Efficient, Accurate and Stable Gradients for Neural ODEs

Sam McCallum* **James Foster**
Department of Mathematical Sciences
University of Bath
Bath, United Kingdom

Abstract

Training Neural ODEs requires backpropagating through an ODE solve. The state-of-the-art backpropagation method is recursive checkpointing that balances recomputation with memory cost. Here, we introduce a class of algebraically reversible ODE solvers that significantly improve upon both the time and memory cost of recursive checkpointing. The reversible solvers presented calculate exact gradients, are high-order and numerically stable – strictly improving on previous reversible architectures. On scientific modeling and time series classification experiments, reversible solvers reduce the training time by more than $2\times$ while using at least $10\times$ less memory on average than recursive checkpointing.

1 Introduction

1.1 Neural ODEs

Neural ODEs introduce a model class where the vector field of an ODE is parameterized as a neural network (R. T. Chen et al. 2018). This strong prior on model space can offer an advantage in many problems.

For example, one of the most exciting areas for Neural ODEs is scientific modeling. This is emphasized by Hamiltonian and Lagrangian networks for learning physical systems (Greydanus et al. 2019; Cranmer et al. 2020); and more generally by universal differential equations for combining theoretical models and neural network vector fields (Rackauckas et al. 2020). Further, if theory is completely unknown, Neural ODEs can be used to identify latent dynamics from data. There are many examples of these approaches being applied to problems across the sciences (Portwood et al. 2019; Lee et al. 2021; Lu et al. 2021; X. Chen et al. 2022; Boral et al. 2024).

The continuous latent state encoded by Neural ODEs can also be beneficial in time series modeling, where data is often partially observed and irregularly sampled. The application to time series was originally explored by latent ODEs (Rubanova et al. 2019) and built upon by Neural Controlled Differential Equations (CDEs) (Kidger, Morrill, et al. 2020) and Stochastic Differential Equations (SDEs) (X. Li et al. 2020; Kidger, Foster, X. Li, et al. 2021; Issa et al. 2024).

1.2 Training

Training Neural ODEs requires that we backpropagate through the ODE solve and proceeds by two main approaches: standard automatic differentiation through the internal numerical solver operations (discretize-then-optimize) or via the continuous adjoint method (optimize-then-discretize) (Kidger 2022).

*Correspondence to sm2942@bath.ac.uk.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Discretize-then-optimize is generally the preferred training method as gradient calculation is exact, rather than the approximation made by optimize-then-discretize which has been shown to result in unstable training dynamics (Gholami et al. 2019; Onken et al. 2020).

However, the memory cost of storing all numerical operations for discretize-then-optimize can become prohibitively large. To keep memory cost low, checkpointing algorithms are used that balance recomputation and memory usage by only storing a subset of the numerical operations (Griewank 1992; Stumm et al. 2010; Gholami et al. 2019).

Specifically, online recursive checkpointing algorithms guarantee an $O(n \log n)$ time cost while storing $O(\sqrt{n})$ checkpoints, where n is the computation length. An online algorithm is required to handle the case where n is unknown in advance, for example when using adaptive step size methods. The online recursive checkpointing algorithm from (Stumm et al. 2010) will form the baseline method for our comparisons here.

1.3 Algebraically reversible Solvers

Interestingly, it is possible to backpropagate through ODEs without storing any intermediate numerical operations *and* calculate exact gradients. This is realized by constructing an algebraically reversible solver, whereby the solver state at step n can be reconstructed exactly in closed form from the solver state at step $n+1$. For example, symplectic solvers for Hamiltonian systems are intrinsically reversible (Greydanus et al. 2019).

In general, there have been two previously proposed algebraically reversible solvers: Zhuang et al. 2021 use the Asynchronous Leapfrog (ALF) method that is a second-order reversible ODE solver; and, Kidger, Foster, X. C. Li, et al. 2021 devise a reversible Heun method that is a second-order ODE solver and 0.5 strong order SDE solver.

Reversible solvers realize an $O(n)$ time and $O(1)$ memory backpropagation algorithm, improving upon the complexity of recursive checkpointing. However, current reversible solvers are low-order and suffer from poor numerical stability. The use of these methods in practice is therefore limited.

1.4 Contributions

In this work we address both the low-order convergence and stability issues of previous reversible solvers for Neural ODEs. We present a general class of algebraically reversible solvers that allows any single-step numerical solver to be made reversible. This class of reversible solvers calculate exact gradients and display the following properties:

1. $O(n)$ time, $O(1)$ memory complexity (Algorithm 1),
2. high-order convergence (Theorem 2.1),
3. improved numerical stability (Theorem 2.3).

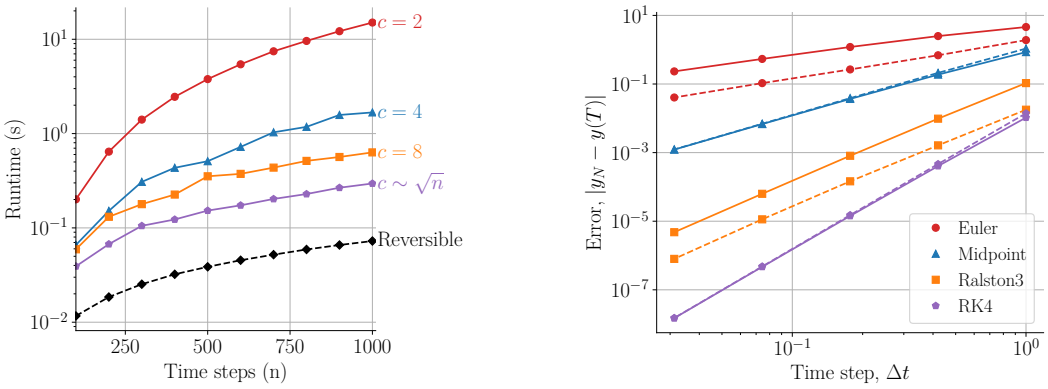


Figure 1: *Left*: Runtime complexity of reversible backpropagation vs recursive checkpointing with c checkpoints. *Right*: Reversible solver convergence (dashed) is inherited from base solver (solid).

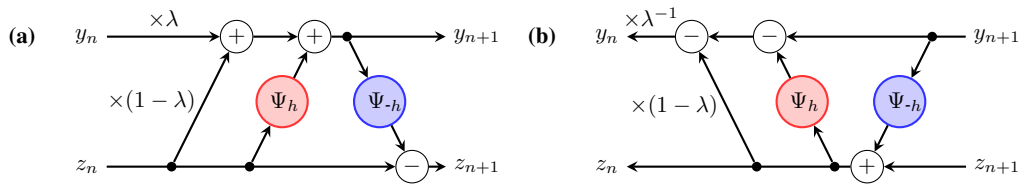


Figure 2: Computation graph of the reversible method. **(a)** Forward solve. **(b)** Backward solve.

2 Reversible Solvers

We introduce a class of reversible solvers where any single-step numerical solver can be made reversible. The algebraic reversibility property allows one to dynamically recompute the forward solve in closed form during backpropagation, thereby obtaining exact gradients in $O(n)$ time and $O(1)$ memory.

We will write our Neural ODE system as

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = f_\theta(t, y(t)), \qquad y(0) = y_0, \tag{1}$$

where $y(t) \in \mathbb{R}^d$ is the solution to (1) and $f_\theta : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ is a neural network parameterized by $\theta \in \mathbb{R}^m$. The initial condition is given by $y_0 \in \mathbb{R}^d$.

2.1 Forward solve

Suppose we have a single-step numerical ODE solver, $\Psi_h(t, y) : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$, such that a numerical step is given by $y_{n+1} = y_n + \Psi_h(t, y)$, where h is the step size. Then we construct a reversible numerical solution $\{y_n, z_n\}_{n \geq 0}$ by

$$\begin{aligned} y_{n+1} &= \lambda y_n + (1-\lambda) z_n + \Psi_h(t_n, z_n), \\ z_{n+1} &= z_n - \Psi_{-h}(t_{n+1}, y_{n+1}), \end{aligned} \tag{2}$$

where $\lambda \in (0, 1]$ is a coupling parameter. The initial state is given by $y_0 = z_0 = y(0)$ and $t_n = nh$ for each $n \geq 0$.

2.2 Backward solve

The numerical scheme in (2) is algebraically reversible, with the reverse scheme given by

$$\begin{aligned} z_n &= z_{n+1} + \Psi_{-h}(t_{n+1}, y_{n+1}), \\ y_n &= \lambda^{-1} y_{n+1} + \left(1 - \lambda^{-1}\right) z_n - \lambda^{-1} \Psi_h(t_n, z_n). \end{aligned} \tag{3}$$

This construction allows us to step between $(y_n, z_n) \leftrightarrow (y_{n+1}, z_{n+1})$ in closed form. It is this property that results in exact gradient calculation as we can dynamically recompute the forward computation graph exactly. See Figure 2 for an illustration.

The above construction of reversibility shares familiarities with other reversible architectures in machine learning. For example, we see a similar coupling between the evolving state in Reversible Networks (RevNets) (Gomez et al. 2017) and in coupling layers for normalizing flows (Dinh et al. 2015). However, the coupling parameter λ is absent from these architectures; we show later that λ is essential for numerical stability when solving ODEs (see Theorem 2.3).

