SOLVER-SKIPPING BACKPROPAGATION: EFFICIENT TRAINING OF CONTINUOUS-TIME NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Continuous-time neural networks (CTNNs) provide an expressive and flexible framework for learning and modeling complex dynamics. However, computing gradients associated with ODE-solver-based state evolution introduces substantial computational and memory overhead, making training a key bottleneck. In this work, we propose Solver-Skipping Backpropagation (SSBP), an efficient CTNN training method that keeps forward integration unchanged while bypassing backpropagation through the solver's internal steps by employing a one-step gradient approximation constructed from the integration interval's initial state, thereby substantially reducing computational and memory costs during training. Relative to conventional backpropagation through time (BPTT), this solver-level decoupling reduces backward-pass computational and memory costs from $\mathcal{O}(Ls)$ to $\mathcal{O}(1)$, where Ls denotes the number of function evaluations in the forward solve. We provide a theoretical explanation for the effectiveness of SSBP by characterizing how its update direction approximates the BPTT gradient. Experiments across diverse CTNN models, tasks, and solvers show that SSBP achieves performance comparable to conventional BPTT while substantially reducing training time and memory usage. Numerical experiments on a spin-torque oscillator, a typical neuromorphic device, further demonstrate its effectiveness and practical utility.

1 INTRODUCTION

Continuous-time neural networks (CTNNs), formulated as ordinary differential equations (ODEs) (Chen et al., 2018), transform the discrete layer-wise mappings of static neural networks or the discrete state updates of recurrent neural networks (RNNs) into continuous state evolution. By parameterizing ODE vector fields with deep neural networks, CTNNs provide an expressive framework for learning and modeling complex dynamics. With their continuous and flexible representation of dynamics, CTNNs are well suited to sequential and irregularly sampled data (Rubanova et al., 2019; Lechner & Hasani, 2020; Kidger et al., 2020) and have shown promise in generative modeling (Grathwohl et al., 2019), scientific computing (Yin et al., 2023; von Bassewitz et al., 2026), as well as sequential decision-making and control (Alvarez et al., 2020; Ghanem & Finkbeiner, 2026).

In practice, the continuous state evolution of CTNNs is typically computed using numerical ODE solvers. To advance the state over a time interval, ODE solvers repeatedly compute state derivatives defined by the model's vector field and use them to update the state through multiple internal integration steps. Training these models, accordingly, requires taking into account the gradients associated with the state evolution within the ODE solver during backpropagation. Existing works commonly follow one of two strategies for propagating gradients through ODE solvers (Onken & Ruthotto, 2020; Kidger, 2022): conventional backpropagation through time (BPTT) (Werbos, 1990) propagates gradients directly through their discrete operations, whereas the adjoint sensitivity method constructs additional adjoint ODEs and solves them backward in time (Chen et al., 2018) (Fig. 1). Of the two, BPTT yields gradients consistent with the discretized forward solve, but requires storing intermediate solver states and the associated computation graph, resulting in substantial memory usage and computational cost, both of which increase with the number of internal integration steps. The adjoint sensitivity method avoids storing the complete computation graph but requires additional backward integration of the adjoint ODEs, potentially involving a large number of function

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

evaluations and considerable runtime. Although the two strategies offer different trade-offs between memory and runtime, both incur substantial solver-related overhead during the backward pass, making training a key bottleneck to the broader adoption of CTNNs.

To alleviate this bottleneck, various approaches have been proposed to improve training efficiency. Some methods seek to reduce the overhead of forward solves, including those that constrain the complexity of the learned dynamics (Kelly et al., 2020; Finlay et al., 2020; Pal et al., 2021) and those that design more efficient numerical integrators (Djeumou et al., 2022; Poli et al., 2020). Others focus on reducing memory consumption during the backward pass while aiming to preserve gradient accuracy, employing techniques such as checkpointing and recomputation (Gholami et al., 2019; Zhuang et al., 2020), as well as integrators designed for memory-efficient gradient computation (Zhuang et al., 2021; Matsubara et al., 2021). Another line of work reformulates the training problem to reduce or avoid differentiation through ODE solvers (Quaglino et al., 2020; Rodriguez et al., 2022; Zhao et al., 2025). Across these approaches, forward and backward processes remain tightly coupled: the backward pass is formulated in accordance with the corresponding forward process, either through direct differentiation or through recomputation to preserve their consistency. By contrast, in a broader context, studies of other models with evolving states have explored the use of time-local or state-local gradient approximations to avoid propagating error signals along the entire state trajectory. For example, temporally local surrogate gradients have been investigated for efficient training of spiking neural networks (SNNs) (Xiao et al., 2022; Meng et al., 2023). Update signals constructed solely from the initial state have also been studied in autonomous state-propagating reservoir computing (Zhang et al., 2026). Similarly, state-local gradient approximations based on selected discretized or equilibrium states have been adopted in hybrid numerical modeling systems (Ouala et al., 2024) and implicit models (Geng et al., 2021). Together, these studies suggest that effective training does not necessarily require tight, step-by-step coupling between the forward and backward processes; useful update signals can be constructed by approximating gradients from local information. This invites consideration of CTNN training: it may not be necessary for all internal solver steps performed during forward integration to participate in backpropagation. This would decouple the backward pass from the solver's internal forward integration, potentially further reducing training time and memory costs.

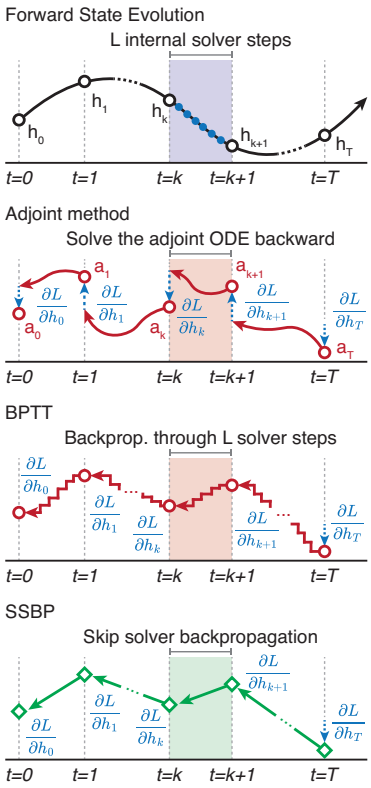

Figure 1: State evolution of CTNNs and the corresponding gradient backpropagation via the adjoint method, BPTT, and our SSBP. SSBP replaces L-step solver backpropagation with a one-step gradient approximation.

Motivated by this insight, we propose Solver-Skipping Backpropagation (SSBP), a method that uses a gradient approximation based on state-local information to bypass backpropagation through the solver, thereby further improving the training efficiency of CTNNs (Fig. 1). SSBP preserves the complete forward integration performed by ODE solvers while substantially simplifying backpropagation through their internal integration steps. Specifically, for each solver integration interval, SSBP no longer propagates gradients step by step through the complete numerical state trajectory. Instead, it constructs a one-step gradient approximation using the initial state of the interval. This enables SSBP to decouple the backward process from forward integration at the solver level, allowing the solver to use the number of function evaluations (NFE) required to meet the desired numerical accuracy during forward integration, while the backward pass neither propagates gradients along the complete internal state trajectory nor stores or recomputes it, thereby substantially reducing computational and memory overhead. Specifically, consider a solver integration interval containing L internal steps, each requiring s vector-field evaluations, such that $\mathrm{NFE} = Ls$. The solver-related backward computation and memory costs of conventional BPTT both scale as $\mathcal{O}(Ls)$, whereas SSBP reduces both to $\mathcal{O}(1)$. Moreover, because it relies only on the initial state of each integration

interval, SSBP is not specific to any particular CTNN model or solver type and is therefore broadly applicable. In this work, our main contributions are summarized as follows:

- We introduce SSBP, an efficient CTNN training method that keeps the forward solve unchanged and bypasses backpropagation through internal solver steps using a one-step gradient approximation constructed from the interval's initial state. This solver-level decoupling reduces backward-pass computational and memory costs from $\mathcal{O}(Ls)$ to $\mathcal{O}(1)$.
- We provide a theoretical analysis of SSBP's effectiveness by characterizing how the update direction produced by SSBP approximates the conventional BPTT gradient, with empirical measurements of their directional alignment further supporting the analysis.
- We conduct extensive experiments across diverse CTNN models, tasks, and numerical solvers, showing that SSBP achieves task performance comparable to conventional BPTT while substantially improving training efficiency. Its application to a spin-torque oscillator (STO) simulator further demonstrates its practical utility.

2 Related works

CTNN models. CTNNs use differential equations to model hidden-state evolution as a continuous dynamical system. Neural ODEs (Chen et al., 2018) established a general continuous-depth framework that replaces a stack of discrete layers with state evolution computed by an ODE solver, enabling parameter-efficient modeling. Continuous-time RNNs (CT-RNNs) and ODE-RNNs extend RNNs with continuous-time state evolution, allowing them to model continuous-time dynamics under changing inputs (Funahashi & Nakamura, 1993; Rubanova et al., 2019). Liquid time-constant networks (LTCs) further enhance expressivity by introducing input- and state-dependent time constants (Hasani et al., 2021; Lechner et al., 2020). Despite the distinct formulations and underlying dynamics for different modeling needs, training these CTNN models largely relies on conventional BPTT (Werbos, 1990) or adjoint methods (Chen et al., 2018) and incurs substantial solver-related costs. We therefore propose SSBP as a general approach for training these models more efficiently while keeping their formulations, forward dynamics, and numerical integration process unchanged.

Efficient training of CTNNs. Existing efforts toward efficient CTNN training can be broadly categorized by whether they target forward solves, backward gradient computation, or the training formulation. Among methods targeting forward solves, regularization schemes constrain the complexity of learned dynamics through higher-order derivative, kinetic, and temporal regularization (Kelly et al., 2020; Finlay et al., 2020; Ghosh et al., 2020); Hypersolvers and TL-NODEs introduce efficient numerical integrators (Poli et al., 2020; Djeumou et al., 2022); and CfCs replace numerical integration with approximate closed-form models (Hasani et al., 2022). However, regularization toward simpler dynamics may trade performance for efficiency, while specialized integrators and closed-form models can limit applicability across solver choices and model families. Methods targeting backward gradient computation reduce memory or runtime through checkpointing and recomputation (Gholami et al., 2019; Zhuang et al., 2020), forward-state interpolation for the adjoint solve (Daulbaev et al., 2020), and accelerated adjoint computation using seminorm-based error control or numerical quadrature (Kidger et al., 2021; Norcliffe & Deisenroth, 2023). Training-problem reformulations can avoid differentiation through ODE solvers using the compact SNODE representation (Quaglino et al., 2020), Lyapunov-based objectives (Rodriguez et al., 2022), or variational integral formulations (Zhao et al., 2025), but may rely on specialized objectives, trajectory representations, or structural assumptions. Across these lines of work, forward and backward processes remain coupled as gradient computation still relies on the complete forward trajectory: some methods differentiate directly through the trajectory, whereas others reconstruct or approximate it. SSBP instead eliminates this dependence, enabling solver-level decoupling and more efficient training.

Time-local and state-local training. More broadly, existing studies have explored constructing update signals from time- or state-local information to avoid full-trajectory error propagation. Truncated BPTT and decoupled neural interfaces, which use finite-window backpropagation and locally predicted synthetic gradients, respectively, are representative time-local training methods for recurrent models (Williams & Peng, 1990; Jaderberg et al., 2017). In SNNs, OTTT tracks presynaptic activity for online updates, while SLTT discards temporal gradient paths; both avoid full tempo-

ral backpropagation (Xiao et al., 2022; Meng et al., 2023). State-skipping training constructs update signals from the initial state of autonomous state-propagating reservoirs, eliminating the need to access all intermediate states (Zhang et al., 2026). Similarly, EGA uses an explicit Euler approximation of gradients for online calibration in hybrid numerical modeling systems (Ouala et al., 2024), while phantom gradients replace exact gradient computation in implicit models with low-cost estimates (Geng et al., 2021). Together, these studies show that effective update signals can be constructed from time- or state-local information, rather than being derived solely from the exact gradient associated with the complete forward process. SSBP applies this principle to CTNNs, replacing backpropagation along the solver's full internal trajectory with a local-information-based approximation, thereby substantially improving efficiency while retaining comparable performance.

