On the Dependency of Neural Differential Equations on the Training Numerical Method: Effects on Optimisation and Predictive Performance

C. Coelho(✉) and O. Niggemann

Institute for Artificial Intelligence, Helmut Schmidt University, Hamburg 22043, Germany
cecilia.coelho@hsu-hh.de

Abstract. Neural Ordinary Differential Equations (Neural ODEs) model data as the solution of a parametrised ordinary differential equation, evaluated numerically at observation times. In practice, both training and prediction require the choice of a numerical method and its hyperparameters. While classical numerical analysis guarantees that all consistent methods converge to the same continuous solution as the step size tends to zero, Neural ODEs are trained and evaluated at finite resolution, where this choice becomes an integral part of the model itself. In this work, we theoretically show that the loss function optimised during training is a surrogate loss determined by the chosen numerical method and its configuration. Under standard regularity assumptions on the dynamics and the loss, we derive explicit bounds on how the order and step size of the integrator perturb the loss surface, its gradient, and the location of the resulting minimisers. These bounds quantify the discrepancy between training and testing losses when different solvers are used, and reveal that method-induced errors behave as a systematic model misspecification term. We validate these theory by providing numerical experiments on a popular spiral ODE benchmark, where we observe large train-test discrepancies and qualitatively different optimisation paths across solvers, even when training errors are nearly identical. Furthermore, the theoretical analysis developed in this work applies to any neural network architecture whose output is defined via a numerical solution of a differential equation, including Neural ODEs, continuous normalizing flows, and related models. This work calls for the attention to, and explicit reporting of, the numerical solvers and their configurations in the Neural ODE literature and in applications that rely on these models.

Keywords: Neural Networks · Neural ode · Continuous-normalizing flows · Differential Equations · Numerical Methods · Numerical Analysis · Optimisation

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16763, pp. 255–271, 2027.
https://doi.org/10.1007/978-3-032-30530-5_16

1 Introduction

Neural Ordinary Differential Equations (Neural ODEs) are an increasingly popular neural network architecture that models ordinary differential equations (ODEs) to data. This allows to have continuous-time representations of datasets, previously impossible with classical neural networks [3]. Neural ODEs have been applied to various domains, including physics [13,21], chemistry [12,16], engineering [4,20], economics [11,14], and biology [17,18].

Furthermore, given the advantages and seamless modelling of systems that are classically modelled using differential equations, several approaches have branched out by improving on the basic ODEs to more complex differential equations, such as Neural Stochastic Differential Equations [10], Neural Delay Differential Equations [22], Neural Fractional Differential Equations [7], and Neural Chronos ODE [5]. Furthermore, Neural ODEs have been used for generative models, such as Continuous Normalizing Flows [3], and Latent ODEs [19].

In all of these approaches, the continuous dynamics are never accessed directly, both training and prediction rely on a numerical method tailored to the specific differential equation type, together with its associated hyperparameters such as step size or error tolerances. Classical numerical analysis guarantees that, for a fixed vector field and under suitable conditions, all consistent methods converge to the same continuous-time solution as the step size tends to zero [8]. In theory, this suggests that the choice of integrator should be largely irrelevant. However, in practice, Neural ODEs are trained and evaluated at finite resolution, with a fixed solver and a finite step size, so that the numerical method becomes an integral part of the model.

In the literature, several applications of Neural ODEs (and related architectures) have been proposed. However, to the best of our knowledge, none explicitly emphasises that the numerical method used during training should be the same as the one used for prediction. This omission can become problematic since domain experts, such as biologists or medical researchers, may not be aware of the subtle numerical dependence, which can lead to seemingly failed implementations and, in turn, distrust in Neural ODEs and related machine learning methods. More broadly, there is currently a gap between the machine learning viewpoint, where discretisation is treated as a minor implementation detail, and the actual practice, where the discretisation is part of the learned model.

In this paper, we build on the preliminary study of [6], which first highlighted empirically that Neural ODE parameters depend on the training numerical method. However, [6] is limited in several respects: the theoretical argument was informal and tied to a specific absolute-error loss; the impact of the solver was only discussed at the level of prediction error, without analysing gradients or minimisers; and the empirical evidence was restricted to a simple regression task evaluating predictive performance only.

To address the aforementioned gap this paper makes the following contributions:

- *Theoretical characterisation of solver-induced loss bias.* Under standard regularity assumptions, we show that the loss function optimised during training is the loss of a surrogate determined by the chosen numerical method and its configuration. We derive bounds of order $\mathcal{O}(h^p)$ that relate the order p and step size h of the integrator to the discrepancy between the *ideal* loss and the method-dependent *effective* loss;
- *Train-test discrepancy for different numerical methods.* We quantify how much the loss changes when the numerical method used at prediction time differs from the one used during training. The derived bounds show explicitly how the orders and step sizes of two different methods control the train-test discrepancy, and show that solver choice acts as a systematic model misspecification term when changed at test time;
- *Gradient-level analysis and parameter displacement.* We prove that the gradient of the *effective* loss obtained by differentiating through the numerical solver is a biased approximation of the *ideal* gradient, with a bias that scales in $\mathcal{O}(h^p)$. Under a local strong convexity assumption, these bounds show that the minimisers associated with different numerical methods can be separated in parameter space by a distance controlled by their respective global errors;
- *Empirical validation of the proposed statement.* Through numerical experiments we prove our statement by varying the training and testing solvers, as well as by performing a fixed-seed study that isolates the effect of the solver on the optimisation path.

As our analysis and experiments will show, the numerical method and its configuration are not mere simple implementation details but part of the model itself. This calls for greater attention to, and explicit reporting of, the chosen solvers and their hyperparameters in the Neural ODE literature and in applications that rely on these models and related neural differential equation architectures.

Note that, although we herein focus on Neural ODEs, the theoretical analysis applies to any architecture whose outputs are obtained by numerically solving an underlying dynamical system, including continuous normalizing flows and related neural differential equation models.

The rest of this paper is organised as follows. Section 2 briefly introduces the necessary concepts on Neural ODEs, and numerical methods. Section 3 develops our theoretical analysis of how the choice of numerical method affects the effective loss, gradients, and learnt parameters of Neural ODE models. Section 4 presents numerical experiments on a controlled benchmark that illustrate and validate the theory. Finally, the paper ends with the conclusions in Sect. 5.

