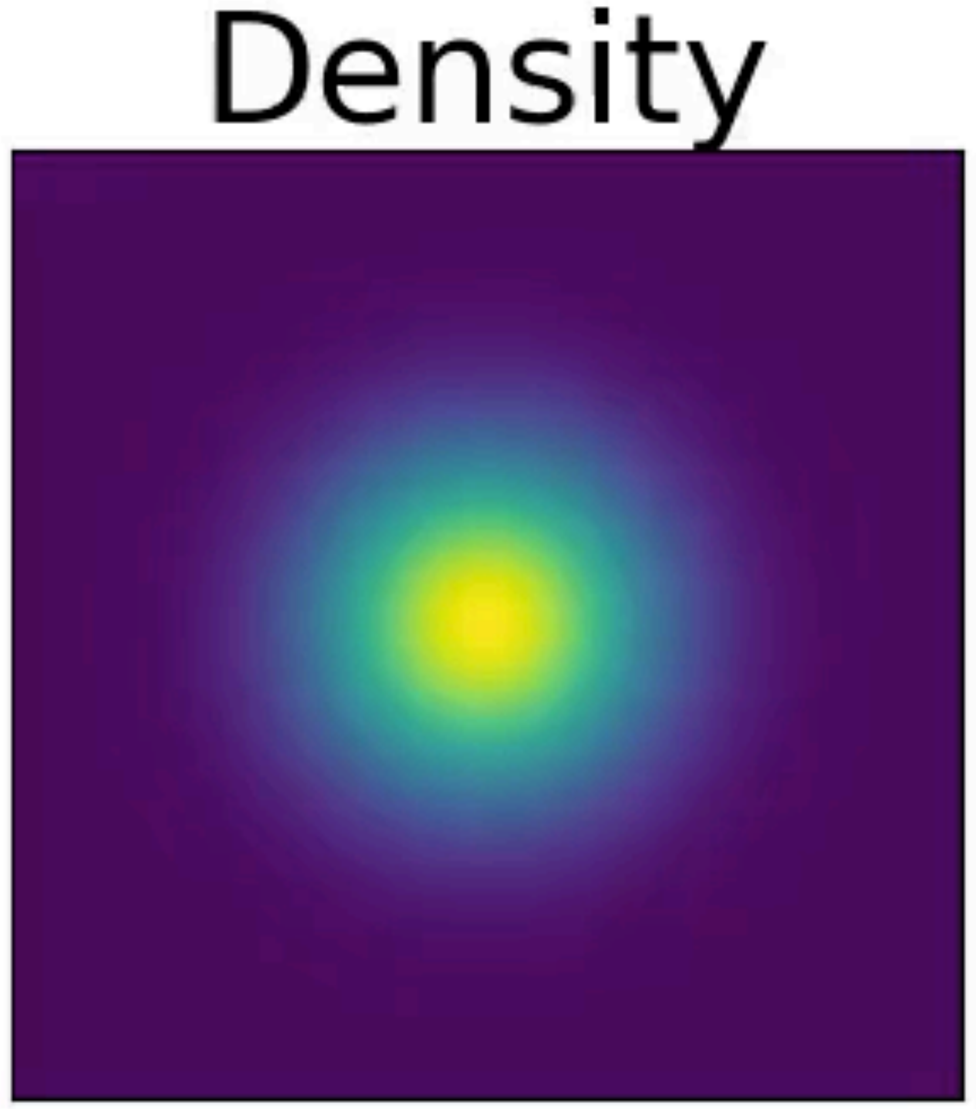


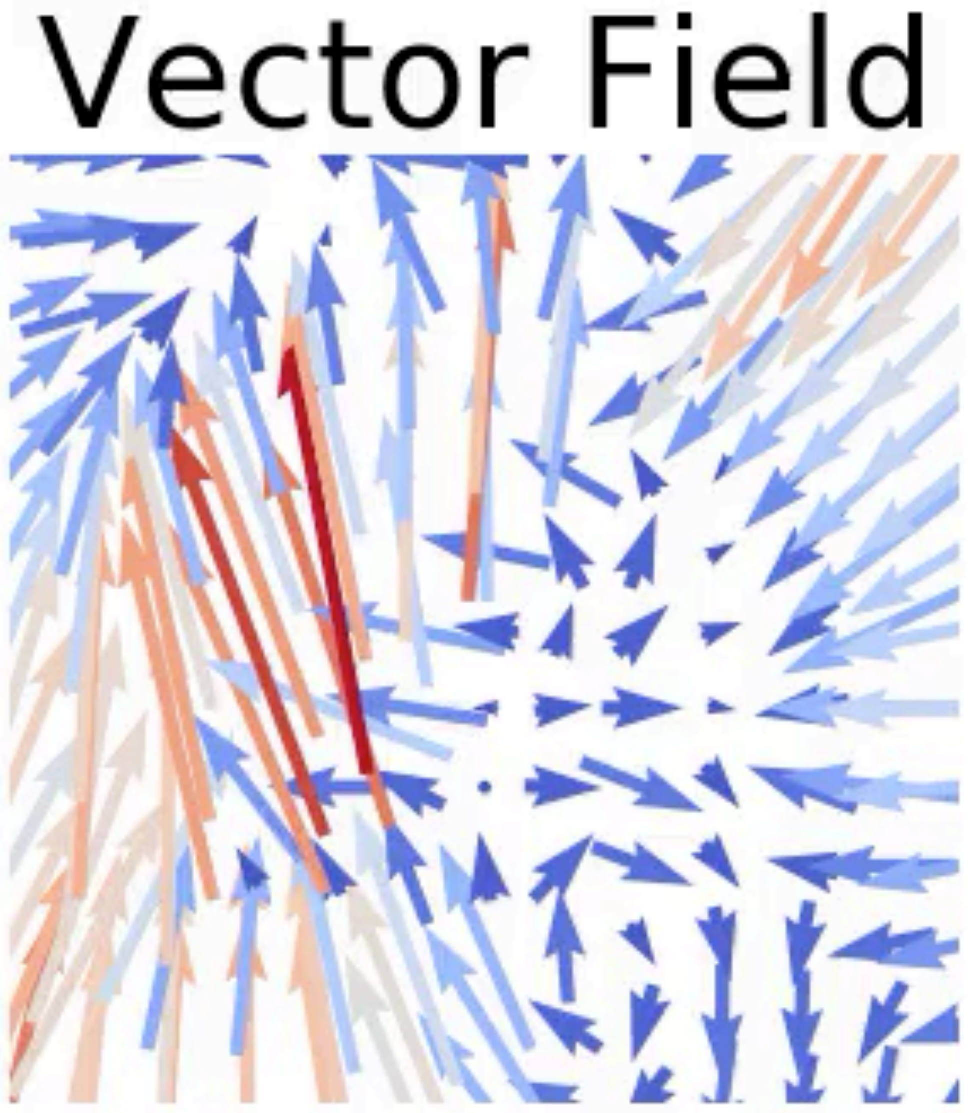
FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models
Grathwohl, Chen, Bettencourt, Sutskever, Duvenaud

PyTorch Code Available

- Adaptive-step solvers w/ O(1) memory backprop
  - Runge-Kutta 4(5)
  - Adaptive-order Adams.
- ODEs in deep nets, and deep nets in ODEs