3 PRELIMINARIES

3.1 CONTINUOUS-TIME NEURAL NETWORKS

CTNNs take various forms, differing in how their state dynamics are formulated. In this work, we focus on three representative models described below. Neural ODEs (Chen et al., 2018) provide a general formulation. Let $h(t) \in \mathbb{R}^d$ denote the hidden state of a Neural ODE and $x(t)$ an optional external input at time $t \in [t_0, T]$, where d is the hidden-state dimension. The state dynamics are typically governed by

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = f_\theta\big(h(t), x(t), t\big), \tag{1}$$

where f_θ denotes a neural network with trainable parameters θ. CT-RNNs (Funahashi & Nakamura, 1993) take the form of Eq. 1 with an explicit state-decay term to promote stable dynamics:

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = -\frac{h(t)}{\tau} + f_\theta\big(h(t), x(t), t\big), \tag{2}$$

where τ denotes the time constant. Building on this recurrent formulation, LTCs (Hasani et al., 2021) further enhance expressivity by coupling their effective time constants to the hidden state and input, yielding the following state dynamics:

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = -\left[\frac{1}{\tau} + f_\theta\big(h(t), x(t), t\big)\right] h(t) + f_\theta\big(h(t), x(t), t\big) A, \tag{3}$$

where $A \in \mathbb{R}^d$ is a learnable saturation vector. This formulation induces a state- and input-dependent effective time constant $\tau_{\text{eff}}(t) = \tau/[1+\tau f_\theta(h(t), x(t), t)]$, with all operations applied element-wise. Each hidden unit can therefore adjust its temporal response based on the current state and input.

To express their state evolution in a unified form, let $F_\Theta(h(t), x(t), t) = \mathrm{d}h(t)/\mathrm{d}t$ denote the complete vector field associated with a given model, where Θ collects all model parameters. Given an integration interval $[t_k, t_{k+1}]$, let $h_k = h(t_k)$ and $h_{k+1} = h(t_{k+1})$ denote the states at its two endpoints. Integrating the state dynamics over this interval gives the following state evolution:

$$h_{k+1} = h_k + \int_{t_k}^{t_{k+1}} F_\Theta\big(h(t), x(t), t\big)\, \mathrm{d}t. \tag{4}$$

In practice, this integral is typically approximated by an ODE solver, which advances h_k to h_{k+1} by discretizing $[t_k, t_{k+1}]$ into multiple internal integration steps. How the solver's internal steps are handled during error backpropagation is a key consideration in CTNN training.

3.2 SOLVER-RELATED GRADIENT COMPUTATION

Over an integration interval $[t_k, t_{k+1}]$, the state evolution in Eq. 4 is numerically computed by an ODE solver that discretizes the interval into L internal steps. Let $h_k^{(\ell)}$ denote the state after ℓ internal steps, with $h_k^{(0)} = h_k$. The state update at each internal integration step can then be written as

$$h_k^{(\ell+1)} = \Phi\left(h_k^{(\ell)}, x_k, t_k^{(\ell)}, \Delta t_k^{(\ell)}; F_\Theta\right), \qquad \ell = 0, \ldots, L-1, \tag{5}$$

where $t_k^{(\ell)}$ denotes the internal time, with $t_k^{(0)} = t_k$ and $t_k^{(L)} = t_{k+1}$, and $\Delta t_k^{(\ell)} = t_k^{(\ell+1)} - t_k^{(\ell)}$ denotes the corresponding internal step size. Φ denotes the solver operator applied to F_Θ, x_k is the input over the interval, and $h_{k+1} = h_k^{(L)}$. If each internal step requires s function evaluations, the NFE over the interval is Ls.

Conventional BPTT. Gradients of the discretized forward solve can be obtained by applying the chain rule to the internal solver steps in Eq. 5. Given a loss $\mathcal{L}$, let $g_k^{(\ell)}$ denote the gradient with respect to the internal state $h_k^{(\ell)}$ after ℓ steps. For simplicity, we write $\Phi(h_k^{(\ell)})$ for the solver operator at the ℓ-th internal step, with the remaining arguments in Eq. 5 suppressed. The state gradient is recursively computed as

$$g_k^{(\ell)} = \frac{\partial \mathcal{L}}{\partial h_k^{(\ell)}} = \left(\frac{\partial \Phi\big(h_k^{(\ell)}\big)}{\partial h_k^{(\ell)}} \right)^\top g_k^{(\ell+1)}, \tag{6}$$

and over the full integration interval $[t_k, t_{k+1}]$, the gradient propagated through the solver to the interval's initial state h_k is

$$g_k = \prod_{\ell=0}^{L-1} \left(\frac{\partial \Phi(h_k^{(\ell)})}{\partial h_k^{(\ell)}} \right)^\top g_{k+1}, \tag{7}$$

where $g_k = g_k^{(0)}$ and $g_{k+1} = g_k^{(L)}$ denote the gradients of the loss with respect to the interval's initial and final states. This gradient computation backpropagates through all L steps and stores their states and computation graph, yielding $\mathcal{O}(Ls)$ solver-related backward computation and memory costs.

Adjoint sensitivity method. The adjoint method propagates the gradient from the integration interval's final state to its initial state by constructing and solving an additional adjoint ODE backward in time. Let $a(t) = \partial \mathcal{L} / \partial h(t)$ denote the continuous adjoint state, with its dynamics given by

$$\frac{\mathrm{d}a(t)}{\mathrm{d}t} = -\left(\frac{\partial F_\Theta(h(t), x(t), t)}{\partial h(t)} \right)^\top a(t). \tag{8}$$

Over the interval $[t_k, t_{k+1}]$, let $a_k = a(t_k)$ and $a_{k+1} = a(t_{k+1})$ denote the adjoint states at its initial and final states. Using a_{k+1} as the terminal condition, integrating Eq. 8 over this interval gives

$$a_k = a_{k+1} - \int_{t_{k+1}}^{t_k} \left(\frac{\partial F_\Theta(h(t), x(t), t)}{\partial h(t)} \right)^\top a(t)\, \mathrm{d}t. \tag{9}$$

Obtaining a_k requires an additional numerical solve of Eq. 8 backward from t_{k+1} to t_k. Typical implementations reconstruct $h(t)$ and reevaluate F_Θ during this solve rather than storing the forward states and computation graph, reducing memory usage but potentially increasing NFE and runtime.

4 Solver-Skipping Backpropagation

SSBP constructs, for each integration interval, a one-step gradient approximation using only local information at its initial state. Accordingly, it avoids gradient propagation along the solver's complete internal trajectory while leaving the forward solve unchanged, thereby decoupling the backward pass from the solver's internal forward integration. SSBP is motivated by the local behavior of interval sensitivities. Within a limited integration interval $[t_k, t_{k+1}]$, when the input is held at x_k or varies mildly and the internal step sizes are sufficiently small, adjacent states remain close. If the state dynamics are locally smooth, the corresponding local derivatives vary gradually along the internal trajectory. Under these conditions, the leading-order contribution of the internal trajectory to gradient propagation can be captured by a gradient approximation based on local information at the interval's initial state h_k. Appendix B provides theoretical analysis of this approximation principle.

To instantiate this gradient approximation, SSBP approximates the state evolution over an entire integration interval with a single update evaluated at its initial state h_k, as follows:

$$\widetilde{h}_{k+1} = \Psi\big(h_k, x_k, t_k, \Delta T_k; F_\Theta\big), \tag{10}$$

where Ψ denotes a one-step approximation operator derived from the corresponding F_Θ and the ODE solver Φ adopted for forward integration, and $\Delta T_k = t_{k+1} - t_k$ denotes the integration-interval duration. Unlike the solver's internal updates using $\Delta t_k^{(\ell)}$ in Eq. 5, Ψ updates h_k once over the full interval duration ΔT_k. The resulting proxy state $\widetilde{h}_{k+1}$ is used only for gradient computation and does not affect the forward state h_{k+1} produced by the original solver. For clarity, we write

$\Psi(h_k)$, suppressing the remaining arguments in Eq. 10. Given the gradient g_{k+1} at the interval's final state, SSBP approximates the gradient at its initial state as

$$\widetilde{g}_k = \left(\frac{\partial \Psi(h_k)}{\partial h_k} \right)^\top g_{k+1}, \tag{11}$$

where $\widetilde{g}_k$ approximates the conventional BPTT gradient g_k in Eq. 7. The corresponding contribution to the parameter-gradient approximation is obtained as $(\partial \Psi(h_k)/\partial \Theta)^\top g_{k+1}$. Computing these gradient approximations requires neither accessing the solver's internal trajectory nor storing its computation graph, thus reducing solver-related backward computation and memory costs from $\mathcal{O}(Ls)$ to $\mathcal{O}(1)$ relative to conventional BPTT. Detailed formulations of the proxy states and gradient approximations for each model–solver setting considered in this work are provided in Appendix E. Moreover, our theoretical analysis identifies positive alignment between the parameter-gradient approximation constructed by SSBP and the BPTT parameter gradient as a sufficient condition for effective optimization. Specifically, if the angle between them is below $90°$, SSBP provides an effective update direction for a sufficiently small learning rate. The theoretical derivation and empirical measurements of this alignment are provided in Appendices C and D.

5 Experiments

We evaluate the effectiveness and efficiency of SSBP across diverse sequence and time-series tasks. We consider three representative CTNN models—Neural ODEs (Chen et al., 2018), CT-RNNs (Funahashi & Nakamura, 1993), and LTCs (Hasani et al., 2021)—in their basic tanh-based forms, together with multiple ODE solvers. Conventional BPTT and the adjoint sensitivity method serve as baselines. All these experiments are implemented in PyTorch (Paszke et al., 2019), with AdamW (Loshchilov & Hutter, 2017) used for parameter optimization. Finally, to assess its practical utility, we apply SSBP to an STO simulator (Imai et al., 2025) for gradient-based optimization. Detailed dataset descriptions and experimental settings are provided in Appendices H and I.

5.1 Effectiveness and Efficiency

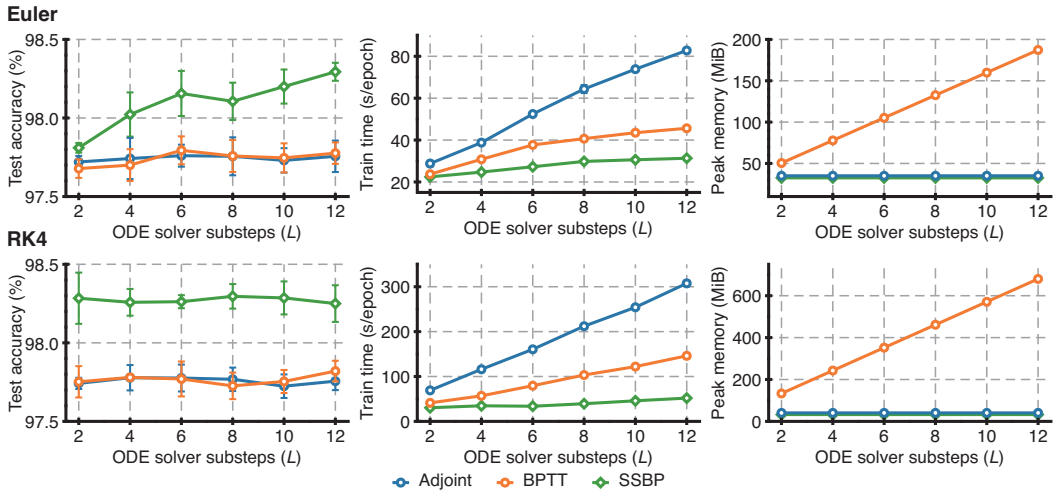

Figure 2: Test accuracy, per-epoch training runtime, and peak GPU memory usage of a Neural ODE on SMNIST trained with SSBP, conventional BPTT, and the adjoint method. The Euler (top) and RK4 (bottom) ODE solvers are evaluated as the number of internal steps L varies. Markers and error bars denote the mean and standard deviation over five runs.

We first evaluate the effectiveness and efficiency of the proposed SSBP. To this end, we train a basic Neural ODE with 500 hidden units on sequential MNIST (SMNIST) (Le Cun et al., 1998), a 10-class row-wise digit-classification task. For forward integration, we consider Euler and fourth-order Runge–Kutta (RK4) solvers. The integration-interval duration is fixed at $\Delta T = 0.6$, and the number

of internal solver steps L is varied over $\{2, 4, 6, 8, 10, 12\}$. For each setting, we compare SSBP with BPTT and the adjoint method under the same forward solve and training configuration, reporting test accuracy, peak memory usage, and runtime. Detailed settings are summarized in Appendix I.

