Curriculum Multiple Shooting for Robust Training of Neural and Universal Differential Equations

Sebastian Persson[*]
Dynamics of Living Systems Laboratory
The Francis Crick Institute

Giacomo Fabrini[*]
Institute for Chemical and Bioengineering
Department of Chemical and Applied Biosciences
ETH Zurich
giacomo.fabrini@chem.ethz.ch

Branwen Snelling
Dynamics of Living Systems Laboratory
The Francis Crick Institute
branwen.snelling@crick.ac.uk

Fabian Fröhlich[†]
Dynamics of Living Systems Laboratory
The Francis Crick Institute
fabian.frohlich@crick.ac.uk

Abstract

Neural ordinary differential equations (NODEs) and universal differential equations (UDEs) provide flexible and popular frameworks for learning interpretable dynamical systems from noisy time-series data. However, training these models remains challenging, and versatile methods that robustly handle sparse and noisy data as well as partially observed models are lacking. To address this, we introduce curriculum multiple shooting (CMS), a general-purpose training strategy for fitting ordinary differential equation (ODE) models to time-series data by integrating curriculum learning with multiple shooting. Across fourteen benchmarks spanning simulated and real data, and covering NODEs, UDEs, and mechanistic ODEs, CMS improves training robustness, often reaching lower training error in fewer optimisation epochs than state-of-the-art-approaches, and ranks among the best methods in generalisation. Finally, we discuss possible explanations for the strong performance of CMS in light of contemporary theories of what makes training on time-series challenging.

1 Introduction

Using mathematical models to predict the time-evolution of dynamical systems is a central challenge across many scientific and engineering disciplines. A classical approach is to iteratively construct mechanistic ordinary differential equation (ODE) models and evaluate them by fitting them to time-series data. As this process is labour-intensive, there is growing interest in using machine learning (ML) to learn interpretable dynamical models directly from data. In particular, neural ODEs (NODEs) combine the widely used ODE-based framework to model dynamical systems with the flexibility of neural networks [1]. Universal differential equations (UDEs), also called grey-box models or hybrid NODEs [2–4], extend this idea by modelling known components mechanistically and unknown components with neural networks [5]. This makes UDEs attractive when prior knowledge is available and/or when data are limited. Accordingly, NODEs and UDEs are increasingly used across domains including biology [6–8], pharmacology/medicine [4, 9–11], epidemiology [12–14], and engineering applications such as automotive modelling [15].

[*]Equal contribution.
[†]Corresponding author.

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

Despite their conceptual appeal, NODEs and UDEs are difficult to train on time-series data [16, 17]. One line of research addresses this challenge by imposing restrictions on what the ML model can learn [18–22]. Here, we instead focus on complementary training strategies given a fixed model structure. Existing strategies include methods from mechanistic ODE training, such as collocation and multiple shooting [23, 24], as well as ML-inspired strategies like curriculum learning and homotopy-based training [16, 25]. However, some strategies trade off applicability against practicality. For example, collocation and homotopy based methods rely on derivative matching from interpolated state trajectories that requires densely sampled data and full observation of the ODE model states [26]. In many application domains [11, 13, 27], however, time-series data are noisy, sparse, irregularly sampled, and the model is often only partially observed. Partial observability can further arise from the model architecture itself, as in augmented NODEs [28]. Meanwhile, hyperparameter tuning can be non-trivial for more broadly applicable training strategies such as multiple shooting [24]. Ideally, a training strategy should be broadly applicable across these settings and robust to tuning, allowing practitioners to focus on modelling rather than training.

Our contributions are threefold. First, we introduce *curriculum multiple shooting* (CMS), a general-purpose training strategy for fitting mechanistic ODE, NODE, and UDE models to noisy time-series data in both fully and partially observed settings. Second, we benchmark this method against state-of-the-art training strategies on six NODE/UDE benchmarks using simulated and real data, as well as on eight real-data mechanistic ODE benchmarks. These benchmarks span partial observability, multiscale and stiff dynamics, and noisy or irregularly sampled time-series data. Third, we analyse optimisation traces and loss landscapes to gain intuition for why CMS often improves training.

2 Related work

Training mechanistic ODEs. A large literature addresses fitting mechanistic ODE models to time-series data. In biology, empirical benchmarks have found that multistart parameter estimation with local second-order optimizers applied to a single-shooting objective (Eq. 2) performs well in practice [27, 29]. Beyond standard single shooting, other training strategies include multiple shooting [30]; collocation methods, which approximate state derivatives from interpolated data and match them to the ODE model right-hand side [26]; and prediction-error [31] and synchronization-based methods [32], both of which use data interpolation to guide model simulations toward the data. All of these strategies have also been tested on NODEs and/or UDEs [12, 14, 16, 21, 23, 24].

Improving performance for NODEs and UDEs. Several approaches have been proposed to improve performance. One is to constrain what the ML component can learn, either through soft constraints in the objective via regularization or auxiliary loss terms [4, 6, 19, 20, 22, 33, 34], or through hard constraints encoded directly in the model structure [18, 21, 33, 35]. State augmentation has likewise been proposed to make the underlying dynamics easier to simulate [28]. Several of these methods aim to stabilize training by learning better-conditioned dynamics, which can help mitigate training instabilities such as exploding/vanishing gradients [18–20, 22]. Another line of work focuses on improving training via gradient computation through the ODE solver [36–39]. Roughly, the two main gradient schemes are discretize-then-optimize and optimize-then-discretize. While benchmarks remain limited, the choice of gradient scheme has been suggested to affect training performance in addition to runtime [36]. In App. A.5, we show that our results are robust across both schemes. Complementary to these approaches, we focus on the training strategy itself. Beyond aforementioned methods such as multiple shooting [12], NODE/UDE-targeted strategies have also been proposed. Curriculum learning, often referred to as "growing fits", has been used but remains under-benchmarked [25, 40]. Related sliding-window training [8] and homotopy-based synchronization [16] have also been applied.

Relation to existing training strategies CMS differs from curriculum and multiple-shooting training by progressively merging shooting windows until recovering the single-shooting objective. Homotopy-based training [16] instead anneals a data-dependent term in the vector field, while differentiable multiple-shooting layers [39] focus on solving the initial-value problem and differentiating through its solution. The contemporaneous MPINeuralODE approach [41] combines curriculum and multiple shooting within a fixed window structure.

3 Problem setup and preliminaries

This section introduces the model and data setting, and the two training strategies (curriculum learning and multiple shooting) forming the basis of our curriculum multiple shooting strategy.

3.1 Training setup for ODE-based models

We consider training ordinary differential equations (ODEs) augmented with neural networks on noisy time-series data. These models, known as universal differential equations (UDEs), also called grey-box models or hybrid NODEs [5], describe the time evolution of a state vector $\mathbf{u} \in \mathbb{R}^M$ as:

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{f}\big(\mathbf{u}, t, \boldsymbol{\theta}_m, \mathbf{NN}(\mathbf{u}, t, \boldsymbol{\theta}_n)\big), \quad \mathbf{u}(t_0) = \mathbf{u}_{t_0}(\boldsymbol{\theta}_m), \tag{1}$$

where $\mathbf{f}$ combines known mechanistic structure with unknown mechanistic parameters $\boldsymbol{\theta}_m$ and a neural network $\mathbf{NN}$ with unknown parameters $\boldsymbol{\theta}_n$. In the absence of a mechanistic component, i.e., when $\mathbf{f} = \mathbf{NN}(\mathbf{u}, t, \boldsymbol{\theta}_n)$, Eq. 1 reduces to a standard NODE [1].