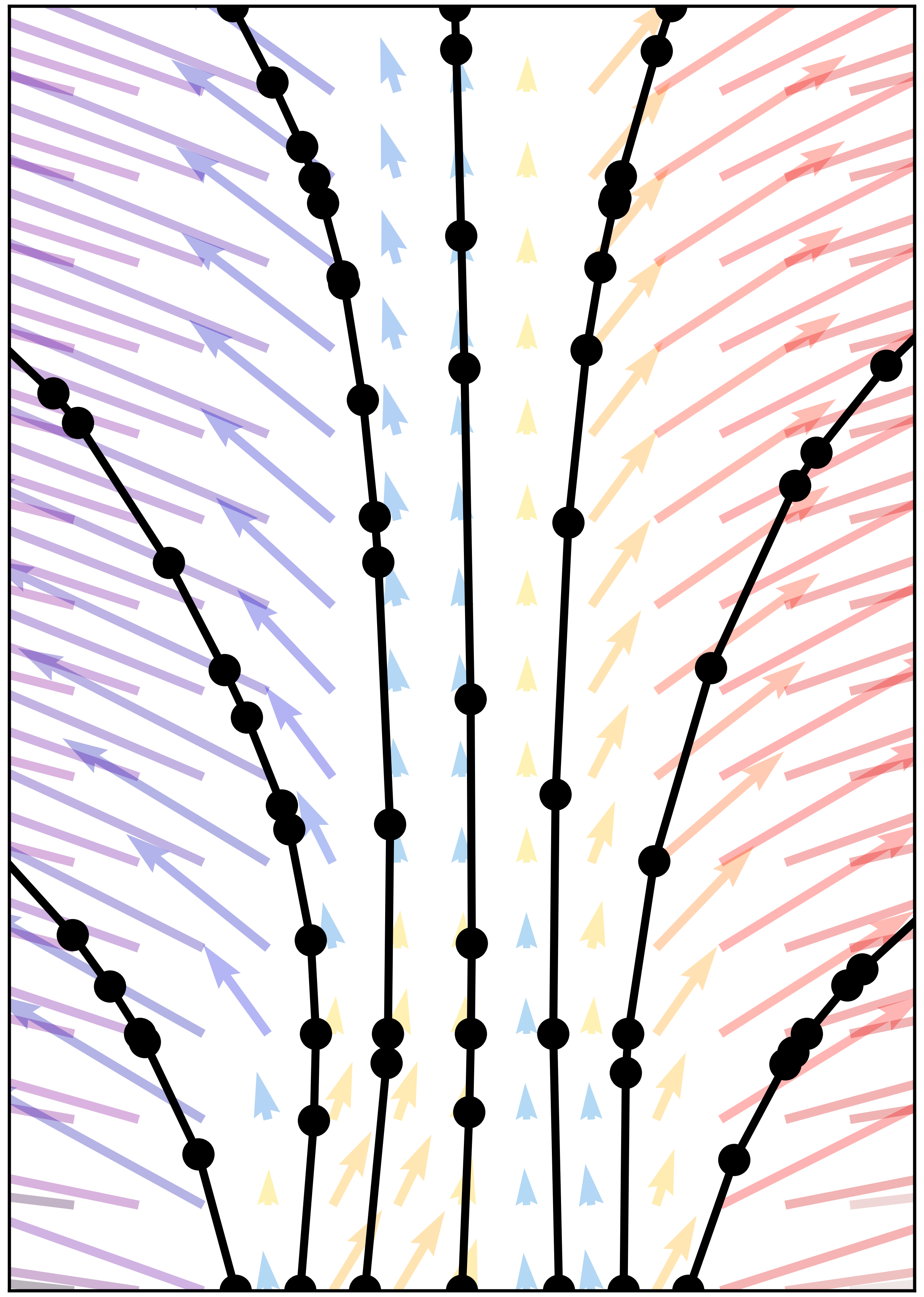
github.com/rtqichen/torchdiffeq
github.com/rtqichen/ffjord