2.3 Convergence

The convergence order of the reversible solver is inherited from the base solver Ψ. That is, provided Ψ is a method with k-th order convergence then the reversible scheme will also have k-th order convergence. This result is illustrated by Theorem 2.1.

For example, we could choose Ψ to be a step of a fourth order Runge-Kutta method (RK4). In which case, the reversible solver in (2) would correspond to a reversible RK4 method and also display fourth order convergence. See Figure 1.

We therefore see that a reversible solver of arbitrarily high order may be constructed from this method. This is in contrast to previous reversible methods where the convergence order is baked in.

Theorem 2.1. *For a fixed time-horizon $T > 0$, we consider the Neural ODE in (1) over $[0, T]$. Let $T = Nh$ where $N > 0$ denotes the number of steps and $h > 0$ is the step size.*

Let Ψ be a k-th order ODE solver satisfying the Lipschitz condition (see A.2) and consider the reversible solution $\{y_n, z_n\}_{n\geq 0}$, given by (2). Then there exist constants $h_{max}, C > 0$ such that, for $h \in [0, h_{max}]$,

$$||y_n - y(t_n)|| \leq Ch^k. \tag{4}$$

Proof. The proof of Theorem 2.1 is given in Appendix A. □

2.4 Backpropagation Algorithm

We now discuss the key result of reversible solvers: exact gradient backpropagation in $O(n)$ time and $O(1)$ memory. The complexity results from dynamically recomputing the forward solve during backpropagation, requiring only the terminal solver state be stored in memory.

The backpropagation algorithm through one solver step is presented in Algorithm 1 and follows the construction of reverse-mode automatic differentiation (Baydin et al. 2018).

We define a scalar-valued loss function $L(y_N) : \mathbb{R}^d \to \mathbb{R}$ on the terminal value of the numerical solve $y_N \approx y(T)$. It is possible to define a loss function on any subset of the numerical solution $\{y_n\}_{0\leq n\leq N}$ but we choose to depend on y_N for ease of presentation.

The gradients (adjoints) are defined as $\bar{v} = \partial L(y_N)/\partial v$. Note that the presence of an adjoint to the left of a Jacobian, $\bar{v}\frac{\partial f}{\partial x}$, indicates a vector-Jacobian product. This can be efficiently computed using automatic differentiation.

The algorithm proceeds by first taking a backward step from $n+1$ to n, recovering the state at n exactly. Then we backpropagate the adjoints from $n+1$ to n by tracing the computation graph of the forward solve, applying the chain rule as we go.

We write the step size as h which can be a constant or selected by an adaptive time stepping algorithm.

Algorithm 1 Reversible backpropagation

Input: $t_{n+1}, y_{n+1}, z_{n+1}, \bar{y}_{n+1}, \bar{z}_{n+1}, \bar{\theta}$

Backward step

$$z_n = z_{n+1} + \Psi_{-h}(t_{n+1}, y_{n+1})$$
$$y_n = \lambda^{-1} y_{n+1} + \left(1 - \lambda^{-1}\right) z_n - \lambda^{-1}\Psi_h(t_n, z_n)$$
$$t_n = t_{n+1} - h$$

Gradients

$$\bar{y}_{n+1} \leftarrow \bar{y}_{n+1} - \bar{z}_{n+1}\frac{\partial \Psi_{-h}(t_{n+1}, y_{n+1})}{\partial y_{n+1}}$$
$$\bar{y}_n = \lambda \bar{y}_{n+1}$$
$$\bar{z}_n = \bar{z}_{n+1} + (1-\lambda)\bar{y}_{n+1} + \bar{y}_{n+1}\frac{\partial \Psi_h(t_n, z_n)}{\partial z_n}$$
$$\bar{\theta} \leftarrow \bar{\theta} - \bar{z}_{n+1}\frac{\partial \Psi_{-h}(t_{n+1}, y_{n+1})}{\partial \theta} + \bar{y}_{n+1}\frac{\Psi_h(t_n, z_n)}{\partial \theta}$$

Return: $t_n, y_n, z_n, \bar{y}_n, \bar{z}_n, \bar{\theta}$

The vector-Jacobian products in Algorithm 1 are computable for an arbitrary (differentiable) step function Ψ. This product requires no knowledge on the structure of Ψ a-priori and no local forward evaluations as Ψ is evaluated on the backward solve.

2.5 Stability

One key ingredient missing from previous reversible solvers is numerical stability. Here we show that the reversible scheme introduced above in (2) has a non-zero linear stability region for Ψ given by any explicit Runge-Kutta method. This is shown by Theorem 2.3.

We prove stability, as is customary, for the real-valued linear test problem (Stewart 2022).

Definition 2.2 (Linear Stability)**.** Let $y \in \mathbb{R}^d$ and consider the ODE,

$$\frac{\mathrm{d}y}{\mathrm{d}t} = \alpha y, \qquad y(0) = y_0, \tag{5}$$

where $\alpha < 0$ is a negative real constant and $y_0 \in \mathbb{R}^d$ is a non-zero initial condition. A numerical solution $\{y_n\}_{n\geq 0}$ to (5) is linearly stable if the spectral radius (maximum eigenvalue) $\rho(T) < 1$, where $T \in \mathbb{R}^{d\times d}$ is given by

$$y_{n+1} = Ty_n. \tag{6}$$

This definition expresses the condition that a numerical method applied to (5) should decay to zero as $t \to \infty$. For this to hold we require that the maximum eigenvalue of $T < 1$. Note that for the reversible scheme in (2) we require that the pair $\{y_n, z_n\}$ tends to zero.

The theorem is simplified by the concept of transfer functions for Runge-Kutta methods, such that we can write $\Psi_h(t_n, y_n) = R(h\alpha)y_n$ for a 'transfer function' $R(h\alpha)$ with step size h. This is a commonly used result in stability analysis, see (Stewart 2022) for a complete discussion.

Theorem 2.3. *Let Ψ be given by an explicit Runge-Kutta solver. Then the reversible numerical solution $\{y_n, z_n\}_{n\geq 0}$ given by (2) is linearly stable iff*

$$|\Gamma| < 1 + \lambda, \tag{7}$$

where

$$\Gamma = 1 + \lambda - (1 - \lambda)R(-h\alpha) - R(-h\alpha)R(h\alpha). \tag{8}$$

Proof. The proof of Theorem 2.3 is given in Appendix B. □

Example. Theorem 2.3 is best illustrated by an example. Consider using Euler's method for the base solver Ψ, then $R(h\alpha) = h\alpha$. Substituting into the stability condition (7), we get $h\alpha > \lambda - 1$. As $\lambda \to 1$, the stability region tends to zero. And, as $\lambda \to 0$ we recover half the stability region of the base Euler solver.

This example illustrates some key points. The stability region of reversible Ψ is decreased in comparison to the stability region of Ψ, where Ψ is the base solver. However, the stability region of the reversible solver is non-zero and for any coupling parameter $\lambda \in (0, 1)$ there exists a step size h such that the reversible scheme is linearly stable. This is in contrast to previous reversible architectures, ALF (Zhuang et al. 2021) and reversible Heun (Kidger, Foster, X. C. Li, et al. 2021), that are nowhere linearly stable for any step size h. See Figure 3 for a visualization.

We see from Theorem 2.3 that the choice of coupling parameter λ is key to the reversible method – stability is increased on the forward solve by decreasing λ. However, decreasing λ will reduce the numerical stability of the backward solve due to the presence of λ^{-1} terms. To balance the stability it is therefore important that λ remains close to 1 and in practice we find $\lambda \in [0.99, 0.999]$ to be good choices.

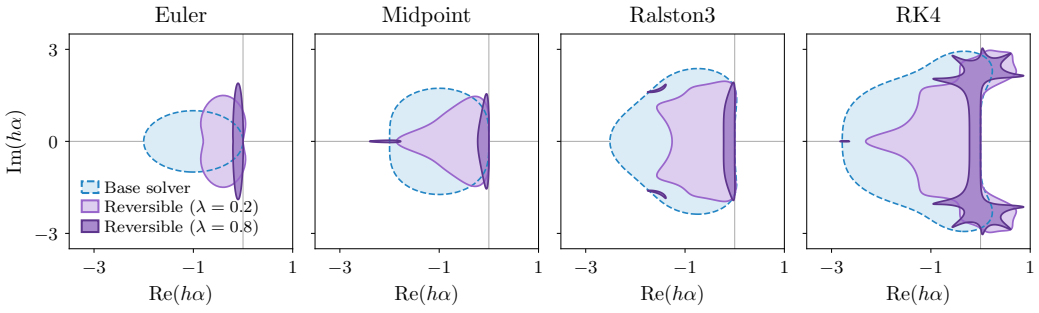


Figure 3: Linear stability regions (complex-valued α) of the reversible solver compared to the base solver for a set of Runge-Kutta methods.

2.6 Adaptive step sizes

It is straightforward to extend the reversible solvers to use adaptive step sizes. This follows naturally from the construction of the forward step (2) due to the dependence on the base solver step Ψ. That is, if the base solver Ψ is capable of providing local error estimates then the reversible method can inherit the estimate.

This allows us to obtain an adaptive reversible solver from an adaptive base solver. We will use an adaptive reversible solver in Section 3.3 to handle chaotic system dynamics.

