Simulation-Free Training of Neural ODEs on Paired Data

Semin Kim[1*] **Jaehoon Yoo**[1*] **Jinwoo Kim**[1]
Yeonwoo Cha[1] **Saehoon Kim**[2] **Seunghoon Hong**[1]
[1]KAIST [2]Kakao Brain

Abstract

In this work, we investigate a method for simulation-free training of Neural Ordinary Differential Equations (NODEs) for learning deterministic mappings between paired data. Despite the analogy of NODEs as continuous-depth residual networks, their application in typical supervised learning tasks has not been popular, mainly due to the large number of function evaluations required by ODE solvers and numerical instability in gradient estimation. To alleviate this problem, we employ the flow matching framework for simulation-free training of NODEs, which directly regresses the parameterized dynamics function to a predefined target velocity field. Contrary to generative tasks, however, we show that applying flow matching directly between paired data can often lead to an ill-defined flow that breaks the coupling of the data pairs (*e.g.*, due to crossing trajectories). We propose a simple extension that applies flow matching in the embedding space of data pairs, where the embeddings are learned jointly with the dynamic function to ensure the validity of the flow which is also easier to learn. We demonstrate the effectiveness of our method on both regression and classification tasks, where our method outperforms existing NODEs with a significantly lower number of function evaluations. The code is available at `https://github.com/seminkim/simulation-free-node`.

1 Introduction

Continuous-depth models [3, 8] have received growing attention as an alternative to deep feedforward networks composed of a stack of discrete layers. As they approximate continuous, infinite-depth models with a constant set of model parameters, they are parameter-efficient models that can reuse their parameters across depth. Additionally, by controlling the number of function evaluations (NFEs) during inference, we can choose the optimal trade-off between performance and computational cost. This trade-off can be adjusted without retraining the model, unlike conventional neural networks that require separately trained models of different capacities to achieve a similar flexibility.

A prominent class of continuous-depth models is Neural Ordinary Differential Equations (NODEs) [8], which utilize continuous-time differential equations to describe evolutions of intermediate states. In NODEs, a parameterized dynamics function learns the time derivative of the continuous transformation of a state. NODEs can be interpreted as a continuous limit of residual networks [8, 15], and hence they offer a versatile framework similarly to ResNet [17]. As a result, NODEs have been successfully applied to tasks such as physically informed modeling [37, 41], time series modeling [5, 23, 36], and generative modeling with normalizing flows [12, 14].

However, despite their success in various tasks, the application of NODEs to learn deterministic mappings between paired data, such as regression or classification, remains under-explored. This is largely due to the substantial computational burden of training NODEs, since numerical ODE solving during training is inherently serial, slow, and requires large NFEs.

[*]Equal Contribution.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

 https://doi.org/10.52202/079017-1924

Recently, an alternative, *simulation-free* training method has been introduced for ODE-based generative models [1, 28, 29]. Known as flow matching, this approach eliminates the need for the expensive ODE solving during training by directly regressing the model on a presumed velocity field. While this method significantly improves training efficiency by requiring only one function evaluation per training step, its effectiveness on deterministic tasks has not been well explored.

In this work, we investigate a simulation-free training method for continuous-depth models in learning deterministic mapping between paired data. We first identify potential problems that occur when applying flow matching objective to NODEs, and then propose a method to mitigate them. With our simulation-free training scheme, we can significantly reduce the computational demands of NODE training while maintaining competitive performance in deterministic tasks. We demonstrate the advantages of our method on both regression and classification problems, providing empirical evidence of its effectiveness and versatility in common supervised learning settings. Furthermore, by leveraging simple flows, our method can achieve superior performance compared to NODEs in low-NFE regime.

2 Preliminary

NODEs as Continuous-Depth Models We aim to learn a deterministic mapping between paired data $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{N}$ with a continuous-depth model. To this end, we consider Neural Ordinary Differential Equations (NODEs) [8], which can be regarded as a continuous limit of residual networks. Formally, a single layer of a residual network transforms a hidden state z with a residual connection, *i.e.* $z_{t+1} = z_t + h_\theta(z_t)$. This update resembles a single step of Euler solver, which uses parameterized dynamics function h_θ to approximate the derivative of z with respective to t. Taking this discretization to the continuous limit, residual networks are equivalent to the following ODE:

$$v_t = \frac{\mathrm{d}z_t}{\mathrm{d}t} = h_\theta(z_t, t). \tag{1}$$

To train a NODE with a loss defined on the state at time t_2, we first solve an initial value problem starting from a known initial state z_{t_1}:

$$z_{t_2} = z_{t_1} + \int_{t_1}^{t_2} h_\theta(z_t, t)dt = \texttt{ODESolve}(z_{t_1}, h_\theta, t_1, t_2). \tag{2}$$

For the task of fitting a paired dataset, each data-label pair (x, y) is placed on the state-space of the ODE as endpoints (z_0, z_1) for time interval $[0, 1]$. This requires matching the dimensions of data x and label y to states z, which is done by projecting data x to initial state $z_0 = f_\phi(x)$,[2] and the solved final state z_1 to predicted label $\hat{y} = d_\psi(z_1)$. Then, NODEs are trained end-to-end to minimize the loss $\mathcal{L}(\hat{y}, y)$. However, this approach is inherently slow and computationally intensive, as it requires a large number of sequential function evaluations by the ODE solver [8, 22, 34]. There is also a hidden cost in integrating the ODE backward in time for gradient estimation (*i.e.*, adjoint method [8]), which necessitates additional serial function evaluations and is numerically unstable [43]. Also, common choices of adaptive-step ODE solvers tend to make NODEs learn arbitrarily complex trajectories [42], making both training and inference computationally expensive.

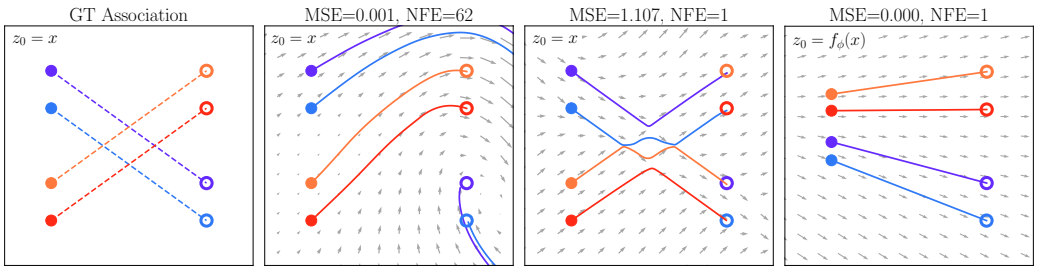
Flow Matching for Simulation-Free Training Instead of optimizing Eq. (1) via the expensive initial value problem of Eq. (2), we can alternatively employ the flow matching framework [1, 28, 29] to directly regress the dynamics function h_θ to the velocity field v_t by:

$$\mathbb{E}_t[||h_\theta(z_t, t) - v_t||_2^2]. \tag{3}$$

Since the intermediate state z_t and its velocity v_t are generally unknown and intractable to compute, existing works [1, 28, 29] employ predefined, tractable conditional velocity fields defined per sample. Specifically, a closed form expression of z_t is given as an interpolant of two endpoints and time (*i.e.* $z_t = \alpha_t z_0 + \beta_t z_1$), and a target velocity field is constructed as a time derivative of the interpolant. One simple instantiation of an interpolant and its corresponding target velocity field is *linear* dynamics with a constant speed [28, 29]:

$$z_t = (1 - t)z_0 + tz_1, \quad v_t = z_1 - z_0. \tag{4}$$

[2] A typical choice for f in the literature is the identity mapping (*i.e.*, $z_0 = x$).



(a) Ground Truth (b) Neural ODE (c) Flow Matching (FM) (d) Embed. + FM (Ours)

Figure 1: Comparison of the learned trajectories. The final train loss (MSE) and training NFE are shown above each plot. (a) We consider deterministic regression task of four data pairs, each of which is represented by two circles (filled and empty circles) connected by dotted lines. (b) NODEs can correctly associate the pairs but through complex paths (solid lines) that require large NFEs. (c) Flow matching with linear velocity can greatly reduce the training NFEs by simulation-free training, but fails to associate the correct pairs due to the crossing trajectories induced by predefined dynamics. (d) The proposed method can alleviate the problems by learning the embeddings for data jointly with the flow matching.