Fig. 2 shows the results. For both Euler- and RK4-based forward integration, SSBP achieves test accuracy comparable to or even higher than those achieved by BPTT and the adjoint method, demonstrating its effectiveness. We attribute this accuracy improvement to the implicit regularization induced by replacing the product of solver-step Jacobians with a single local approximation, which may prevent the optimizer from overfitting fine-grained variations along the internal trajectory and thereby improve generalization. Moreover, with the Euler solver, SSBP accuracy improves as L increases, indicating that it benefits from the more accurate forward integration provided by finer internal steps. With the RK4 solver, its performance remains stable across L and matches the best performance achieved with the Euler solver, suggesting that sufficiently accurate state evolution is already achieved at small L. In contrast, BPTT and the adjoint method show marginal variation with L, possibly because the aforementioned tendency to overfit limits the benefit of more accurate forward integration. Regarding training efficiency, the training runtime increases with L for all methods due to the growing forward-integration cost, but SSBP remains the fastest and shows the slowest growth because its backward computation is independent of the solver's internal steps. BPTT training runtime grows faster as its backward computation graph expands with L, while the adjoint method is the slowest and shows the steepest growth due to the additional backward ODE solve. For peak GPU memory usage, BPTT exhibits a linear increase with L because it stores all solver-internal states and the computation graph, whereas the adjoint method and SSBP remain constant because they avoid this storage. Moreover, SSBP requires the least memory, as it avoids the additional state variables introduced by the augmented backward ODE system. Notably, this efficiency advantage of SSBP is more pronounced when using the RK4 solver, as each RK4 internal step requires $s = 4$ function evaluations, further increasing the solver-related backward overhead of BPTT and the adjoint method while leaving SSBP's backward computation unchanged. Specifically, at $L = 12$, where the efficiency gap is largest, SSBP achieves a speedup of $1.46\times$ and a peak-memory reduction of 82.6% over BPTT with the Euler solver; with the RK4 solver, these gains increase to $2.82\times$ and 95.2%. Compared with the adjoint method, the corresponding gains are $2.64\times$ and 7.2% with the Euler solver, and $5.93\times$ and 19.7% with the RK4 solver.

5.2 Generality across Tasks and Models

We next assess the generality of SSBP across Neural ODEs, CT-RNNs, and LTCs on six sequence and time-series benchmarks—SMNIST, Hand Gesture Segmentation (Gesture) (Wagner et al., 2014), Room Occupancy (Occ.) (Candanedo & Feldheim, 2016), Human Activity Recognition (HAR) (Anguita et al., 2013), Traffic Estimation (Traffic) (Hogue, 2019), and Person Activity (PA) described in Rubanova et al. (2019). Here, Neural ODEs and CT-RNNs are evaluated with both Euler and RK4 solvers, while LTCs are evaluated with the fused ODE solver proposed in the original work (Hasani et al., 2021). For PA, we consider both regular sampling and irregular sampling with a fixed number of internal solver steps $L = 6$. For all other regularly sampled benchmarks, we fix the integration-interval duration at $\Delta T = 0.6$ and use $L = 6$. We compare conventional BPTT and SSBP in terms of task performance, per-epoch training runtime, and peak GPU memory usage, across five runs. Detailed dataset descriptions and experimental settings are provided in Appendices H and I. Table 1 summarizes the results. In terms of task performance, SSBP generally outperforms BPTT for Neural ODEs, which we attribute to the potential regularization effect discussed in Section 5.1. It also remains competitive for CT-RNNs and across most LTC settings. Its effectiveness under both regular and irregular sampling further confirms its generality across different temporal sampling patterns. An exception is the modest performance gap for LTCs on the PA task, which we attribute to their state- and input-dependent effective time constants τ_{eff}. As the state evolves, τ_{eff} changes across the fused solver's internal steps, making the initial-state information used by SSBP less representative of the entire interval. The unnormalized and strongly varying inputs retained by the PA preprocessing further amplify these changes, increasing the discrepancy between the SSBP gradient approximation and the BPTT gradient. In terms of efficiency, SSBP consistently requires less per-epoch training time and peak GPU memory than conventional BPTT across all evaluated tasks, models, and solvers. Taken together, these results demonstrate that SSBP substantially simplifies gradient computation during the backward pass with little to no loss in task performance, thereby providing a general and efficient approach to CTNN training.

Table 1: Generality of SSBP across tasks, CTNN models, and ODE solvers. Within each result cell, task performance (test accuracy (%) or mean squared error (MSE) for Traffic), per-epoch training runtime (s), and peak GPU memory usage (MiB) are listed from top to bottom. Performance and runtime are reported as mean ± standard deviation over five runs. The better results between BPTT and SSBP are shown in bold. Reg. and Irreg. denote regular and irregular sampling, respectively.

Task	Neural ODE				CT-RNN				LTC	
	Euler		RK4		Euler		RK4		Fused	
	BPTT	**SSBP**	BPTT	**SSBP**	BPTT	**SSBP**	BPTT	**SSBP**	BPTT	**SSBP**
SMNIST	97.79% ±0.09%	**98.16%** ±0.14%	97.77% ±0.11%	**98.26%** ±0.04%	**98.85%** ±0.00%	**98.85%** ±0.07%	**98.82%** ±0.07%	**98.82%** ±0.04%	**99.10%** ±0.05%	98.91% ±0.04%
	37.70 ±0.57	**27.51** ±0.61	80.44 ±1.40	**34.18** ±0.49	40.99 ±0.45	**34.67** ±0.11	99.43 ±1.85	**49.54** ±0.09	46.02 ±0.07	**35.53** ±0.06
	105.21	**32.57**	353.04	**32.57**	105.53	**32.58**	354.34	**32.58**	227.39	**33.08**
Gesture	44.06% ±2.06%	**46.74%** ±1.74%	**45.07%** ±0.98%	44.53% ±0.92%	62.21% ±1.19%	**62.45%** ±1.36%	61.55% ±0.90%	**62.43%** ±1.10%	**69.62%** ±1.16%	69.49% ±0.88%
	1.06 ±0.02	**0.84** ±0.00	2.67 ±0.00	**1.29** ±0.07	1.35 ±0.03	**0.95** ±0.00	3.60 ±0.00	**1.86** ±0.05	1.64 ±0.28	**1.10** ±0.07
	25.67	**20.17**	43.67	**20.17**	25.76	**20.17**	44.05	**20.17**	34.77	**20.17**
Occ.	90.62% ±3.57%	**92.06%** ±2.81%	91.10% ±2.48%	**92.11%** ±3.58%	**98.02%** ±0.66%	97.93% ±0.61%	**98.07%** ±0.67%	97.96% ±0.61%	**97.57%** ±0.71%	97.53% ±0.51%
	2.08 ±0.00	**1.63** ±0.00	5.18 ±0.01	**2.40** ±0.00	2.32 ±0.03	**1.79** ±0.06	6.99 ±0.00	**3.51** ±0.02	2.78 ±0.01	**1.90** ±0.04
	21.85	**19.10**	30.85	**19.10**	21.94	**19.10**	31.23	**19.10**	26.45	**19.10**
HAR	97.98% ±0.31%	**98.06%** ±0.19%	97.90% ±0.16%	**98.09%** ±0.13%	**98.16%** ±0.13%	98.11% ±0.17%	**98.10%** ±0.25%	97.97% ±0.21%	**98.30%** ±0.22%	98.29% ±0.19%
	15.24 ±0.04	**8.76** ±0.50	51.93 ±0.33	**19.02** ±0.95	18.34 ±0.56	**10.61** ±0.42	66.80 ±0.48	**25.98** ±0.07	25.03 ±3.52	**12.53** ±0.71
	28.84	**23.37**	46.88	**23.37**	28.93	**23.37**	47.26	**23.37**	37.84	**23.40**
Traffic	0.201 ±0.035	**0.187** ±0.026	0.224 ±0.015	**0.190** ±0.012	**0.071** ±0.002	0.079 ±0.004	**0.072** ±0.002	0.088 ±0.004	**0.066** ±0.002	0.076 ±0.002
	9.87 ±0.06	**7.96** ±0.06	25.35 ±0.05	**11.74** ±0.00	11.20 ±0.07	**9.03** ±0.01	34.29 ±0.03	**17.19** ±0.03	13.66 ±0.06	**9.51** ±0.11
	25.50	**20.00**	43.50	**20.00**	25.59	**20.00**	43.88	**20.00**	34.60	**20.00**
PA-Reg.	76.44% ±0.57%	**76.54%** ±0.26%	76.26% ±0.47%	**76.53%** ±0.17%	83.25% ±0.29%	**83.40%** ±0.22%	**83.36%** ±0.27%	83.16% ±0.35%	**85.09%** ±0.22%	84.19% ±0.23%
	8.44 ±0.14	**6.82** ±0.02	21.57 ±0.06	**10.10** ±0.00	9.62 ±0.08	**7.65** ±0.03	29.13 ±0.01	**14.57** ±0.01	11.74 ±0.01	**8.23** ±0.01
	32.65	**21.65**	68.65	**21.65**	32.74	**21.65**	69.02	**21.65**	50.74	**21.65**
PA-Irreg.	73.11% ±0.55%	**73.27%** ±0.42%	**73.20%** ±0.27%	73.12% ±0.37%	**83.85%** ±0.41%	83.32% ±0.26%	**83.72%** ±0.29%	83.12% ±0.25%	**84.89%** ±0.30%	83.23% ±0.31%
	11.11 ±0.85	**7.97** ±0.23	35.71 ±0.99	**17.30** ±1.10	13.13 ±0.40	**8.93** ±1.39	39.67 ±1.07	**19.01** ±1.71	18.66 ±0.42	**13.42** ±0.73
	41.62	**24.41**	98.31	**24.41**	41.77	**24.41**	98.90	**24.41**	69.75	**24.41**

5.3 Evaluation with an Adaptive Solver

In this section, we evaluate SSBP with adaptive integration using a Neural ODE on SMNIST. Unlike fixed-step solvers, an adaptive solver dynamically adjusts its internal step sizes according to local error estimates. We employ the adaptive Dormand–Prince (Dopri5) solver (Dormand & Prince, 1980) and compare SSBP with BPTT in terms of test accuracy, mean forward NFE per integration interval, per-epoch training runtime, and peak GPU memory usage over five runs. We use the same 500-hidden-unit Neural ODE and training configuration as in Section 5.1, with the integration-interval duration fixed at $\Delta T = 0.6$. Dopri5 uses relative and absolute tolerances of 10^{-5} and 10^{-8}, respectively, with the initial step size set to ΔT. The reported NFE includes all forward function evaluations from both accepted and rejected solver step attempts and is averaged over all integration intervals within each training epoch. Detailed settings are provided in Appendix I. Fig. 3 shows the results. SSBP achieves test accuracy comparable to BPTT throughout training and reaches a best accuracy of 98.08%, compared with 97.54% for BPTT, demonstrating that the gradient approximation constructed by SSBP remains effective under adaptive integration. As training progresses, the mean forward NFE of SSBP becomes higher than that of BPTT but remains at a comparable level and fol-

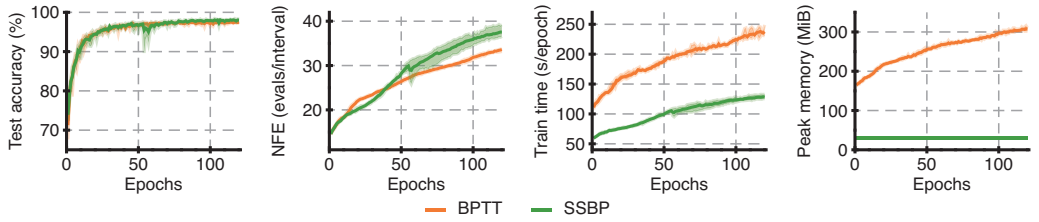

Figure 3: Test accuracy, mean forward NFE per integration interval, per-epoch training runtime, and peak GPU memory usage of a Neural ODE on SMNIST trained using BPTT and SSBP with the adaptive Dormand–Prince (Dopri5) solver. Lines denote means over five runs, and shaded regions indicate the corresponding minimum–maximum ranges.

lows a similar trend. This indicates that the dynamics learned by the two methods evolve similarly in their numerical integration difficulty, and SSBP does not drive the model toward substantially more difficult-to-integrate dynamics. Benefiting from the decoupling of the backward pass from adaptive forward integration, SSBP consistently requires substantially less training time and memory than BPTT. As forward NFE increases during training, SSBP exhibits slower runtime growth and constant memory usage, whereas BPTT exhibits faster growth in both runtime and memory usage.