Training Eq. 1 corresponds to estimating the unknown parameters $(\boldsymbol{\theta}_m, \boldsymbol{\theta}_n)$ by fitting the model to noisy time-series data. We assume data are available for O observables, where for each observable $o = 1, \ldots, O$, measurements are taken at time points $\{t_{o,j}\}_{j=1}^{T_o}$ and the corresponding model output is given by the possibly nonlinear observation function $h_o(\mathbf{u}(t_{o,j}), \boldsymbol{\theta}_m)$. In many applications, such as biology and pharmacology [27, 42], models are only partially observed ($O < M$) and data are irregularly sampled. Both settings are considered in our experiments (Sec. 5).

Given this observation model, the parameters in Eq. 1 are estimated by minimizing a single-shooting objective function that quantifies the mismatch between model output and time-series data:

$$g = \sum_{o=1}^{O} \sum_{j=1}^{T_o} g_o\big(h_o(\mathbf{u}(t_{o,j}), \boldsymbol{\theta}_m), \tilde{m}_{o,t_{o,j}}\big), \tag{2}$$

where $\tilde{m}_{o,t_{o,j}}$ denotes the noisy measurement of observable o at time $t_{o,j}$. Common choices for g_o include mean squared error (MSE) and likelihood-based objectives. When fitting UDEs, optimizing only Eq. 2 can allow the neural component to dominate the dynamics and "cannibalise" the mechanistic component [43]. One way to mitigate this is to add neural-network output regularization [21]:

$$g_{\text{out}} = \lambda_O \left(\int_{t_0}^{T} \|\mathbf{NN}(\mathbf{u}(t), t, \boldsymbol{\theta}_n)\|_2 \, \mathrm{d}t \right)^2, \tag{3}$$

where λ_O controls the regularization strength. When used, g_{out} is added to the objective in Eq. 2.

In general, evaluating Eq. 2 requires numerically solving Eq. 1 with an ODE solver. A possible alternative is to reconstruct interpolants $\tilde{\mathbf{u}}(t)$ and $\mathrm{d}\tilde{\mathbf{u}}/\mathrm{d}t$ from the measurements, and then directly compare $\mathrm{d}\tilde{\mathbf{u}}/\mathrm{d}t$ to the right-hand side of Eq. 1. We do not consider that setting here, since it requires the full state $\mathbf{u}$ to be observed, which is uncommon in many application domains like biology [27].

3.2 Curriculum learning for ODE-based models

Curriculum learning (CL) is a training strategy in which a model is trained over n_c stages on progressively harder tasks [44, 45]. What constitutes a harder task depends on the application. Since training often becomes more difficult over longer time horizons [16, 46] for differential equations of the form in Eq. 1, a natural curriculum is to train on progressively longer time intervals.

More formally, if measurements are available on $[t_0, T]$, a curriculum with n_c stages is defined by the intervals $[t_0, t_1], \ldots, [t_0, t_i], \ldots, [t_0, t_{n_c}]$, where $t_{n_c} = T$. At stage i, the model is trained using only measurements in $[t_0, t_i]$.

3.3 Multiple shooting for ODE-based models

Multiple shooting (MS) is a training strategy in which the time interval $[t_0, T]$ is divided into n_w windows, and the model is fitted separately on each window [24, 30]. As with curriculum learning, the motivation is that fitting multiple shorter trajectories is typically easier than fitting a long trajectory.

To define the MS training objective, we partition the time interval $[t_0, T]$ into n_w windows with boundaries $t_0 < t_1 < \cdots < t_{n_w} = T$. We denote the resulting set of windows by $\mathcal{W}_{n_w} = \{w_i\}_{i=1}^{n_w}$, where $w_i := [t_{i-1}, t_i]$. For ease of notation, we drop the observable index o from Eq. 2, write $(t_j, \tilde{m}_j)$ for the resulting measurement time points and values, and let $\mathcal{M}_i := \{\, t_j : t_j \in w_i \,\}$ denote the measurement time points in w_i. As we assume accurate interpolants of the model state $\mathbf{u}$ are unavailable (Sec. 3.1), we introduce additional parameters to estimate, $\boldsymbol{\theta}_{u_0,i}$, for the initial state of each window. The multiple-shooting objective is computed by solving the model on each window:

$$\frac{\mathrm{d}\mathbf{u}^{(i)}}{\mathrm{d}t} = \mathbf{f}\big(\mathbf{u}^{(i)}, t, \boldsymbol{\theta}_m, \mathbf{NN}(\mathbf{u}^{(i)}, t, \boldsymbol{\theta}_n)\big), \qquad \mathbf{u}^{(i)}(t_{i-1}) = \boldsymbol{\theta}_{u_0,i}, \qquad t \in w_i, \tag{4}$$

followed by summing the data-fitting loss over windows:

$$g_{\mathrm{MS}} = \sum_{i=1}^{n_w} \sum_{t_j \in \mathcal{M}_i} g\Big(h\Big(\mathbf{u}^{(i)}(t_j), \boldsymbol{\theta}_m\Big), \tilde{m}_j\Big). \tag{5}$$

To enforce continuity between windows, a quadratic penalty is typically added [14, 25]. While plain MS penalizes discrepancies at the window boundary, to allow longer overlapping windows we define the penalty on shared measurement time points between windows, i.e. $\mathcal{P}_i := \mathcal{M}_i \cap \mathcal{M}_{i+1}$:

$$g_{\mathrm{cont}} = \lambda_{\mathrm{cont}} \sum_{i=1}^{n_w - 1} \sum_{t \in \mathcal{P}_i} \left\| \mathbf{u}^{(i)}(t) - \mathbf{u}^{(i+1)}(t) \right\|_2^2. \tag{6}$$

Here, λ_{cont} controls the penalty strength. While other ways of handling continuity have been proposed [24, 30, 47], we use this common formulation [25], which our CMS strategy builds on.

4 Curriculum multiple shooting

Our curriculum multiple shooting (CMS) strategy aims to combine the strengths of curriculum learning (CL; Sec. 3.2) and multiple shooting (MS; Sec. 3.3) while mitigating their main limitations. A general challenge in CL is to design curriculum stages that are representative of the final objective [44, 45]. When the time-series data in early CL stages are not representative of the full trajectory, any overfitting incurred there must be corrected during later stages, potentially slowing training. In contrast, MS exposes the full trajectory from the start, but never directly trains on the final single-shooting objective, instead relying on the hard-to-tune continuity penalty (λ_{cont} Eq. 6) to make the MS objective a good surrogate. CMS addresses this by starting from the MS objective and progressively merging shooting windows across curriculum stages until recovering single shooting. This yields easier initial optimization on short trajectories, retains a full-trajectory view throughout training, and gradually transitions to the final objective which reduces sensitivity to λ_{cont}.

Specifically, CMS starts from a multiple-shooting objective with n_w windows spanning $[t_0, T]$. At curriculum stage $s \in \{1, \ldots, n_w\}$, the window set is $\mathcal{W}^{(s)} = \{\, [t_{i-1},\, t_{i+s-1}] \,\}_{i=1}^{n_w - s + 1}$, so adjacent windows are merged into longer overlapping windows. The number of windows thereby decreases by one per curriculum stage, and at the final stage $s = n_w$ only a single window $[t_0, T]$ remains, recovering the standard single-shooting objective (Eq. 2). When the windows grow, the data-fitting term (Eq. 5) increasingly promotes agreement between neighboring windows in addition to the continuity penalty (Eq. 6), reducing sensitivity to λ_{cont}. Pseudocode is given in Alg. 1.

While Alg. 1 is presented with a gradient-based optimizer (e.g. Adam), the method is compatible with most local optimizers; for example, in the ODE experiments (Sec. 5) we use a Newton trust-region method. We also found that, instead of penalizing all shared measurement points (Eq. 6), penalizing only the first overlapping point as in plain multiple shooting also works (App. A.5).