Since we can obtain v_t at arbitrary time t, this allows the dynamics function to be trained in a simulation-free manner, where the regression of the vector field is performed parallel in time. Simulation-free training in the flow matching models is intriguing, since there is no need for serialized function evaluations during training as well as backpropagation through time. Employing a simple velocity field such as Eq. (4) also greatly simplifies the trajectory of the dynamics function, reducing the number of function evaluations for inference. However, flow matching has been mainly studied for generative modeling, which aims to find a transportation map between two marginal distributions $p(z_0)$ and $p(z_1)$ while learning arbitrary *per-sample* couplings between $z_0 \sim p(z_0)$ and $z_1 \sim p(z_1)$. It makes it difficult to be applied in deterministic regression tasks where the per-sample coupling is defined by data pairs $(z_0, z_1) \sim (x, y)$. We elaborate on this issue in the next section.

3 Challenges in Flow Matching for Paired Data

We seek to adopt the simulation-free training objective in Eq. (3) to learn a continuous-depth model of Eq. (1). However, we find that careful consideration is required when employing the flow-matching objective to model a deterministic mapping between paired data.

To illustrate this point, we provide a toy example in Fig. 1. Here, we consider a simple regression task with two-dimensional inputs and outputs, translating four points at $x = -1$ to the ones at $x = 1$ but in a different order in the y axis. For ease of analysis, we consider simple linear velocity field in Eq. (4) for flow matching.

First, we observe that NODE successfully learns to associate inputs and outputs by solving Eq. (2) (Fig. 1 (b)). However, the learned trajectories are highly non-linear and complex which leads to a large number of function evaluations for training and inference.

The flow matching can greatly improve the computation cost by optimizing the simulation-free objective in Eq. (3). However, we observe that it fails to infer the correct input-output correspondence according to the learned trajectories (Fig. 1 (c)). This is because some target trajectories induced by the predefined velocity field are crossing each other, which cannot be modeled by ODEs [40]. As a result, the learned dynamics function at the intersection of crossing trajectories produces their mean velocity, which makes the inferred trajectory fail to arrive at the correct output.

Note that the crossing trajectories induced by the predefined flow are less problematic in generative tasks. It is because their objective is to transport between two marginal distributions $p(z_0)$ and $p(z_1)$ while allowing arbitrary association between samples $z_0 \sim p(z_0)$ and $z_1 \sim p(z_1)$. It is evident from Fig. 1 (c), where the model fails to preserve the initial coupling in the dataset but still yields valid marginal distribution at the output. However, it is not desirable for deterministic regression, since the goal is to preserve *per-sample* correspondence between data and label.

https://doi.org/10.52202/079017-1924

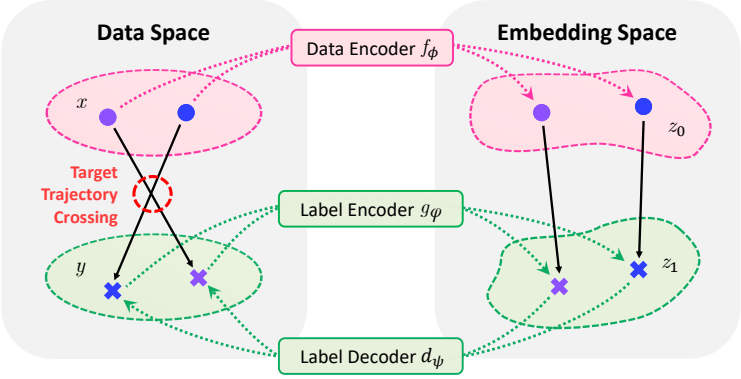


Figure 2: An overview of our framework. We avoid the crossing trajectory problem in data space by introducing learnable encoders that project data and label to embedding space. In the learned embedding space, the presumed dynamics induce valid target velocity field.

4 End-to-End Latent Flow Matching

Previous discussions suggest that predefined flow between paired data can yield crossing trajectories that cannot be modeled by the well-defined ODE, which leads to invalid paths by the dynamics function that break the coupling of data and label. As a simple solution, we propose to learn the embeddings of data and label jointly with dynamics function, in such a way that the predefined flow induces non-crossing trajectories in the embedding space (Fig. 1 (d)).

Fig. 2 illustrates the overall framework. The proposed framework comprises the data encoder f_ϕ, label encoder g_φ, label decoder $d_\psi \approx g_\varphi^{-1}$, and dynamics function h_θ. Given a data pair (x, y), our model first projects the data and label using respective encoders by $(z_0, z_1) = (f_\phi(x), g_\varphi(y))$. Then, for all $t \in [0, 1]$, the state z_t and the corresponding target velocity v_t are obtained by some predefined dynamics (*e.g.,* Eq. (4)). We train the encoders and dynamics function jointly using the flow matching loss, while an additional label autoencoding objective is applied to the label encoder and decoder. The inference procedure remains similar to that of NODEs: given the input embedding $z_0 = f_\phi(x)$, we obtain the state $\hat{z}_1$ by solving ODE in Eq. (2) and decode it to a label by the label decoder $\hat{y} = d_\psi(z_1)$. Below, we describe the learning objective and optimization of our method in detail.

Flow Loss With the learnable projections $z_0 = f_\phi(x), z_1 = g_\varphi(y)$ and any simple dynamics assumption $z_t = \alpha_t z_0 + \beta_t z_1$, our flow matching objective is defined by:

$$\mathcal{L}_{flow}(\theta, \phi, \varphi) = \mathbb{E}_{t,(x,y)\sim\mathcal{D}}[||h_\theta(z_t, t) - v_t||_2^2]. \tag{5}$$

In contrast to conventional flow matching (Eq. (3)) that optimizes the dynamics function given the fixed endpoints z_0 and z_1, the loss in Eq. (5) is optimized jointly with two encoders f_ϕ and g_φ. Since the loss has the optimum at $\mathcal{L}_{flow}(\theta^*, \phi^*, \varphi^*) = 0$ when the dynamics function h_{θ^*} perfectly fits to the velocity field v_t, optimizing Eq. (5) guarantees that the optimal encoders f_{ϕ^*} and g_{φ^*} induce non-crossing target trajectories in the embedding space.

Conceptually, our approach resembles the *reflow* procedure in Rectified Flow [29], which straightens the target trajectories by recursively rewiring z_0 and z_1 according to the trajectory obtained by solving an ODE. In contrast, our approach straightens the trajectory by learning the encoders such that the trajectories defined by the predefined flow do not cross in the embedding space, which does not involve ODE solving and, most importantly, preserves the initial coupling (z_0, z_1).

However, contrary to Eq. (3), we observe that optimizing the flow matching objective with learnable encoders in Eq. (5) can lead to trivial degenerate solutions. One such example is when both encoders collapse to output constant ignoring data, which produces trivial targets for the dynamic model (*e.g.*, $z_0 = z_1 = h_\theta(\cdot, \cdot) = 0$). To avoid trivial solutions, both encoders require strong regularization to be relevant to the inputs. Fortunately, such regularization is naturally provided by the learning objective for the label decoder d_ψ, which is described below.

Label Autoencoding Loss We define the *label autoencoding loss* as a simple mean squared error (MSE) between the label y and its reconstruction obtained by label encoder and decoder:

$$\mathcal{L}_{label\_ae}(\psi, \varphi) = \mathbb{E}[||d_\psi(g_\varphi(y)) - y||_2^2]. \tag{6}$$

The primary purpose of Eq. (6) is to provide learning signals for the label decoder d_ψ, which is used to decode the predicted z_1 to the label $\hat{y} = d_\psi(z_1)$ at inference. However, Eq. (6) also functions as a regularization to avoid trivial solutions of Eq. (5) by preventing the label encoder g_φ from ignoring the input label y. Regularizing the label encoder is sufficient to prevent trivial solutions in principle, since it eliminates the trivial targets for both data encoders f_ϕ and the dynamics function h_θ.

Objective Function The overall objective function for the proposed framework is given by combining the two losses introduced above:

$$\min_{\theta,\phi,\varphi,\psi} \mathcal{L}_{flow}(\theta, \phi, \varphi) + \mathcal{L}_{label\_ae}(\psi, \varphi) \tag{7}$$

Theoretically, we can show that optimizing the combination of flow loss and autoencoding loss in Eq. 7 can indeed mitigate the problem of target trajectory crossing:

Proposition 1. *There exist f_ϕ and g_φ that induce non-crossing target trajectory for any data pair in $\mathcal{D}$ while minimizing $\mathcal{L}_{label\_ae}$.*

Proof. The proof can be found in App. A.1. □

Proposition 2. *If g_φ is injective, the following equivalence holds: f_ϕ, g_φ and h_θ minimize $\mathcal{L}_{flow}$ to zero for all $t \in [0, 1)$ if and only if f_ϕ and g_φ induce non-crossing target trajectory and h_θ perfectly fits the induced target velocity, for any data pair in $\mathcal{D}$.*

Proof. The proof can be found in App. A.2. □

Proposition 1 ensures the existence of encoders that do not induce crossing in the target trajectory, while Proposition 2 suggests that the encoders are optimal when flow loss is minimized, assuming the label encoder is injective. Since the label autoencoding task enforces g_φ to be injective, combining the two propositions implies that optimizing the objective function in Eq. (7) prevents the encoders from inducing target trajectory crossing and enables the dynamics function to accurately fit the induced trajectories.