We choose to inherit the error estimate provided by the local forward step, Ψ_h. It is also possible to include the error estimate provided by the local backward step, Ψ_{-h}. In general, two error estimates are available per step at t_n and t_{n+1} which may prove useful for stiff problems.

3 Experiments

We perform three experiments that compare the test accuracy, runtime and memory cost of reversible solvers to recursive checkpointing. First, we consider discovery of latent ODE dynamics from scientific modeling data. Second, we train Neural CDE models on time series classification problems. Third, we verify the performance of reversible solvers with adaptive time steps on both scientific modeling data and CDE time series classification.

3.1 Discovery of latent dynamics

For both experiments in this section, we parameterize the Neural ODE in (1) with a feed-forward neural network vector field. The model is trained to minimise the mean-squared-error. Gradient calculation is performed by both the reversible method and recursive checkpointing for a range of numerical solvers: Euler, Midpoint, Ralston's 3rd order method (Ralston3) and the classic Runge-Kutta 4 method (RK4).

We compare to recursive checkpointing over a range of checkpoints c up to the optimal memory/runtime trade-off $c \sim \sqrt{n}$. Note that the memory usage of the reversible solver is equivalent to 2 checkpoints as the method must store the augmented state $\{y_n, z_n\}$.

Chandrasekhar's Equation The first task we consider is discovery of Chandrasekhar's white dwarf equation from generated data. The system describes the density of a white dwarf φ as a function of the radial distance r from the center of the star. The density follows the second-order ODE,

$$\frac{1}{r^2}\frac{\mathrm{d}}{\mathrm{d}r}\left(r^2\frac{\mathrm{d}\varphi}{\mathrm{d}r}\right) + \left(\varphi^2 - C\right)^{3/2} = 0,$$
$$\varphi(0) = 1, \quad \frac{\mathrm{d}\varphi}{\mathrm{d}r}(0) = 0,$$

where $\sqrt{C}$ is a constant that defines the lower bound on the density φ. The system provides a strong test case for Neural ODEs as the vector field is both non-linear and 'time-dependent'.

We generate training data by simulating the white dwarf system over $r \in [0, 5]$ with $C = 0.001$. The task is to learn the dynamics of the white dwarf system. The runtime and memory usage of both training methods is shown in Table 1.

We see from Table 1 that reversible backpropagation is at least $2.5\times$ faster than recursive checkpointing while using $22\times$ less memory. Further, reversible backpropagation is at least $100\times$ faster for the same memory usage. These results hold across all solvers considered. The mean final loss obtained is the same for both methods, 0.9×10^{-4}, due to exact gradient calculation.

Coupled oscillators The second experiment we consider investigates the performance of the reversible method for identifying dynamics from real-system data. We use the dataset from (Schmidt et al. 2009), designed to facilitate research in discovery of physical laws from data. Specifically, we consider the real coupled oscillator system.

Table 1: Memory usage (in terms of checkpoints c) and runtime for Chandrasekhar experiment. Mean $\pm$ standard deviation over three repeats.

Method	Memory	Runtime (min)			
		Euler	Midpoint	Ralston3	RK4
Reversible	**2**	$\mathbf{1.5 \pm 0.3}$	$\mathbf{1.7 \pm 0.4}$	$\mathbf{1.8 \pm 0.2}$	$\mathbf{2.2 \pm 0.3}$
Recursive	2	264.0 ± 17.5	280.0 ± 13.7	259.9 ± 22.9	254.3 ± 16.1
	4	30.5 ± 2.3	30.3 ± 1.6	29.8 ± 2.2	29.2 ± 1.8
	16	8.9 ± 1.2	9.6 ± 0.6	8.4 ± 0.2	8.2 ± 0.2
	32	8.1 ± 0.8	8.7 ± 0.7	7.6 ± 0.2	7.5 ± 0.2
	44	5.2 ± 0.7	5.5 ± 0.8	4.90 ± 0.04	4.95 ± 0.02

Table 2: Memory usage (in terms of checkpoints c), runtime and final loss for coupled oscillator experiment. Mean $\pm$ standard deviation over three repeats.

Method	Memory	Runtime (min)			Loss ($\times 10^{-3}$)		
		Midpoint	Ralston3	RK4	Midpoint	Ralston3	RK4
Reversible	**2**	$\mathbf{14.3 \pm 3.1}$	$\mathbf{13.2 \pm 0.3}$	$\mathbf{19.7 \pm 6.6}$	$\mathbf{1.0 \pm 0.2}$	1.3 ± 0.3	$\mathbf{1.2 \pm 0.4}$
Recursive	2	632.2 ± 20.0	645.2 ± 7.2	654.9 ± 0.8	1.0 ± 0.2	$\mathbf{1.2 \pm 0.1}$	1.4 ± 0.3
	4	99.0 ± 10.7	95.7 ± 1.9	97.5 ± 0.1			
	8	63.4 ± 9.8	57.5 ± 1.7	58.1 ± 0.2			
	16	53.8 ± 8.8	48.7 ± 2.8	48.6 ± 0.1			
	31	36.6 ± 7.9	30.8 ± 2.6	30.8 ± 0.1			

The coupled oscillator data was sampled over $t \in [0, 3]$. The task is to learn a Neural ODE that approximates the dynamics. The runtime, memory usage and final loss obtained is compared for both methods in Table 2.

We see that the reversible method is more that $2\times$ faster on average than recursive checkpointing while using $15.5\times$ less memory. Further, for the same memory usage the reversible method is more than $40\times$ faster. The final loss obtained is equal across both methods (within one standard deviation).

3.2 Time series classification

The reversible method is easily extended to modeling controlled differential equations (CDEs). This follows from the fact that for a differentiable control path the CDE reduces to an ODE with an augmented vector field (Kidger, Morrill, et al. 2020).

Further, we may also apply the reversible method to rough differential equations (RDEs). This follows similarly by the reduction of the RDE to an ODE via the log-ODE method. This is explored in (Morrill, Salvi, et al. 2021) which enables Neural CDE models to classify long time series efficiently.

We therefore perform two time series classification experiments: time series classification on the PhysioNet Sepsis 2019 challenge data (Reyna et al. 2020) using a Neural CDE model and a long time series classification task using a Neural RDE model on the EigenWorms dataset (Bagnall et al. 2018). See Appendix C for details.

Neural CDE on PhysioNet Sepsis We first test the performance of the reversible methods on the PhysioNet Sepsis 2019 challenge dataset (Reyna et al. 2020). This dataset is irregularly sampled and partially observed – a setting where Neural CDEs are most advantageous over discrete-time models.

The training setup follows (Kidger, Morrill, et al. 2020): we consider the first 72 hours of the patient's stay with 34 time-dependent measurements and 5 static features. The task is to predict whether the patient develops sepsis over the course of their stay. Observational intensity is computed as a feature for the model.

Table 3: Sepsis classification results across solvers and adjoint methods (mean $\pm$ std, $n = 5$).

		Euler	Midpoint	Ralston3	RK4
Test AUC (%)	Reversible	87.0 ± 0.2	87.6 ± 0.6	87.6 ± 0.2	87.7 ± 0.3
	Checkpointed	87.6 ± 0.3	87.9 ± 0.2	87.6 ± 0.4	87.7 ± 0.3
Runtime (hrs)	**Reversible**	$\mathbf{0.5 \pm 0.1}$	$\mathbf{0.6 \pm 0.1}$	$\mathbf{0.7 \pm 0.1}$	$\mathbf{0.7 \pm 0.1}$
	Checkpointed	1.1 ± 0.1	1.3 ± 0.1	1.3 ± 0.1	1.3 ± 0.1

Table 4: EigenWorms classification results. Mean $\pm$ std over 5 seeds.

Step	Method	Memory (c)	Accuracy (%)	Time (min)
16	Reversible	**2**	79.0 ± 5.6	$\mathbf{12.0 \pm 0.7}$
	Checkpointed	46	$\mathbf{81.5 \pm 6.1}$	27.9 ± 2.7
64	Reversible	**2**	$\mathbf{79.5 \pm 4.1}$	$\mathbf{3.35 \pm 0.25}$
	Checkpointed	23	$\mathbf{79.5 \pm 6.0}$	7.54 ± 2.24
128	**Reversible**	**2**	$\mathbf{82.6 \pm 3.8}$	$\mathbf{1.46 \pm 0.03}$
	Checkpointed	16	81.0 ± 2.9	3.47 ± 0.45

We compare the reversible backpropagation algorithm to recursive checkpointing for a range of base solvers: Euler, Midpoint, Ralston's 3rd order Runge-Kutta method (Ralston3), and the classic Runge-Kutta 4 method (RK4). For the checkpointed algorithm we choose the number of checkpoints, c, according to the optimal memory-runtime tradeoff which results in $c = 11$.

Performance is measured by AUC on the test set, due to class imbalance. The results can be seen in Table 3. We see that, across all base solvers, the reversible method matches performance of exact checkpointed backpropagation. Notably, the reversible method matches performance while reducing runtime by a factor of $2\times$ on average and memory usage by $5.5\times$.

Neural RDE on EigenWorms Next we investigate the application of reversible solvers to a Neural Rough Differential Equation (RDE) where the log-ODE method is used to derive an approximating ODE. We use the EigenWorms dataset consisting of long time series samples (length 17984) that records 6 channels corresponding to the movement of a roundworm. The task is to classify the worm type from 5 classes (Bagnall et al. 2018).