Algorithm 1 Curriculum multiple shooting (CMS)

Input: Time-series data $\mathcal{D} = \{(t_j, \tilde{m}_j)\}_{j=1}^{M}$, initial model parameters $(\boldsymbol{\theta}_m^{(0)}, \boldsymbol{\theta}_n^{(0)})$
Hyperparameters: Initial number of windows n_w, continuity penalty λ_{cont}, cumulative stage epoch budget $\{E_s\}_{s=1}^{n_w}$ with $E_1 < \cdots < E_{n_w}$, learning rate η, initial window values $\{\boldsymbol{\theta}_{u_0,i}^{(1)}\}_{i=1}^{n_w}$

Initialize $(\boldsymbol{\theta}_m, \boldsymbol{\theta}_n) \leftarrow (\boldsymbol{\theta}_m^{(0)}, \boldsymbol{\theta}_n^{(0)})$
Set initial window boundaries $t_0 < t_1 < \cdots < t_{n_w} = T$ and let $\mathcal{W}^{(1)} = \{[t_{i-1}, t_i]\}_{i=1}^{n_w}$
for $s = 1, \ldots, n_w$ **do**
  $k^{(s)} \leftarrow |\mathcal{W}^{(s)}|$ ▷ At $s = n_w$, the loss reduces to single-shooting as $k^{(n_w)} = 1$
  for epoch $= E_{s-1} + 1, \ldots, E_s$ **do** ▷ $E_0 := 0$
    Solve Eq. 4 on all $w_i^{(s)} \in \mathcal{W}^{(s)}$ with $(\boldsymbol{\theta}_m, \boldsymbol{\theta}_n, \boldsymbol{\theta}_{u_0,i}^{(s)})$ to obtain $\mathbf{u}^{(i,s)}(t)$
    Form the stage objective by evaluating Eqs. 5- 6 on $\mathcal{W}^{(s)}$: $g_{\text{CMS}}^{(s)} \leftarrow g_{\text{MS}}^{(s)} + g_{\text{cont}}^{(s)}$
    Update $(\boldsymbol{\theta}_m, \boldsymbol{\theta}_n, \{\boldsymbol{\theta}_{u_0,i}^{(s)}\}_{i=1}^{n_w^{(s)}})$ with one optimizer step using $\nabla g_{\text{CMS}}^{(s)}$ and η
  end for
  Form longer overlapping windows for the next stage: $\mathcal{W}^{(s+1)} = \{[\, t_{i-1},\, t_{i+s}\,]\}_{i=1}^{n_w - s}$
  Set $\boldsymbol{\theta}_{u_0,i}^{(s+1)} \leftarrow \boldsymbol{\theta}_{u_0,i}^{(s)}$ for $i = 1, \ldots, k^{(s)} - 1$
end for
return $(\boldsymbol{\theta}_m, \boldsymbol{\theta}_n)$

4.1 Hyperparameters

CMS introduces five hyperparameters. In our benchmark protocol (Sec. 5 and App. C), we tuned three (learning rate η, number of initial multiple-shooting windows n_w, and continuity penalty λ_{cont}), whereas the remaining two (cumulative stage epoch budget $\{E_s\}_{s=1}^{n_w}$, and window-state initialization $\boldsymbol{\theta}_{u_0,i}^{(1)}$) were fixed to reasonable default values. To provide practical tuning guidance for CMS, we summarize our approach below.

Tuned: For η, values used for conventional single-shooting training (Eq. 2) were sufficient, around 1×10^{-3}. The number of initial multiple-shooting windows, equivalently the number of curriculum stages, controls the initial window length; we recommend starting with eight windows where permissible by the data, and less with measurements at fewer time-points. Placing boundaries at the observation times ensures shared measurement points without requiring every state to be observed (App. C3) . The continuity penalty controls the strength of agreement between adjacent windows; we recommend starting with zero and tuning a positive penalty (<10) if needed.

Fixed: We found it sufficient to set $\{E_s\}_{s=1}^{n_w}$ such that the first third of the total epoch budget ($E_{n_w}/3$) was allocated equally across the first $n_w - 1$ curriculum stages, with the remaining two thirds allocated to the final single-shooting stage. For $\boldsymbol{\theta}_{u_0,i}^{(1)}$, setting all initial window values to a small constant (e.g. 0.01) was sufficient. When the system is fully observed, the observed states $\mathbf{u}$ can be used directly.

5 Experiments

We benchmarked our curriculum multiple shooting (CMS) approach in two settings: (i) fitting UDEs and NODEs, and (ii) fitting mechanistic ODE models to noisy time-series data. The UDE and NODE benchmarks comprised four models of varying complexity and covered both simulated (two models) and real (two models) time-series data. The mechanistic ODE benchmarks provided a complementary robustness test and comprised eight real-data models from the PEtab benchmark collection (Tab. B.1) [27, 48].

All UDE/NODE benchmarks were implemented in Python using the JAX-based packages `equinox` and `diffrax` [38], whereas the mechanistic benchmarks were implemented in Julia using `PEtab.jl` [49]. All experiments were run on CPUs, with a single model fit typically requiring less than 15 minutes (App. C). Code for all benchmarks is available at `https://github.com/frohlich-lab/CurriculumMultipleShooting_NeurIPS_2026`.

5.1 Models and datasets

We briefly summarize the UDE and mechanistic ODE models below; full details on model architectures and datasets are provided in Appendix B.

Lotka–Volterra UDE and NODE. The Lotka–Volterra system models prey (x) and predator (y) populations and exhibits oscillatory dynamics, making it challenging to fit [50]. We used simulated noise-corrupted data as in [5, 16], with x and y observed over one period. We considered two model architectures. The first, following [5], is a UDE of the form $\mathrm{d}x/\mathrm{d}t = \alpha x + \mathbf{NN}_1(x, y; \boldsymbol{\theta}_n)$ and $\mathrm{d}y/\mathrm{d}t = -\delta y + \mathbf{NN}_2(x, y; \boldsymbol{\theta}_n)$, where $\mathbf{NN}$ is a feed-forward neural network and $\boldsymbol{\theta}_m = (\alpha, \delta)$ are mechanistic parameters. We jointly learned the neural-network parameters $\boldsymbol{\theta}_n$ and the mechanistic parameters (α, δ). To discourage the network from absorbing the full dynamics and driving (α, δ) toward zero, we applied output regularization (Eq. 3) [21]. Second, we considered a NODE (Eq. B.3).

SIR UDE. We used an SIR (susceptible, infected, and recovered) UDE model of the initial COVID-19 outbreak in the United Kingdom (UK), as described in [12]. The model augments a classical SIR system with a feed-forward neural network to capture quarantine strength (Eq. B.4). We fitted the model to real time-series data from the UK. The model is partially observed, since only the infected and recovered populations are measured, and it exhibits pronounced scale differences across states.

Double pendulum UDE. We used real data from [51] describing the dynamics of a hard-to-fit double-pendulum system. As in [16], we employed a physics-informed second-order UDE (Eq B.8).

Reaction–diffusion UDE and NODE. We used simulated data from the FitzHugh–Nagumo (FHN) reaction–diffusion system in its oscillatory spiral-wave regime [52], discretised on a 64×64 grid with Neumann boundaries (8192 states; App. B.4). Following APHYNITY [53], the UDE combines a known diffusion operator with free coefficients (D_u, D_v) and a three-layer convolutional network for the reaction term. The NODE replaces the full right-hand side with a convolutional network.