2 Background

2.1 Neural Ordinary Differential Equations

A Neural ODE is a neural network architecture that fits a network $\boldsymbol{f_\theta}$, with learnable parameters $\boldsymbol{\theta} \in \Theta$, to the right-hand side of an ODE, or system of

ODEs. The end goal is that solving this ODE as an initial value problem (IVP) fits the data:

$$\frac{d\boldsymbol{y}}{dt}(t) = \boldsymbol{f}_{\boldsymbol{\theta}}(t, \boldsymbol{y}(t)) \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0, \tag{1}$$

where $t \in [t_0, t_M]$ and $\boldsymbol{y}(t) \in \mathbb{R}^d$ [3].

To obtain the predictions $\{\hat{\boldsymbol{y}}_m\}_{m=1}^M$, (1) needs to be solved numerically using a classical ODE solver:

$$\{\hat{\boldsymbol{y}}_m\}_{m=1}^M = \text{ODESolve}(\boldsymbol{f}_{\boldsymbol{\theta}}, \boldsymbol{y}_0, (t_0, t_M)).$$

These predictions are then used in a loss functional to optimise $\boldsymbol{\theta}$ by computing the error of the fit to the data [3].

2.2 Numerical Methods

Consider the classical ODE IVP:

$$\frac{d\boldsymbol{y}}{dt}(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)) \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0,$$

on $[t_0, t_M]$ and a discretisation grid $t_0 < t_1 < \cdots < t_M$, $t_{m+1} - t_m = h$, with constant step size $h > 0$, for simplicity. A (one-step) numerical method $\mathcal{A}$ computes an approximation $\boldsymbol{y}_m^{\mathcal{A}} \approx \boldsymbol{y}(t_m)$ by applying an update rule starting from $\boldsymbol{y}_0$. Different update rules lead to different solvers, with the most well known being: *Explicit Euler* (order $p = 1$) $\boldsymbol{y}_{m+1} = \boldsymbol{y}_m + h\boldsymbol{f}(t_m, \boldsymbol{y}_m)$; *explicit midpoint / Heun* (order $p = 2$), which uses an additional intermediate evaluation of $\boldsymbol{f}$ to improve accuracy; *runge-kutta* (*rk4*, order $p = 4$), which combines four stages per step; *dormand-prince* (*dopri5*), an adaptive runge-kutta method that adjusts h based on an embedded error estimate [1].

In the context of Neural ODEs, these numerical methods are used both in the forward pass to compute the predictions, and in the backward pass to compute gradients via backpropagation or the adjoint method [3].

3 Dependence on the Numerical Method

We now formalise how the numerical method enters the effective loss optimised by a Neural ODE and how changing the method between training and prediction perturbs this loss, thus affecting the resulting model.

For simplicity, consider a Neural ODE with a single trajectory and let $\{(t_m, \boldsymbol{y}_m)\}_{m=0}^M$ be observed data points, with $\boldsymbol{y}_0$ the initial condition at time t_0. For each parameter vector $\boldsymbol{\theta}$, the continuous dynamics are given by (1) with flow $\varphi_{\boldsymbol{\theta}}$.

We introduce the following assumptions:

A1) *Regularity of the vector field.*
each $\boldsymbol{\theta} \in \Theta$, the function $\boldsymbol{f_\theta}$ is globally Lipschitz in $\boldsymbol{y}$ and continuous in t on $[t_0, t_M] \times \mathbb{R}^d$, with Lipschitz constant $L_f(\boldsymbol{\theta})$. Then the IVP (1) has a unique solution on $[t_0, t_M]$, and the flow $\varphi_{\boldsymbol{\theta}}$ is well defined [8].
A2) *Numerical method, order, and global error.*
$\mathcal{A}$ denote a fixed-step numerical method of order $p \in \mathbb{N}$ with step size $h > 0$, used to integrate (1) from t_0 to t_m. We denote by $\Phi^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)$ the resulting numerical approximation to $\varphi_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)$. Following classical numerical analysis for stable one-step methods, we assume that there exists a constant $C(\boldsymbol{\theta})$, independent of h and m, such that the global error satisfies:

$$\left\|\Phi^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0) - \varphi_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)\right\| \leq C(\boldsymbol{\theta})h^p, \qquad \forall m = 1, \ldots, M, \tag{2}$$

where the order p appears explicitly as the exponent governing how fast the numerical solution converges to the true flow as $h \to 0$ [9].
A3) *Lipschitz loss on bounded trajectories.*
the discrepancy between predictions and observations to be measured via a pointwise loss $\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$. Then, assuming there exists $R > 0$ such that all true and predicted states satisfy $\|\boldsymbol{y}_m\| \leq R$ and $\|\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A})\| \leq R$ for all relevant $\boldsymbol{\theta}$ and m. Then there exists $L_\rho(R) > 0$ such that, for all $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{y} \in \mathbb{R}^d$ with $\|\boldsymbol{u}\|, \|\boldsymbol{v}\|, \|\boldsymbol{y}\| \leq R$,

$$|\rho(\boldsymbol{u}, \boldsymbol{y}) - \rho(\boldsymbol{v}, \boldsymbol{y})| \leq L_\rho(R)\|\boldsymbol{u} - \boldsymbol{v}\|.$$

This condition is mild and standard in error-propagation arguments, *i.e.* if $\rho(\cdot, \boldsymbol{y})$ is continuously differentiable then one can take $L_\rho(R) = sup_{\|\boldsymbol{u}\|, \|\boldsymbol{y}\|} \leq R\|\lambda_{\boldsymbol{u}}\rho(\boldsymbol{u}, \boldsymbol{y})\|$.
Note that, the boundeness requirement on trajectories can be ensured in practise by restricting the data range using standard regularisation or applying mild growth controls on $\boldsymbol{f_\theta}$ over finite time intervals.

Given $\boldsymbol{\theta}$, consider the *ideal* loss function:

$$\mathcal{L}^*(\boldsymbol{\theta}) = \frac{1}{M}\sum_{m=1}^{M} \rho\big(\hat{\boldsymbol{y}}^*_m(\boldsymbol{\theta}), \boldsymbol{y}_m\big), \qquad \hat{\boldsymbol{y}}^*_m(\boldsymbol{\theta}) = \varphi_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0).$$

In practice, the numerical method $\mathcal{A}$ is used to compute predictions, $\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}) = \Phi^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)$, so the *effective* training loss is instead given by:

$$\mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{m=1}^{M} \rho\big(\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}), \boldsymbol{y}_m\big). \tag{3}$$

Thus, given the dependence of $\hat{\boldsymbol{y}}$ on $\mathcal{A}$, the numerical method defines the actual objective surface seen by the optimiser.