Optimization During the optimization of Eq. (7), we find that two additional regularizations on the encoders are useful in further preventing suboptimal solutions and stabilizing training and inference.

First, although the autoencoding loss in Eq. (6) prevents degenerate encoders in principle, we find that flow matching loss in Eq. (5) can still induce an ill-behaved local minima for the data encoder f_ϕ. For instance, under linear velocity field (Eq. (4)), a collapsed, constant data encoder, *e.g.*, $z_0 = 0$, makes both intermediate state and target velocity depend only on z_1 at $t \in (0, 1]$, *e.g.*, $z_t = tz_1$ and $v_t = z_1$. In this case, a dynamics function that scales with time, $h_\theta(z, t) = z/t$, can fit the target velocity field for all $t \in (0, 1]$ although it does not yield meaningful mapping between (x, y). We find that explicitly sampling $t = 0$ with a certain probability during training effectively resolves the issue.

Second, we empirically find that the label encoder tends to reduce the scale of the output embeddings to optimize the flow matching loss. Although this is not a fundamental problem in principle, we observe that it often affects the generalization performance by making the model prone to small numerical errors at inference, such as prediction errors in the dynamics function or discretization errors in the ODE solver. To address this, we encourage the label embeddings to *repel* each other, so that they construct a robust destination point for ODE solving during inference. Specifically, during training we add random noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ to label embedding and let the label decoder to reconstruct original label from it, *i.e.*, we minimize $\mathbb{E}[||d_\psi(g_\varphi(y) + \epsilon) - y||_2^2]$ instead of Eq. (6). We empirically find these techniques helpful for better optimization, as shown in Sec. 6.3.

 https://doi.org/10.52202/079017-1924

5 Related Work

Enhancing Efficiency of NODEs Several previous works have addressed the problem of high numbers of function evaluations (NFEs) in the forward process of Neural Ordinary Differential Equations (NODEs). Most approaches involve imposing regularization on the learned trajectory, such as penalizing higher-order derivatives [20] or incorporating kinetic regularization [12]. Similar effects can be achieved through weight decay [14], augmenting dimensions [11], or using internal solver heuristics [34]. Additionally, sampling the end time of the integration interval [13] has also been explored as a simple solution. These methods encourage the model to learn simpler trajectories, thereby effectively reducing the NFE required to solve the ODE. However, with the common choice of adaptive-step solvers, there often remain dozens of sequential function evaluations during a single training step, making the training of NODEs slow and computationally intensive.

Simulation-Free Training on Paired Dataset Some studies have explored simulation-free training methods for fitting dynamics functions on paired datasets, primarily within the context of diffusion probabilistic models (DPMs) [18]. DPMs achieve simulation-free training by learning to denoise data at multiple noise levels in parallel [30]. During inference, DPMs generate data from standard Gaussian noise through iterative denoising. These models have been applied to various vision tasks aimed at learning deterministic mappings, including segmentation [2, 4, 9], object detection [7], and image restoration tasks [27, 35, 39], among others. While these applications are impressive, they are tailored specifically to their target tasks and do not represent general methods for paired data with diverse label structures.

One notable work in this area is CARD [16], which introduced a new conditional diffusion process for classification and regression tasks, making it applicable to arbitrary regression and classification data. Although these works, based on diffusion models trained with denoising objectives, align more closely with neural SDEs rather than ODEs, we include a comparison with CARD in our main experiments to provide a comprehensive evaluation.

6 Experiments

6.1 Experimental Setup

Baselines and Datasets We validate the effectiveness of our method on various datasets for both regression and classification tasks. For baselines, we compare our method with the standard NODE [8] and previous works that utilize regularization to reduce NFEs. Specifically, we compare against STEER [13], which introduces stochasticity in the integration interval, and RNODE [12], which regularizes the norm of the velocity field. Additionally, we include a comparison with CARD [16], a classification and regression model based on diffusion. Following prior work [11, 13], we use MNIST [26], SVHN [33], and CIFAR10 [25] for image classification experiments. For regression tasks, we use UCI regression datasets [21], adhering to the protocol used by CARD.

Evaluation We report classification accuracy and root mean square error (RMSE) as the main performance metrics. To quantify computational costs, we report average per-sample NFEs along with training throughput measured by the total training iterations divided by training time. For all NODE baselines, we use the dopri5 [10] adaptive-step solver implemented in torchdiffeq [6] package for both training and inference. Additionally, we report few-step inference results using the Euler solver. For CARD, we perform few-step inference by periodically skipping intermediate steps, similar to DDPM in few-step inference [18, 38], and report the metric of full-step inference in the place of adaptive-step solver in NODEs.

Implementation Details We use the same network architecture across baselines, employing an MLP-based architecture for MNIST and UCI and a convolutional architecture for SVHN and CIFAR10. For classification tasks, we use one-hot encoded labels for y and assign the predicted label $\hat{y}$ by applying argmax on the channel dimension. To handle the significant memory requirements of training NODE-based baselines, we use the adjoint sensitivity method [8] for SVHN and CIFAR10 experiments. When using the adjoint method, we report the total NFE by summing the number of function evaluations in both the forward and backward passes. For our model, we primarily use

Table 1: Experiment results on image classification. Training cost and few-/full-step performances are reported in three datasets. For classification accuracy, numbers indicate the number of function evaluations with Euler solver, where ∞ denotes the result of dopri5 adaptive-step solver. For CARD, we report the 1000-step decoding results instead of using the adaptive solver, as the model was trained on discrete timesteps. CARD† is trained with 4 times longer steps.

		Training Cost		Accuracy over NFE (%)				
Dataset	Model	NFE	Throughput (Batch / sec.)	1	2	10	20	∞
MNIST	NODE	354	0.93	14.20	7.71	8.81	10.83	98.36
	STEER	90	3.22	24.79	29.39	49.57	62.68	99.23
	RNODE	43	3.27	**99.35**	**99.35**	**99.35**	**99.35**	**99.35**
	CARD	**1**	28.95	9.72	15.88	98.92	99.12	98.83
	Ours	**1**	**29.41**	99.33	99.25	**99.35**	99.34	99.30
SVHN	NODE	75	0.42	87.72	91.89	95.18	95.16	95.09
	STEER	110	0.27	30.55	65.00	93.22	94.44	94.55
	RNODE	130	0.14	93.36	95.01	95.37	95.39	95.39
	CARD	**1**	7.94	9.95	16.01	76.87	79.49	65.31
	CARD†	**1**	7.94	9.95	35.91	95.36	95.31	95.23
	Ours	**1**	**9.48**	**96.16**	**96.14**	**96.03**	**96.03**	**96.12**
CIFAR10	NODE	91	0.34	81.18	82.72	86.25	86.33	86.30
	STEER	96	0.33	76.80	77.78	83.55	84.24	84.51
	RNODE	89	0.19	79.61	81.62	85.68	85.96	86.08
	CARD	**1**	7.67	10.22	18.78	84.48	84.68	81.77
	CARD†	**1**	7.67	10.61	33.69	86.67	86.54	86.42
	Ours	**1**	**9.00**	**88.87**	**88.85**	**88.71**	**88.88**	**88.89**

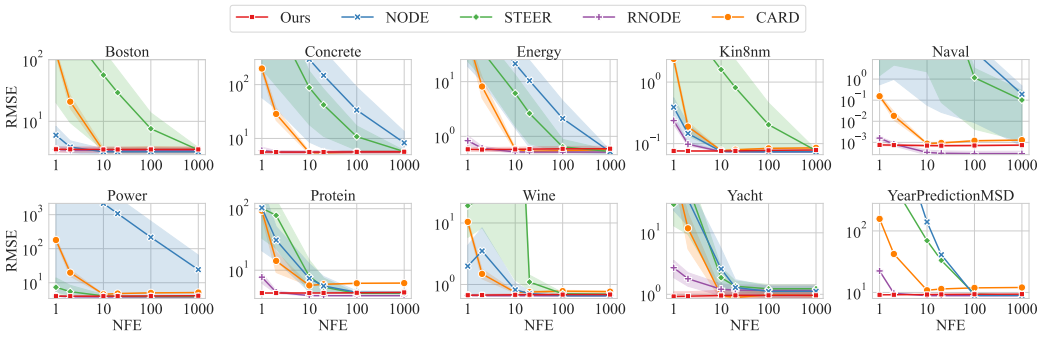


Figure 3: RMSE over NFEs on UCI regression tasks. To control the NFE, we use Euler solver for the evaluation. By assuming linear dynamics, our model shows better performance in low NFE regime.

a simple linear dynamics assumption, which is shown to be effective according to the analysis in Sec. 6.3. Further experimental details can be found in App. B.