Mechanistic ODE models. We used eight mechanistic models from the PEtab benchmark collection [27, 48]. These models span diverse biological applications (e.g. cell signaling, gene regulation, and disease spread), and are all paired with real data. They cover a wide range of challenges for model fitting, including stiffness, partial observability, model misspecification, irregular sampling, variation in model size, and changing simulation conditions across measurements (Tab. B.1).

5.2 Evaluation metrics and baselines

We briefly summarize the evaluation metrics and baselines used in the experiments. Additional implementation details and hyperparameter tuning procedure are provided in App. C.

UDE and NODE models. For evaluation, we split the time series into training and validation sets (50/50 for Lotka–Volterra, 2:1 for FHN and 70/30 for the other models), and tuned hyperparameters on the training data. Performance was evaluated using mean-squared error (MSE) on each set, hereafter referred to as interpolation and extrapolation MSE, following [16]. MSE is reported at the end of training, with all strategies run for the same number of epochs per model and repeated over 100 random initializations of the neural-network and mechanistic parameters. As some training runs diverged and produced skewed MSE distributions with outliers (Fig. 2), we summarized performance using the median MSE and compared methods using the Wilcoxon rank-sum test, with per-model Holm-corrected p-values [54]. Tables of values are provided in App A. In figures, we used ns ($p \geq 0.05$), * ($p < 0.05$), ** ($p < 0.01$), and *** ($p < 0.001$).

For **baselines**, because our experiments include both fully and partially observed models, we compared against four state-of-the-art methods applicable to both settings. These where single-shooting training with the Adam optimizer (Adam) [55], curriculum learning with Adam (CL; Sec. 3.2), multiple shooting with Adam (MS; Sec. 3.3), and single shooting Adam followed by a second-order optimizer with a BFGS Hessian approximation (Adam+BFGS). The latter is a common strategy for training both UDEs and PINNs [5, 56]. Results were reported for the best-performing hyperparameter configuration for each training strategy (Tabs. C.1–C.4).

Mechanistic ODE models. For evaluation, we counted how many of 1000 training runs from randomly sampled parameter initializations converge to the global minimum (Fig. 4a), a standard metric in systems biology [27, 29]. Since second-order optimizers are effective for mechanistic ODEs [29], we use the Fides trust-region Newton optimizer [57]. We compare CMS against three

baselines: plain Fides (control), curriculum learning with Fides (CL), and multiple shooting with Fides (MS). Because our goal was to assess robustness of our CMS strategy rather than maximize performance, we used a small fixed grid of relevant hyperparameter values and report the best-performing combinations (Tab. C.5).

5.3 Results on UDE and NODE benchmarks

For the Lotka–Volterra UDE with simulated data, CMS achieved the lowest median extrapolation MSE, with a value of 0.0148 compared to 0.0251 for the second-best method, multiple shooting (Fig. 1b; $p = 1.3 \times 10^{-3}$). For interpolation MSE (Fig. 1a), CMS was on par with both curriculum learning (0.010 vs. 0.012; $p = 0.37$) and multiple shooting (0.010 vs. 0.0099; $p = 0.12$). Adam+BFGS achieved the lowest interpolation MSE (0.0053; $p = 1 \times 10^{-9}$), but performed substantially worse than CMS in extrapolation (0.0148 vs. 1.52; $p = 7 \times 10^{-4}$). Inspection of the fitted training trajectories showed that Adam+BFGS overfitted the training data noise (Fig. A.4), explaining its worse extrapolation error. As a robustness check, we verified that CMS remained top-performing under higher measurement noise and longer simulation horizons (Fig. A.2).

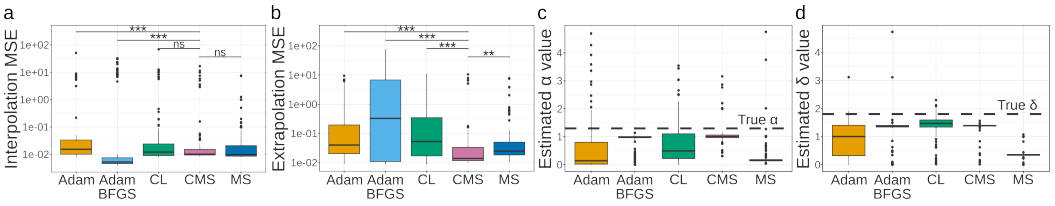

Figure 1: Benchmark results for the Lotka–Volterra UDE model from 100 multistarts with random parameter initializations. (a,b) Interpolation and extrapolation MSE for each strategy on $\log_{10}$ scale; p-values reported using the Wilcoxon test. (c,d) Estimated values of the mechanistic parameters.

Strikingly, CMS also recovered the Lotka–Volterra UDE mechanistic parameters close to the ground truth (true $(\alpha, \delta) = (1.3, 1.8)$, fitted median $(1.0, 1.45)$). In contrast, multiple shooting drove both parameters toward zero (Fig. 1c-d), indicating that the UDE degenerated to a NODE. This was likely because CMS could be trained with stronger output regularization (Eq. 3) without increasing the interpolation MSE (Fig. A.1). While too strong regularization eventually degraded performance for all strategies (Fig. A.1), CMS was more robust, indicating that it is better suited to harder optimization objectives like those induced by strong regularization.

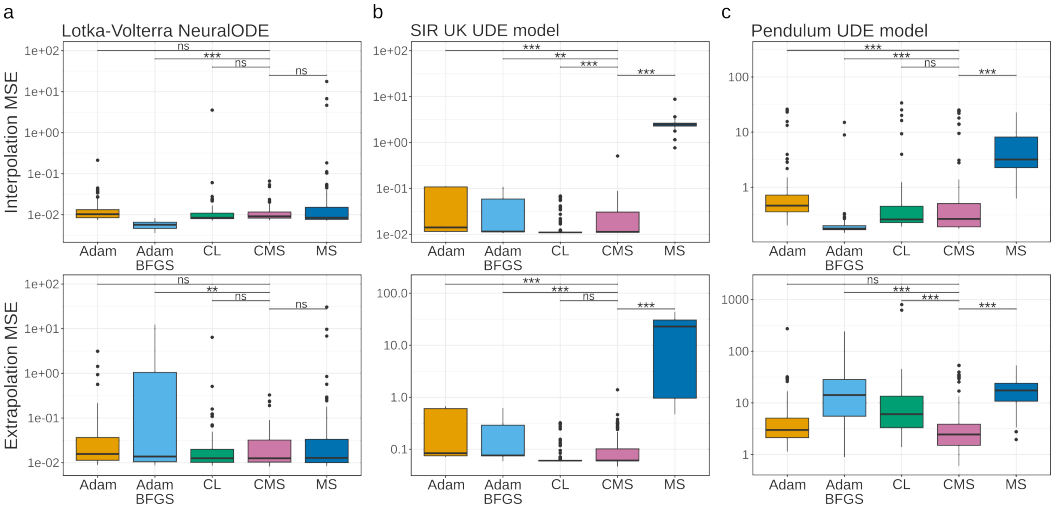

Figure 2: Benchmark UDE results from 100 multistarts with random parameter initializations. Interpolation MSE (top) and extrapolation MSE (bottom) on $\log_{10}$ scale for the (a) Lotka–Volterra NODE, (b) SIR UDE, and (c) double pendulum models. P-values reported using the Wilcoxon test.