Proposition 1 (Loss perturbation induced by a numerical method). *Suppose Assumptions A1, A2, and A3 hold. Then for every* $\boldsymbol{\theta} \in \Theta$,

$$\left|\mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta}) - \mathcal{L}^{*}(\boldsymbol{\theta})\right| \leq L_{\rho}(R)C(\boldsymbol{\theta})h^{p}. \tag{4}$$

Note that, by A2 the constant $C(\boldsymbol{\theta})$ *is uniform in the time index* m *(and independent of* h*), so the bound* $\left\|\hat{\boldsymbol{y}}_m(\boldsymbol{\theta};\mathcal{A}) - \hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta})\right\| \leq C(\boldsymbol{\theta})h^{p}$ *holds for every summand. Thus, averaging over* m *leaves the right-hand side unchanged.*

Proof. Consider a fix set of network parameters $\boldsymbol{\theta}_1 \in \Theta$. Then, by Assumption A2, for all $m \in 1, \ldots, M$,

$$\|\hat{\boldsymbol{y}}_m(\boldsymbol{\theta};\mathcal{A}) - \hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta})\| = \left\|\Phi^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0) - \varphi_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)\right\| \leq C(\boldsymbol{\theta})h^{p}.$$

Using Assumption A3, and considering $\boldsymbol{u} = \hat{\boldsymbol{y}}_m(\boldsymbol{\theta};\mathcal{A})$, $\boldsymbol{v} = \hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta})$, and $\boldsymbol{y} = \boldsymbol{y}_m$, comes:

$$\left|\rho\big(\hat{\boldsymbol{y}}_m(\boldsymbol{\theta};\mathcal{A}), \boldsymbol{y}_m\big) - \rho\big(\hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta}), \boldsymbol{y}_m\big)\right| \leq L_{\rho}(R)\left\|\hat{\boldsymbol{y}}_m(\boldsymbol{\theta};\mathcal{A}) - \hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta})\right\| \leq L_{\rho}(R)C(\boldsymbol{\theta})h^{p}.$$

Averaging over all m data points results:

$$\begin{aligned}
\left|\mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta}) - \mathcal{L}^{*}(\boldsymbol{\theta})\right| &= \left|\frac{1}{M}\sum_{m=1}^{M}\left(\rho\big(\hat{\boldsymbol{y}}_m(\boldsymbol{\theta};\mathcal{A}), \boldsymbol{y}_m\big) - \rho\big(\hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta}), \boldsymbol{y}_m\big)\right)\right| \\
&\leq \frac{1}{M}\sum_{m=1}^{M} L_{\rho}(R)C(\boldsymbol{\theta})h^{p} \\
&= L_{\rho}(R)C(\boldsymbol{\theta})h^{p},
\end{aligned}$$

which is (4).

Proposition 1 shows that the numerical method perturbs the loss by an amount of order $\mathcal{O}(h^p)$. For a fixed h, this perturbation is systematic, meaning that even with infinite data and perfect optimisation, the parameters $\boldsymbol{\theta}$ are chosen to minimise $\mathcal{L}^{\mathcal{A}}$, not $\mathcal{L}^{*}$. The factor h^p in (4) shows how quickly the solver-induced bias decays as the step size $h \to 0$. For a first-order method (p=1, such as the explicit Euler), halving the step size halves the error term, since it scales in h. For a second-order method (p=2), the same refinement reduces the error by a factor of four, since it scales in h^2. For a fourth-order method (p=4, such as the Runge-Kutta), halving h reduces the error by a factor of sixteen, as it scales in h^4 [1]. Thus, higher-order methods make the discrepancy between the *ideal* $\mathcal{L}^{*}$ and *effective* $\mathcal{L}^{\mathcal{A}}$ losses vanish much faster as the time discretisation is refined.

3.1 Training Versus Prediction with Different Methods

We now compare the usage of two different numerical methods, one during training and another during prediction.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ denote two numerical methods of orders p_1 and p_2 with step sizes h_1 and h_2, respectively. Then, the induced numerical flows $\Phi_{\boldsymbol{\theta}}^{\mathcal{A}_1}$ and $\Phi_{\boldsymbol{\theta}}^{\mathcal{A}_2}$, and corresponding losses $\mathcal{L}^{\mathcal{A}_1}$ and $\mathcal{L}^{\mathcal{A}_2}$ can be defined as in (3).

Let us assume that each method satisfies its own global error bound given by [9]:

$$\left\|\Phi_{\boldsymbol{\theta}}^{\mathcal{A}_1}(t_m, t_0, \boldsymbol{y}_0) - \varphi_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)\right\| \leq C_1(\boldsymbol{\theta}) h_1^{p_1}, \tag{5}$$

$$\left\|\Phi_{\boldsymbol{\theta}}^{\mathcal{A}_2}(t_m, t_0, \boldsymbol{y}_0) - \varphi_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)\right\| \leq C_2(\boldsymbol{\theta}) h_2^{p_2}. \tag{6}$$

Proposition 2 (Discrepancy between training and testing losses). *Under Assumptions A1, A2 and A3, and the error bounds* (5) *and* (6)*, for every* $\boldsymbol{\theta} \in \Theta$ *comes:*

$$\left|\mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta}) - \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta})\right| \leq L_\rho(R)\left(C_1(\boldsymbol{\theta}) h_1^{p_1} + C_2(\boldsymbol{\theta}) h_2^{p_2}\right), \tag{7}$$

and when the same method is used for training and testing ($\mathcal{A}_1 = \mathcal{A}_2$)*, comes* $\mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta}) = \mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta})$*, for all* $\boldsymbol{\theta} \in \Theta$*. Any remaining discrepancy is then solely the bias of* $\mathcal{L}^{\mathcal{A}}$ *with respect to* $\mathcal{L}^*$*, which is of order* $\mathcal{O}(h^p)$ *by Proposition 1.*

Proof. Using the triangle inequality [9], and adding and subtracting the losses,

$$\left|\mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta}) - \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta})\right| \leq \left|\mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta}) - \mathcal{L}^*(\boldsymbol{\theta})\right| + \left|\mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta}) - \mathcal{L}^*(\boldsymbol{\theta})\right|. \tag{8}$$

Applying Proposition 1 to $\mathcal{A}_1$ and $\mathcal{A}_2$ separately, with constants $C_1(\boldsymbol{\theta})$ and $C_2(\boldsymbol{\theta})$, comes:

$$\left|\mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta}) - \mathcal{L}^*(\boldsymbol{\theta})\right| \leq L_\rho(R) C_2(\boldsymbol{\theta}) h_2^{p_2}, \qquad \left|\mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta}) - \mathcal{L}^*(\boldsymbol{\theta})\right| \leq L_\rho(R) C_1(\boldsymbol{\theta}) h_1^{p_1},$$

and by substitution of these inequalities in (8) we reach to (7).