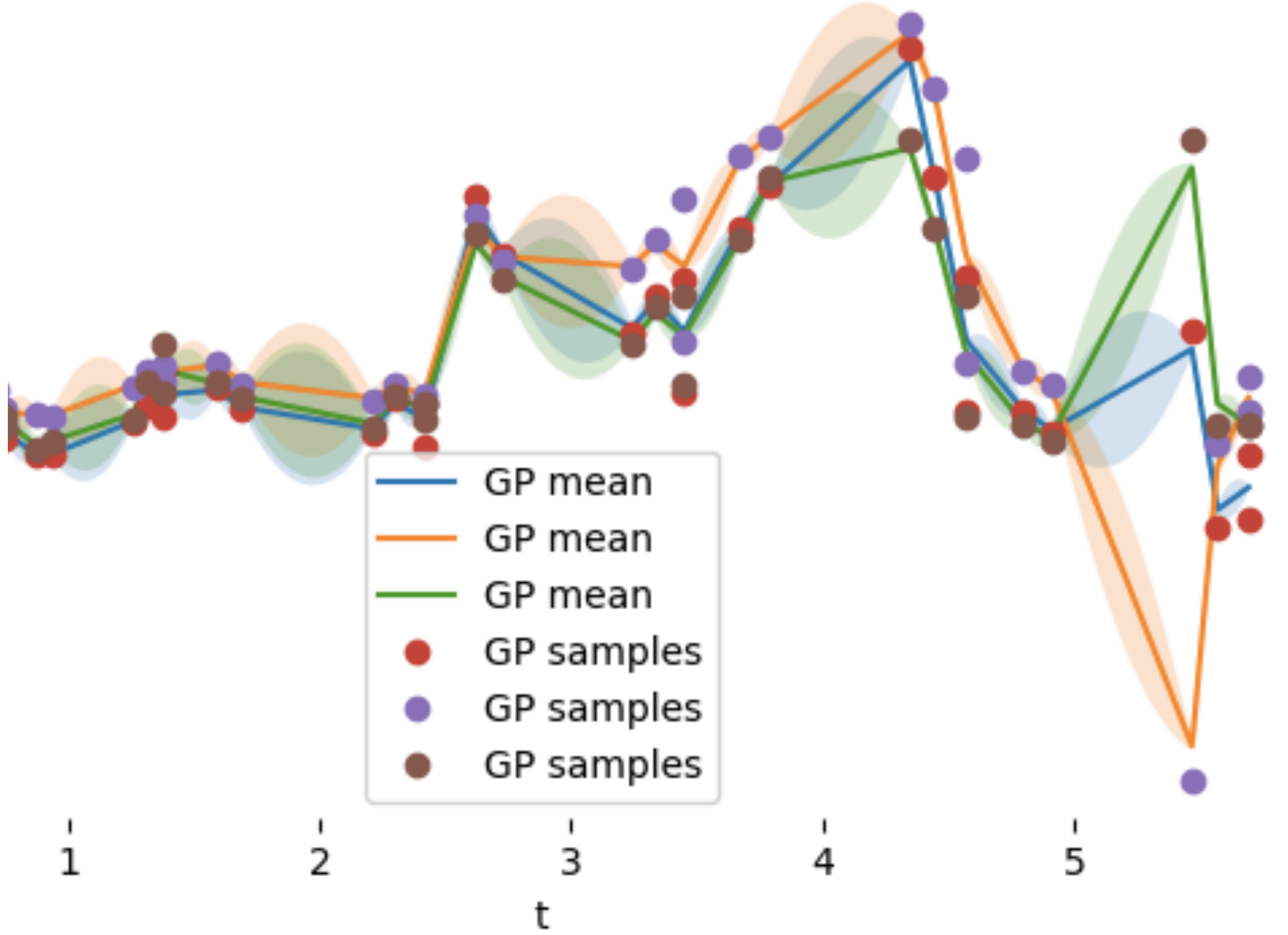
Recap

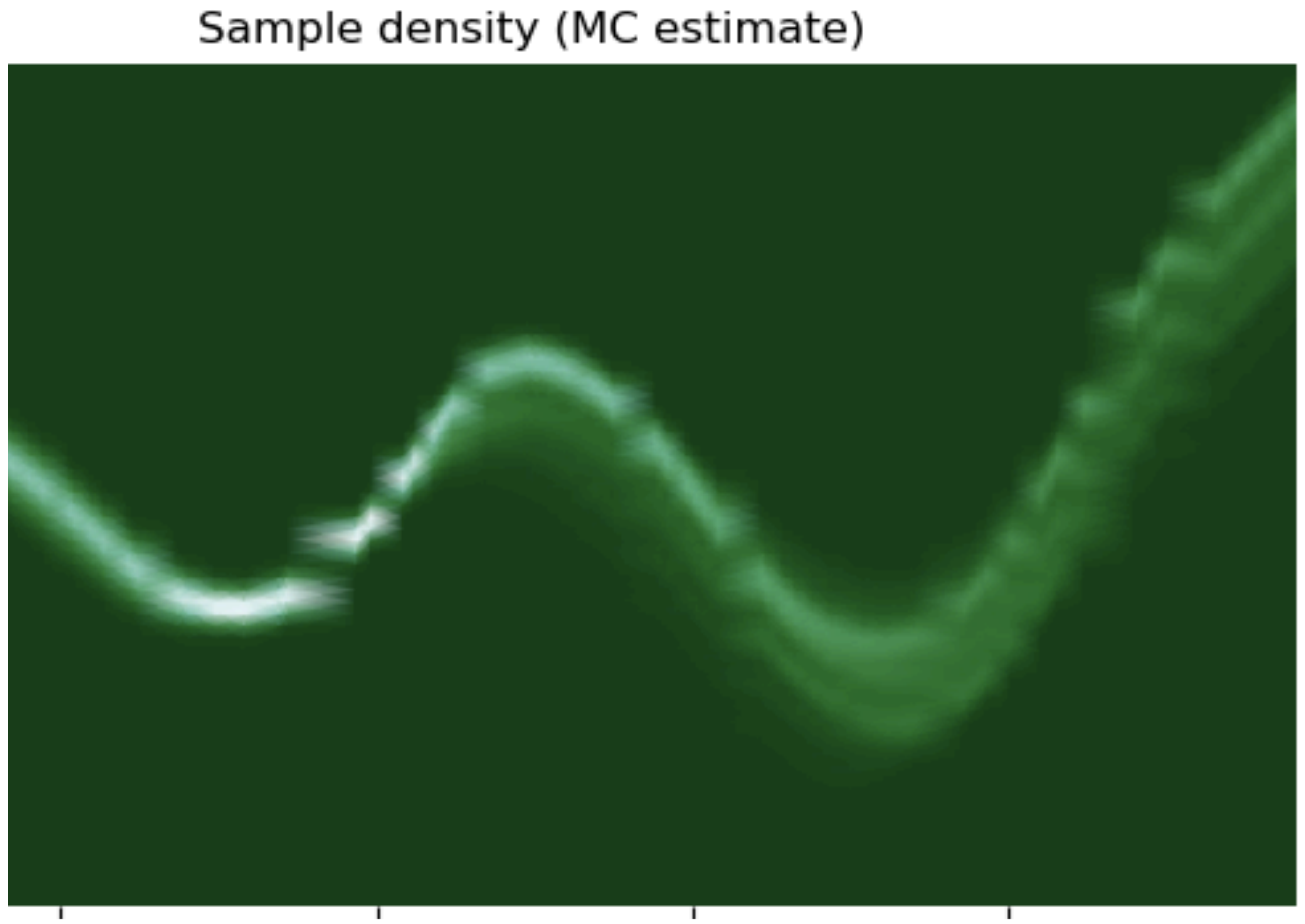
- New family of continuous-depth architectures
  - Adaptive computation
  - Constant memory cost
  - Tradeoff speed vs precision
- Time series with irregular observation times
- New class of generative density models



Next steps

- Latent Stochastic Differential Equations
- Regularize dynamics to need fewer evaluations
- Experiment with architectures & solvers





Ricky Chen*, Yulia Rubanova*, Jesse Bettencourt*,
David Duvenaud

Thanks!

VECTOR INSTITUTE

https://github.com/rtqichen/torchdiffeq