Across the remaining NODE and UDE models, CMS was top-performing in extrapolation MSE (Fig. 2). For the Lotka–Volterra NODE (Fig. 2a), CMS matched the best method, curriculum learning (CL) (0.0126 vs. 0.0126; $p = 0.99$). Adam+BFGS again extrapolated worse than CMS (0.0126 vs. 0.0180; $p = 6.8 \times 10^{-3}$), consistent with its lower interpolation MSE reflecting overfitting. For the SIR UDE (Fig. 2b), CMS again matched the best method (CL) (0.0606 vs. 0.0605; $p = 0.47$), while outperforming the second-best (Adam+BFGS) (0.0606 vs. 0.0759; $p = 7 \times 10^{-11}$). For the double pendulum (Fig. 2c), CMS had the lowest median extrapolation MSE (2.45 vs. 2.99 for Adam; $p = 0.058$), and significantly lower interpolation MSE (0.267 vs. 0.466; $p = 6 \times 10^{-6}$). As CMS was designed to improve upon multiple shooting, we note it substantially improved extrapolation MSE over multiple shooting for the SIR (0.0606 vs. 24.9; $p = 5 \times 10^{-33}$) and the pendulum models (2.45 vs. 17.6; $p = 7 \times 10^{-19}$). On the Lotka–Volterra NODE and UDE models, where multiple shooting performed well (Figs. 1–2), CMS was also more robust to hyperparameter choice, particularly the continuity penalty λ_{cont} (Fig. A.3). This robustness also transferred across implementations and gradient-computation schemes, with CMS performing well on the Lotka–Volterra models and SIR UDE in complementary Julia-based benchmarks (App. A.5).

To assess epoch-wise convergence, we further inspected single-shooting interpolation-MSE curves over epochs (Fig. A.7). CMS reached low MSE in fewer epochs than the other strategies on $3/4$ models. The only exception is the non-oscillatory SIR model, where CL converged faster. This is discussed in more detail in App. A.3.

In summary, CMS achieved competitive extrapolation performance and typically reached low training error in fewer training epochs.

5.4 Results on a high-dimensional reaction–diffusion system

To test whether the benefits of CMS carry over to high-dimensional dynamics, we trained UDE and NODE models of a spatially extended FHN reaction–diffusion system with 8192 states (Fig. 3, note that here all curriculum stages received equal epochs and training was restricted to 500 epochs, App. C.5.2). The curriculum-based strategies were the only ones to extrapolate better than a trivial zero predictor (extrapolation MSE 0.78). Adam (UDE 0.787, NODE 1.59) and multiple shooting (UDE 1.00, NODE 7.44) did not. CMS substantially improved over multiple shooting in extrapolation MSE, for both the UDE (0.315 vs. 1.00; p = 2×10^{-19}) and the NODE (0.210 vs. 7.44; p = 2×10^{-33}), and over Adam (UDE 0.315 vs. 0.787, p = 7×10^{-6}; NODE 0.210 vs. 1.59, p = 5×10^{-25}). For the UDE (Fig. 3a), CL outperformed CMS in both interpolation (0.124 vs. 0.168; p = 4×10^{-4}) and extrapolation MSE (0.197 vs. 0.315; p = 3×10^{-5}). For the NODE (Fig. 3b), CMS matched the best method, CL (0.210 vs. 0.208; p = 0.45). Overall, curriculum-based training remained effective at three orders of magnitude more states than in the other benchmarks, whereas multiple shooting alone failed to generalise.

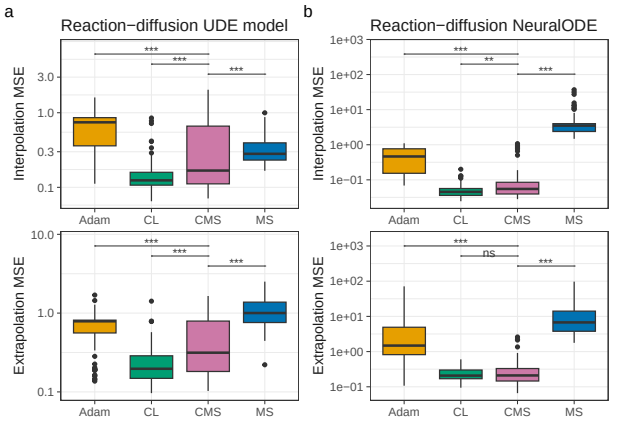

Figure 3: Reaction–diffusion benchmark (64×64 grid, 8192 states) from 100 multistarts with random parameter initializations. Interpolation MSE (top) and extrapolation MSE (bottom) on $\log_{10}$ scale for the (a) UDE and (b) NODE models. P-values reported using the Wilcoxon test.

5.5 Results on mechanistic ODE benchmarks

To assess the robustness of CMS more broadly, we evaluated it on eight diverse mechanistic ODE models with features that make model fitting challenging (e.g. stiffness, partial observability; Tab. B.1). Using the number of runs, out of 1000 random initialisations, converging to the global optimum as the metric (Fig. 4a), CMS outperformed the single-shooting control on all models by an average fold-change of 8.9 (range: 1.3–40.2). CMS also outperformed multiple shooting on

all models and was the best-performing method on 5/8 models (Fig. 4b). CL outperformed the single-shooting control on 7/8 models and was best on the remaining 3/8 models. As all methods used small, method-specific hyperparameter grids, the relative performance of CL and CMS may partly reflect tuning difficulty, since CL had fewer hyperparameters to tune.

Because CL performed well on a subset of benchmarks and has fewer tuning parameters than CMS, making it attractive when effective, we next investigated under which conditions it worked well. Although no single definitive pattern emerged, the results suggested that CL performed best when early curriculum stages were representative of the full problem, for example when each curriculum stage contained all observables (i.e. all model outputs). This is discussed in more detail in Appendix A.4.

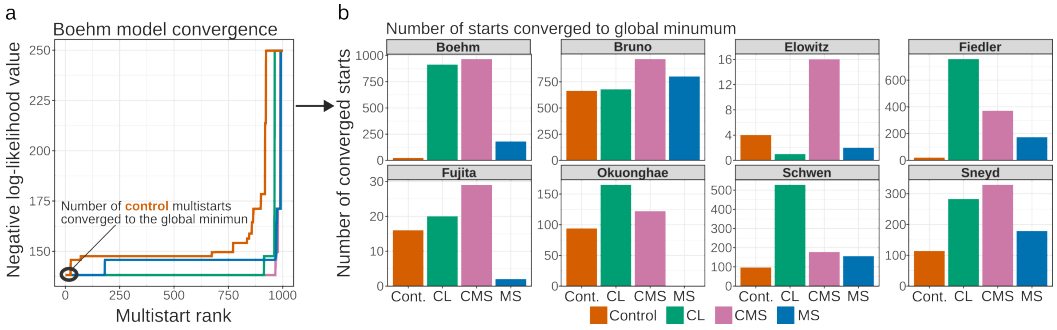

Figure 4: Benchmark results on eight mechanistic ODE models from 1000 multistarts with random parameter initializations. (a) Illustration of the evaluation criterion, i.e. the number of runs converging to the global optimum. (b) Results for each model. Control denotes plain single-shooting optimization.

5.6 Why does curriculum multiple shooting improve training?

To gain intuition for the strong performance of CMS, we analysed the training trajectories on the UDE/NODE models using Hessian trace profiles [58]. Hessian traces provide a descriptive curvature diagnostic, where small, positive value are considered favourable for training [16, 58]. CL and CMS consistently showed lower Hessian traces than Adam and MS early in training (Fig. 5a–c), often followed by a gradual increase, consistent with the expected progressive learning behaviour of curriculum approaches. This was also reflected in gradient ℓ_2-norm traces (Fig. A.6): CL and CMS kept gradient magnitudes controlled throughout training, with gradients decreasing within stages and increasing at curriculum stage boundaries. By contrast, poor training performance for Adam and MS coincided with larger gradient norms. Large gradients for Adam may result from long time spans for numerical integration by the ODE solver, which, building on the interpretation of NODEs as continuous-depth residual neural networks [1, 59], can make gradients more prone to vanishing or exploding [60]. While CL, MS, and CMS all initially train on shorter trajectories, MS additionally relies on a strong continuity penalty to make its windowed loss an effective surrogate for single shooting training, which may introduce larger gradients.