Proposition 2 makes it explicit that changing the numerical method between training and prediction perturbs the loss landscape by an amount controlled by the global errors and by the orders p_1 and p_2 of both methods. Furthermore, changing the associated step sizes h_1 and h_2 also affects the magnitude of this perturbation through the factors $h_1^{p_1}$ and $h_2^{p_2}$. Also, changing to lower order methods (small p) or larger step sizes h lead to larger perturbations.

Moreover, for any fixed $\boldsymbol{\theta}$, the difference in predictions produced by the two numerical methods can be bounded directly using (5) and (6):

$$\left\|\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}_2) - \hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}_1)\right\| \leq C_1(\boldsymbol{\theta}) h_1^{p_1} + C_2(\boldsymbol{\theta}) h_2^{p_2}, \qquad \forall m = 1, \ldots, M.$$

Thus, even with the same parameters $\boldsymbol{\theta}$, changing the solver to one of different order changes the outputs by a computable amount depending on $h_1^{p_1}$ and $h_2^{p_2}$.

3.2 Gradient-Level Effects of the Numerical Method

During training, the Neural ODE parameter updates are driven by the gradient of the *effective* loss, $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta})$, which is obtained by differentiating through the numerical solver (using backpropagation or the adjoint method [3]). We now show that, under Assumptions A1-A3 and some additinal assumptions, $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta})$ is a perturbation of the *ideal* loss gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{*}(\boldsymbol{\theta})$ of order $\mathcal{O}(h^p)$, with an explicit dependence on the solver order p and step size h.

We impose the following additional assumptions:

A4) *Smoothness in parameters and states.*
each $t \in [t_0, t_M]$, the vector field $\boldsymbol{f}_{\boldsymbol{\theta}}(t, \boldsymbol{y})$ is continuously differentiable in both $\boldsymbol{\theta}$ and $\boldsymbol{y}$, with bounded derivatives on compact sets. Moreover, the pointwise loss $\rho(\boldsymbol{u}, \boldsymbol{y})$ is continuously differentiable in $\boldsymbol{u}$ with bounded gradient.
Under this assumption, the sensitivity of the continuous flow with respect to the parameters,

$$S_{\boldsymbol{\theta}}(t) = \frac{\partial \varphi_{\boldsymbol{\theta}}(t, t_0, \boldsymbol{y}_0)}{\partial \boldsymbol{\theta}},$$

exists and solves the linear ODE, $\dot{S}_{\boldsymbol{\theta}}(t) = \dfrac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}}{\partial \boldsymbol{y}}(t, \boldsymbol{y}(t)) S_{\boldsymbol{\theta}}(t) + \dfrac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(t, \boldsymbol{y}(t))$,
with initial condition $S_{\boldsymbol{\theta}}(t_0) = 0$.
A5) *Global error for sensitivities.*
to the state variables, the numerical method $\mathcal{A}$ induces a discrete sensitivity:

$$S^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m) = \frac{\partial \Phi^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m, t_0, \boldsymbol{y}_0)}{\partial \boldsymbol{\theta}}, \qquad \forall m = 1, \ldots, M$$

which is the quantity used by automatic differentiation. Then we assume that there exists a constant $C_S(\boldsymbol{\theta})$, independent of h and m, such that [9]:

$$\left\| S^{\mathcal{A}}_{\boldsymbol{\theta}}(t_m) - S_{\boldsymbol{\theta}}(t_m) \right\| \leq C_S(\boldsymbol{\theta}) h^p.$$

Proposition 3 (Gradient perturbation induced by the numerical method). *Suppose Assumptions A1-A5 hold, then for every $\boldsymbol{\theta} \in \Theta$,*

$$\left\| \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \mathcal{L}^{*}(\boldsymbol{\theta}) \right\| \leq K(\boldsymbol{\theta}) h^p, \tag{9}$$

for some constant $K(\boldsymbol{\theta})$ depending on the bounds for $\boldsymbol{f}_{\boldsymbol{\theta}}$, its derivatives, and the loss ρ, but independent of h and m. Thus, the training gradient obtained by differentiating through the numerical solver is a biased approximation of the ideal *gradient, with bias of order $\mathcal{O}(h^p)$.*

Proof. By the chain rule, the *ideal* gradient can be written as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^{*}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \Big(D_{\boldsymbol{u}} \rho\big(\hat{\boldsymbol{y}}^{*}_m(\boldsymbol{\theta}), \boldsymbol{y}_m\big) \Big) S_{\boldsymbol{\theta}}(t_m), \tag{10}$$

where $D_{\boldsymbol{u}} \rho$ is the gradient of ρ with respect to its first component.

Similarly, the *effective* gradient can be expressed as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \Big(D_{\boldsymbol{u}} \rho\big(\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}), \boldsymbol{y}_m\big) \Big) S_{\boldsymbol{\theta}}^{\mathcal{A}}(t_m). \tag{11}$$

Subtracting (10) and (11) comes:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}}(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \mathcal{L}^{*}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \Big[\Delta_m^{(1)}(\boldsymbol{\theta}) + \Delta_m^{(2)}(\boldsymbol{\theta}) \Big],$$

with $\Delta_m^{(1)}(\boldsymbol{\theta}) = \Big(D_{\boldsymbol{u}} \rho\big(\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}), \boldsymbol{y}_m\big) - D_{\boldsymbol{u}} \rho\big(\hat{\boldsymbol{y}}_m^{*}(\boldsymbol{\theta}), \boldsymbol{y}_m\big) \Big) S_{\boldsymbol{\theta}}(t_m)$, and $\Delta_m^{(2)}(\boldsymbol{\theta}) = D_{\boldsymbol{u}} \rho\big(\hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}), \boldsymbol{y}_m\big) \big(S_{\boldsymbol{\theta}}^{\mathcal{A}}(t_m) - S_{\boldsymbol{\theta}}(t_m) \big)$.