5.4 Application to a Spin-Torque Oscillator

Table 2: Performance and efficiency of BPTT, the adjoint method, and SSBP for the coupled STO simulator on MNIST. Accuracy and time are reported as mean $\pm$ standard deviation over five runs.

	BPTT	**Adjoint**	**SSBP**
Test accuracy (%) $\uparrow$	$98.29\% \pm 0.07\%$	$98.31\% \pm 0.06\%$	$\mathbf{98.38\% \pm 0.07\%}$
Time (s/epoch) $\downarrow$	49.29 ± 1.15	77.34 ± 5.99	$\mathbf{36.80 \pm 0.67}$
Memory (MiB) $\downarrow$	199.24	51.75	**43.93**

Motivated by the potential of neuromorphic computing for efficient deep learning and its natural compatibility with continuous-time dynamics, we apply SSBP to a coupled macrospin-type STO simulator (Imai et al., 2025). It comprises 200 coupled STOs whose magnetization dynamics are governed by the Landau–Lifshitz–Gilbert equation. Detailed dynamics and settings are provided in Appendix F. As shown in Fig. 4, the input image $\mathbf{u}$ is projected into the initial STO states via $\mathbf{W}_{\mathrm{in}}$, the coupled STO system is then evolved using an RK4 solver for 10 steps with a step size of 50 ps, and its final state is mapped to the output through $\mathbf{W}_{\mathrm{out}}$. We jointly optimize $\mathbf{W}_{\mathrm{in}}$, $\mathbf{W}_{\mathrm{out}}$, and physical parameters of the STOs, including applied magnetic fields, current, and Gilbert damping. We compare SSBP with BPTT and the adjoint method, reporting test accuracy, per-epoch training runtime, and peak GPU memory usage over five runs. As shown in Table 2, SSBP achieves improved performance with substantially reduced runtime and memory usage, demonstrating its practical utility.

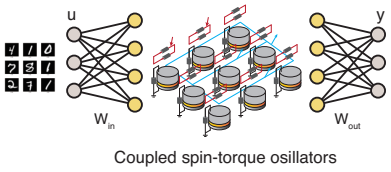

Figure 4: Schematic of the coupled STO simulator.

6 Conclusion

We propose solver-skipping backpropagation (SSBP), which replaces complete solver-trajectory gradient propagation with a one-step approximation constructed from the integration interval's initial state. Leaving the forward solve unchanged, SSBP decouples the backward pass from internal solver steps and reduces solver-related gradient computation and memory costs from $\mathcal{O}(Ls)$ to $\mathcal{O}(1)$. Across diverse CTNN models, solvers, and tasks, SSBP achieves competitive or improved performance with substantially lower training runtime and memory usage. Results with adaptive integration and coupled-STO optimization further demonstrate its generality and practical utility.

AI USE STATEMENT

In this work, generative AI tools were used to assist with language polishing, LaTeX formatting, table formatting, and background literature searches. They were also used as an aid in code development and organization, as well as in developing mathematical proofs. We independently developed the research idea, proposed method, hypotheses, and experimental design and determined the final scientific conclusions. Generative AI tools did not originate or determine any of these contributions. Moreover, although generative AI tools assisted with background literature searches, every source cited in this paper was independently identified, read, selected, and verified by us. Generative AI tools were not used to preprocess or aggregate experimental data, execute experiments, generate plots, or produce the reported numerical results. Although generative AI tools assisted with mathematical proof development, we developed the core proof ideas, independently reviewed every derivation, and made all final judgments regarding mathematical correctness. All AI-assisted work was reviewed by us, and all code developed with AI assistance was independently inspected, tested, and verified for correctness by two authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops an efficient training method for continuous-time neural networks and evaluates it using publicly available benchmark datasets and numerical simulations. It involves no new human-subject studies, collection of personal data, or any form of invasive technology. All datasets are used in accordance with their stated licenses and usage policies.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide model-specific formulations and pseudocode for our method in Appendices E and G. Detailed descriptions of the datasets and preprocessing procedures are provided in Appendix H, while model architectures, solver configurations, hyperparameters, initialization, training and evaluation procedures, and computational resources are documented in Appendix I. The dynamics and optimization settings of the utilized STO simulator are provided in Appendix F. We will publicly release the source code and configurations required to reproduce all experiments upon publication.

REFERENCES

Victor M Martinez Alvarez, Rareş Roşca, and Cristian G Fălcuţescu. Dynode: Neural ordinary differential equations for dynamics modeling in continuous control. *arXiv preprint arXiv:2009.04278*, 2020.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, pp. 3–4, 2013.

Luis M Candanedo and Véronique Feldheim. Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and buildings*, 112:28–39, 2016.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolation technique to speed up gradients propagation in neural odes. *Advances in Neural Information Processing Systems*, 33:16689–16700, 2020.

Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Taylor-lagrange neural ordinary differential equations: Toward fast training and evaluation of neural odes. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 2923–2929. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/405. URL https://doi.org/10.24963/ijcai.2022/405. Main Track.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. *Advances in neural information processing systems*, 34:24247–24260, 2021.

Mohamed Ghanem and Bernd Finkbeiner. Flowing through states: Neural ODE regularization for reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=FHFDCsB9UC.

Amir Gholami, Kurt Keutzer, and George Biros. Anode: unconditionally accurate memory-efficient gradients for neural odes. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, pp. 730–736. AAAI Press, 2019. ISBN 9780999241141.

Arnab Ghosh, Harkirat Behl, Emilien Dupont, Philip Torr, and Vinay Namboodiri. Steer: Simple temporal regularization for neural ode. *Advances in neural information processing systems*, 33: 14831–14843, 2020.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJxgknCcK7.

Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid time-constant networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 7657–7666, 2021.

Ramin Hasani, Mathias Lechner, Alexander Amini, Lucas Liebenwein, Aaron Ray, Max Tschaikowski, Gerald Teschl, and Daniela Rus. Closed-form continuous-time neural networks. *Nature Machine Intelligence*, 4(11):992–1003, 2022.

John Hogue. Metro Interstate Traffic Volume. UCI Machine Learning Repository, 2019. DOI: https://doi.org/10.24432/C5X60B.

Yusuke Imai, Shuhong Liu, Nozomi Akashi, and Kohei Nakajima. Gradient-based optimization of spintronic devices. *Applied Physics Letters*, 126(8), 2025.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International conference on machine learning*, pp. 1627–1635. PMLR, 2017.

Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33:4370–4380, 2020.

Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in neural information processing systems*, 33:6696–6707, 2020.

Patrick Kidger, Ricky TQ Chen, and Terry J Lyons. " hey, that's not an ode": Faster ode adjoints via seminorms. In *ICML*, pp. 5443–5452, 2021.

Yann Le Cun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998. URL http://leon.bottou.org/papers/lecun-98h.

Mathias Lechner and Ramin Hasani. Learning long-term dependencies in irregularly-sampled time series. *arXiv preprint arXiv:2006.04418*, 2020.

Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10): 642–652, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Takashi Matsubara, Yuto Miyatake, and Takaharu Yaguchi. Symplectic adjoint method for exact gradient of neural ODE with minimal memory. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=46J_l-cpc1W.

Qingyan Meng, Mingqing Xiao, Shen Yan, Yisen Wang, Zhouchen Lin, and Zhi-Quan Luo. Towards memory- and time-efficient backpropagation for training spiking neural networks. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6143–6153, 2023. doi: 10.1109/ICCV51070.2023.00567.

Alexander Norcliffe and Marc Peter Deisenroth. Faster training of neural odes using gau {\ss}-legendre quadrature. *arXiv preprint arXiv:2308.10644*, 2023.

Derek Onken and Lars Ruthotto. Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows. *arXiv preprint arXiv:2005.13420*, 2020.

Said Ouala, Bertrand Chapron, Fabrice Collard, Lucile Gaultier, and Ronan Fablet. Online calibration of deep learning sub-models for hybrid numerical modeling systems. *Communications Physics*, 7(1):402, 2024.

Avik Pal, Yingbo Ma, Viral Shah, and Christopher V Rackauckas. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In *International Conference on Machine Learning*, pp. 8325–8335. PMLR, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21105–21117. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f1686b4badcf28d33ed632036c7ab0b8-Paper.pdf.

Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Snode: Spectral discretization of neural odes for system identification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Sye0XkBKvS.

Ivan Dario Jimenez Rodriguez, Aaron Ames, and Yisong Yue. LyaNet: A Lyapunov framework for training neural ODEs. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18687–18703. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/rodriguez22a.html.

Yulia Rubanova, Ricky TQ Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

Jan-Philipp von Bassewitz, Adeel Pervez, Marco Fumero, Matthew R Robinson, Theofanis Karaletsos, and Francesco Locatello. Learning explicit single-cell dynamics using ODE representations. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=DzSNH5APPl.

Priscilla Koch Wagner, Sarajane Marques Peres, Renata Cristina Barros Madeo, Clodoaldo Aparecido de Moraes Lima, and Fernando de Almeida Freitas. Gesture unit segmentation using spatial-temporal information and machine learning. In *FLAIRS*, 2014.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

Mingqing Xiao, Qingyan Meng, Zongpeng Zhang, Di He, and Zhouchen Lin. Online training through time for spiking neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 20717–20730. Curran Associates, Inc., 2022. doi: 10.52202/068431-1506. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/82846e19e6d42ebfd4ace4361def29ae-Paper-Conference.pdf.

Jun Yin, Jiali Li, Iftekhar A Karimi, and Xiaonan Wang. Generalized reactor neural ode for dynamic reaction process modeling with physical interpretability. *Chemical Engineering Journal*, 452: 139487, 2023.

Yongbo Zhang, Mitsumasa Nakajima, Katsuma Inoue, Toshikazu Hashimoto, Yasuo Kuniyoshi, and Kohei Nakajima. Scalable physical deep learning using optical dynamics with state-skipping training. *Research Square*, January 2026. doi: 10.21203/rs.3.rs-8145052/v1. URL https://doi.org/10.21203/rs.3.rs-8145052/v1. Preprint.

Hongjue Zhao, Yuchen Wang, Hairong Qi, Zijie Huang, Han Zhao, Lui Sha, and Huajie Shao. Accelerating neural odes: a variational formulation-based approach. In *International Conference on Learning Representations*, volume 2025, pp. 17913–17942, 2025.

Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *International Conference on Machine Learning*, pp. 11639–11649. PMLR, 2020.

Juntang Zhuang, Nicha C Dvornek, sekhar tatikonda, and James s Duncan. {MALI}: A memory efficient and reverse accurate integrator for neural {ode}s. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=blfSjHeFM_e.

A Limitations and Future Work

Our theoretical analysis and experiments mainly consider locally smooth dynamics, sufficiently short integration intervals, and moderate within-interval variations. When an interval is long, or when the state, input, or effective time constants vary substantially within it, local information at the initial state may not adequately represent the complete internal trajectory, increasing the gradient-approximation error. The modest performance gap observed for LTCs on PA illustrates this limitation. Addressing this limitation is an important direction for future work. Potential approaches include introducing appropriate regularization or incorporating information from a small number of additional internal steps to improve approximation accuracy and, consequently, task performance. We have also not evaluated SSBP under strongly stiff or nonsmooth dynamics, discrete events, or complex constraints, which may require specialized approximation forms and further theoretical analysis. Moreover, our theoretical analysis establishes positive gradient alignment as a sufficient condition for SSBP to provide an effective update direction, but does not guarantee that this condition holds for every model or setting. Finally, SSBP currently decouples backpropagation from the solver's internal steps, while temporal gradient propagation across integration intervals still relies on BPTT. Further decoupling of cross-interval gradient propagation and extending SSBP to ResNet-style continuous-depth Neural ODEs are promising directions for future work.