6.2 Main Results

Training Cost and Performance In Tab. 1, we report the training cost and classification accuracy on MNIST, SVHN, and CIFAR10. NODE-based baselines suffer from large NFEs—ranging from tens to hundreds—during training, resulting in a low throughput. In contrast, simulation-free training method (*i.e.*, ours and CARD) require only a single function evaluation per training step, significantly boosting training speed.

The trade-off for faster training in simulation-free methods is a constraint on the dynamics that can be learned. However, we observe that this reduced flexibility does not lead to significant performance degradation. Compared to NODE-based baselines, our method shows only minor degradation on MNIST and even improvements on SVHN and CIFAR10 in terms of final classification accuracy. On SVHN and CIFAR10, NODE-based baselines are trained with the adjoint sensitivity method [8] to meet memory requirements, which is known to suffer from inaccurate gradient estimation [42, 43].

https://doi.org/10.52202/079017-1924

Thus we hypothesize that the performance gains on SVHN and CIFAR10 may be due to the accurate gradient calculation achieved through direct backpropagation in our model, which is a positive byproduct of utilizing simulation-free training.

When compared to the diffusion-based baseline, our model tends to show better performance and faster convergence. Specifically, our model outperforms CARD on all three datasets, even compared to CARD variants trained for longer iterations. While diffusion models also benefit from simulation-free training, they are based on stochastic differential equations that induce stochastic and non-linear trajectories. We conjecture that this characteristic is not ideal for few-step inference and also contributes to slower convergence.

Few-Step Inference Ideally, our model trained with linear dynamics will yield a perfectly straight solution trajectory, which can be accurately estimated even with a one-step Euler solver. Consequently, with our linear dynamics assumption, we can significantly enhance inference speed by utilizing few-step inference while maintaining competitive performance compared to many-step solving. To demonstrate this, we report few-step inference results with the Euler solver in Tab. 1. Our model, by avoiding crossing points in the learnable embedding space, produces a linear trajectory and thus exhibits superior few-step performance. This observation aligns with findings in flow matching models [28, 29], which highlight the advantage of linear dynamics for generating high quality samples with low inference cost.

Regression We further analyze the effectiveness of our method in regression tasks, as shown in Fig. 3. See App. D.4 for full results. Despite differences in label structure, we observe similar trends for both classification and regression tasks: our method significantly reduces computational burden during training and demonstrates superior performance in few-step inference with linear dynamics. Similar to the original NODE, our model can be effectively applied to a wide range of common supervised learning settings, regardless of whether the labels are categorical or continuous.

6.3 Analysis and Discussion

Learning Encoders with Flow Loss To support our key claim that learning encoders with flow loss allows our model to avoid crossing trajectories, we compare our model with two variants that do not utilize flow loss for learning encoders. Specifically, we consider: (1) *ANODE+FM* which augments the data and label to same dimensionality by zero-padding and learns dynamics function with flow matching; and (2) *Autoencoder+FM* which employs a two-stage approach by first learning the embedding space using an independent autoencoding objective for both data and labels, and then learning the dynamics function on the fixed embedding space[3]. To quantify the crossing points in trajectories, we train all models with the linear dynamics assumption and measure the proportion of samples where the predicted labels from a one-step Euler solver and an adaptive-step solver disagree[4]. We report the disagreement ratio and training accuracy in Tab. 2.

Table 2: The effectiveness of learning encoders with flow loss. Training accuracy and the proportion of disagreement in prediction between a one-step Euler solver and an adaptive-step solver are shown. Simply augmenting dimensions (ANODE+FM) does not effectively prevent trajectory crossing. Furthermore, learning encoders without flow loss (Autoencoder+FM) also fails to preserve the original coupling due to crossing trajectories.

Training	Disagreement	Accuracy
ANODE + FM	47.30%	30.23%
Autoencoder + FM	11.56%	55.73%
Ours	**0.02%**	**99.80%**

As discussed in Sec. 3, our results indicate that merely augmenting the dimension (ANODE+FM) does not resolve the issue of crossing trajectories induced by the predefined dynamics, resulting in a poor performance even on training data. Although employing more sophisticated encoders (Autoencoder+FM) partially reduces disagreements, it still fails to fit the training data properly. Such crossing trajectories can be eliminated by learning encoders with the flow matching loss, allowing our model to fit successfully to training data with high accuracy.

[3]The samples reconstructed by the pretrained data autoencoder are presented in App. D.1.

[4]We also analyze the relationship between time t and the occurrence of trajectory crossing in App. D.2.

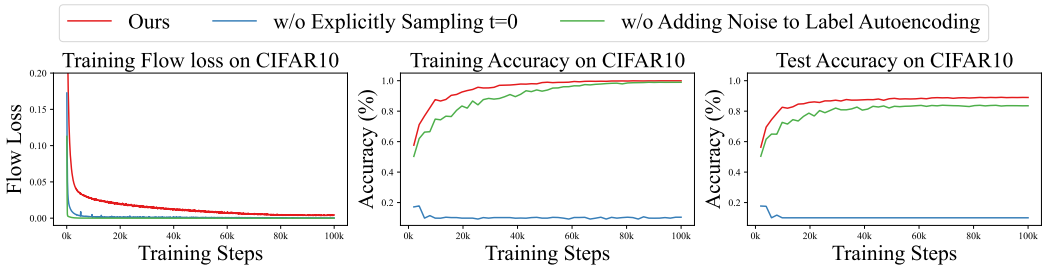


Figure 5: Ablation study of optimization techniques on CIFAR10. Explicitly sampling $t = 0$ in training prevents suboptimal solutions while adding noise to label autoencoding improves generalization.

Role of the Learned Dynamics Function A potential concern is that learning a nonlinear transformation to embed data might lead the data encoder to perfectly predict the label. This issue, noted in previous work [32], warns that learning a complex input transformation could result in a collapse where the dynamics function becomes a simple identity map. To investigate whether this issue occurs in our case, we conduct a 1-NN classification using the learned data encoder on CIFAR10 image classification. The 1-NN classification accuracy with the learned data embedding z_0 is 65.66%, which is significantly lower than the accuracy of 88.47% when we utilize the learned dynamics function to obtain the predicted label embedding $\hat{z}_1$. This result indicates that the learned data embedding is not yet linearly separable enough, and the learned flow can further enhance accuracy. Based on this observation, we conclude that the collapse scenario, where the velocity field becomes an identity map, does not occur in our model. Instead, we find that the learned dynamics function clearly plays a role in processing the data, thereby avoiding the aforementioned pitfall.

Nonlinear Predefined Dynamics Our method embraces not only the linear dynamics but also dynamics having fixed form of $z_t = \alpha_t z_0 + \beta_t z_1$ in general. Here, we investigate the effect of utilizing different dynamics assumptions, including nonlinear ones. To be specific, we choose three different dynamics that is easy to implement:

- Concave: $\alpha_t = \cos(\frac{\pi}{2}t), \beta_t = \sin(\frac{\pi}{2}t)$
- Linear: $\alpha_t = (1-t), \beta_t = t$
- Convex: $\alpha_t = 1 - \sin(\frac{\pi}{2}t), \beta_t = 1 - \cos(\frac{\pi}{2}t)$.

We visualize the difference of dynamics and their effect on final performance in Fig. 4.

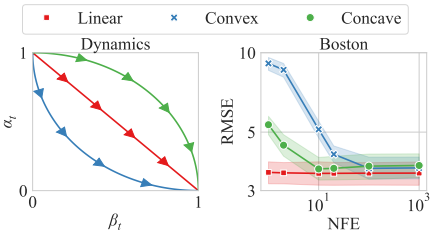


Figure 4: Analysis on predefined dynamics. (Left) Change of coefficients in interpolant with respective to time. (Right) Prediction RMSE over NFE on UCI Boston dataset.