From Assumption A4 we have that $D_{\boldsymbol{u}} \rho$ is Lipschitz in $\boldsymbol{u}$, and the sensitivity $S_{\boldsymbol{\theta}}(t_m)$ is bounded. Thus, there exists a constant $C_{S_1}(\boldsymbol{\theta}) > 0$ such that:

$$\big\| \Delta_m^{(1)}(\boldsymbol{\theta}) \big\| \leq C_{S_1}(\boldsymbol{\theta}) \big\| \hat{\boldsymbol{y}}_m(\boldsymbol{\theta}; \mathcal{A}) - \hat{\boldsymbol{y}}_m^{*}(\boldsymbol{\theta}) \big\|.$$

And by combing with the global error bound (2) comes:

$$\big\| \Delta_m^{(1)}(\boldsymbol{\theta}) \big\| \leq C_{S_1}(\boldsymbol{\theta}) C(\boldsymbol{\theta}) h^p.$$

Similarly, from Assumption A5 comes:

$$\| \Delta_m^{(2)}(\boldsymbol{\theta}) \| \leq C_{S_2}(\boldsymbol{\theta}) \| S_{\boldsymbol{\theta}}^{\mathcal{A}}(t_m) - S_{\boldsymbol{\theta}}(t_m) \| \leq C_{S_2}(\boldsymbol{\theta}) C_S(\boldsymbol{\theta}) h^p,$$

and by combining these bounds and averaging over all m we reach to (9), with $K(\boldsymbol{\theta}) = C_{S_1}(\boldsymbol{\theta}) C(\boldsymbol{\theta}) + C_{S_2}(\boldsymbol{\theta}) C_S(\boldsymbol{\theta})$.

Proposition 3 formalises that the choice of numerical method, its step size h, and its order p does not only perturb the value of the loss, but also the gradient directions and magnitudes during optimisation. Consequently, training with different numerical integrators is equivalent to performing gradient descent on different, method-dependent approximations of the optimisation objective, where the approximation error scales with h^p.

3.3 Consequences for the Learnt Parameters and Results

Let $\boldsymbol{\theta}_1^{\star} \in \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta})$ be a local minimiser of the training loss associated with method $\mathcal{A}_1$. Then, from Proposition 2 comes that the testing loss evaluated with a different method satisfies:

$$\mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta}_1^{\star}) \leq \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta}_1^{\star}) + L_{\rho}(R) \big(C_1(\boldsymbol{\theta}_1^{\star}) h_1^{p_1} + C_2(\boldsymbol{\theta}_1^{\star}) h_2^{p_2} \big).$$

Thus, even if $\boldsymbol{\theta}_1^{\star}$ is nearly optimal for $\mathcal{L}^{\mathcal{A}_1}$, it may be sub-optimal for $\mathcal{L}^{\mathcal{A}_2}$, with a difference controlled by the orders p_1 and p_2, and by the respective step sizes.

To clearly show the impact on the optimisation process, we now relate the numerical method to the location of the minimisers in the parameter

space. Assume that $\mathcal{L}^*$ is locally μ-strongly convex around a minimiser $\boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta})$, in the sense of [15, Definition 2.1.2], and that the gradient perturbation bounds hold as in Proposition 3. Let $\boldsymbol{\theta}_1^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta})$ and $\boldsymbol{\theta}_2^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}_2}(\boldsymbol{\theta})$. Then, by Theorem 2.1.9 in [15], strong convexity implies that for any $\boldsymbol{\theta}$ in a neighbourhood of $\boldsymbol{\theta}^\star$ we have:

$$\left\langle \nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}^\star), \boldsymbol{\theta} - \boldsymbol{\theta}^\star \right\rangle \;\geq\; \mu \|\boldsymbol{\theta} - \boldsymbol{\theta}^\star\|^2,$$

and since $\nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}^\star) = \mathbf{0}$, then it reduces to:

$$\left\langle \nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}), \boldsymbol{\theta} - \boldsymbol{\theta}^\star \right\rangle \;\geq\; \mu \|\boldsymbol{\theta} - \boldsymbol{\theta}^\star\|^2.$$

Applying the Cauchy-Schwarz inequality and considering $\boldsymbol{\theta} \neq \boldsymbol{\theta}^\star$:

$$\mu \|\boldsymbol{\theta} - \boldsymbol{\theta}^\star\| \;\leq\; \left\| \nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}) \right\|.$$

By taking $\boldsymbol{\theta} = \boldsymbol{\theta}_1^\star$ and using that $\nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta}_1^\star) = \mathbf{0}$, we get:

$$\mu \|\boldsymbol{\theta}_1^\star - \boldsymbol{\theta}^\star\| \leq \left\| \nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}_1^\star) \right\| = \left\| \nabla_{\boldsymbol{\theta}} \mathcal{L}^*(\boldsymbol{\theta}_1^\star) - \nabla_{\boldsymbol{\theta}} \mathcal{L}^{\mathcal{A}_1}(\boldsymbol{\theta}_1^\star) \right\| \leq K_1(\boldsymbol{\theta}_1^\star) h_1^{p_1},$$

and therefore

$$\|\boldsymbol{\theta}_1^\star - \boldsymbol{\theta}^\star\| \leq \frac{K_1}{\mu} h_1^{p_1}.$$

Following an analogous argument for $\mathcal{A}_2$ comes:

$$\|\boldsymbol{\theta}_2^\star - \boldsymbol{\theta}^\star\| \leq \frac{K_2}{\mu} h_2^{p_2}.$$

Finally, using the triangle inequality, we reach:

$$\|\boldsymbol{\theta}_1^\star - \boldsymbol{\theta}_2^\star\| \leq \|\boldsymbol{\theta}_1^\star - \boldsymbol{\theta}^\star\| + \|\boldsymbol{\theta}^\star - \boldsymbol{\theta}_2^\star\| \lesssim \frac{K_1}{\mu} h_1^{p_1} + \frac{K_2}{\mu} h_2^{p_2}.$$

Thus, we conclude that lower order methods (smaller p) or larger step sizes can lead to minimisers that are significantly displaced from the ideal minimiser $\boldsymbol{\theta}^\star$ and from each other.

4 Numerical Experiments

To support the theoretical analysis in Sect. 3, we compare different solvers during training and testing on a popular benchmark ODE by using the Mean Squared Error (MSE) as the loss and evaluation metric, and explicitly studying the effect of solver order on both the optimisation process and the final predictions.

We use a two-dimensional linear spiral ODE with an analytic solution, which allows us to separate discretisation error from optimisation effects. The ground-truth dynamics are given by:

$$\dot{\boldsymbol{y}}(t) = A\boldsymbol{y}(t), \text{ with } A = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}, \boldsymbol{y}(t) \in \mathbb{R}^2,$$

where $\alpha < 0$ controls radial decay and $\beta > 0$ controls rotational speed. In all experiments we fix $\alpha = -0.1$, $\beta = 1.0$, and the initial condition $y_0 = (2, 0)^\top$. The exact solution is:

$$\boldsymbol{y}(t) = \exp(tA)\boldsymbol{y}_0,$$

which we evaluate analytically. We generate a training/reconstruction dataset by sampling $N_{\text{train}} = 50$ time points on a uniform grid $t_i \in [0, T_{\text{train}}]$, $T_{\text{train}} = 5.0$, to get pairs $(t_i, \boldsymbol{y}(t_i))$. A test/extrapolation dataset is also generated by sampling $N_{\text{test}} = 100$ time points on a uniform grid $t_j \in [0, T_{\text{test}}]$, $T_{\text{test}} = 10.0$.