B Theory Analysis: SSBP's Approximation Principle for the Leading-Order Contribution in BPTT

Consider the continuous dynamics

$$\frac{\mathrm{d}h}{\mathrm{d}t} = F_\Theta(h, x, t) \tag{12}$$

over an integration interval $[t_k, t_{k+1}]$ of duration $\Delta T_k = t_{k+1} - t_k$. The forward solver performs L internal steps and generates

$$h_k^{(0)} = h_k, \qquad h_k^{(L)} = h_{k+1}, \tag{13}$$

where $h_k^{(\ell)}$ denotes the state after the ℓ-th internal step. Let $t_k^{(\ell)}$ be the corresponding internal time point and define

$$\delta_\ell = t_k^{(\ell+1)} - t_k^{(\ell)}, \qquad \sum_{\ell=0}^{L-1} \delta_\ell = \Delta T_k. \tag{14}$$

Here, δ_ℓ is the ℓ-th internal step size, and $x_k^{(\ell)} = x(t_k^{(\ell)})$ denotes the input at the corresponding internal time point.

Assumption 1 (Local regularity and first-order consistency)**.** *Along the state trajectory over $[t_k, t_{k+1}]$, $F_\Theta(h, x, t)$ is continuously differentiable, and both F_Θ and its state Jacobian $\partial F_\Theta / \partial h$ are locally bounded. The state Jacobian is also locally Lipschitz continuous, such that local changes in the state, input, and time do not induce abrupt changes in the Jacobian. The input $x(t)$ is either held fixed within the interval or locally Lipschitz continuous in time.*

Let Φ_{δ_ℓ} denote one internal solver update with step size δ_ℓ. The numerical update and its state Jacobian are assumed to be first-order consistent:

$$\Phi_{\delta_\ell}(h, x, t) = h + \delta_\ell F_\Theta(h, x, t) + \mathcal{O}(\delta_\ell^2), \tag{15}$$

and

$$\frac{\partial \Phi_{\delta_\ell}(h, x, t)}{\partial h} = I + \delta_\ell \frac{\partial F_\Theta(h, x, t)}{\partial h} + \mathcal{O}(\delta_\ell^2), \tag{16}$$

where I is the identity matrix with the same dimension as the state.

Similarly, using the abbreviated notation introduced in Eq. 10, the SSBP one-step operator satisfies

$$\Psi(h_k) = h_k + \Delta T_k F_\Theta(h_k, x_k, t_k) + \mathcal{O}(\Delta T_k^2), \tag{17}$$

and

$$\frac{\partial \Psi(h_k)}{\partial h_k} = I + \Delta T_k \frac{\partial F_\Theta(h_k, x_k, t_k)}{\partial h_k} + \mathcal{O}(\Delta T_k^2). \tag{18}$$

The terms $\mathcal{O}(\delta_\ell^2)$ and $\mathcal{O}(\Delta T_k^2)$ collect contributions of second and higher orders in the internal step size and the integration-interval duration, respectively. More precisely, their norms are bounded by constant multiples of δ_ℓ^2 and ΔT_k^2 for sufficiently small step sizes. Therefore, Eqs. 15 and 16 show that the numerical solver update and its Jacobian preserve, to first order, the state evolution and local gradient propagation of the continuous dynamics, respectively.

Notably, the conditions in Assumption 1 are standard local properties commonly satisfied by continuous-time models and numerical ODE solvers. CTNNs constructed from smooth operations, including affine maps and smooth activation functions, generally have locally bounded and locally Lipschitz continuous state Jacobians. The tanh-based Neural ODE, CT-RNN, and LTC models considered in this work satisfy these conditions. The Euler and Runge–Kutta solvers, as well as the LTC fused solver used in this work, reproduce the first-order behavior of the corresponding continuous dynamics for sufficiently small step sizes and therefore satisfy the expansions in Assumption 1. The model-specific one-step approximation used by SSBP also satisfy these expansions, as detailed in Appendix E. The approximation operator of SSBP need not reproduce every internal stage of the forward solver; it only needs to share the solver's first-order state-update and Jacobian terms. Thus, Assumption 1 characterizes the usual local behavior of smooth CTNNs and their numerical solvers rather than imposing a model- or solver-specific restriction.

Remark 1 (Origin of the first-order expansions). Here, we derive the first-order expansions in Assumption 1 from the integral form of the continuous dynamics and the local smoothness of F_Θ. Specifically, the exact state evolution over a step of size δ satisfies

$$h(t+\delta) = h(t) + \int_0^\delta F_\Theta\big(h(t+\tau), x(t+\tau), t+\tau\big)\,\mathrm{d}\tau, \tag{19}$$

where $\tau \in [0, \delta]$ denotes the time offset from the beginning of the step. Under the local regularity conditions,

$$F_\Theta\big(h(t+\tau), x(t+\tau), t+\tau\big) = F_\Theta(h(t), x(t), t) + \mathcal{O}(\tau). \tag{20}$$

Substituting this relation into the integral gives

$$h(t+\delta) = h(t) + \delta F_\Theta(h(t), x(t), t) + \mathcal{O}(\delta^2). \tag{21}$$

Differentiating this expansion with respect to the initial state $h(t)$ further gives

$$\frac{\partial h(t+\delta)}{\partial h(t)} = I + \delta \frac{\partial F_\Theta(h(t), x(t), t)}{\partial h(t)} + \mathcal{O}(\delta^2). \tag{22}$$

Therefore, Eqs. 21 and 22 derive the first-order state-update and Jacobian expansions used in Assumption 1.

Define the vector-field Jacobian at the ℓ-th internal state as

$$A_k^{(\ell)} = \frac{\partial F_\Theta}{\partial h}\left(h_k^{(\ell)}, x_k^{(\ell)}, t_k^{(\ell)}\right), \tag{23}$$

and let

$$A_k = A_k^{(0)} = \frac{\partial F_\Theta}{\partial h}(h_k, x_k, t_k) \tag{24}$$

denote this Jacobian evaluated at the beginning of the interval.

Lemma 1 (First-order expansion of the solver-trajectory Jacobian)**.** *Under Assumption 1, the Jacobian of the complete internal solver trajectory satisfies*

$$J_k := \frac{\partial h_{k+1}}{\partial h_k} = I + \Delta T_k A_k + \mathcal{O}(\Delta T_k^2). \tag{25}$$

Here, J_k describes how a local perturbation of the interval's initial state h_k affects its final state h_{k+1}.

Proof. Define the state Jacobian of the ℓ-th internal solver step as

$$D_k^{(\ell)} = \frac{\partial h_k^{(\ell+1)}}{\partial h_k^{(\ell)}}. \tag{26}$$

By the first-order consistency in Eq. 16,

$$D_k^{(\ell)} = I + \delta_\ell A_k^{(\ell)} + \mathcal{O}(\delta_\ell^2). \tag{27}$$

The Jacobian of the complete trajectory is the ordered product

$$J_k = D_k^{(L-1)} D_k^{(L-2)} \cdots D_k^{(0)}. \tag{28}$$

Substituting the expansion of each internal-step Jacobian gives

$$J_k = \prod_{\ell=L-1}^{0} \left(I + \delta_\ell A_k^{(\ell)} + \mathcal{O}(\delta_\ell^2) \right), \tag{29}$$

where the product follows the backward order from $\ell = L-1$ to $\ell = 0$. In this product, selecting the identity I from every factor gives the zeroth-order term I, while selecting $\delta_\ell A_k^{(\ell)}$ from one factor and I from all remaining factors gives the first-order terms $\sum_{\ell=0}^{L-1} \delta_\ell A_k^{(\ell)}$. All other terms contain either an $\mathcal{O}(\delta_\ell^2)$ remainder or at least two step-size factors, such as $\delta_j \delta_i A_k^{(j)} A_k^{(i)}$.

Because the Jacobians $A_k^{(\ell)}$ are locally bounded, the norms of all matrix factors in the product are bounded by a common finite constant. Consequently, each local remainder contributes at most a constant multiple of δ_ℓ^2, while each product involving first-order terms from two distinct steps contributes at most a constant multiple of $\delta_i \delta_j$. Terms containing three or more step-size factors are of third or higher order and can also be absorbed into the second-order bound for sufficiently small ΔT_k. Therefore, the total magnitude of the higher-order terms is bounded by a constant multiple of

$$\sum_{\ell=0}^{L-1} \delta_\ell^2 + \sum_{0 \le i < j \le L-1} \delta_i \delta_j \le \sum_{\ell=0}^{L-1} \delta_\ell^2 + 2 \sum_{0 \le i < j \le L-1} \delta_i \delta_j = \left(\sum_{\ell=0}^{L-1} \delta_\ell \right)^2 = \Delta T_k^2. \tag{30}$$

Therefore,

$$J_k = I + \sum_{\ell=0}^{L-1} \delta_\ell A_k^{(\ell)} + \mathcal{O}(\Delta T_k^2). \tag{31}$$

The remainder contains the second-order terms of individual solver steps and products involving at least two internal step sizes.

To relate the internal Jacobians to the interval's initial state, we first accumulate the state changes over the preceding internal steps. By the first-order solver expansion,

$$h_k^{(\ell)} - h_k = \sum_{r=0}^{\ell-1} \left(h_k^{(r+1)} - h_k^{(r)} \right) \tag{32}$$

$$= \sum_{r=0}^{\ell-1} \delta_r F_\Theta \left(h_k^{(r)}, x_k^{(r)}, t_k^{(r)} \right) + \sum_{r=0}^{\ell-1} \mathcal{O}(\delta_r^2), \tag{33}$$

where r indexes the internal steps preceding step ℓ. Because F_Θ is locally bounded and $\sum_{r=0}^{\ell-1} \delta_r = t_k^{(\ell)} - t_k$, the accumulated first-order contribution is $\mathcal{O}(t_k^{(\ell)} - t_k)$. Moreover,

$$\sum_{r=0}^{\ell-1} \delta_r^2 \le \left(\sum_{r=0}^{\ell-1} \delta_r \right)^2 = \left(t_k^{(\ell)} - t_k \right)^2, \tag{34}$$

so the accumulated local remainders are $\mathcal{O}((t_k^{(\ell)} - t_k)^2)$. Therefore,

$$h_k^{(\ell)} - h_k = \mathcal{O}\left(t_k^{(\ell)} - t_k \right) + \mathcal{O}\left((t_k^{(\ell)} - t_k)^2 \right) \tag{35}$$

$$= \mathcal{O}\left(t_k^{(\ell)} - t_k \right). \tag{36}$$

The last equality follows because the second-order term is absorbed into the first-order bound for a sufficiently small local time span.

By the local Lipschitz continuity of the state Jacobian with respect to (h, x, t), there exists a constant $C_A > 0$ such that

$$\left\|A_k^{(\ell)} - A_k\right\| \le C_A \left(\left\|h_k^{(\ell)} - h_k\right\| + \left\|x_k^{(\ell)} - x_k\right\| + \left|t_k^{(\ell)} - t_k\right|\right). \tag{37}$$

Under the input condition,

$$\left\|x_k^{(\ell)} - x_k\right\| = \mathcal{O}(t_k^{(\ell)} - t_k).$$

Together with Eq. 36, this yields

$$A_k^{(\ell)} = A_k + \mathcal{O}\left(t_k^{(\ell)} - t_k\right). \tag{38}$$

Consequently,

$$\sum_{\ell=0}^{L-1} \delta_\ell A_k^{(\ell)} = \Delta T_k A_k + \sum_{\ell=0}^{L-1} \delta_\ell \mathcal{O}\left(t_k^{(\ell)} - t_k\right) \tag{39}$$

$$= \Delta T_k A_k + \mathcal{O}\left(\sum_{\ell=0}^{L-1} \delta_\ell (t_k^{(\ell)} - t_k)\right) \tag{40}$$

$$= \Delta T_k A_k + \mathcal{O}(\Delta T_k^2), \tag{41}$$

where the last equality follows from $0 \le t_k^{(\ell)} - t_k \le \Delta T_k$ and $\sum_{\ell=0}^{L-1} \delta_\ell = \Delta T_k$. Substituting this result into Eq. 31 proves Eq. 25. □

Proposition 1 (Leading-order approximation of the BPTT gradient)**.** *Define the Jacobian of the SSBP one-step approximation as*

$$\widetilde{J}_k = \frac{\partial \Psi(h_k)}{\partial h_k}. \tag{42}$$

Under Assumption 1,

$$\widetilde{J}_k - J_k = \mathcal{O}(\Delta T_k^2). \tag{43}$$

Given the gradient g_{k+1} at the interval's final state, let

$$g_k = J_k^\top g_{k+1}, \qquad \widetilde{g}_k = \widetilde{J}_k^\top g_{k+1} \tag{44}$$

denote the conventional BPTT gradient and its SSBP approximation, respectively. Then there exists a finite local constant $C_k > 0$ such that

$$\|\widetilde{g}_k - g_k\| \le C_k \Delta T_k^2 \|g_{k+1}\|. \tag{45}$$

Here, $\|\cdot\|$ denotes the Euclidean norm for vectors and its induced operator norm for matrices. The constant C_k depends on the local dynamics and solver but not on the internal step partition.