As it shows, the performance of the nonlinear variants (*i.e.*, convex and concave) clearly improves with an increased number of function evaluations. For these variants, the many-step inference performance increases as we invest more NFEs and then later saturates, indicating that the model's approximation of an infinite-depth model becomes sufficiently accurate as discretization error diminishes. In contrast, our model trained on the linear dynamics shows consistently good performance, even with a few function evaluations. This behavior is somewhat expected, as the ideal linear trajectory can be already accurately inferred using a one-step Euler solver. Surprisingly, we also empirically observe that the choice of linear dynamics leads to better performance, compared to more complex choice of dynamics assumption. Thus, similar to the claims of Liu et al.(2022) [29], we believe that linear dynamics should be considered as a default choice unless specific constraints on hidden states are required.

Ablation on Optimization Techniques We conduct an ablation study on CIFAR10 dataset to investigate the effects of optimization techniques introduced in Sec. 4, and report the results in Fig. 5. Similar ablation studies were also conducted on other datasets, as detailed in App. D.3. The variant without explicitly sampling $t = 0$ fails to fit on training data, despite the convergence of the flow loss. Furthermore, the variant without adding noise to the label autoencoding objective succeeds

https://doi.org/10.52202/079017-1924

in fitting the training set, but its test accuracy significantly degrades compared to the version with noise in label autoencoding. Thus, as discussed in Sec. 4, we conclude that explicitly sampling $t = 0$ and introducing noise in label autoencoding effectively regularize both encoders and produces label embedding that is robust to test-time errors.

7 Limitations and Future Work

In this work, we primarily study the problem of adopting flow matching by imposing a fixed and simple dynamics assumption, observing that a fixed, closed-form equation for the intermediate state is already sufficient to bring the advantages of flow matching. Although employing simple dynamics (*e.g.* linear) may seem overly restrictive, this approach can be justified in the context of Koopman operator theory, which aims to find embeddings that globally linearize the dynamics. Since our method of flow matching with linear dynamics shares a high-level concept with Koopman operator theory, exploring the relationship between the two could be a promising direction for future research. We leave further discussions in App. C.

On the other hand, while we studied simple and predefined dynamics, the form of dynamics assumption could be further generalized to be a learnable component. To be specific, introducing a learnable target dynamics that determines per-sample dynamics would be a promising future direction to study. This would be advantageous for handling inputs with varying complexity, by efficiently and adaptively allocating more computation on hard samples.

Furthermore, extending our method to a broader range of paired data applications might be a useful future direction. The reduced computational burden achieved through simulation-free training could offer several benefits of continuous-depth models to diverse applications. For example, applying our method to model compression or knowledge distillation could be particularly promising, leveraging the parameter efficiency of continuous-depth models.

8 Conclusion

In this work, we adopted a flow matching objective to achieve simulation-free training of continuous-depth models for learning deterministic mappings between paired data. We proposed learning an embedding space where flow matching occurs, which we identified as a crucial component for ensuring the validity of the target velocity field. Our proposed method significantly reduces the computational burden of training NODEs while maintaining competitive performance. Additionally, we found that our method, leveraging simple linear dynamics, demonstrates impressive performance on low-NFE regime.

Acknowledgments and Disclosure of Funding

This work was in part supported by the National Research Foundation of Korea (RS-2024-00351212 and RS-2024-00436165), and Institute of Information and communications Technology Planning and Evaluation (IITP) grant (RS-2022-II220926, RS-2024-00509279, RS-2021-II212068, and RS-2019-II190075) funded by the Korean government (MSIT). We thank Youngmin Ryou (KAIST) for his valuable input in formulating the theoretical analysis presented in this work.

References

[1] Michael S. Albergo and Eric Vanden-Eijnden. 2023. Building Normalizing Flows with Stochastic Interpolants. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[2] Tomer Amit, Eliya Nachmani, Tal Shaharabany, and Lior Wolf. 2021. SegDiff: Image Segmentation with Diffusion Probabilistic Models. *CoRR*, abs/2112.00390.

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019. Deep Equilibrium Models. In *NeurIPS*, pages 688–699.

[4] Dmitry Baranchuk, Andrey Voynov, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. 2022. Label-Efficient Semantic Segmentation with Diffusion Models. In *ICLR*.

[5] Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. 2019. GRU-ODE-Bayes: Continuous Modeling of Sporadically-Observed Time Series. In *NeurIPS*.

[6] Ricky T. Q. Chen. 2018. torchdiffeq.

[7] Shoufa Chen, Peize Sun, Yibing Song, and Ping Luo. 2023. DiffusionDet: Diffusion Model for Object Detection. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 19773–19786. IEEE.

[8] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583.

[9] Ting Chen, Lala Li, Saurabh Saxena, Geoffrey E. Hinton, and David J. Fleet. 2023. A Generalist Framework for Panoptic Segmentation of Images and Videos. In *ICCV*.

[10] J.R. Dormand and P.J. Prince. 1980. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26.

[11] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. 2019. Augmented Neural ODEs. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3134–3144.

[12] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman. 2020. How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 3154–3164. PMLR.

[13] Arnab Ghosh, Harkirat Behl, Emilien Dupont, Philip Torr, and Vinay Namboodiri. 2020. STEER : Simple Temporal Regularization For Neural ODE. In *Advances in Neural Information Processing Systems*, volume 33, pages 14831–14843. Curran Associates, Inc.

[14] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. 2019. FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models. In *ICLR*. OpenReview.net.

[15] Eldad Haber and Lars Ruthotto. 2017. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004.

[16] Xizewen Han, Huangjie Zheng, and Mingyuan Zhou. 2022. CARD: Classification and Regression Diffusion Models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778. IEEE Computer Society.

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *NeurIPS*.

 https://doi.org/10.52202/079017-1924

[19] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *ICML*.

[20] Jacob Kelly, Jesse Bettencourt, Matthew J. Johnson, and David Duvenaud. 2020. Learning Differential Equations that are Easy to Solve. In *NeurIPS*.

[21] Markelle Kelly, Rachel Longjohn, and Kolby Nottingham. 2023. The UCI machine learning repository.

[22] Patrick Kidger. 2022. On Neural Differential Equations. *CoRR*, abs/2202.02435.

[23] Patrick Kidger, James Morrill, James Foster, and Terry J. Lyons. 2020. Neural Controlled Differential Equations for Irregular Time Series. In *NeurIPS*.

[24] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR (Poster)*.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images.

[26] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. 1989. Handwritten Digit Recognition with a Back-Propagation Network. In *NIPS*, pages 396–404. Morgan Kaufmann.

[27] Haoying Li, Yifan Yang, Meng Chang, Shiqi Chen, Huajun Feng, Zhihai Xu, Qi Li, and Yueting Chen. 2022. SRDiff: Single image super-resolution with diffusion probabilistic models. *Neurocomputing*, 479:47–59.

[28] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. 2023. Flow Matching for Generative Modeling. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[29] Xingchao Liu, Chengyue Gong, and Qiang Liu. 2023. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[30] Calvin Luo. 2022. Understanding Diffusion Models: A Unified Perspective. *CoRR*, abs/2208.11970.

[31] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. 2018. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950.

[32] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. 2020. Dissecting Neural ODEs. In *NeurIPS*.

[33] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*. Granada, Spain.

[34] Avik Pal, Yingbo Ma, Viral B. Shah, and Christopher Vincent Rackauckas. 2021. Opening the Blackbox: Accelerating Neural Differential Equations by Regularizing Internal Solver Heuristics. In *ICML*.

[35] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-Resolution Image Synthesis with Latent Diffusion Models. In *CVPR*.

[36] Yulia Rubanova, Tian Qi Chen, and David Duvenaud. 2019. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *NeurIPS*.

[37] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter W. Battaglia. 2019. Hamiltonian Graph Networks with ODE Integrators. *CoRR*, abs/1909.12790.

[38] Jiaming Song, Chenlin Meng, and Stefano Ermon. 2021. Denoising Diffusion Implicit Models. In *ICLR*.

[39] Jay Whang, Mauricio Delbracio, Hossein Talebi, Chitwan Saharia, Alexandros G. Dimakis, and Peyman Milanfar. 2022. Deblurring via Stochastic Refinement. In *CVPR*.

[40] Laurent Younes. 2010. *Shapes and Diffeomorphisms*, volume 171. Springer Science & Business Media.