Motivated by evidence that skip-connections can convexify loss landscapes of residual neural networks by reducing the effective depth [61], we also analysed the loss landscape of the Lotka–Volterra UDE along the top two Hessian eigenvectors, evaluated at the final epoch of stages or overall training (Fig. 5d–g and Fig. A.5). Consistent with [16], Adam exhibited a rugged, non-convex landscape (Fig. 5d), whereas MS (Fig. 5e) and early stages of CL and CMS (Fig. 5f–g) showed smoother profiles in the displayed slices. Unlike MS, which never transitions to the full single-shooting objective (Eq. 2), CL and CMS eventually progressed to more complex landscapes with multiple plateaus and local minima (see also Fig. A.5). However, earlier stages often positioned optimisation in a favourable region before the harder landscape was encountered, as reflected by the small displacement between starting (white square) and final (yellow star) points in later stages. This was particularly evident for CMS, which is exposed to the full trajectory from the start, whereas CL introduces additional training data abruptly, which can lead to larger shifts between stages (Fig. A.5c). Analysis of single-shooting interpolation MSE traces revealed non-monotonic progression in both CL and CMS (Fig. A.7). While such non-monotonicity may reflect a desirable ability to escape local minima, the larger spikes in CL suggest that its early stages are not representative of the single-shooting objective.

In summary, CMS likely achieves strong performance by easing early-phase optimisation while remaining aligned with the single-shooting objective, ultimately recovering it and reducing reliance on a strong continuity penalty.

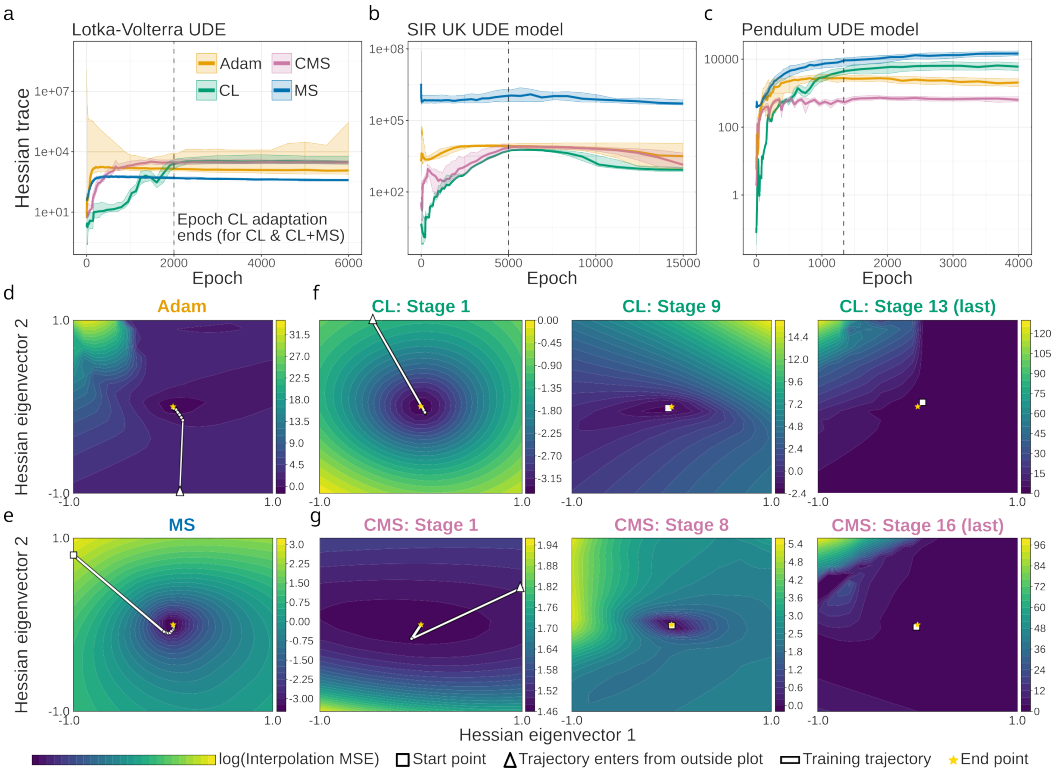

Figure 5: Training trajectory analysis across training strategies. (a–c) Hessian trace profiles for the Lotka–Volterra UDE, SIR UDE, and double pendulum UDE models. Data are shown as median (solid lines) and 25th–75th percentiles (shaded intervals) across 100 multistarts. (d–g) Loss-landscape visualisations along the top two Hessian eigenvectors for the Lotka–Volterra UDE, one representative multistart per strategy reaching interpolation MSE < 0.02. For Adam and MS, the Hessian eigenvectors are computed at the final epoch; for CL and CMS at the final epoch of each curriculum stage.

6 Discussion and limitations

We propose *curriculum multiple shooting* (CMS), a general-purpose training strategy for fitting mechanistic ODEs, NODEs, and hybrid UDEs to noisy time-series data. Across diverse benchmark problems spanning full and partial observability, noisy and irregularly sampled measurements, and challenging dynamics, CMS was consistently among the top-performing methods (Fig. 1–4). Its strong performance on classical mechanistic ODE fitting tasks (Fig. 4) further demonstrates that our results extend beyond ML to disciplines where mechanistic modelling remains central, such as biology, epidemiology, and pharmacology. More broadly, this suggests that training NODEs and UDEs is fundamentally a hard ODE fitting problem, and that training strategies can transfer from mechanistic ODEs to NODEs/UDEs, and likely also to related discrete-time nonlinear state-space models where multiple-shooting ideas have been used [46].

Our study also has several limitations pointing to relevant future directions. First, we consider only a limited number of NODE and UDE benchmarks, partly because identifying and implementing suitable benchmark problems is labour-intensive. This underscores the need for broader, standardized benchmark collections similar to those available for mechanistic ODEs [27]. Second, on the high-dimensional reaction–diffusion system, CMS remained far better than multiple shooting and Adam but with limited runtime it did not improve on CL; whether CMS retains an advantage over CL in high dimensions remains open. Third, CMS introduces several tuning parameters. Improving default heuristics and developing robust tuning schemes would further increase its practical usability.

Acknowledgments and Disclosure of Funding

Funding

B.S, F.F., G.F., and S.P. were supported by the Francis Crick Institute, which receives its core funding from Cancer Research UK (CC2242), the UK Medical Research Council (CC2242), and the Wellcome Trust (CC2242). F.F., G.F., and S.P. were also supported by the European Union (ERC, DeepMechanism, grant no 101163005).

Competing Interests

F.F. reports consulting honoraria from Deep Origin, but this had no influence on the study.

References

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural Ordinary Differential Equations. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[2] Jennifer Brucker, René Behmann, Wolfgang G. Bessler, and Rainer Gasper. Neural Ordinary Differential Equations for Grey-Box Modelling of Lithium-Ion Batteries on the Basis of an Equivalent Circuit Model. *Energies*, 15(7):2661, January 2022. ISSN 1996-1073. doi: 10.3390/en15072661.