Proof. Lemma 1 gives

$$J_k = I + \Delta T_k A_k + \mathcal{O}(\Delta T_k^2). \tag{46}$$

From Eq. 18,

$$\widetilde{J}_k = I + \Delta T_k A_k + \mathcal{O}(\Delta T_k^2). \tag{47}$$

Therefore, J_k and $\widetilde{J}_k$ share the same zeroth- and first-order terms, and their difference satisfies Eq. 43. The corresponding gradient difference is

$$\widetilde{g}_k - g_k = \left(\widetilde{J}_k - J_k\right)^\top g_{k+1}. \tag{48}$$

Because the induced norm satisfies $\|Av\| \le \|A\|\|v\|$ for any matrix A and vector v, we obtain

$$\|\widetilde{g}_k - g_k\| \le \left\|\widetilde{J}_k - J_k\right\| \|g_{k+1}\| \tag{49}$$

$$\le C_k \Delta T_k^2 \|g_{k+1}\|, \tag{50}$$

which proves Eq. 45. □

Remark 2 (Interpretation). Proposition 1 shows that the complete solver trajectory and the SSBP one-step approximation share the same leading Jacobian structure,

$$I + \Delta T_k A_k. \tag{51}$$

Thus, SSBP retains the zeroth- and first-order contributions of the complete internal trajectory to gradient propagation, while the difference between $\widetilde{J}_k$ and J_k is of order $\mathcal{O}(\Delta T_k^2)$. For sufficiently small ΔT_k, this second-order discrepancy is small relative to the shared zeroth- and first-order terms, so $\widetilde{J}_k$ closely approximates J_k and, consequently, $\widetilde{g}_k$ closely approximates g_k. This establishes that the leading-order contribution of the internal trajectory to gradient propagation can be captured by a gradient approximation constructed from local information at the interval's initial state.

C Theory Analysis: Positive Alignment Supports Effective Optimization

We next establish a sufficient condition under which the parameter gradient constructed by SSBP provides an effective optimization direction. Let

$$G_\Theta = \nabla_\Theta \mathcal{L}(\Theta) \tag{52}$$

denote the parameter gradient computed by conventional BPTT, and let $\widetilde{G}_\Theta$ denote the parameter-gradient approximation computed by SSBP. We regard all trainable parameters and their gradients as vectors. For nonzero G_Θ and $\widetilde{G}_\Theta$, their directional similarity is

$$\mathrm{CosSim}\left(\widetilde{G}_\Theta, G_\Theta\right) = \frac{\left\langle \widetilde{G}_\Theta, G_\Theta \right\rangle}{\|\widetilde{G}_\Theta\| \|G_\Theta\|}. \tag{53}$$

Proposition 2 (Effective update under positive alignment)**.** *Suppose that $\mathcal{L}$ is differentiable in a neighborhood of Θ and that its gradient is locally Lipschitz continuous in this neighborhood. Consider the SSBP update*

$$\Theta^+ = \Theta - \eta \widetilde{G}_\Theta, \qquad \eta > 0. \tag{54}$$

If

$$\left\langle G_\Theta, \widetilde{G}_\Theta \right\rangle > 0, \tag{55}$$

or equivalently,

$$\mathrm{CosSim}\left(\widetilde{G}_\Theta, G_\Theta\right) > 0 \quad \Longleftrightarrow \quad \angle\left(\widetilde{G}_\Theta, G_\Theta\right) < 90^\circ, \tag{56}$$

then there exists $\bar{\eta} > 0$ such that

$$\mathcal{L}\left(\Theta - \eta \widetilde{G}_\Theta\right) < \mathcal{L}(\Theta) \tag{57}$$

for every $0 < \eta < \bar{\eta}$.

Proof. Define the scalar function

$$q(\eta) = \mathcal{L}\left(\Theta - \eta \widetilde{G}_\Theta\right), \tag{58}$$

which represents the loss along the SSBP update direction as a function of the learning rate η. Under the stated regularity conditions, the first-order Taylor expansion of q around $\eta = 0$ gives

$$q(\eta) = q(0) + \eta q'(0) + \mathcal{O}(\eta^2), \qquad \eta \to 0. \tag{59}$$

At $\eta = 0$,

$$q(0) = \mathcal{L}(\Theta). \tag{60}$$

By the chain rule,

$$q'(0) = -\left\langle \nabla_\Theta \mathcal{L}(\Theta), \widetilde{G}_\Theta \right\rangle = -\left\langle G_\Theta, \widetilde{G}_\Theta \right\rangle. \tag{61}$$

Therefore,

$$\mathcal{L}\left(\Theta - \eta \widetilde{G}_\Theta\right) = \mathcal{L}(\Theta) - \eta \left\langle G_\Theta, \widetilde{G}_\Theta \right\rangle + \mathcal{O}(\eta^2). \tag{62}$$

If

$$\left\langle G_\Theta, \widetilde{G}_\Theta \right\rangle > 0, \tag{63}$$

then the first-order term is strictly negative. Since $\mathcal{O}(\eta^2)/\eta \to 0$ as $\eta \to 0$, the higher-order term is dominated by the negative first-order term for sufficiently small $\eta > 0$. Hence, there exists $\bar{\eta} > 0$ such that, for every $0 < \eta < \bar{\eta}$,

$$\mathcal{L}\left(\Theta - \eta \widetilde{G}_\Theta\right) < \mathcal{L}(\Theta). \tag{64}$$

□

Proposition 2 shows that SSBP does not need to exactly reproduce the magnitude or direction of the BPTT parameter gradient. Positive directional alignment is sufficient for the SSBP gradient approximation to remain descent-compatible and provide an effective optimization direction for a sufficiently small learning rate.

D Empirical Analysis of Gradient Alignment

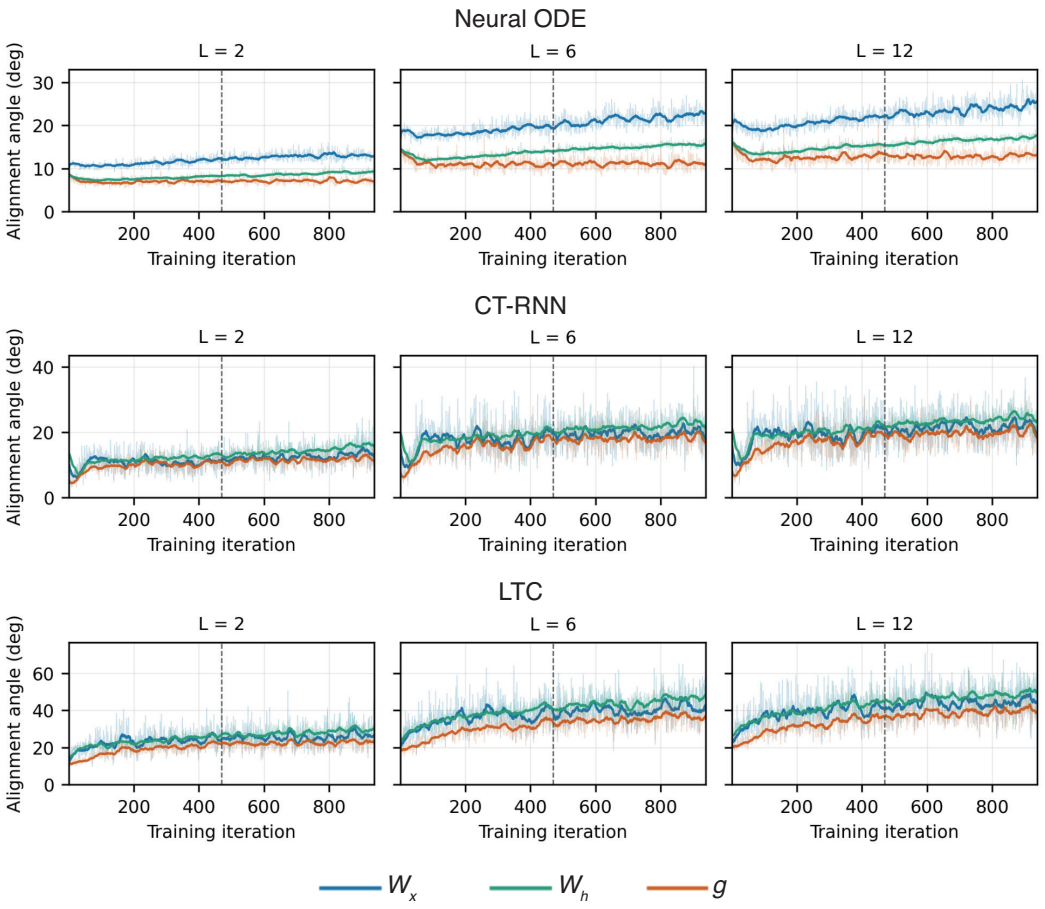

Figure 5: Directional angles between the BPTT gradients and their SSBP approximations during the first two training epochs on SMNIST. At each iteration, both gradients are evaluated using the same BPTT parameter point and mini-batch. Neural ODEs and CT-RNNs use the Euler solver, while LTCs use their fused semi-implicit solver. Angles are shown for the gradients with respect to the input weights W_x and state weights W_h, as well as the state gradient g. Light curves show the mini-batch measurements, dark curves show their moving averages, and dashed vertical lines mark the boundary between the two epochs.

We empirically examine the directional alignment between the gradients computed by conventional BPTT and their approximations constructed by SSBP. We use SMNIST as a representative example and consider Neural ODEs, CT-RNNs, and LTCs with $L \in \{2, 6, 12\}$. Neural ODEs and CT-RNNs are integrated using Euler solver, while LTCs use their fused semi-implicit solver. For each setting, we track gradient alignment over the first two epochs of training. At each training iteration, the SSBP model is synchronized with the BPTT model, and both gradients are computed using the same parameters, forward states, and mini-batch. Only the BPTT gradient is subsequently used to update the model. This procedure ensures the comparison directly reflects the difference between the SSBP approximation and the BPTT gradient at the same model state. We measure the angles between the BPTT gradients and their SSBP approximations for the input weights W_x, state weights W_h, and state gradient g. As shown in Fig. 5, the alignment angles increase gradually over the first two training epochs and are generally larger for settings with more internal solver steps, but remain well below 90° for all three models and all evaluated values of L. The corresponding BPTT gradients and SSBP approximations therefore remain positively aligned. For W_x and W_h, this provides empirical support for the sufficient condition established in Proposition 2. The alignment of g further shows that the state-gradient approximation remains directionally consistent with BPTT.

Because Fig. 5 covers only the first two epochs, we further evaluate gradient alignment using the best-performing models obtained from the complete 120-epoch BPTT and SSBP training runs. Both models are trained with $L = 6$. For each checkpoint, its parameters are loaded into paired BPTT and SSBP implementations. The BPTT gradient and its SSBP approximation are then computed using the same forward states and mini-batch, without updating the model parameters. We repeat this comparison over all 469 mini-batches covering the $60{,}000$ SMNIST training examples and report the resulting angles as mean $\pm$ standard deviation. For the Neural ODE, the BPTT-trained model produces angles of $(47.48 \pm 13.19)^\circ$, $(47.60 \pm 18.48)^\circ$, and $(38.81 \pm 4.44)^\circ$ for W_x, W_h, and g, respectively, whereas the corresponding angles for the SSBP-trained model are $(32.19 \pm 7.69)^\circ$, $(27.89 \pm 7.03)^\circ$, and $(31.67 \pm 6.10)^\circ$. For the CT-RNN, the BPTT-trained model produces angles of $(62.78 \pm 16.72)^\circ$, $(58.10 \pm 15.51)^\circ$, and $(50.65 \pm 8.66)^\circ$, compared with $(48.39 \pm 12.72)^\circ$, $(52.96 \pm 12.64)^\circ$, and $(47.98 \pm 7.03)^\circ$ for the SSBP-trained model. For the LTC, the corresponding angles are $(64.06 \pm 11.93)^\circ$, $(60.53 \pm 10.83)^\circ$, and $(61.35 \pm 7.78)^\circ$ for the BPTT-trained model, and $(45.77 \pm 10.43)^\circ$, $(44.61 \pm 8.37)^\circ$, and $(45.48 \pm 8.60)^\circ$ for the SSBP-trained model. All mean angles remain below 90°. Moreover, the SSBP-trained models exhibit smaller angles than their BPTT-trained counterparts for every model and measured gradient. Thus, despite the gradual increase observed during the first two epochs, positive alignment is preserved in the best-performing models obtained from the complete training runs. Taken together, these results show that the gradient approximations constructed by SSBP remain positively aligned with the corresponding BPTT gradients both during early training and at the best-performing models obtained from complete training runs. This empirical alignment supports the sufficient condition established in Proposition 2 and helps explain why SSBP maintains competitive task performance despite skipping gradient propagation through the solver's internal trajectory.