Following (Morrill, Salvi, et al. 2021), the log-signature is computed for a given depth over each sub-interval of the original time series. Each sub-interval is defined by a step size parameter. For example, a step size of 128 coarsens the original time series by a factor of 128. Full details are given in Appendix C.

The results are shown in Table 4. Across all step sizes the reversible method matches performance of exact backpropagation while running consistently more than $2\times$ faster. For small step sizes the memory usage of the reversible method is $23\times$ less than recursive checkpointing.

3.3 Adaptive step sizes

Lastly, we investigate the performance of the reversible method when using adaptive steps sizes. We consider two experiments: identification of chaotic non-linear dynamics using the (real-system) chaotic double pendulum dataset from (Schmidt et al. 2009) and time series classification on the Speech Commands dataset (Warden 2018).

The reversible adaptive Bosh3 method is compared to recursive checkpointing for a range of checkpoints c (Bogacki et al. 1989). For adaptive step sizes, the optimal number of checkpoints is unknown a-priori; we therefore choose to vary up to $c = 32$ (double pendulum) and $c = 128$ (Speech Commands). This guarantees that we are in the regime $c > \sqrt{n}$ to ensure $O(n \log n)$ runtime.

Table 5: Memory usage, runtime, final loss and number of adaptive solver steps for double pendulum experiment. Mean $\pm$ standard deviation over five repeats.

Method	Memory (c)	Runtime (min)	Loss ($\times 10^{-3}$)	Solver steps (mean)
Reversible	**2**	$\mathbf{21.9 \pm 2.1}$	8.3 ± 3.2	445 ± 14
Recursive	2	818.2 ± 21.5	9.5 ± 2.0	422 ± 7
	4	135.2 ± 7.1	8.6 ± 1.9	$\mathbf{417 \pm 7}$
	8	82.8 ± 1.2	12.8 ± 7.4	434 ± 14
	16	70.8 ± 3.7	$\mathbf{7.8 \pm 1.3}$	432 ± 11
	32	62.4 ± 2.7	7.9 ± 1.6	429 ± 10

Table 6: Speech Commands classification results.

Method	Memory (c)	Solver steps (mean)	Accuracy (%)	Time (hrs)
Checkpointed	16	**695**	**91.3**	10.5
	32	**695**	**91.3**	9.38
	64	**695**	**91.3**	6.27
	128	699	90.0	5.34
Reversible	**2**	706	90.9	**3.28**

Chaotic double pendulum The spatial dynamics of a double pendulum are recorded in (Schmidt et al. 2009). In this experiment, we train a feed-forward Neural ODE to identify the dynamics of the double pendulum system. The memory usage, runtime and best loss achieved is shown in Table 5.

We see from Table 5 that reversible backpropagation is at least $2.9\times$ faster than recursive checkpointing while using $16\times$ less memory. For the same memory usage the reversible method is $37\times$ faster. The final loss is comparable across both methods.

Adaptive Neural CDE on Speech Commands The Speech Commands time series dataset consists of audio recordings of 10 different spoken words (Warden 2018). The task is to classify the word from the recording. The data is preprocessed by calculating the mel-frequency cepstrum coefficients, resulting in a 20 channel time series (Kidger, Morrill, et al. 2020).

The results are shown in Table 6. We see that the reversible method achieves comparable test accuracy to checkpointing. It is also clear to see the memory/runtime trade-off for the checkpointing algorithm; at low memory usage (16 checkpoints – $8\times$ reversible memory usage) the checkpointing algorithm is $3.2\times$ slower than the reversible method and at high memory usage (128 checkpoints – $64\times$ reversible memory usage) checkpointing is $1.6\times$ slower.

Further, it is interesting to see in both adaptive step size experiments that the mean number of steps taken with the reversible solver is comparable to the base solver despite the stability differences.

4 Discussion

4.1 Implementation

An implementation of the reversible solver method was written in JAX (Bradbury et al. 2018). The code is available at `https://github.com/sammccallum/reversible-solvers`. This is part of ongoing work to implement reversible solvers in the popular differential equation solving library Diffrax (Kidger 2022).

4.2 Limitations

The forward solve of the reversible method requires twice the computational cost of the base solver. However, this additional cost is overshadowed by the reduced cost of backpropagation, such that overall the reversible method is significantly faster than recursive checkpointing.

The reversible method can be less numerically stable than the recursive checkpointing algorithm. While the numerical stability of the reversible solvers presented is a strict improvement over previous reversible architectures, stability remains an area to be further improved.

4.3 Extensions

The reversible solvers presented naturally extend to solving stochastic differential equations (SDEs) and semi-discrete partial differential equations (PDEs). For SDEs, each forward solve depends on a set of random variables which must be stored for the backward pass or reconstructed by a virtual Brownian tree algorithm (X. Li et al. 2020; Jelinčič et al. 2024). For PDEs, the discretized state can be memory-intensive for finely discretized domains and as a result the memory usage standard backpropagation often becomes prohibitively expensive. The application of the $O(1)$ memory reversible backpropagation algorithm to PDEs could therefore be advantageous.

Conclusion

A class of reversible ODE solvers was introduced that significantly outperform recursive checkpointing algorithms for training Neural ODEs on both runtime and memory usage. The reversible method presented also strictly improves upon the stability properties and convergence order of previous reversible solvers. Further, the reversible method naturally handles adaptive step sizes and extends to the wider family of neural differential equations.

Broader Impacts

This work introduced a class of reversible numerical solvers for neural differential equations with improved computational complexity. The improved runtime and memory usage has the potential to reduce the energy requirements of learning differential equation models from data. We do not expect any specific negative impacts from this method.

Acknowledgments

Thanks to Patrick Kidger for helpful feedback on an earlier version of the paper and guiding the Diffrax implementation of the reversible solvers. SM was supported by the EPSRC grant EP/S022945/1. JF was supported by the Department of Mathematical Sciences at the University of Bath.

References

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018). "The UEA multivariate time series classification archive, 2018". In: *arXiv preprint arXiv:1811.00075* (cit. on pp. 7, 8).

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2018). "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18.153, pp. 1–43 (cit. on p. 4).

Bhattacharyya, S. P. and Keel, L. H. (1995). "Robust control: the parametric approach". In: *Advances in control education 1994*. Elsevier, pp. 49–52 (cit. on p. 16).

Bogacki, P. and Shampine, L. F. (1989). "A 3 (2) pair of Runge-Kutta formulas". In: *Applied Mathematics Letters* 2.4, pp. 321–325 (cit. on pp. 8, 17).

Boral, A., Wan, Z. Y., Zepeda-Núñez, L., Lottes, J., Wang, Q., Chen, Y.-f., Anderson, J., and Sha, F. (2024). "Neural ideal large eddy simulation: Modeling turbulence with neural stochastic differential equations". In: *Advances in Neural Information Processing Systems* 36 (cit. on p. 1).

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018). *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. URL: `http://github.com/jax-ml/jax` (cit. on p. 9).

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). "Neural ordinary differential equations". In: *Advances in neural information processing systems* 31 (cit. on p. 1).

Chen, X., Araujo, F. A., Riou, M., Torrejon, J., Ravelosona, D., Kang, W., Zhao, W., Grollier, J., and Querlioz, D. (2022). "Forecasting the outcome of spintronic experiments with neural ordinary differential equations". In: *Nature communications* 13.1, p. 1016 (cit. on p. 1).

Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D., and Ho, S. (2020). "Lagrangian Neural Networks". In: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations* (cit. on p. 1).

Dinh, L., Krueger, D., and Bengio, Y. (2015). "NICE: Non-linear Independent Components Estimation". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings* (cit. on p. 3).

Gholami, A., Keutzer, K., and Biros, G. (2019). "ANODE: unconditionally accurate memory-efficient gradients for neural ODEs". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pp. 730–736 (cit. on pp. 2, 18, 19).

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. (2017). "The reversible residual network: Backpropagation without storing activations". In: *Advances in neural information processing systems* 30 (cit. on p. 3).

Greydanus, S., Dzamba, M., and Yosinski, J. (2019). "Hamiltonian neural networks". In: *Advances in neural information processing systems* 32 (cit. on pp. 1, 2).

Griewank, A. (1992). "Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation". In: *Optimization Methods and software* 1.1, pp. 35–54 (cit. on p. 2).

Issa, Z., Horvath, B., Lemercier, M., and Salvi, C. (2024). "Non-adversarial training of Neural SDEs with signature kernel scores". In: *Advances in Neural Information Processing Systems* 36 (cit. on p. 1).

Jelinčič, A., Foster, J., and Kidger, P. (2024). "Single-seed generation of Brownian paths and integrals for adaptive and high order SDE solvers". In: *arXiv preprint arXiv:2405.06464* (cit. on p. 10).

Kidger, P. (2022). "On neural differential equations". In: *PhD thesis, University of Oxford. Diffrax avaiable at* `https://github.com/patrick-kidger/diffrax` (cit. on pp. 1, 9, 18).

Kidger, P., Foster, J., Li, X., and Lyons, T. J. (2021). "Neural sdes as infinite-dimensional gans". In: *International conference on machine learning*. PMLR, pp. 5453–5463 (cit. on p. 1).

Kidger, P., Foster, J., Li, X. C., and Lyons, T. (2021). "Efficient and accurate gradients for neural sdes". In: *Advances in Neural Information Processing Systems* 34, pp. 18747–18761 (cit. on pp. 2, 5, 19).

Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). "Neural controlled differential equations for irregular time series". In: *Advances in Neural Information Processing Systems* 33, pp. 6696–6707 (cit. on pp. 1, 7, 9, 17, 18).

Lee, K. and Parish, E. J. (2021). "Parameterized neural ordinary differential equations: Applications to computational physics problems". In: *Proceedings of the Royal Society A* 477.2253, p. 20210162 (cit. on p. 1).

Li, X., Wong, T.-K. L., Chen, R. T., and Duvenaud, D. (2020). "Scalable gradients for stochastic differential equations". In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 3870–3882 (cit. on pp. 1, 10).

Loshchilov, I. and Hutter, F. (2019). "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations* (cit. on p. 17).

Lu, J., Deng, K., Zhang, X., Liu, G., and Guan, Y. (2021). "Neural-ODE for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens". In: *Iscience* 24.7 (cit. on p. 1).

Morrill, J., Kidger, P., Yang, L., and Lyons, T. (2022). "On the choice of interpolation scheme for neural cdes". In: *Transactions on Machine Learning Research* (cit. on p. 17).

Morrill, J., Salvi, C., Kidger, P., and Foster, J. (2021). "Neural rough differential equations for long time series". In: *International Conference on Machine Learning*. PMLR, pp. 7829–7838 (cit. on pp. 7, 8, 18).

Onken, D. and Ruthotto, L. (2020). "Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows". In: *arXiv preprint arXiv:2005.13420* (cit. on pp. 2, 18, 19).

Portwood, G. D., Mitra, P. P., Ribeiro, M. D., Nguyen, T. M., Nadiga, B. T., Saenz, J. A., Chertkov, M., Garg, A., Anandkumar, A., Dengel, A., et al. (2019). "Turbulence forecasting via Neural ODE". In: *Second Workshop on Machine Learning and the Physical Sciences (NeurIPS)* (cit. on p. 1).

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. (2020). "Universal differential equations for scientific machine learning". In: *arXiv preprint arXiv:2001.04385* (cit. on p. 1).

Reyna, M. A., Josef, C. S., Jeter, R., Shashikumar, S. P., Westover, M. B., Nemati, S., Clifford, G. D., and Sharma, A. (2020). "Early prediction of sepsis from clinical data: the physionet/computing in cardiology challenge 2019". In: *Critical care medicine* 48.2, pp. 210–217 (cit. on p. 7).

Rubanova, Y., Chen, R. T., and Duvenaud, D. K. (2019). "Latent ordinary differential equations for irregularly-sampled time series". In: *Advances in neural information processing systems* 32 (cit. on p. 1).

Schmidt, M. and Lipson, H. (2009). "Distilling free-form natural laws from experimental data". In: *science* 324.5923, pp. 81–85 (cit. on pp. 6, 8, 9, 17).

Söderlind, G. (2003). "Digital Filters in Adaptive Time-Stepping". In: *ACM Transactions on Mathematical Software* 20.1, pp. 1–26 (cit. on p. 17).

Stewart, D. E. (2022). *Numerical Analysis: A Graduate Course*. Vol. 258. Springer (cit. on pp. 4, 5, 13).

Stumm, P. and Walther, A. (2010). "New algorithms for optimal online checkpointing". In: *SIAM Journal on Scientific Computing* 32.2, pp. 836–854 (cit. on p. 2).

Warden, P. (2018). "Speech commands: A dataset for limited-vocabulary speech recognition". In: *arXiv preprint arXiv:1804.03209* (cit. on pp. 8, 9).

Zhuang, J., Dvornek, N. C., Duncan, J. s, et al. (2021). "MALI: A memory efficient and reverse accurate integrator for Neural ODEs". In: *International Conference on Learning Representations* (cit. on pp. 2, 5, 19).

A Proof of convergence (Theorem 2.1)

We will consider the following ODE,

$$\frac{dy}{dt} = f(y), \tag{9}$$
$$y(0) = y_0,$$

where the state (t, y) has been concatenated to $y \in \mathbb{R}^n$ without loss of generality. The vector field is given by the Lipschitz function $f : \mathbb{R}^n \to \mathbb{R}^n$ and the initial condition is $y_0 \in \mathbb{R}^n$.

We start by defining the base solver Ψ and associated error analysis. Throughout, the norm $\|\cdot\|$ corresponds to the Euclidean norm $\|\cdot\|_2$.

Definition A.1. For $h_{\max} > 0$, consider the map Ψ given by

$$\Psi : [-h_{\max}, h_{\max}] \times \mathbb{R}^n \to \mathbb{R}^n,$$
$$(h, x) \mapsto \Psi_h(x).$$

We say that Ψ is an ODE solver with order $\alpha > 0$ if there exists a constant $C_1 > 0$ such that

$$\big\|x(h) - \big(x + \Psi_h(x)\big)\big\| \leq C_1 |h|^{\alpha+1}, \tag{10}$$

for $h \in [-h_{\max}, h_{\max}]$ where $x(h)$ denotes the solution $y(t)$ at time $t = |h|$ of the ODE:

$$y' = \operatorname{sgn}(h) \cdot f(y),$$
$$y(0) = x.$$

Definition A.2. Let Ψ denote an order α solver for an ODE governed by a Lipschitz vector field $f : \mathbb{R}^n \to \mathbb{R}^n$. We say that Ψ is an ODE solver satisfying the Lipschitz condition if there exists a constant $C_2 > 0$ such that

$$\big\|\Psi_h(x) - \Psi_h(y)\big\| \leq C_2 |h| \big\|x - y\big\|, \tag{11}$$

for all $x, y \in \mathbb{R}^n$ and $h \in [-h_{\max}, h_{\max}]$. In particular, this may be written as

$$\big\|\Psi_h\big\|_{\text{Lip-1}} \leq C_2 |h|. \tag{12}$$

We note that the Lipschitz condition defined in A.2 is true for all explicit Runge-Kutta methods; this result is shown in Section 6.1.4.2 of (Stewart 2022).

Lemma A.3. *Let Ψ denote an order α ODE solver satisfying the Lipschitz condition. Then there exists a constant $C_3 > 0$ such that*

$$\big\|\Psi_h(x + \Psi_{-h}(x)) + \Psi_{-h}(x)\big\| \leq C_3 |h|^{\alpha+1},$$

for $h \in [-h_{\max}, h_{\max}]$.

Proof. By definition A.1, A.2 and the triangle inequality, we have

$$\begin{aligned}
\big\|\Psi_h(x + \Psi_{-h}(x)) + \Psi_{-h}(x)\big\| &= \big\|\big(x(-h) + \Psi_h(x(-h)) - x\big) + \big(x + \Psi_{-h}(x) - x(-h)\big) \\
&\qquad + \Psi_h(x + \Psi_{-h}(x)) - \Psi_h(x(-h))\big\| \\
&\leq \big\|x(-h) + \Psi_h(x(-h)) - x\big\| + \big\|x + \Psi_{-h}(x) - x(-h)\big\| \\
&\qquad + \big\|\Psi_h(x + \Psi_{-h}(x)) - \Psi_h(x(-h))\big\| \\
&\leq 2C_1 |h|^{\alpha+1} + C_2 |h| \big\|x + \Psi_{-h}(x) - x(-h)\big\| \\
&\leq \underbrace{C_1\big(2 + C_2\, h_{\max}\big)}_{=: C_3} |h|^{\alpha+1}.
\end{aligned}$$

□

Next, we bound the error between the reversible solver states in lemma A.4.

Lemma A.4. *For a fixed time horizon $T > 0$, we consider the ODE (9) over the interval $[0, T]$. Suppose that $T = Nh$ were $N > 0$ denote the number of steps and $h > 0$ denotes the step size. Consider the reversible numerical solution $\{(y_k, z_k)\}_{0 \le k \le N}$ with $y_0 = z_0 = y(0)$ and, for $k \ge 0$,*

$$y_{k+1} := \lambda y_k + (1-\lambda) z_k + \Psi_h(z_k),$$
$$z_{k+1} := z_k - \Psi_{-h}(y_{k+1}),$$

where $\lambda \in (0, 1]$ and Ψ denotes an order α ODE solver satisfying the Lipschitz condition. Then there exists $h_{\max}, C_4 > 0$ such that

$$\|y_k - z_k\| \le C_4 h^\alpha \,, \tag{13}$$

for $h \in (0, h_{\max}]$ and $0 \le k \le N$.