We use the same Neural ODE configuration across experiments, where $\boldsymbol{f_\theta}$ is a fully-connected neural network with one hidden layer with 32 neurons and a hyperbolic tangent activation function. We considered four ODE solvers: *euler* (explicit Euler, order $p = 1$); *midpoint* (explicit midpoint/Heun, order $p = 2$); *rk4* (Runge-Kutta, order $p = 4$); *dopri5* (adaptive Dormand–Prince method). For the fixed-step solvers (*euler*, *midpoint*, *rk4*) we use the same uniform time grid on $[0, T_{\text{train}}]$, so that they share a common step size h. The adaptive solver *dopri5* is run with its default tolerances (see [2]).

All models were trained with the Adam optimiser, using a learning rate 10^{-2} and 200 epochs, with batch size equal to the full training set. Each configuration is repeated with 3 random seeds and we report mean and standard deviation across seeds. All implementations were done using the *Torchdiffeq* library [2].

Note that, all Neural ODE models were trained by fixing seeds as to isolate the effect of changing the numerical method.

4.1 Effect of Different Solvers for Training and Testing

To study the impact of using different solvers during training and prediction, we consider a full grid of training and testing solvers. Let $\mathcal{S} = \{$*euler*, *midpoint*, *rk4*, *dopri5*$\}$. For each training solver $\mathcal{A}_{\text{tr}} \in \mathcal{S}$ and each random seed, we train a Neural ODE model by minimising $\mathcal{L}^{\mathcal{A}_{\text{tr}}}(\boldsymbol{\theta})$ on the training dataset, $[0, T_{\text{train}}]$ and we evaluate the trained model on the training grid using $\mathcal{A}_{\text{tr}}$ and compute the MSE against $\boldsymbol{y}(t)$ on $[0, T_{\text{train}}]$. Then, we evaluate the trained model on the training grid using all $\mathcal{A}_{\text{te}} \in \mathcal{S}$ and compute the MSE against $\boldsymbol{y}(t)$ on $[0, T_{\text{train}}]$. Finally, we do an extrapolation experiment by evaluating the same model on the extended test grid using all $\mathcal{A}_{\text{te}}$ and compute the MSE against $\boldsymbol{y}(t)$ on $[0, T_{\text{test}}]$.

Table 1, Table 2, and Table 3 present the reconstruction and extrapolation results, respectively, with the average MSE and its respective standard deviation (std), over the three seeds, for each pair $(\mathcal{A}_{\text{tr}}, \mathcal{A}_{\text{te}})$. Figure 1 and Fig. 2 show the predictions of the Neural ODE models trained with $\mathcal{A}_{\text{tr}}$ and using $\mathcal{A}_{\text{te}}$ for the reconstruction and extrapolation experiments, respectively.

Table 1. Training error (mean MSE ± std) for different training (row) and testing (column) solvers.

Training	Testing			
	dopri5	euler	midpoint	rk4
dopri5	1.62e-03 ± 3.56e-04	1.62e-03 ± 3.56e-04	1.62e-03 ± 3.56e-04	1.62e-03 ± 3.56e-04
euler	1.38e-03 ± 2.83e-04	1.38e-03 ± 2.83e-04	1.38e-03 ± 2.83e-04	1.38e-03 ± 2.83e-04
midpoint	1.63e-03 ± 3.61e-04	1.63e-03 ± 3.61e-04	1.63e-03 ± 3.61e-04	1.63e-03 ± 3.61e-04
rk4	1.62e-03 ± 3.56e-04	1.62e-03 ± 3.56e-04	1.62e-03 ± 3.56e-04	1.62e-03 ± 3.56e-04

Table 2. Reconstruction error (mean MSE ± std) for different training (row) and testing (column) solvers.

Training	Testing			
	dopri5	euler	midpoint	rk4
dopri5	1.62e-03 ± 3.56e-04	2.71e-02 ± 3.23e-03	1.65e-03 ± 3.54e-04	1.62e-03 ± 3.56e-04
euler	1.94e-02 ± 2.23e-03	1.38e-03 ± 2.83e-04	2.01e-02 ± 2.16e-03	1.94e-02 ± 2.23e-03
midpoint	1.63e-03 ± 3.63e-04	2.78e-02 ± 3.14e-03	1.63e-03 ± 3.61e-04	1.63e-03 ± 3.63e-04
rk4	1.62e-03 ± 3.56e-04	2.71e-02 ± 3.23e-03	1.65e-03 ± 3.54e-04	1.62e-03 ± 3.56e-04

From Table 1, we see that a Neural ODE trained with *euler* achieves the lowest training error, while the higher-order solvers *dopri*, *rk4*, and *midpoint* get slightly larger values, showing that all solvers can fit the data on the training grid to essentially the same level, with small variability across seeds.

In Table 2, the diagonal entries coincide with the training errors, as expected when training and testing solvers agree and the training and testing datasets are the same. Off-diagonal entries, however, can be an order of magnitude larger, especially when testing with *euler* a model trained with a higher-order solver, or vice versa. In conjunction with Fig. 1, this directly illustrates the train-test discrepancy showed in our theoretical analysis, Sect. 3, in which different solvers induce different effective loss landscapes, so a model optimised under one solver can perform poorly when evaluated under another.

Table 3. Extrapolation error (mean MSE ± std) for different training (row) and testing (column) solvers.