E SSBP Formulations for Different Model–Solver Settings

This section specifies the SSBP proxy states and the corresponding gradient approximations used for the model–solver settings considered in our experiments. In all settings, the forward state h_{k+1} is computed by the original numerical solver and remains unchanged. SSBP only replaces differentiation through the solver's internal trajectory with differentiation of a one-step proxy evaluated at h_k. For Neural ODEs and CT-RNNs, the same Euler-based one-step approximation of the state evolution is used for gradient computation, regardless of whether forward integration is performed using Euler, RK4, or an adaptive solver. For LTCs, the proxy follows the fused-solver update used in forward integration. This choice follows the leading-order approximation principle established in Appendix B. Euler, RK4, and adaptive solvers share the same first-order Euler term, allowing the Euler-based proxy to capture their leading-order gradient propagation. For LTCs, the fused update is retained to preserve the model-specific semi-implicit structure of their solver. The formulas below are given for a single sample; parameter-gradient contributions are summed over samples and integration intervals during training. We use $\odot$ and $\oslash$ to denote element-wise multiplication and division, respectively.

E.1 NEURAL ODE

The tanh-based Neural ODE considered in this work is defined by

$$\frac{\mathrm{d}h}{\mathrm{d}t} = F_\Theta(h, x) = \tanh\left(W_x x + W_h h + b\right). \tag{65}$$

At the beginning of the k-th integration interval, define

$$z_k = W_x x_k + W_h h_k + b, \qquad r_k = \tanh(z_k), \qquad \phi_k = \tanh'(z_k) = 1 - r_k \odot r_k. \tag{66}$$

Although the forward state is produced by the selected Euler, RK4, or adaptive solver, SSBP uses the following full-interval Euler proxy for backward computation:

$$\widetilde{h}_{k+1} = \Psi_{\mathrm{NODE}}(h_k) = h_k + \Delta T_k r_k. \tag{67}$$

Given the gradient g_{k+1} at the solver-produced forward state h_{k+1}, define the local preactivation-gradient approximation as

$$v_k = \Delta T_k \left(g_{k+1} \odot \phi_k\right). \tag{68}$$

Differentiating Eq. 67 gives the SSBP state-gradient approximation

$$\widetilde{g}_k = g_{k+1} + W_h^\top v_k. \tag{69}$$

The corresponding local input- and parameter-gradient contributions are

$$\begin{aligned} \widetilde{\nabla}_{x_k}\mathcal{L} &= W_x^\top v_k, \\ \widetilde{G}_{W_x,k} &= v_k x_k^\top, \\ \widetilde{G}_{W_h,k} &= v_k h_k^\top, \\ \widetilde{G}_{b,k} &= v_k. \end{aligned} \tag{70}$$

Consequently, the backward computation rule is identical for forward integration using Euler, RK4, or an adaptive solver and does not depend on the corresponding solver's internal steps or stages. This also explains why SSBP provides greater efficiency gains with RK4 and adaptive solvers, whose internal steps require multiple vector-field evaluations, while its memory usage remains nearly the same as with Euler.

E.2 CT-RNN

The CT-RNN dynamics are

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -\tau^{-1} \odot h + \tanh\left(W_x x + W_h h + b\right), \tag{71}$$

where $\tau > 0$ denotes the time-constant vector. To ensure positivity, we use the exponential parameterization $\tau = \exp(\lambda)$. In our experiments, τ is initialized to 1 and held fixed throughout training.

$$\lambda = \log \tau, \qquad \tau^{-1} = \exp(-\lambda). \tag{72}$$

Using z_k, r_k, and ϕ_k as defined in Eq. 66, SSBP employs the following full-interval Euler proxy:

$$\widetilde{h}_{k+1} = \Psi_{\text{CT-RNN}}(h_k) = h_k + \Delta T_k \left(-\tau^{-1} \odot h_k + r_k\right). \tag{73}$$

Similarly, this proxy is used for backward computation regardless of whether the forward state is obtained using Euler, RK4, or adaptive integration.

With

$$v_k = \Delta T_k \left(g_{k+1} \odot \phi_k\right), \tag{74}$$

the resulting state-gradient approximation is

$$\widetilde{g}_k = \left(1 - \Delta T_k \tau^{-1}\right) \odot g_{k+1} + W_h^\top v_k. \tag{75}$$

The input- and parameter-gradient contributions are

$$\begin{aligned} \widetilde{\nabla}_{x_k}\mathcal{L} &= W_x^\top v_k, \\ \widetilde{G}_{W_x,k} &= v_k x_k^\top, \\ \widetilde{G}_{W_h,k} &= v_k h_k^\top, \\ \widetilde{G}_{b,k} &= v_k, \\ \widetilde{G}_{\lambda,k} &= \Delta T_k \left(g_{k+1} \odot h_k \odot \tau^{-1}\right). \end{aligned} \tag{76}$$

The last expression is omitted when the time constants are fixed.

E.3 LTC WITH THE FUSED SEMI-IMPLICIT SOLVER

The tanh-based LTC dynamics are

$$\frac{\mathrm{d}h}{\mathrm{d}t} = -\left(\tau^{-1} + r(h,x)\right) \odot h + r(h,x) \odot A, \tag{77}$$

where

$$r(h,x) = \tanh\left(W_x x + W_h h + b\right), \tag{78}$$

and A is a trainable LTC parameter. To ensure $\tau > 0$, the time constant is parameterized as $\tau = \exp(\lambda)$, where λ is the trainable log-time-constant parameter.

For forward integration, the interval is divided into L internal steps of size $\delta_k = \Delta T_k / L$. Starting from $h_k^{(0)} = h_k$, the fused semi-implicit solver applies

$$\begin{aligned} r_k^{(\ell)} &= \tanh\left(W_x x_k + W_h h_k^{(\ell)} + b\right), \\ d_k^{(\ell)} &= 1 + \delta_k \left(\tau^{-1} + r_k^{(\ell)}\right), \\ h_k^{(\ell+1)} &= \left(h_k^{(\ell)} + \delta_k r_k^{(\ell)} \odot A\right) \oslash d_k^{(\ell)}, \end{aligned} \tag{79}$$

and returns $h_{k+1} = h_k^{(L)}$.

Rather than differentiating through these L fused steps, SSBP applies the same fused update once over the entire interval. Let

$$\begin{aligned} z_k &= W_x x_k + W_h h_k + b, \\ r_k &= \tanh(z_k), \\ \phi_k &= \tanh'(z_k) = 1 - r_k \odot r_k, \\ \widetilde{d}_k &= 1 + \Delta T_k \left(\tau^{-1} + r_k\right). \end{aligned} \tag{80}$$

The LTC proxy state is then

$$\widetilde{h}_{k+1} = \Psi_{\mathrm{LTC}}(h_k) = (h_k + \Delta T_k r_k \odot A) \oslash \widetilde{d}_k. \tag{81}$$

Define the corresponding local preactivation-gradient approximation as

$$v_k = \left[\Delta T_k g_{k+1} \odot \left(A - \widetilde{h}_{k+1}\right) \oslash \widetilde{d}_k\right] \odot \phi_k. \tag{82}$$

Differentiating Eq. 81 gives

$$\widetilde{g}_k = g_{k+1} \oslash \widetilde{d}_k + W_h^\top v_k. \tag{83}$$

The corresponding input- and parameter-gradient contributions are

$$\begin{aligned} \widetilde{\nabla}_{x_k} \mathcal{L} &= W_x^\top v_k, \\ \widetilde{G}_{W_x,k} &= v_k x_k^\top, \\ \widetilde{G}_{W_h,k} &= v_k h_k^\top, \\ \widetilde{G}_{b,k} &= v_k, \\ \widetilde{G}_{A,k} &= \Delta T_k \left(g_{k+1} \odot r_k \oslash \widetilde{d}_k\right), \\ \widetilde{G}_{\lambda,k} &= \Delta T_k \left(g_{k+1} \odot \widetilde{h}_{k+1} \odot \tau^{-1} \oslash \widetilde{d}_k\right). \end{aligned} \tag{84}$$

Therefore, the Neural ODE and CT-RNN settings use a solver-independent one-step Euler proxy, whereas the LTC setting uses a one-step proxy matched to its fused semi-implicit solver. In every case, the original forward state is retained, and the proxy state is used only to construct the gradient approximation during the backward process. Consequently, the backward computation requires neither storing nor recomputing the solver's internal trajectory.

F STO Simulator and Optimization Details

F.1 Coupled STO Dynamics

We construct the coupled macrospin-type spin-torque oscillator (STO) simulator following Imai et al. (2025). The dynamics of a single STO are governed by the Landau–Lifshitz–Gilbert (LLG) equation:

$$\begin{aligned}\frac{\mathrm{d}\mathbf{m}}{\mathrm{d}t} = &-\gamma H_{\mathrm{appl}}\mathbf{m}\times\mathbf{e}_z - \gamma(H_K - 4\pi M)m_z\mathbf{m}\times\mathbf{e}_z \\ &-\gamma H_s(t)\mathbf{m}\times(\mathbf{p}\times\mathbf{m}) + \alpha\mathbf{m}\times\frac{\mathrm{d}\mathbf{m}}{\mathrm{d}t},\end{aligned} \tag{85}$$

where $\mathbf{m}$ denotes the magnetization and

$$H_s(t) = \frac{\hbar\eta j_{\mathrm{DC}}}{2e\,(1+\lambda\mathbf{m}\cdot\mathbf{p})\,M\pi R^2 d}. \tag{86}$$

Here, $\hbar$ is the Dirac constant, e is the elementary charge, and $\mathbf{p}$ denotes the magnetization direction of the fixed layer, which is aligned with the x-axis.

For the coupled system, the coupling field applied to the i-th STO is

$$\mathbf{H}_i^{\mathrm{cp}} = A_{\mathrm{cp}}\sum_{j=1}^{N} w_{ij}^{\mathrm{cp}} m_{j,x}\mathbf{e}_x. \tag{87}$$

where $N = 200$ and w_{ij}^{cp} is an element of the coupling matrix W^{cp}, which is normalized to have unit spectral radius. Accordingly, A_{cp} determines the coupling magnitude (CM). We use $A_{\mathrm{cp}} = 100$ in our experiments. The definitions and values of the physical parameters are summarized in Table 3.

F.2 Experimental Settings

For the MNIST task, the coupled STO system is connected to the input and output layers as

$$x(0) = W_{\mathrm{in}}u^{(n)}, \tag{88}$$

$$x(t) = x(0) + \int_0^t \frac{\mathrm{d}x(s)}{\mathrm{d}s}\,\mathrm{d}s, \tag{89}$$

and

$$y^{(n)} = W_{\mathrm{out}}\begin{bmatrix}x(T)\\1\end{bmatrix}, \tag{90}$$

where $u^{(n)}$ is the n-th input image, $x(t)$ denotes the state of the coupled STO system, and $y^{(n)}$ is the corresponding output.

In our implementation, $x(t)\in\mathbb{R}^{3N}$ contains the three magnetization components of all STOs. The output of the input layer is reshaped into N three-dimensional magnetization vectors, each of which is normalized before simulation. The coupled dynamics are integrated for $T = 500\,\mathrm{ps}$ using a fixed-step RK4 solver with 10 steps of 50 ps.

Following the full-tuning setting, we jointly optimize W_{in}, W_{out}, the individual applied fields $H_{\mathrm{appl},i}$, the shared current j_{DC}, and the shared Gilbert damping α. Their initial values are $H_{\mathrm{appl},i} = 200\,\mathrm{Oe}$, $j_{\mathrm{DC}} = 2.5\times10^{-3}\,\mathrm{A}$, and $\alpha = 5.0\times10^{-3}$. The remaining physical parameters are listed in Table 3.