[3] Tongli Zhang, Ioannis P. Androulakis, Peter Bonate, Limei Cheng, Tomáš Helikar, Jaimit Parikh, Christopher Rackauckas, Kalyanasundaram Subramanian, Carolyn R. Cho, Ioannis P. Androulakis, Peter Bonate, Ivan Borisov, Gordon Broderick, Limei Cheng, Valeriu Damian, Rafael Dariolli, Oleg Demin, Nicholas Ellinwood, Dirk Fey, Abhishek Gulati, Tomas Helikar, Eric Jordie, Cynthia Musante, Jaimit Parikh, Christopher Rackauckas, Julio Saez-Rodriguez, Eric Sobie, Kalyanasundaram Subramanian, Carolyn R. Cho, and on behalf of the Working Group. Two heads are better than one: current landscape of integrating QSP and machine learning. *Journal of Pharmacokinetics and Pharmacodynamics*, 49(1):5–18, February 2022. ISSN 1573-8744. doi: 10.1007/s10928-022-09805-z.

[4] Bob Junyi Zou, Matthew E. Levine, Dessi P. Zaharieva, Ramesh Johari, and Emily B. Fox. Hybrid2 neural ODE causal modeling and an application to glycemic response. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 62934–62963, Vienna, Austria, July 2024. JMLR.org.

[5] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal Differential Equations for Scientific Machine Learning, November 2021. arXiv:2001.04385 [cs].

[6] Max de Rooij, Natal A. W. van Riel, and Shauna D. O'Donovan. Conditional universal differential equations capture population dynamics and interindividual variation in c-peptide production. *npj Systems Biology and Applications*, 11(1):84, July 2025. ISSN 2056-7189. doi: 10.1038/s41540-025-00570-6.

[7] Marie Steinacker, Yuri Kheifetz, and Markus Scholz. Developing hybrid mechanistic and data-driven personalized prediction models for platelet dynamics, May 2025. arXiv:2505.21204 [cs].

[8] Xiaojun Wu, MeiLu McDermott, and Adam L. MacLean. Data-driven model discovery and model selection for noisy biological systems. *PLOS Computational Biology*, 21(1):e1012762, January 2025. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1012762.

[9] Zhaozhi Qian, William Zame, Lucas Fleuren, Paul Elbers, and Mihaela van der Schaar. Integrating Expert ODEs into Neural ODEs: Pharmacology and Disease Progression. In *Advances in Neural Information Processing Systems*, volume 34, pages 11364–11383. Curran Associates, Inc., 2021.

[10] James Lu, Kaiwen Deng, Xinyuan Zhang, Gengbo Liu, and Yuanfang Guan. Neural-ODE for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens. *iScience*, 24(7):102804, July 2021. ISSN 2589-0042. doi: 10.1016/j.isci.2021.102804.

[11] Dominic Stefan Bräm, Uri Nahum, Johannes Schropp, Marc Pfister, and Gilbert Koch. Low-dimensional neural ODEs and their application in pharmacokinetics. *Journal of Pharmacokinetics and Pharmacodynamics*, 51(2):123–140, April 2024. ISSN 1573-8744. doi: 10.1007/s10928-023-09886-4.

[12] Raj Dandekar, Chris Rackauckas, and George Barbastathis. A Machine Learning-Aided Global Diagnostic and Comparative Tool to Assess Effect of Quarantine Control in COVID-19 Spread. *Patterns*, 1(9), December 2020. ISSN 2666-3899. doi: 10.1016/j.patter.2020.100145.

[13] Yang Ye, Abhishek Pandey, Carolyn Bawden, Dewan Md Sumsuzzman, Rimpi Rajput, Affan Shoukat, Burton H. Singer, Seyed M. Moghadas, and Alison P. Galvani. Integrating artificial intelligence with mechanistic epidemiological modeling: a scoping review of opportunities and challenges. *Nature Communications*, 16(1):581, January 2025. ISSN 2041-1723. doi: 10.1038/s41467-024-55461-x.

[14] Sharv Murgai, Utkarsh Utkarsh, Kyle C. Nguyen, Alan Edelman, Erin C. S. Acquesta, and Christopher Vincent Rackauckas. ABM-UDE: Developing Surrogates for Epidemic Agent-Based Models via Scientific Machine Learning, February 2026. arXiv:2602.21588 [cs].

[15] Tobias Thummerer, Johannes Stoljar, and Lars Mikelsons. NeuralFMU: Presenting a Workflow for Integrating Hybrid NeuralODEs into Real-World Applications. *Electronics*, 11(19):3202, January 2022. ISSN 2079-9292. doi: 10.3390/electronics11193202.

[16] Joon-Hyuk Ko, Hankyul Koh, Nojun Park, and Wonho Jhe. Homotopy-based training of NeuralODEs for accurate dynamics discovery. *Advances in Neural Information Processing Systems*, 36:64725–64752, December 2023.

[17] Maren Philipps, Nina Schmid, and Jan Hasenauer. Current state and open problems in universal differential equations for systems biology. *npj Systems Biology and Applications*, 11(1):101, August 2025. ISSN 2056-7189. doi: 10.1038/s41540-025-00550-w.

[18] Krzysztof M Choromanski, Jared Quincy Davis, Valerii Likhosherstov, Xingyou Song, Jean-Jacques Slotine, Jacob Varley, Honglak Lee, Adrian Weller, and Vikas Sindhwani. Ode to an ODE. In *Advances in Neural Information Processing Systems*, volume 33, pages 3338–3350. Curran Associates, Inc., 2020.

[19] Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization. In *Proceedings of the 37th International Conference on Machine Learning*, pages 3154–3164. PMLR, November 2020.

[20] Ivan Dario Jimenez Rodriguez, Aaron Ames, and Yisong Yue. LyaNet: A Lyapunov Framework for Training Neural ODEs. In *Proceedings of the 39th International Conference on Machine Learning*, pages 18687–18703. PMLR, June 2022.

[21] Maren Philipps, Antonia Körner, Jakob Vanhoefer, Dilan Pathirana, and Jan Hasenauer. Non-Negative Universal Differential Equations With Applications in Systems Biology. *IFAC-PapersOnLine*, 58(23):25–30, January 2024. ISSN 2405-8963. doi: 10.1016/j.ifacol.2024.10.005.

[22] Chaoyang Luo, Yan Zou, Wanying Li, and Nanjing Huang. FxTS-Net: Fixed-time stable learning framework for Neural ODEs. *Neural Networks*, 185:107219, May 2025. ISSN 0893-6080. doi: 10.1016/j.neunet.2025.107219.

[23] Elisabeth Roesch, Christopher Rackauckas, and Michael P. H. Stumpf. Collocation based training of neural ordinary differential equations. *Statistical Applications in Genetics and Molecular Biology*, 20(2):37–49, April 2021. ISSN 1544-6115. doi: 10.1515/sagmb-2020-0025.

[24] Evren Mert Turan and Johannes Jäschke. Multiple Shooting for Training Neural Differential Equations on Time Series. *IEEE Control Systems Letters*, 6:1897–1902, 2022. ISSN 2475-1456. doi: 10.1109/LCSYS.2021.3135835.

[25] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. DiffEqFlux.jl - A Julia Library for Neural Differential Equations, February 2019.

[26] Hua Liang and Hulin Wu. Parameter Estimation for Differential Equation Models Using a Framework of Measurement Error in Regression Models. *Journal of the American Statistical Association*, 103(484):1570–1583, December 2008. ISSN 0162-1459, 1537-274X. doi: 10.1198/016214508000000797.

[27] Helge Hass, Carolin Loos, Elba Raimúndez-Álvarez, Jens Timmer, Jan Hasenauer, and Clemens Kreutz. Benchmark problems for dynamic modeling of intracellular processes. *Bioinformatics*, 35(17):3073–3082, September 2019. ISSN 1367-4803.

[28] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented Neural ODEs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[29] Andreas Raue, Marcel Schilling, Julie Bachmann, Andrew Matteson, Max Schelke, Daniel Kaschek, Sabine Hug, Clemens Kreutz, Brian D. Harms, Fabian J. Theis, Ursula Klingmüller, and Jens Timmer. Lessons Learned from Quantitative Dynamical Modeling in Systems Biology. *PLOS ONE*, 8(9):e74335, September 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0074335.