Training	Testing			
	dopri5	euler	midpoint	rk4
dopri5	5.91e-02 ± 5.91e-02	5.25e-02 ± 2.72e-02	6.11e-02 ± 6.06e-02	5.91e-02 ± 5.91e-02
euler	1.02e-01 ± 4.87e-02	5.25e-02 ± 5.06e-02	1.04e-01 ± 4.95e-02	1.02e-01 ± 4.86e-02
midpoint	5.77e-02 ± 5.82e-02	5.24e-02 ± 2.60e-02	5.96e-02 ± 5.97e-02	5.77e-02 ± 5.82e-02
rk4	5.91e-02 ± 5.91e-02	5.25e-02 ± 2.72e-02	6.11e-02 ± 6.06e-02	5.91e-02 ± 5.91e-02

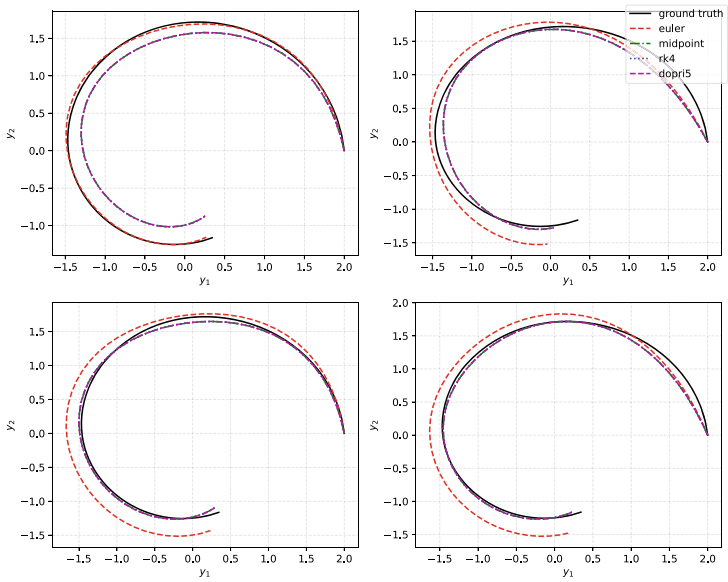

Fig. 1. Reconstruction test predictions for Neural ODE models trained and tested with *euler*, *dopri*, *rk4*, and *midpoint*. The same seed is used to allow to isolate the influence of the numerical method.

Table 3 and Fig. 2, show that errors grow substantially on the longer horizon, and here the differences between training solvers become more noticeable. Models trained with higher-order solvers (*dopri* or *midpoint*) that are then evaluated also with a high-accuracy solver achieve extrapolation MSE around $10e{-}2$, while training with *euler* leads to doubling the test error to $10e{-}1$. Note that, the low quality visual predictions in Fig. 2 are also caused by the extreme extrapolation test in which the models are predicting a time horizon twice as bigger than the one seen during training.

Overall, the numerical results support the statement that: (i) using a sufficiently accurate and consistent solver is important both for reconstruction and for long-term extrapolation; (ii) changing the solver between training and testing can have a large quantitative impact even when the training loss is nearly identical.

4.2 Effect of the Solver Order on the Optimisation Process

The previous experiments compare final errors across many solver configurations, but do not isolate how the solver affects the optimisation path itself. To do so, we perform a fixed-seed study that keeps all sources of randomness constant and varies only the training solver, and analyse the models' MSE over epochs for all solvers, Fig. 3.

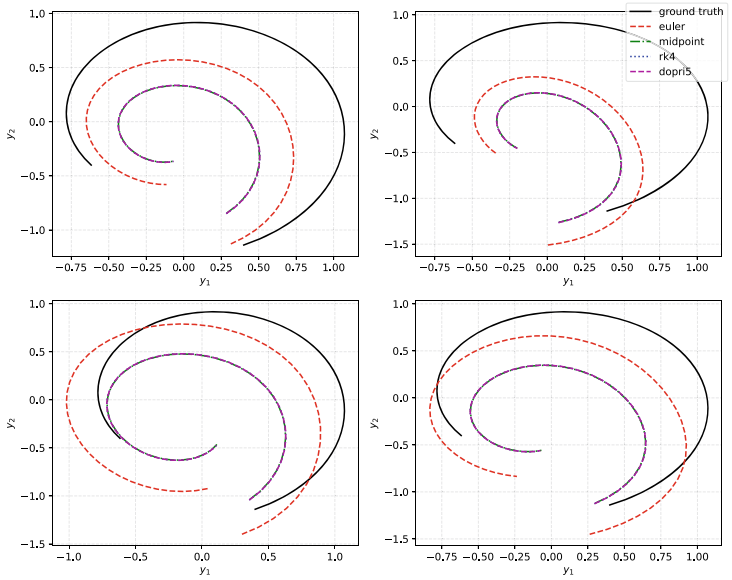

Fig. 2. Extrapolation test predictions for Neural ODE models trained and tested with *euler*, *dopri*, *rk4*, and *midpoint*. The same seed is used to allow to isolate the influence of the numerical method.

This experiment directly illustrates the gradient-level effects discussed in Sect. 3. Even when starting from the same initial parameters and observing the same data, different training solvers induce different effective gradients $\nabla \mathcal{L}^{\mathcal{A}_{\text{tr}}}(\boldsymbol{\theta})$, leading to different optimisation paths and converging to different parameter values and trajectories.

From Fig. 3, we observe that during the first 30 epochs all training solvers show almost identical MSE, consistent with the fact that the trajectories and hence the propagated numerical errors are still very small near the initialisation. As training progresses, solver-induced truncation error accumulates along the integration interval, leading to different effective gradients. In this regime *Euler* induces dynamics that decreases its effective loss faster and maintains the lowest training MSE during the whole training.

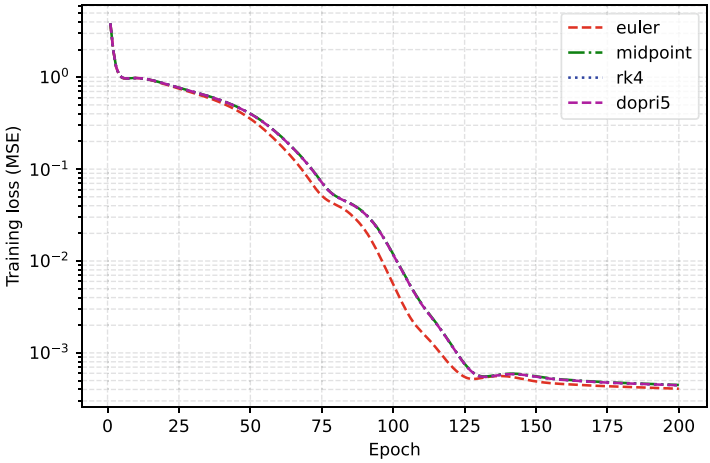

Fig. 3. Training MSE over epoch for four Neural ODE models trained from the same initialisation and data, but with different numerical integrators (*euler*, *dopri5*, *rk4*, *midpoint*).