All trainable parameters are optimized using Adam with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The learning rates are 10^{-3} for the input and output layers, 10^{-2} for the applied fields, and 10^{-6} for both the current and Gilbert damping. All learning rates are multiplied by 0.85 after each epoch. We train for 20 epochs using cross-entropy loss and a batch size of 128. The same RK4 forward integration is used for BPTT, the adjoint method, and SSBP; these methods differ only in their backward gradient computation.

Table 3: Physical parameters of the coupled STO simulator.

Parameter	Symbol	Value
Gyromagnetic ratio	γ	$1.764 \times 10^7\,\mathrm{rad/(Oe\,s)}$
Saturation magnetization	M	$1448.3\,\mathrm{emu/cm^3}$
Anisotropy field	H_K	$1.8616 \times 10^4\,\mathrm{Oe}$
Thickness	d	$2.0 \times 10^{-7}\,\mathrm{cm}$
Radius	R	$60.0 \times 10^{-7}\,\mathrm{cm}$
Spin polarization	η	0.537
Torque asymmetry	λ	0.288
Coupling magnitude	A_{cp}	100

G Pseudocode for SSBP

Algorithm 1 summarizes the training procedure of SSBP. During the forward pass, the state evolution is computed entirely by the original solver. During the backward pass, gradient propagation through each solver-internal trajectory is replaced by its corresponding one-step approximation. Therefore, the backward computation does not depend on the number of internal solver steps or stages. The model-specific one-step approximations are provided in Appendix E.

Let $X = (x_0, \ldots, x_{T-1})$ denote an input sequence of length T. Each sequence element x_k is associated with an integration interval $[t_k, t_{k+1}]$, over which the forward solver advances the state from h_k to h_{k+1}. Within this interval, the solver performs L_k internal steps indexed by ℓ. For fixed-step solvers, $L_k = L$ for all intervals, whereas L_k may vary under adaptive integration.

Algorithm 1: Training with Solver-Skipping Backpropagation

Require: Input sequence $X = (x_0, \ldots, x_{T-1})$, target, CTNN model, forward solver, model-specific one-step approximation, and optimizer

Initialize the state h_0
for $k = 0, \ldots, T-1$ **do**
  Retain only h_k, x_k, t_k, and ΔT_k for backward computation
  Set the solver-initial state $h_k^{(0)} \leftarrow h_k$
  for $\ell = 0, \ldots, L_k - 1$ **do**
    Compute the next internal solver state $h_k^{(\ell+1)}$ using the original forward solver
  end for
  Set the interval output $h_{k+1} \leftarrow h_k^{(L_k)}$
end for
Compute the model output and loss from the forward states
Compute the loss gradient with respect to the final state h_T
for $k = T-1, \ldots, 0$ **do**
  Construct the one-step approximation using the local information evaluated at the initial state h_k of the interval $[t_k, t_{k+1}]$
  Backpropagate the gradient with respect to h_{k+1} through the one-step approximation
  Accumulate the parameter-gradient approximation
  Pass the resulting state gradient to h_k
end for
Update all trainable parameters using the optimizer

H Dataset Descriptions and Preprocessing

We follow the dataset preprocessing protocols used by Hasani et al. (2021). Predictions are made and evaluated at every time step for all tasks except SMNIST, which is treated as a sequence-classification task. The irregularly sampled PA setting follows the timestamp- and mask-based representation used by Rubanova et al. (2019).

Sequential MNIST (SMNIST). We use the standard MNIST training and test sets (Le Cun et al., 1998) and scale pixel values to $[0, 1]$. Each 28×28 image is interpreted row by row as a sequence of length 28, with 28 input features at each time step. The final state is used to classify the sequence into one of the ten digit classes. We reserve 10% of the original training set for validation.

Hand Gesture Segmentation (Gesture). The Gesture dataset consists of seven recordings of hand movements (Wagner et al., 2014). Each time step contains 32 motion-sensor features and is assigned to one of five gesture phases: rest, preparation, stroke, hold, or retraction. Each recording is divided into 50%-overlapping sequences of length 32. The features are standardized to zero mean and unit variance, and the resulting sequences are randomly divided into training, validation, and test sets using a 75:10:15 ratio.

Room Occupancy (Occ.). The Occupancy dataset contains binary room-occupancy labels and five sensor features: temperature, humidity, light, CO_2 concentration, and humidity ratio (Candanedo & Feldheim, 2016). Measurements are sampled at one-minute intervals. We retain the predefined training and test traces and normalize all features using the mean and standard deviation of the raw training trace. The traces are divided into sequences of length 32 using a stride of 8. We randomly reserve 10% of the training sequences for validation and combine the two predefined test traces to form the test set.

Human Activity Recognition (HAR). The HAR dataset contains smartphone inertial measurements from 30 participants performing six activities (Anguita et al., 2013). Each time step is represented by the 561 preprocessed features provided with the dataset. We retain the official training and test split and divide both sets into overlapping sequences of length 32, using strides of 1 and 8 for the training and test traces, respectively. We randomly reserve 10% of the training sequences for validation. No additional feature normalization is applied.

Traffic Estimation (Traffic). The Traffic task uses hourly measurements from the Metro Interstate Traffic Volume dataset (Hogue, 2019) to predict westbound traffic volume on Interstate 94. We selected the seven columns of the data as input features: a holiday indicator, centered temperature, rainfall, snowfall, cloud coverage, the weekday index, and a sinusoidal encoding of the hour of day. The traffic-volume target is standardized to zero mean and unit variance. The trace is divided into partially overlapping sequences of 32 hours using a stride of 8, followed by a random 75:10:15 training–validation–test split. Mean squared error (MSE) is used as both the training objective and evaluation metric.

Person Activity (PA). The PA dataset contains 25 recordings collected from four body-worn sensors and includes three-dimensional position measurements for different human activities (Rubanova et al., 2019). The original activity labels are merged into seven classes.

For the regularly sampled setting, each observation is represented by a seven-dimensional vector consisting of a four-dimensional one-hot sensor identifier and the corresponding three-dimensional sensor value. Following Hasani et al. (2021), timing information is discarded and no input normalization is applied. Each recording is divided into 50%-overlapping sequences of length 32, and the resulting sequences are randomly divided into training, validation, and test sets using a 75:10:15 ratio.

For the irregularly sampled setting, observations sharing the same quantized timestamp are grouped together. The input at each time point contains 12 sensor values, corresponding to four sensors with three coordinates each, together with a 12-dimensional binary observation mask. Missing sensor values are filled with zero, resulting in a 24-dimensional input representation. Each sequence retains its own time grid, and the elapsed times between consecutive observations define the integration intervals. We use sequences of length 50 with a stride of 25 and randomly divide them into training and test sets using an 80:20 ratio. No coordinate normalization is applied.

I Experimental Settings

Common setup. All sequence and time-series models consist of a single dense continuous-time recurrent layer followed by a linear output layer. The initial hidden state is set to zero. SMNIST

is treated as a sequence-classification task, for which the linear readout is applied only to the final hidden state. For the remaining sequence-labeling and time-series tasks, the readout and loss are evaluated at every time step. Classification models are trained using cross-entropy loss, whereas the Traffic models are trained using mean squared error. We use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and zero weight decay. The gradient norm is clipped to 1.0. The learning rates listed in Table 4 are held fixed throughout training. For all three CTNN models, the input weights are initialized using Xavier uniform initialization, the recurrent weights are initialized orthogonally and subsequently scaled by 0.9, and the recurrent bias is initialized to zero. The linear output layer uses the Kaiming uniform initialization. For LTCs, the parameter A is initialized to 1 and optimized during training. Task-specific model sizes, training hyperparameters, and initial time constants are summarized in Table 4.

Table 4: Task-specific model and training settings. H, T, and B denote the hidden size, sequence length, and training batch size, respectively.

					Learning rate		
Task	**H**	**T**	**Epochs**	**B**	**Neural ODE**	**CT-RNN**	**LTC**
SMNIST	500	28	120	128	2.5×10^{-4}	2.5×10^{-4}	5.0×10^{-4}
Gesture	128	32	120	32	1.0×10^{-3}	1.0×10^{-3}	1.0×10^{-2}
Occupancy	64	32	120	32	1.0×10^{-4}	1.0×10^{-4}	8.0×10^{-4}
HAR	128	32	120	32	5.0×10^{-4}	5.0×10^{-4}	1.0×10^{-4}
Traffic	128	32	120	32	1.5×10^{-3}	1.5×10^{-3}	2.5×10^{-3}
PA (regular)	128	32	300	64	5.0×10^{-4}	5.0×10^{-4}	7.5×10^{-4}
PA (irregular)	128	50	300	64	7.5×10^{-4}	7.5×10^{-4}	1.5×10^{-3}

Fixed-step solver experiments. For regularly sampled tasks, the duration of each integration interval is fixed at $\Delta T = 0.6$. Neural ODEs and CT-RNNs are evaluated using both Euler and RK4 solvers, whereas LTCs use their fused semi-implicit solver. Unless otherwise stated, each integration interval contains $L = 6$ internal solver steps, corresponding to an internal step size of 0.1. For irregularly sampled PA, the interval duration is determined by the time difference between consecutive observations, and each interval is divided into $L = 6$ internal steps. For the effectiveness and efficiency evaluation in Section 5.1, we use a 500-hidden-unit Neural ODE on SMNIST and vary the number of internal solver steps over $L \in \{2, 4, 6, 8, 10, 12\}$. All remaining settings follow the SMNIST configuration in Table 4.

Regarding the time-constant τ settings, Neural ODEs have no time-constant parameter. For CT-RNNs, all time constants are set to 1 and held fixed throughout training. For LTCs, the time constants are trainable and initialized to 1 for all tasks except PA, where they are initialized to 0.25 under both regular and irregular sampling.

For LTC on Gesture, we use a decay-preserving variant of SSBP because we empirically found this fine-grained phase-segmentation task to be particularly sensitive to the accumulation of state decay across the fused solver's internal steps. Using the full interval duration $\Delta T = 0.6$ in a single fused approximation gives a relatively large approximation step and only coarsely represents the accumulated decay. This coarse approximation results in an accuracy gap of approximately two percentage points relative to BPTT and less stable optimization across runs. We therefore evaluate the effective time constant $\tau_{\text{eff},k}$ at the interval's initial state and replace the one-step decay factor $(1 + \Delta T_k/\tau_{\text{eff},k})^{-1}$ with its accumulated L-step form $(1 + \Delta T_k/(L\tau_{\text{eff},k}))^{-L}$. The forward solve remains unchanged, and this variant does not store or recompute the solver-internal trajectory. All other LTC experiments use the standard one-step fused approximation for the SSBP backward pass.

Adaptive-solver experiment. The adaptive-solver experiment uses the same 500-hidden-unit Neural ODE and SMNIST training configuration as the fixed-step experiment.

Evaluation protocol. For each run of a classification task, we report the highest test accuracy across all epochs. For Traffic, we report the lowest test MSE across all epochs. Each configuration is repeated over five independent runs, and the reported mean and standard deviation are computed

Table 5: Settings for the adaptive-solver experiment on SMNIST.

Setting	**Value**
Model	Neural ODE
Hidden size	500
Sequence length	28
Training epochs	120
Training batch size	128
Learning rate	2.5×10^{-4}
Integration-interval duration	$\Delta T = 0.6$
Solver	Dopri5
Relative tolerance	10^{-5}
Absolute tolerance	10^{-8}
Initial solver step size	0.6
Minimum solver step size	10^{-5}
Maximum step attempts per interval	100
Step-size safety factor	0.9
Step-size adjustment range	$[0.2, 5.0]$

from the resulting per-run best test performance. Per-epoch training runtime measures only the training phase and excludes validation and test evaluation. The first epoch is treated as warm-up. Within each run, we report the median runtime and the largest peak GPU memory allocation over the following five epochs. The resulting per-run values are then averaged across five runs. For the adaptive-solver experiment, runtime and peak memory are recorded at every training epoch and averaged across the five runs at each epoch.

All experiments are performed on an HPC server running Ubuntu 20.04.6 LTS and equipped with two Intel Xeon Platinum 8360Y CPUs operating at 2.40 GHz, 2.0 TiB RAM, and 8 NVIDIA A100 80GB PCIe GPUs. The code is implemented in Python 3.11.13 and PyTorch 2.9.1 with CUDA 12.8 and cuDNN 9.10.2.