Proof. Letting $E_k^{yz} := \|y_k - z_k\|$, it follows from definition A.2 and lemma A.3 that

$$\begin{aligned}
E_{k+1}^{yz} &= \|y_{k+1} - z_{k+1}\| \\
&= \big\|\lambda\big(y_k - z_k\big) + \Psi_h(z_k) + \Psi_{-h}(y_{k+1})\big\| \\
&= \big\|\lambda\big(y_k - z_k\big) + \Psi_h(z_k) + \Psi_{-h}(z_{k+1}) + \Psi_{-h}(y_{k+1}) - \Psi_{-h}(z_{k+1})\big\| \\
&\le \lambda\|y_k - z_k\| + \big\|\Psi_h(z_k) + \Psi_{-h}(z_{k+1})\big\| + \big\|\Psi_{-h}(y_{k+1}) - \Psi_{-h}(z_{k+1})\big\| \\
&\le \lambda\|y_k - z_k\| + \big\|\Psi_h(z_{k+1} + \Psi_{-h}(y_{k+1})) + \Psi_{-h}(z_{k+1})\big\| + \|\Psi_{-h}\|_{\text{Lip-1}}\big\|y_{k+1} - z_{k+1}\big\| \\
&\le \lambda\|y_k - z_k\| + \big\|\Psi_h(z_{k+1} + \Psi_{-h}(z_{k+1})) + \Psi_{-h}(z_{k+1})\big\| + C_2\, h\big\|y_{k+1} - z_{k+1}\big\| \\
&\qquad + \big\|\Psi_h(z_{k+1} + \Psi_{-h}(y_{k+1})) - \Psi_h(z_{k+1} + \Psi_{-h}(z_{k+1}))\big\| \\
&\le \lambda E_k^{yz} + C_3 h^{\alpha+1} + C_2 h E_{k+1}^{yz} + C_2^2 h^2 E_{k+1}^{yz} \,.
\end{aligned}$$

Therefore, we have the following inequality,

$$E_{k+1}^{yz} \le \frac{1}{1 - C_2 h - C_2^2 h^2}\big(\lambda E_k^{yz} + C_3 h^{\alpha+1}\big),$$

provided $C_2 h < \frac{1+\sqrt{5}}{2}$. Note that $(1 - x - x^2)^{-1} \le 16^x$ for $x \in [0, \frac{1}{2}]$. Hence, for $h \le \frac{1}{2C_2}$,

$$E_{k+1}^{yz} \le 16^{C_2 h}\big(\lambda E_k^{yz} + C_3 h^{\alpha+1}\big).$$

Applying the above inequality k times and using $E_0^{yz} = 0$, we have

$$\begin{aligned}
E_k^{yz} &\le 16^{C_2 h} C_3\, h^{\alpha+1} \frac{16^{C_2 k h} - 1}{16^{C_2 h} - 1} \\
&= C_3 \frac{16^{C_2 h}}{\log(16) C_2}\big(16^{C_2 k h} - 1\big) \frac{\log(16) C_2 h}{e^{\log(16) C_2 h} - 1} h^\alpha \\
&\le \underbrace{C_3 \frac{16^{C_2 h}}{\log(16) C_2}\big(16^{C_2 T} - 1\big)}_{=: C_4} h^\alpha,
\end{aligned}$$

where we used $kh \le T$ and the inequality $1 + x \le e^x$. □

Bringing the above analysis together, we now prove Theorem 2.1.

Theorem. *For a fixed time horizon $T > 0$, we consider the ODE (9) over the interval $[0, T]$. Let $T = Nh$ where $N > 0$ denotes the number of steps and $h > 0$ is the step size. Consider the reversible numerical solution $\{(y_k, z_k)\}_{0 \le k \le N}$ with $y_0 = z_0 = y(0)$ and, for $k \ge 0$,*

$$y_{k+1} := \lambda y_k + (1-\lambda) z_k + \Psi_h(z_k), \tag{14}$$
$$z_{k+1} := z_k - \Psi_{-h}(y_{k+1}), \tag{15}$$

where $\lambda \in (0, 1]$ and Ψ denotes an order α ODE solver satisfying the Lipschitz condition. Then there exist constants $h_{\max}, C_5 > 0$ such that, for $h \in (0, h_{\max}]$,

$$\|y_k - y(t_k)\| \le C_5 h^\alpha \,, \tag{16}$$

for $0 \le k \le N$ where $t_k := kh$.

Proof. Let $x_k := \lambda^{N-k} y_k + (1-\lambda^{N-k}) z_k$ and consider the error $E_k := \|x_k - y(t_k)\|$. Then, by the triangle inequality and previous lemmas, we have

$$
\begin{aligned}
E_{k+1} &= \big\|\lambda^{N-(k+1)} y_{k+1} + (1-\lambda^{N-(k+1)}) z_{n+1} - y(t_{k+1})\big\| \\
&= \big\|\lambda^{N-(k+1)}\big(\lambda y_k + (1-\lambda) z_k + \Psi_h(z_k)\big) + (1-\lambda^{N-(k+1)})\big(z_k - \Psi_{-h}(y_{k+1})\big) - y(t_{k+1})\big\| \\
&= \big\|\lambda^{N-k} y_k + (1-\lambda^{N-k}) z_k + \lambda^{N-(k+1)}\Psi_h(z_k) - (1-\lambda^{N-(k+1)})\big(\Psi_{-h}(y_{k+1})\big) - y(t_{k+1})\big\| \\
&\le E_k + \big\| y(t_k) + \lambda^{N-(k+1)}\Psi_h(z_k) - (1-\lambda^{N-(k+1)})\big(\Psi_{-h}(y_{k+1})\big) - y(t_{k+1})\big\| \\
&\le E_k + \lambda^{N-(k+1)}\big\|\Psi_h(z_k) + \Psi_{-h}(y_{k+1})\big\| + \big\|y(t_k) - \big(y(t_{k+1}) + \Psi_{-h}(y_{k+1})\big)\big\| \\
&\le E_k + \lambda^{N-(k+1)}\big\|\Psi_h(z_{k+1} + \Psi_{-h}(y_{k+1})) + \Psi_{-h}(y_{k+1})\big\| \\
&\qquad + \big\|y(t_k) - \big(y(t_{k+1}) + \Psi_{-h}(y(t_{k+1}))\big)\big\| + \big\|\Psi_{-h}(y(t_{k+1})) - \Psi_{-h}(y_{k+1})\big\| \\
&\le E_k + \lambda^{N-(k+1)}\big\|\Psi_h(y_{k+1} + \Psi_{-h}(y_{k+1})) + \Psi_{-h}(y_{k+1})\big\| \\
&\qquad + \lambda^{N-(k+1)}\big\|\Psi_h(z_{k+1} + \Psi_{-h}(y_{k+1})) - \Psi_h(y_{k+1} + \Psi_{-h}(y_{k+1}))\big\| \\
&\qquad + C_1 h^{\alpha+1} + \big\|\Psi_{-h}(y(t_{k+1})) - \Psi_{-h}(x_{k+1})\big\| + \big\|\Psi_{-h}(x_{k+1}) - \Psi_{-h}(y_{k+1})\big\| \\
&\le E_k + \lambda^{N-(k+1)} C_3 h^{\alpha+1} + \lambda^{N-(k+1)} C_2 h \|z_{k+1} - y_{k+1}\| \\
&\qquad + C_1 h^{\alpha+1} + C_2 h\, E_{k+1} + C_2 h \|x_{k+1} - y_{k+1}\| \\
&\le E_k + \lambda^{N-(k+1)} C_3 h^{\alpha+1} + \lambda^{N-(k+1)} C_2 h \|y_{k+1} - z_{n+1}\| \\
&\qquad + C_1 h^{\alpha+1} + C_2 h\, E_{k+1} + (1-\lambda^{N-(k+1)}) C_2 h \|y_{k+1} - z_{n+1}\| \\
&\le E_k + C_2 h\, E_{k+1} + (C_1 + C_2 C_4 + C_3) h^{\alpha+1}.
\end{aligned}
$$

Therefore, provided $C_2 h < 1$,

$$
E_{k+1} \le \frac{1}{1 - C_2 h} E_k + \frac{C_1 + C_2 C_4 + C_3}{1 - C_2 h} h^{\alpha+1}.
$$

Since $(1-x)^{-1} \le 1 + 2x \le e^{2x}$ for $x \in [0, \frac{1}{2}]$, we have for $h \le \frac{1}{2C_2}$,

$$
E_{k+1} \le e^{2C_2 h} E_k + (C_1 + C_2 C_4 + C_3)(1 + 2C_2 h) h^{\alpha+1}.
$$

Applying the above inequality k times and using $E_0 = 0$, we have

$$
\begin{aligned}
E_k &\le (C_1 + C_2 C_4 + C_3)(1 + 2C_2 h) h^{\alpha+1} \frac{e^{2C_2 k h} - 1}{e^{2C_2 h} - 1} \\
&= (C_1 + C_2 C_4 + C_3)(1 + 2C_2 h) \frac{e^{2C_2 k h} - 1}{2C_2} \frac{2C_2 h}{e^{2C_2 h} - 1} h^{\alpha} \\
&\le (C_1 + C_2 C_4 + C_3)(1 + 2C_2 h_{\max}) \frac{e^{2C_2 T} - 1}{2C_2} h^{\alpha},
\end{aligned}
$$

where we used $kh \le T$ and the inequality $1 + x \le e^x$. Therefore

$$
\begin{aligned}
\|y_k - y(t_k)\| &\le \|y_k - x_k\| + \|x_k - y(t_k)\| \\
&= E_k + (1 - \lambda^{N-k}) \|y_k - z_k\| \\
&\le \underbrace{\left((C_1 + C_2 C_4 + C_3)(1 + 2C_2 h_{\max}) \frac{e^{2C_2 T} - 1}{2C_2} + C_4 \right)}_{=: C_5} h^{\alpha}.
\end{aligned}
$$

□

B Proof of stability (Theorem 2.3)

For convenience we provide the description of Theorem 2.3 here.