[41] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. 2020. Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control. In *ICLR*.

[42] Juntang Zhuang, Nicha C. Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James S. Duncan. 2020. Adaptive Checkpoint Adjoint Method for Gradient Estimation in Neural ODE. In *ICML*.

[43] Juntang Zhuang, Nicha C. Dvornek, Sekhar Tatikonda, and James S. Duncan. 2021. MALI: A memory efficient and reverse accurate integrator for Neural ODEs. In *ICLR*. OpenReview.net.

 https://doi.org/10.52202/079017-1924

Appendix

A Proofs

In this section, we provide full proofs for the propositions in Sec. 4. We first start with formal definition of target trajectory crossing. Assume that we have a data encoder f_ϕ and label encoder g_φ that transform data $x \in \mathbb{R}^{d_x}$ and labels $y \in \mathbb{R}^{d_y}$ to latent $z_0, z_1 \in \mathbb{R}^d$, where $z_0 = f_\phi(x)$ and $z_1 = g_\varphi(y)$, respectively. We also choose a predefined dynamics $F(z_0, z_1, t) = \alpha_t z_0 + \beta_t z_1 = z_t$. Under the assumption of $d > d_x, d_y$[5] and both α_t, β_t being smooth and nonzero except for $t = 0$ and $t = 1$, we define the target trajectory crossing as follows:

Definition 1 (Target Trajectory Crossing)**.** *The encoders (f_ϕ, g_φ) are said to induce a target trajectory crossing if there exists a tuple (t, x, y, x', y') such that $\alpha_t f_\phi(x) + \beta_t g_\varphi(y) = \alpha_t f_\phi(x') + \beta_t g_\varphi(y')$ for $x \neq x'$ and $y \neq y'$.*

We now provide the proofs for each proposition in the following subsections.

A.1 Proof of Proposition 1 (Section 4)

Proposition 1. *There exist (f_ϕ, g_φ) that induces non-crossing target trajectory for any data pair in $\mathcal{D}$ while minimizing $\mathcal{L}_{label\_ae}$.*

Proof. Let the latent space constructed by a set of basis $\mathbb{I} = \{e_1, e_2, ..., e_d\}$. Since $d > d_y$, we can find a label encoder g_φ such that utilizes k basis $\mathbb{J} = \{e_1, e_2, ..., e_k\}$ ($d > k \geq d_y$) and minimizes the autoencoding loss (*i.e.*, $g_\varphi(y) = g_\varphi(y')$ if and only if $y = y'$). Also, we can find a data encoder f_ϕ such that $\text{proj}_{\text{span}(\mathbb{K})} f_\phi(x) = \text{proj}_{\text{span}(\mathbb{K})} f_\phi(x')$ if and only if $x = x'$, where $\mathbb{K} = \{e_{k+1}, ..., e_d\}$.

Then, suppose that there exists a tuple (t, x, y, x', y') such that $\alpha_t f_\phi(x) + \beta_t g_\varphi(y) = \alpha_t f_\phi(x') + \beta_t g_\varphi(y')$, *i.e.*, $\alpha_t(f_\phi(x) - f_\phi(x')) + \beta_t(g_\varphi(y) - g_\varphi(y')) = 0$.

Since $g_\varphi(y) - g_\varphi(y') = 0$ if and only if $y = y'$ and $\text{proj}_{\text{span}(\mathbb{K})}(f_\phi(x) - f_\phi(x')) = 0$ if and only if $x = x'$ by construction, such a tuple does not exist. Therefore, there exists f_ϕ, g_φ such that does not induce target trajectory crossing, while minimizing the autoencoding loss. □

A.2 Proof of Proposition 2 (Section 4)

Proposition 2. *If g_φ is injective, the following equivalence holds: $(f_\phi, g_\varphi, h_\theta)$ minimizes $\mathcal{L}_{flow}$ to zero for all $t \in [0, 1)$ if and only if (f_ϕ, g_φ) induces non-crossing target trajectory and h_θ perfectly fits the induced target velocity, for any data pair in $\mathcal{D}$.*

Proof. ($\Longleftarrow$) If (f_ϕ, g_φ) induces non-crossing target trajectory for any data pair in $\mathcal{D}$, there is a well-defined target velocity $\frac{d}{dt} z_t$ at every z_t which is continuous on t. If h_θ perfectly fits this target velocity for all (z_t, t), the flow loss is zero.

($\Longrightarrow$) We prove by contradiction. Suppose the flow loss is zero but there is a crossing trajectory, *i.e.*, there exists a tuple (t, x, y, x', y') that $z_t = z'_t$ for $x \neq x'$ and $y \neq y'$. Since the loss is zero for all $t \in [0, 1)$, the dynamics function h_θ must output $\frac{d}{dt} F(z_0, z_1, t)$ at z_t, and $\frac{d}{dt} F(z'_0, z'_1, t)$ at z'_t. This is a contradiction since at the point of crossing we have $z_t = z'_t$ but $\frac{d}{dt} F(z_0, z_1, t) \neq \frac{d}{dt} F(z'_0, z'_1, t)$. □

B Experiment Details

This section describes the implementation detail in our experiments (Sec. 6).

[5] As d_x and d_y are dimensions in observation space, we also conjecture that the latent dimension d can be made smaller if the data lives on a low-dimensional manifold.

B.1 Experiment Details for Classification Tasks

Network Architecture (MNIST) We employ an MLP-based architecture for the MNIST dataset. The data encoder maps each image to a 784-dimensional embedding, using a three-layer MLP with a hidden dimension of 784 and BatchNorm [19]. The dynamics function consists of a three-layer MLP with 2048 hidden dimension. Following NODE [8], we concatenate the time variable to the input of each layer. Consistent with common practices for NODEs [22], normalization layers are not included in the dynamics function. For the label autoencoder, class labels are converted into one-hot vectors and encoded with a single linear layer, while the label decoder utilizes a two-layer MLP with BatchNorm.

Network Architecture (SVHN, CIFAR10) We adopt a CNN-based architecture for both the SVHN and CIFAR10 datasets. The data encoder utilizes 7 convolutional layers with a hidden dimension of 64. By default, we employ 3x3 convolution kernels, while the final two layers utilize 4x4 kernels with a stride of 2 for downsampling, resulting in the data being encoded into states with dimensions of 7x7x64. The dynamics function consists of 6 convolutional layers with 3x3 kernels and a hidden dimension of 256. At each layer, the time variable is concatenated to the input. We utilize a single linear layer for label encoding, reshaping the output to match the size of embedding. In the label decoding process, we average the feature map over the spatial dimension and apply a single linear layer.

Training We train all models for 100,000 iterations using the Adam optimizer [24] with a cosine learning rate scheduler. For all classification experiments, we utilize a batch size of 1024. Additionally, we set a maximum training time of 48 hours for the feasibility of experiments. By default, we set the learning rate to 1e-3 for MNIST and 3e-4 for CIFAR10 and SVHN. For our method, we set the ratio of explicitly sampling $t = 0$ to 10% for all datasets. Regarding the noise introduced to the label autoencoder, we set the standard deviation σ to 3 for MNIST, 7 for SVHN, and 10 for CIFAR10. In cases where training of NODE baselines fails, we adjust the learning rate accordingly. The failure cases of NODE baselines are illustrated in Fig. 6.

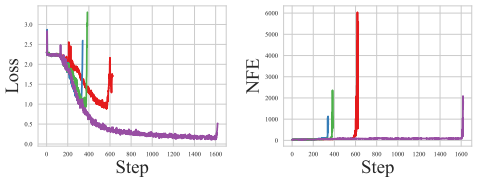


Figure 6: Failure cases of NODEs.

B.2 Experiment Details for Regression Tasks

Network Architecture We utilize a 2-layer MLP with a hidden dimension of 64 for the data encoder, and a 7-layer MLP with the same hidden dimension for the dynamics function. In the dynamics function, we concatenate the time variable to the input of each layer. On the label side, we employ a single linear layer for encoding and another single linear layer for decoding.

Training We trained all models for 100,000 iterations using the Adam optimizer with a constant learning rate of 3e-3. Additionally, we split the training set into a train-validation split with a ratio of 6:4, and utilized the validation metric for early-stopping. Early stopping was implemented by measuring the validation metric every 1,000 iterations and setting the patience level to 10. For our model, we sample the noise added to the label autoencoding from $\mathcal{N}(0, 3^2)$ and the proportion of explicitly sampling $t = 0$ as 10%.