5 Conclusion

In this work we demonstrated that the numerical integrator used in a Neural ODE is not a minor implementation detail, but an integral part of the model that shapes both the loss landscape and the optimisation dynamics. On the theoretical side, we proved that a fixed-step method of order p and step size h perturbs the ideal loss and its gradient by order $\mathcal{O}(h^p)$. Thus, different solvers induce different surrogate objectives and different gradient fields, with the discrepancy between two methods scaling with their respective global errors. Under some assumptions, we further demonstrated that the corresponding minimisers in the parameter space can be displaced from the ideal minimiser, and from each other, by h^p.

The numerical experiments on a simple spiral ODE corroborate these statement demonstrating that, in general, different numerical integrators can fit the training data to essentially the same MSE. Using different solvers in training and testing can increase prediction errors by up to an order of magnitude, even when the training loss is nearly identical. Specifically, mismatched train-test pairs involving a low-order solver, such as *euler*, produce larger discrepancies, while higher-order methods, when used consistently at train and test time, show markedly better performance Furthermore, starting from the same initial parameters and using the same data, models trained with different solvers initially follow similar loss curves, but their trajectories and parameter updates diverge as solver-induced truncation errors accumulate. This leads to distinct optimisation paths and different final parameter vectors, in line with our theoretical analysis of solver-dependent gradient perturbations.

As a whole, these results support the view that the choice of numerical method-order, and step size should be treated as part of the Neural ODE model specification. For reliable predictions, it is important to: (i) use a sufficiently accurate solver during both training and testing; (ii) avoid changing the solver (or its configurations) at prediction time without accounting for the induced bias; (iii) document the numerical configuration alongside architectural and optimisation details.

Beyond the specific results reported here, this work is as a call to action when developing, publishing, or deploying Neural ODEs and other solver-based neural networks, the numerical integrator and its settings should be treated as first-class modelling choices, documented with the same care as the architecture and optimisation procedure, and kept consistent between training and prediction whenever possible.

Note that, this theoretical analysis is valid to any neural network using a numerical method to output a response, such as Neural ODEs, and continuous-normalizing flows.

Future work could extend this analysis to stiff systems, adaptive-step training, and more complex architectures, as well as investigate principled strategies for jointly selecting or learning the numerical method and the Neural ODE parameters.

Acknowledgment. C. Coelho would like to thank the KIBIDZ project funded by dtec.bw - Digitalization and Technology Research Center of the Bundeswehr; dtec.bw is funded by the European Union-NextGenerationEU. We also acknowledge the computational resources (HPC cluster HSUper) that have been provided by the project hpc.bw, funded by dtec.bw.

References

1. Chapra, S.C.: Numerical Methods for Engineers. Mcgraw-Hill (2010)
2. Chen, R.T.Q.: torchdiffeq (2018). https://github.com/rtqichen/torchdiffeq
3. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. Adv. Neural Inf. Process. Syst. **31** (2018)
4. Coelho, C., Costa, M.F.P., Ferrás, L., Niggemann, O.: Neural ODEs for optimal control: learning continuous trajectories and feedback policies via adaptive discretisation. Authorea Preprints (2026)
5. Coelho, C., Costa, M.F.P., Ferrás, L.L.: Neural Chronos ODE: modelling bidirectional temporal patterns in time-series data. Exp. Syst. Appl. **273**, 126784 (2025)
6. Coelho, C., Costa, M.P., Ferrás, L.L.: Consistency matters: neural ODE parameters are dependent on the training numerical method. In: ICLR 2024 Workshop on AI4DifferentialEquations in Science (2024). https://openreview.net/forum?id=tPDA7Zt1bf
7. Coelho, C., Costa, M.F.P., Ferrás, L.L.: Neural fractional differential equations. Appl. Math. Model. **144**, 116060 (2025)
8. Dahlquist, G.: Convergence and stability in the numerical integration of ordinary differential equations. Mathematica Scandinavica, 33–53 (1956)

9. Hairer, E., Wanner, G., Nørsett, S.P.: Solving Ordinary Differential Equations I: Nonstiff Problems. SSCM. Springer, Heidelberg (1993). https://doi.org/10.1007/978-3-540-78862-1
10. Jia, J., Benson, A.R.: Neural jump stochastic differential equations. Adv. Neural Inf. Process. Syst. **32** (2019)
11. Khoo, Z.-Y., Lee, K.H., Huang, Z., Bressan, S.: Neural ordinary differential equations for the regression of macroeconomics data under the Green Solow model. In: Strauss, C., Kotsis, G., Tjoa, A.M., Khalil, I. (eds.) DEXA 2021. LNCS, vol. 12923, pp. 78–90. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86472-9_7
12. Kumar, T., Kumar, A., Pal, P.: A physics-constrained neural ordinary differential equations approach for robust learning of stiff chemical kinetics. Combust. Theor. Model. **29**(3), 332–347 (2025)
13. Lee, K., Parish, E.J.: Parameterized neural ordinary differential equations: applications to computational physics problems. Proc. R. Soc. A Math. Phys. Eng. Sci. **477**(2253) (2021)
14. Li, J., Chen, W., Liu, Y., Yang, J., Zeng, D., Zhou, Z.: Neural ordinary differential equation networks for fintech applications using internet of things. IEEE Internet Things J. **11**(12), 21763–21772 (2024)
15. Nesterov, Y.: Introductory Lectures on Convex Programming Volume I: Basic Course. Lecture Notes, vol. 3, no. 4, p. 5 (1998)
16. Owoyele, O., Pal, P.: ChemNODE: a neural ordinary differential equations framework for efficient chemical kinetic solvers. Energy AI **7**, 100118 (2022)
17. Qian, Z., Zame, W., Fleuren, L., Elbers, P., van der Schaar, M.: Integrating Expert ODEs into Neural ODEs: pharmacology and disease progression. Adv. Neural. Inf. Process. Syst. **34**, 11364–11383 (2021)
18. Richter, T., Wang, W., Palma, A., Theis, F.J.: Generative models of cell dynamics: from Neural ODEs to flow matching. Commun. Biol. (2026)
19. Rubanova, Y., Chen, R.T., Duvenaud, D.K.: Latent ordinary differential equations for irregularly-sampled time series. Adv. Neural Inf. Process. Syst. **32** (2019)
20. Yang, Z., et al.: Detection of wind turbine blade abnormalities through a deep learning model integrating VAE and neural ODE. Ocean Eng. **302**, 117689 (2024)
21. Zakwan, M., Di Natale, L., Svetozarevic, B., Heer, P., Jones, C.N., Trecate, G.F.: Physically consistent neural odes for learning multi-physics systems. IFAC-PapersOnLine **56**(2), 5855–5860 (2023)
22. Zhu, Q., Guo, Y., Lin, W.: Neural delay differential equations. arXiv preprint arXiv:2102.10801 (2021)