Theorem. *Let Ψ be given by an explicit Runge-Kutta solver. Then the reversible numerical solution $\{y_n, z_n\}_{n\geq 0}$ given by*

$$\begin{aligned} y_{n+1} &= \lambda y_n + (1-\lambda) z_n + \Psi_h(t_n, z_n), \\ z_{n+1} &= z_n - \Psi_{-h}(t_{n+1}, y_{n+1}), \end{aligned}$$

is linearly stable iff

$$|\Gamma| < 1 + \lambda,$$

where

$$\Gamma = 1 + \lambda - (1-\lambda) R(-h\alpha) - R(-h\alpha) R(h\alpha).$$

Proof. Firstly, we re-write the reversible method applied to the linear test problem (definition 2.2) using the Runge-Kutta transfer function. We get

$$\begin{aligned} y_{n+1} &= \lambda y_n + (1-\lambda) z_n + R(h\alpha) z_n, \\ z_{n+1} &= z_n - R(-h\alpha) y_{n+1}. \end{aligned}$$

Writing the coupled system as a matrix-vector product results in

$$\begin{pmatrix} y_{n+1} \\ z_{n+1} \end{pmatrix} = \begin{pmatrix} \lambda & 1 - \lambda + R(h\alpha) \\ -\lambda R(-h\alpha) & 1 + (1-\lambda) R(h\alpha) - R(-h\alpha) R(h\alpha) \end{pmatrix} \begin{pmatrix} y_n \\ z_n \end{pmatrix}.$$

For stability we require that the eigenvalues of the coupling matrix are by magnitude less than one. The eigenvalues are given by the characteristic equation,

$$P(e_k) = e_k^2 - \left(1 + \lambda - (1-\lambda) R(-h\alpha) - R(-h\alpha) R(h\alpha)\right) e_k + \lambda = 0.$$

Routh-Hurwitz test

The Routh-Hurwitz test is a condition on the coefficients of a polynomial such that the roots lie in the left half complex plane. Instead we require that the roots lie in the unit disc. Therefore, we apply the following map to the characteristic equation $P(e_k)$,

$$Q(e_k) = (e_k - 1)^2 P\left(\frac{e_k + 1}{e_k - 1}\right),$$

such that if Q is Routh-Hurwitz stable, then the roots of P lie in the open unit disc (referred to as Schur stable) (Bhattacharyya et al. 1995).

The polynomial Q obtained by the transformation is given by

$$Q(e_k) = (1 + \lambda - \Gamma) e_k^2 + 2(1-\lambda) e_k + (1 + \lambda + \Gamma) = 0,$$

where

$$\Gamma = 1 + \lambda - (1-\lambda) R(-h\alpha) - R(-h\alpha) R(h\alpha).$$

We proceed to check when Q is Routh-Hurwitz stable. For quadratic polynomials, the Routh-Hurwitz test reduces to checking that the coefficients of Q are all positive. Therefore, for stability, we require

$$\begin{aligned} 1 + \lambda - \Gamma &> 0 \qquad \text{(i)} \\ 2(1-\lambda) &> 0 \qquad \text{(ii)} \\ 1 + \lambda + \Gamma &> 0 \qquad \text{(iii)} \end{aligned}$$

Condition (ii) is immediately satisfied by the definition of $\lambda \in (0, 1)$. Condition (i) and (iii) may be combined into the condition

$$|\Gamma| < 1 + \lambda.$$

□

C Experimental details

All experiments were run on one Dual NVIDIA RTX A6000, 48 GB.

For all ODE experiments the Neural ODE model is defined with a small feed-forward neural network vector field with two layers and hidden size 10. The activation function used was $\tanh$. For the CDE/RDE experiments the model was significantly larger, see Table 7. Each training run was performed for multiple repeats with random network initializations.

The reversible solver was initialized with coupling parameter $\lambda = 0.99$.

C.1 Chandrasekhar's White Dwarf Equation

Training data was generated by simulating the white dwarf system over $r \in [0, 5]$ with $C = 0.001$ for 1000 time steps. The Neural ODE model was trained to minimise mean squared error over 1000 time steps. The number of training steps was 1000 and the optimizer was AdamW with learning rate 10^{-2} and weight decay 10^{-5} (Loshchilov et al. 2019).

C.2 Coupled oscillators

The experimental coupled oscillator data from (Schmidt et al. 2009) was linearly interpolated, sampled at 500 time steps over $t \in [0, 3]$ and normalized. The Neural ODE model was trained to minimise mean squared error over 500 time steps. The number of training steps was 10000 and the optimizer was AdamW with learning rate 10^{-2} and weight decay 10^{-5}.

C.3 Chaotic double pendulum

The experimental double pendulum data from (Schmidt et al. 2009) was linearly interpolated, sampled at 500 time steps over $t \in [0, 2]$ and normalized. The Neural ODE model was trained to minimise mean squared error over an adaptively selected number of time steps. The number of training steps was 10000 and the optimizer was AdamW with learning rate 10^{-2} and weight decay 10^{-5}.

The adaptive time stepping algorithm was a PID controller from (Söderlind 2003) with error estimates provided by the embedded second order method from Bogacki-Shampine (Bogacki et al. 1989). The PID controller absolute and relative error tolerances were 10^{-6}.

C.4 Time series classification

The Neural CDE/RDE models are given by the Neural ODE in (1) but with the vector field replaced with the augmented field $g_{\theta,X}(t, y_t)$ defined below.

Neural CDEs For the Neural CDE experiments, we define the model following (Kidger, Morrill, et al. 2020). Consider a continuous control path $X_t : \mathbb{R} \to \mathbb{R}^{d_x}$ of bounded variation, then we define the Neural CDE as

$$dy_t = f_\theta(t, y(t))dX_t, \quad y(0) = y_0, \tag{17}$$

where $y_t \in \mathbb{R}^{d_y}$ denotes the hidden state, $f_\theta : \mathbb{R} \times \mathbb{R}^{d_y} \to \mathbb{R}^{d_y} \times \mathbb{R}^{d_x}$ is the (matrix-valued) vector field and $y_0 = \zeta_\theta(t_0, X_0)$ is the initial condition taken as a learnt map from the initial condition of the control path.

Since the control path is differentiable, then the CDE model can be reduced to an ODE via

$$y_t = y_0 + \int_0^t f_\theta(u, y_u)dX_u = y_0 + \int_0^t f_\theta(u, y_u)\frac{dX_u}{du}du = y_0 + \int_0^t g_{\theta,X}(u, y_u)du, \tag{18}$$

where the derivative of the control path has been subsumed into the vector field $g_{\theta,X}(t, y_t)$. To compute the interpolation of the path X_t, we used cubic Hermite splines with backward differences (Morrill, Kidger, et al. 2022).

Table 7: Model hyperparameters for each experiment. NCDE = Neural CDE; NRDE = Neural RDE.

	Sepsis	Speech Commands	EigenWorms
Model class	NCDE	NCDE	NRDE
Parameter count	322K	96.5K	1.22M
Hidden size h	64	90	64
Vector field width	64	40	128
Vector field depth	4	4	4
Vector field activation	ReLU	ReLU	ReLU
Vector field final activation	tanh	tanh	tanh
Initial MLP width	256	40	128
Initial MLP depth	1	4	1
Base solver	All	Bosh3	RK4
Batch size	1024	1024	181
Epochs	200	200	200

Neural RDEs For the Neural RDE experiment, we define the model following (Morrill, Salvi, et al. 2021). Specifically, the approximating ODE over a sub-interval $[r_i, r_{i+1}] \subset [0, T]$ is obtained by taking

$$g_{\theta,X}(t, y_t) = f_\theta(t, y_t) \frac{\log(S^N(X_t)_{[r_i, r_{i+1}]})}{r_{i+1} - r_i}, \tag{19}$$

where $S^N(X_t)_{[r_i, r_{i+1}]}$ is the depth-N signature over the sub-interval. We take $N = 3$ for the EigenWorms experiment.

The model hyperparameters for each experiment are given in Table 7. Each experiment used the Adam optimizer with default parameters. The PhysioNet sepsis learning rate schedule was a cosine decay with initial value 3.2×10^{-3}. The Speech Commands experiment used a reduce-on-plateau schedule with initial value 1.6×10^{-3}, dividing the learning rate by 10 if the loss did not improve for 10 epochs. The EigenWorms experiment used a reduce-on-plateau schedule with initial value 1×10^{-3}, dividing the learning rate by 10 if the loss did not improve for 15 epochs.

The PhysioNet sepsis model was trained with a weighted binary cross-entropy loss with weight $w = 10$ on positive examples and L^2 regularization with a large 0.03 weight following (Kidger, Morrill, et al. 2020). Both Speech Commands and EigenWorms models were trained with categorical cross entropy. The Speech Commands loss included L^2 regularization with weight 10^{-4}.

All accuracy results reported were evaluated on the test set. The train/val/test split for each dataset was 70/15/15.

D Additional results

D.1 Comparison to continuous adjoint method

The continuous adjoint method derives an adjoint differential equation, where the solution is the gradient of a given loss function. To obtain the gradient, one must therefore numerically integrate the adjoint system during backpropagation. This is often referred to as optimize-then-discretize (Gholami et al. 2019; Kidger 2022; Onken et al. 2020). This method achieves a linear-in-time $O(n)$ and constant-in-memory $O(1)$ backpropagation algorithm.