B.3 Experiment Details about Baselines

NODE [8] Following the tolerance values used in NODE [8] and ANODE [11], we used dopri5 [10] solver with the absolute and relative tolerance of 1e-3 for both training and inference.

STEER [13] STEER introduces a new hyperparameter b that controls the integration interval. Instead of integrating the dynamics function from 0 to 1, STEER integrates from 0 to $t \sim \text{Uniform}(1-b, 1+b)$ during training. We follow the default configuration of using $b = 0.99$ for MNIST classification. For other tasks, we set b to 0.1 since higher values resulted in training failures.

https://doi.org/10.52202/079017-1924

RNODE [12] RNODE introduces two hyperparameters used for the coefficients of regularization terms. Specifically, it regularizes the Jacobian norm and the kinetic energy of the dynamics function to encourage the model to learn straight and constant-speed dynamics. The coefficient of 0.01 was generally used throughout the experiments in the original paper. Therefore, we set both coefficients for the Jacobian norm and kinetic energy as 0.01 for our experiments.

CARD [16] CARD utilizes discrete timesteps and a hyperparameter β_t to schedule the noise level for diffusion modeling. Following the paper's approach, we use 1000 discrete timesteps and set a linear noise schedule from β_1 =1e-4 to $\beta_{1000} = 0.02$.

B.4 Computation Resources

We conducted experiments on our internal cluster with two types of machine. We list their specifications below.

1. Intel Xeon Gold 6330 CPU and NVIDIA RTX A6000 GPU (with 48GB VRAM)
2. Intel Xeon Gold 6230 CPU and NVIDIA RTX 3090 GPU (with 24GB VRAM)

We utilized the first machine for image classification experiments, and latter one for regression experiments. We expect training our model will take about 90 min., 270 min, 230 min. for classification experiments on MNIST, SVHN and CIFAR10, respectively. For regression tasks, we estimate training cost, summing up for all splits and datasets, would cost about 288 GPU hours.

C Relation to Koopman Autoencoder

Our model with linear dynamics shares the high-level motivation with Lusch et al. (2018) [31], which aims to find an embedding space that yields linear dynamics between source and target. Regardless of the theoretical background, both flow matching and Koopman operator theory are promising approaches that seek to interpret a nonlinear system within a well-studied linear framework.

At the same time, we identify several differences between our work and the line of research based on Koopman operator theory. While those works mainly focus on a systematic way to obtain a linearized representation of the underlying nonlinear dynamics (with eigenfunctions), our work aims to find a way to learn it in a simulation-free manner, avoiding the heavy computation of forward simulation (*e.g.*, which appears in $\mathcal{L}_{lin}$ of Lusch et al. (2018) [31]) from an initial state to an end state. Additionally, compared to the discrete depth neural networks that have a single linear layer processor, our proposed method is generally applicable to any nonlinear dynamics that connects two endpoints z_0 and z_1, exemplified as *convex* or *concave* as discussed in Sec. 6.3. This implies that in our case, it is possible to have a latent trajectory as a curve in non-Euclidean geometry whenever the interpolated state z_t is tractable.

D Additional Results

In this section, we extend the discussions in Sec. 6 with additional results.

D.1 Additional Result of Learning Encoders with Flow Loss

We present the reconstructed images from the trained data autoencoder in Fig. 7, which are used to analyze the effectiveness of the flow loss as reported in Tab. 2. While the pretrained autoencoder reconstructs the images holistically with minimal information loss (Tab. 2), the embedding space learned without the flow loss fails to maintain the coupling with the predefined dynamics.

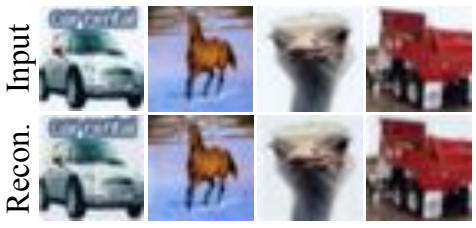


Figure 7: Reconstruction from the autoencoder.

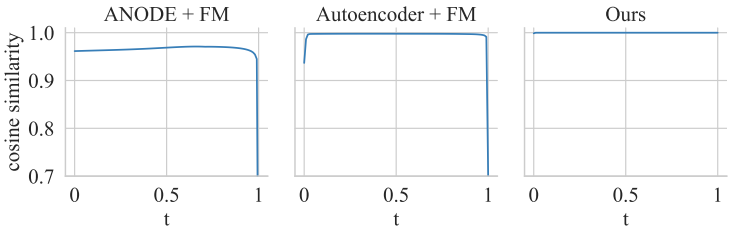


Figure 8: Cosine similarity between target velocity and predicted velocity over time t. Similar to the flow loss, we measure the cosine similarity between target velocity and predicted velocity. Low cosine similarity near $t = 0$ and $t = 1$ indicates the occurrence of target trajectory intersection near the endpoints.

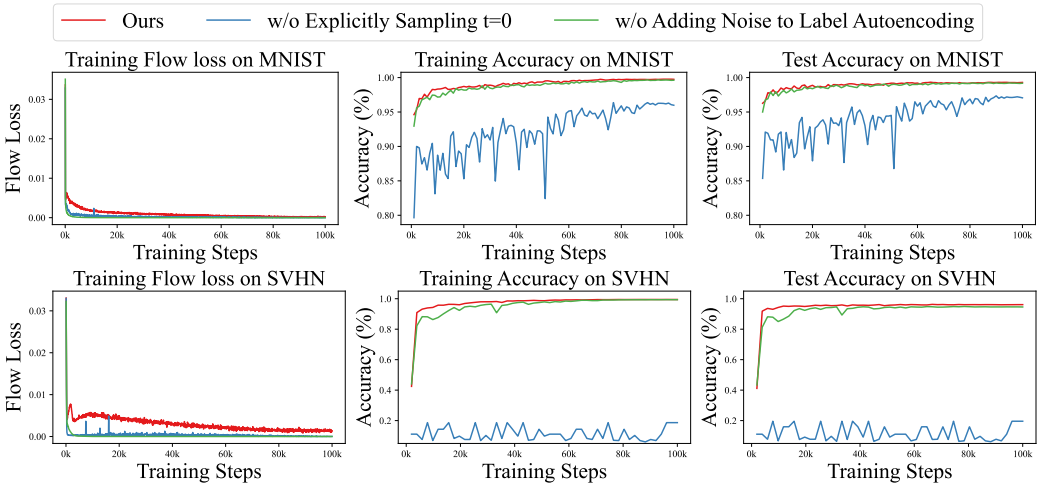


Figure 9: Ablation study of optimization techniques on MNIST and SVHN datasets, corresponding to Fig. 5. Explicitly sampling $t = 0$ prevents suboptimal solutions, while adding noise to label autoencoding improves generalization performance.

D.2 Location of Crossing Points

In Fig. 8, we extend the analysis in Sec. 6.3 to identify where target trajectory intersections occur. Specifically, we measure the cosine similarity between target velocity and predicted velocity across t. By measuring cosine similarity instead of MSE, we can ignore the effect of absolute scale in the embedding space and accurately compare ANODE+FM, Autoencoder+FM, and our method. Model variants that do not learn encoders with flow loss particularly suffer from trajectory crossing near both endpoints, showing low prediction accuracy for the direction of target velocity. In contrast, our proposed method shows consistently high cosine similarity, mitigating the issue of target trajectory intersections near these regions. Since both endpoints are constructed from encoders, the result also supports our claim that the encoders should be trained with flow loss to effectively penalize such intersections.

D.3 Additional Results of Ablation on Optimization Techniques

We provide additional results on the optimization techniques in Fig. 9. Consistent with the findings on CIFAR10 (Fig. 5), the model without explicitly sampling at $t = 0$ converges to suboptimal solutions, while introducing noise during label autoencoding improves test accuracy.