However, due to numerical integration, the gradient obtained is an approximation to the exact gradient obtained by standard automatic differentiation through the discrete forward graph. This approximation can often lead to training instability (Gholami et al. 2019; Onken et al. 2020). We include a characterization of this instability here on the PhysioNet sepsis dataset and demonstrate that, in contrast, the exact gradient obtained by the reversible solver results in stable training dynamics.

We use the training setup described in Section C.4 and select Euler for the base solver. The continuous adjoint method, recursive checkpointing and reversible backpropagation algorithms are applied with all other training settings fixed. The training dynamics for each method are shown in Figure 4.

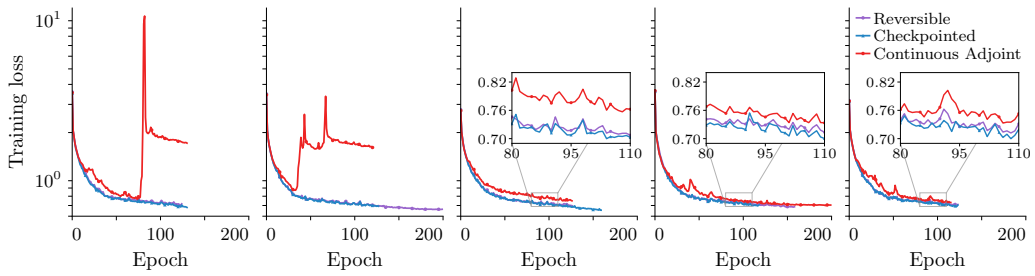


Figure 4: Training loss for each backpropagation method on the PhysioNet Sepsis classification task.

We see that the reversible method obtains similar training dynamics to checkpointed backpropagation, as expected due to gradient exactness. The continuous adjoint method results in unstable loss spiking, with two of five repeats producing diverged training losses larger than the initial loss. This replicates the instability discussed in Gholami et al. 2019 and Onken et al. 2020.

D.2 Comparison to previous reversible solvers

There have been two previous reversible solver constructions presented: Zhuang et al. 2021 propose the Asynchronous Leapfrog method and Kidger, Foster, X. C. Li, et al. 2021 propose a reversible Heun method. The main limitation of these previous solvers is low-order convergence and poor stability properties.

Theorem 2.3 shows that the coupling parameter between the joint state (y_n, z_n) is essential for stability. We understand this to result from the contractivity of the interpolation $\lambda y_n + (1-\lambda)z_n$ which forces the two states to remain close during integration. This coupling is absent from Asynchronous Leapfrog and Reversible Heun and therefore the two states are able to diverge during integration.

To empirically compare the reversible solver presented in this work to Asynchronous Leapfrog and Reversible Heun, we compute the integration error on the linear stability test problem. This isolates the improved convergence and stability properties directly; see Figure 5. Note that on this linear ODE, the Asynchronous Leapfrog and Reversible Heun updates coincide.

We see that at small step sizes h the Reversible RK4 method achieves an integration mean-absolute-error five orders of magnitude lower than Asynchronous Leapfrog and Reversible Heun. At $h = 0.17$, Asynchronous Leapfrog and Reversible Heun become unstable with a relative error passing 1.0, while the Reversible RK4 method remains stable with an integration error below 10^{-7}. The Reversible RK4 method does not become unstable until $h = 0.5$, at which point Asynchronous Leapfrog and Reversible Heun display a 10^2 relative absolute error.

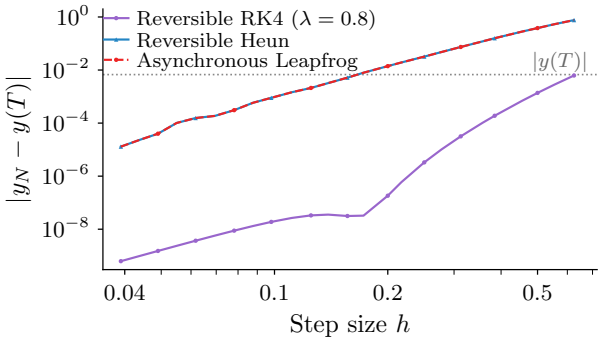


Figure 5: Mean-absolute-error of integration on the linear stability ODE for Reversible RK4, Reversible Heun and Asynchronous Leapfrog methods.

Table 8: Training step runtime breakdown between the forward solve and backpropagation for recursive checkpointing and the Reversible method using RK4 base solver.

Method	Forward solve (s)	Backpropagation (s)	Full train step (s)
Checkpointing	$\mathbf{0.0516 \pm 0.0007}$	1.5712 ± 0.0009	1.6239 ± 0.0005
Reversible	0.0901 ± 0.0001	$\mathbf{0.5971 \pm 0.0006}$	$\mathbf{0.6899 \pm 0.0005}$

Table 9: Training step runtime as a function of network size for recursive checkpointing and the Reversible method using RK4 base solver.

Width	Params	Checkpointed (s/step)	Reversible (s/step)	Speedup
128	1.2M	1.62	0.69	$2.36\times$
256	2.5M	3.13	1.16	$2.70\times$
512	5.4M	6.42	2.14	$2.99\times$
1024	12.4M	14.36	4.58	$3.13\times$

D.3 Computational results

In this section we report additional computational results obtained for the EigenWorms experiment.

Forward / backward breakdown. The forward solve of the reversible solver has increased runtime compared to the base solver due to evaluating the vector field twice per step but decreased backpropagation runtime compared to checkpointed backpropagation. We measure this runtime breakdown on the EigenWorms dataset; see Table 8. We see that the mean training step forward cost is $1.8\times$ higher than the base solver, as expected. While the backpropagation cost is $2.6\times$ faster than checkpointing.

Scaling network size. Since the runtime and memory complexity of the backpropagation methods are independent of the single-step / network cost, the runtime and memory improvements of the reversible method should hold for larger vector fields. To numerically verify this, we ablated the width size of the Neural RDE for the EigenWorms experiment and measured the training step runtime; see Table 9. We see that the runtime advantage holds as the network size increases.

Comparison to non-checkpointed backpropagation. Standard (non-checkpointed) backpropagation through the forward-solve graph stores the state at every timestep. This is equivalent to recursive checkpointing with checkpoints c equal to the number of solver steps. We measure the performance of this algorithm on the EigenWorms dataset; see Table 10. We see that for a memory cost $70\times, 140\times, 562\times$ larger than the reversible backpropagation algorithm (respectively for each step size), standard backpropagation obtains an average $1.3\times$ runtime improvement. The final test accuracy obtained is comparable across all methods. This large memory cost for small runtime improvement demonstrates why checkpointed algorithms form the default backpropagation method for differential equations.

Table 10: EigenWorms classification results. Mean $\pm$ std over 5 seeds.

Step	Method	Memory (c)	Accuracy (%)	Time (min)
16	Reversible	**2**	79.0 ± 5.6	12.0 ± 0.7
	Checkpointed	46	$\mathbf{81.5 \pm 6.1}$	27.9 ± 2.7
	Standard	1124	78.5 ± 3.5	$\mathbf{8.67 \pm 0.55}$
64	Reversible	**2**	$\mathbf{79.5 \pm 4.1}$	3.35 ± 0.25
	Checkpointed	23	$\mathbf{79.5 \pm 6.0}$	7.54 ± 2.24
	Standard	281	78.5 ± 4.8	$\mathbf{2.58 \pm 0.62}$
128	Reversible	**2**	$\mathbf{82.6 \pm 3.8}$	1.46 ± 0.03
	Checkpointed	16	81.0 ± 2.9	3.47 ± 0.45
	Standard	140	81.0 ± 3.5	$\mathbf{1.14 \pm 0.08}$

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

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

   Answer: [Yes]

   Justification: The statements made in the abstract are either proved or experimentally shown in the main text.

   Guidelines:

   - The answer [N/A] 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 [N/A] 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: See section 4.2.

Guidelines:

- The answer [N/A] 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: The proofs are provided in the Appendix and any assumptions required to complete the proof are stated.

   Guidelines:

   - The answer [N/A] 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: See Appendix C.

   Guidelines:

- The answer [N/A] 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?

   Answer: [Yes]

   Justification: All code to reproduce the experiments has been provided as supplementary material.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.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://neurips.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) necessary to understand the results?

   Answer: [Yes]

   Justification: See Appendix C.

   Guidelines:

   - The answer [N/A] 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: [Yes]

   Justification: All experiments contain error bars obtained over multiple repeats of each experiment. The adaptive step size experiment on Speech Commands does not contain error bars as it was too computationally expensive to achieve.

   Guidelines:

   - The answer [N/A] 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: See Appendix C.

   Guidelines:

   - The answer [N/A] 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:

   Guidelines:

   - The answer [N/A] 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: [Yes]

    Justification: See Broader Impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] 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.
    - 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., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

Justification: Data and models are open-source.

Guidelines:

- The answer [N/A] 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: All code and data used is appropriately credited. Does not use models from third parties.

    Guidelines:

    - The answer [N/A] 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: [N/A]

    Justification: No new assets.

    Guidelines:

    - The answer [N/A] 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: [N/A]

Justification: No human subjects.

Guidelines:

- The answer [N/A] 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: [N/A]

Justification: No human subjects.

Guidelines:

- The answer [N/A] 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.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: This work does not use LLMs in any important, original, or non-standard component.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.