D.4 Additional Experiment Result for UCI Regression Tasks

The full results on UCI regression tasks with various solvers are shown in from Tab. 3 to 9. As discussed in the main text, our method with linear dynamics assumption shows superior performance compared to baselines in low-NFE (1-2 steps) regime, which aligns with our observations in image classification experiments.

https://doi.org/10.52202/079017-1924

Table 3: Experiment results on UCI regression tasks with Euler 1-step solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.52±0.79	5.95±5.10	552.10±1638.54	3.50±0.84	130.23±38.91
Concrete	5.59±0.58	2922.76±12615.38	1056.15±2607.27	5.98±1.66	198.91±72.58
Energy	0.58±0.11	253.57±707.63	74.06±164.27	0.83±0.41	63.87±12.81
Kin8nm	7.61±0.02	39.04±3.31	1694.59±700.19	23.69±0.82	238.18±6.44
Naval	0.08±0.00	8447.16±2968.35	3011.13±907.95	0.16±0.01	15.42±0.59
Power	4.03±0.19	$> 10^4$	7.37±14.16	4.06±0.35	179.91±65.08
Protein	4.26±0.08	103.14±141.55	102.17±97.95	7.70±1.88	92.82±23.84
Wine	0.67±0.05	1.99±4.31	19.08±36.97	0.67±0.05	10.42±2.87
Yacht	0.92±0.42	62.19±108.46	29.06±41.13	2.71±1.94	111.21±62.53
Year	9.15±NA	1366.06±NA	2452.93±NA	22.37±NA	157.11±NA

Table 4: Experiment results on UCI regression tasks with Euler 2-step solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.50±0.78	3.81±2.04	278.23±827.35	3.41±0.81	20.86±12.26
Concrete	5.58±0.57	1560.21±6301.49	1437.21±5229.38	5.59±0.80	28.49±17.08
Energy	0.57±0.11	114.46±310.32	36.50±82.19	0.61±0.11	8.15±8.57
Kin8nm	7.59±0.02	14.62±1.23	813.00±338.36	9.77±0.17	18.87±0.78
Naval	0.07±0.00	$> 10^4$	$> 10^4$	0.08±0.00	1.81±0.22
Power	4.03±0.19	$> 10^4$	5.46±6.52	3.86±0.23	19.78±10.59
Protein	4.26±0.08	30.85±24.65	77.94±77.06	4.40±0.35	14.17±6.97
Wine	0.67±0.05	3.50±10.84	24.95±46.23	0.66±0.04	1.49±0.64
Yacht	0.93±0.43	34.66±62.06	57.13±131.04	1.79±1.01	11.83±19.40
Year	9.17±NA	6053.32±NA	542.82±NA	9.67±NA	42.18±NA

Table 5: Experiment results on UCI regression tasks with Euler 10-step solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.49±0.79	3.20±0.82	56.54±164.64	3.39±0.80	3.32±0.89
Concrete	5.57±0.54	294.54±1260.67	88.00±190.75	5.59±0.77	5.47±0.67
Energy	0.58±0.12	21.24±57.69	6.13±15.60	0.52±0.06	0.58±0.09
Kin8nm	7.59±0.02	7.46±0.02	159.73±64.92	7.43±0.02	7.71±0.02
Naval	0.07±0.00	$> 10^4$	$> 10^4$	0.03±0.00	0.09±0.00
Power	4.03±0.18	2144.77±9543.08	4.10±0.71	3.82±0.21	4.57±0.19
Protein	4.26±0.08	7.38±6.01	7.92±7.56	3.87±0.05	5.66±0.12
Wine	0.68±0.05	0.82±0.49	$> 10^4$	0.66±0.04	0.75±0.05
Yacht	0.96±0.45	2.58±6.37	1.86±1.93	1.21±0.60	0.95±0.34
Year	9.25±NA	139.85±NA	69.53±NA	8.87±NA	10.85±NA

Table 6: Experiment results on UCI regression tasks with Euler 20-step solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.49±0.79	3.20±0.82	29.29±80.54	3.39±0.80	3.32±0.89
Concrete	5.58±0.55	148.43±629.86	42.13±85.51	5.59±0.77	5.43±0.66
Energy	0.58±0.13	10.43±28.28	2.63±7.20	0.52±0.06	0.53±0.09
Kin8nm	7.62±0.02	7.39±0.02	81.51±31.91	7.38±0.02	7.95±0.03
Naval	0.07±0.00	$> 10^4$	$> 10^4$	0.03±0.00	0.09±0.00
Power	4.04±0.18	1073.63±4769.25	3.97±0.19	3.82±0.21	4.81±0.18
Protein	4.27±0.08	5.55±2.65	5.33±2.38	3.86±0.04	5.90±0.07
Wine	0.68±0.05	0.71±0.17	1.10±0.72	0.66±0.04	0.75±0.06
Yacht	0.96±0.46	1.29±0.90	1.35±0.67	1.17±0.60	0.89±0.36
Year	9.30±NA	40.97±NA	33.25±NA	8.86±NA	11.40±NA

Table 7: Experiment results on UCI regression tasks with Euler 100-step solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.49±0.80	3.20±0.82	7.58±13.13	3.40±0.80	3.44±0.97
Concrete	5.63±0.55	33.72±124.75	10.81±13.46	5.60±0.77	5.59±0.61
Energy	0.59±0.14	2.12±4.92	0.64±0.52	0.52±0.06	0.55±0.10
Kin8nm	7.77±0.02	7.37±0.02	20.30±5.69	7.36±0.03	8.38±0.02
Naval	0.07±0.00	2095.27±667.36	116.16±43.60	0.03±0.00	0.12±0.00
Power	4.08±0.16	216.78±950.18	3.95±0.18	3.82±0.21	5.03±0.18
Protein	4.30±0.09	4.38±0.29	4.20±0.06	3.86±0.04	6.13±0.09
Wine	0.68±0.05	0.66±0.04	0.71±0.07	0.66±0.04	0.78±0.06
Yacht	0.96±0.46	1.12±0.54	1.23±0.48	1.16±0.60	0.98±0.38
Year	9.35±NA	8.94±NA	9.41±NA	8.86±NA	11.87±NA

Table 8: Experiment results on UCI regression tasks with Euler 1000-step solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.50±0.80	3.20±0.82	3.44±0.69	3.40±0.80	3.34±0.90
Concrete	5.66±0.55	8.29±11.12	5.70±0.58	5.60±0.77	5.56±0.52
Energy	0.59±0.14	0.52±0.08	0.53±0.07	0.52±0.06	0.58±0.07
Kin8nm	7.87±0.02	7.36±0.02	7.69±0.06	7.36±0.03	8.53±0.03
Naval	0.07±0.00	19.23±5.63	10.21±3.75	0.03±0.00	0.13±0.00
Power	4.14±0.14	24.24±90.84	3.95±0.18	3.82±0.21	5.15±0.12
Protein	4.36±0.09	4.33±0.23	4.20±0.06	3.86±0.04	6.17±0.11
Wine	0.69±0.05	0.66±0.04	0.70±0.05	0.66±0.04	0.76±0.06
Yacht	0.96±0.46	1.12±0.54	1.23±0.48	1.16±0.60	0.95±0.34
Year	9.39±NA	8.95±NA	8.95±NA	8.86±NA	12.05±NA

Table 9: Experiment results on UCI regression tasks with dopri5 solver.

Dataset	Ours	NODE	STEER	RNODE	CARD
Boston	3.50±0.80	3.20±0.81	3.44±0.69	3.40±0.80	3.34±0.90
Concrete	5.67±0.55	5.76±0.69	5.70±0.57	5.60±0.77	5.56±0.52
Energy	0.60±0.14	0.52±0.08	0.53±0.07	0.51±0.06	0.58±0.07
Kin8nm	7.89±0.02	7.35±0.02	7.56±0.02	7.37±0.03	8.53±0.03
Naval	0.07±0.00	0.05±0.00	0.10±0.00	0.03±0.00	0.13±0.00
Power	4.17±0.14	3.91±0.19	3.95±0.17	3.83±0.21	5.15±0.12
Protein	4.37±0.11	4.33±0.23	4.20±0.06	3.86±0.04	6.17±0.11
Wine	0.69±0.05	0.66±0.04	0.70±0.05	0.66±0.04	0.76±0.06
Yacht	0.96±0.46	1.12±0.54	1.23±0.48	1.16±0.60	0.95±0.34
Year	9.41±NA	8.95±NA	8.95±NA	8.88±NA	12.05±NA

https://doi.org/10.52202/079017-1924

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of our work in Sec. 7.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provide the full set of assumptions and proof for theoretical analysis.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide all the information needed to reproduce our work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

https://doi.org/10.52202/079017-1924

Answer: [Yes]

Justification: We release the code with instructions at `https://github.com/seminkim/simulation-free-node`.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We specified all the experiment details either in the main text or in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We reported either 95% CI or standard deviation for UCI regression tasks. However, we were not able to repeat image classification experiment due to the limited computational resources.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provided sufficient information on the computer resources in the appendix.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: We observed no particular societal impact to address.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

https://doi.org/10.52202/079017-1924

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not have experiments with dataset having such risk.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We properly cited all the datasets and assets that we used.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provided the code with documentation in our submission.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

https://doi.org/10.52202/079017-1924