[30] H. G. Bock and K. J. Plitt. A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems*. *IFAC Proceedings Volumes*, 17(2):1603–1608, July 1984. ISSN 1474-6670. doi: 10.1016/S1474-6670(17)61205-9.

[31] Lennart Ljung. Prediction error estimation methods. *Circuits, Systems and Signal Processing*, 21(1):11–21, January 2002. ISSN 1531-5878. doi: 10.1007/BF01211648.

[32] Henry D. I. Abarbanel, Daniel R. Creveling, Reza Farsian, and Mark Kostuk. Dynamical State and Parameter Estimation. *SIAM Journal on Applied Dynamical Systems*, 8(4):1341–1381, January 2009. doi: 10.1137/090749761.

[33] Yuan Yin, Vincent Le Guen, Jérémie Dona, Emmanuel de Bézenac, Ibrahim Ayed, Nicolas Thome, and Patrick Gallinari. Augmenting physical models with deep networks for complex dynamics forecasting*. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124012, December 2021. ISSN 1742-5468. doi: 10.1088/1742-5468/ac3ae5.

[34] Hananeh Aliee, Till Richter, Mikhail Solonin, Ignacio Ibarra, Fabian Theis, and Niki Kilbertus. Sparsity in Continuous-Depth Neural Networks. *Advances in Neural Information Processing Systems*, 35:901–914, December 2022.

[35] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian Neural Networks. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[36] Derek Onken and Lars Ruthotto. Discretize-Optimize vs. Optimize-Discretize for Time-Series Regression and Continuous Normalizing Flows, July 2020. arXiv:2005.13420 [cs].

[37] Suyong Kim, Weiqi Ji, Sili Deng, Yingbo Ma, and Christopher Rackauckas. Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31 (9), September 2021. ISSN 1054-1500. doi: 10.1063/5.0060697.

[38] Patrick Kidger. On Neural Differential Equations, February 2022. arXiv:2202.02435 [cs].

[39] Facundo Sapienza, Jordi Bolibar, Frank Schäfer, Brian Groenke, Avik Pal, Victor Boussange, Patrick Heimbach, Giles Hooker, Fernando Pérez, Per-Olof Persson, and Christopher Rackauckas. Differentiable Programming for Differential Equations: A Review, October 2025. arXiv:2406.09699 [math].

[40] Joel Laudo and Adrian Buganza Tepole. Stable Long-Horizon Neural ODE Reduced-Order Models via Learned Feedback for Biological Growth and Remodeling, April 2026. arXiv:2604.13820 [cs].

[41] Lake Yang, Antonio Malpica-Morales, Frank Ioannis Papadakis Wood, and Serafim Kalliadasis. Mpineuralode: Multiple-initial-condition physics-informed neural odes for globally consistent dynamical system learning, 2026.

[42] Johan Gabrielsson and Daniel Weiner. *Pharmacokinetic and Pharmacodynamic Data Analysis: Concepts and Applications, Third Edition*. CRC Press, November 2001. ISBN 978-91-86274-92-4.

[43] Benjamin Whipple and Esteban A. Hernandez-Vargas. Hybrid Neural Differential Equations to Model Unknown Mechanisms and States in Biology, December 2024. Pages: 2024.12.08.627408 Section: New Results.

[44] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, June 2009. Association for Computing Machinery. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553380.

[45] Petru Soviany, Radu Tudor Ionescu, Paolo Rota, and Nicu Sebe. Curriculum Learning: A Survey. *International Journal of Computer Vision*, 130(6):1526–1565, June 2022. ISSN 1573-1405. doi: 10.1007/s11263-022-01611-x.

[46] Antônio H. Ribeiro, Koen Tiels, Jack Umenberger, Thomas B. Schön, and Luis A. Aguirre. On the smoothness of nonlinear system identification. *Automatica*, 121:109158, November 2020. ISSN 0005-1098. doi: 10.1016/j.automatica.2020.109158.

[47] Stefano Massaroli, Michael Poli, Sho Sonoda, Taiji Suzuki, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Differentiable Multiple Shooting Layers. In *Advances in Neural Information Processing Systems*, volume 34, pages 16532–16544. Curran Associates, Inc., 2021.

[48] Leonard Schmiester, Yannik Schälte, Frank T. Bergmann, Tacio Camba, Erika Dudkin, Janine Egert, Fabian Fröhlich, Lara Fuhrmann, Adrian L. Hauber, Svenja Kemmer, Polina Lakrisenko, Carolin Loos, Simon Merkt, Wolfgang Müller, Dilan Pathirana, Elba Raimúndez, Lukas Refisch, Marcus Rosenblatt, Paul L. Stapor, Philipp Städter, Dantong Wang, Franz-Georg Wieland, Julio R. Banga, Jens Timmer, Alejandro F. Villaverde, Sven Sahle, Clemens Kreutz, Jan Hasenauer, and Daniel Weindl. PEtab—Interoperable specification of parameter estimation problems in systems biology. *PLOS Computational Biology*, 17(1):e1008646, January 2021. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1008646.

[49] Sebastian Persson, Fabian Fröhlich, Stephan Grein, Torkel Loman, Damiano Ognissanti, Viktor Hasselgren, Jan Hasenauer, and Marija Cvijovic. PEtab.jl: advancing the efficiency and utility of dynamic modelling. *Bioinformatics*, 41(9):btaf497, September 2025. ISSN 1367-4811. doi: 10.1093/bioinformatics/btaf497.

[50] Jake Alan Pitt and Julio R. Banga. Parameter estimation in models of biological oscillators: an automated regularised estimation approach. *BMC Bioinformatics*, 20(1):82, February 2019. ISSN 1471-2105. doi: 10.1186/s12859-019-2630-y.

[51] Michael Schmidt and Hod Lipson. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85, April 2009. doi: 10.1126/science.1165893.

[52] Daniel Cebrián-Lacasa, Pedro Parra-Rivas, Daniel Ruiz-Reynés, and Lendert Gelens. Six decades of the fitzhugh–nagumo model: A guide through its spatio-temporal dynamics and influence across disciplines. *Physics Reports*, 1096:1–39, 2024. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2024.09.014. Six decades of the FitzHugh-Nagumo model: A guide through its spatio-temporal dynamics and influence across disciplines.

[53] Yuan Yin, Vincent Le Guen, Jérémie Dona, Emmanuel de Bézenac, Ibrahim Ayed, Nicolas Thome, and Patrick Gallinari. Augmenting physical models with deep networks for complex dynamics forecasting*. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12): 124012, December 2021. ISSN 1742-5468. doi: 10.1088/1742-5468/ac3ae5.

[54] Sture Holm. A Simple Sequentially Rejective Multiple Test Procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979. ISSN 0303-6898.

[55] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[56] Pratik Rathore, Weimu Lei, Zachary Frangella, Lu Lu, and Madeleine Udell. Challenges in Training PINNs: A Loss Landscape Perspective. In *Proceedings of the 41st International Conference on Machine Learning*, pages 42159–42191. PMLR, July 2024.

[57] Fabian Fröhlich and Peter K. Sorger. Fides: Reliable trust-region optimization for parameter estimation of ordinary differential equation models. *PLOS Computational Biology*, 18(7): e1010322, July 2022. ISSN 1553-7358. doi: 10.1371/journal.pcbi.1010322.

[58] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. PyHessian: Neural Networks Through the Lens of the Hessian. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 581–590, December 2020. doi: 10.1109/BigData50022.2020.9378171.

[59] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting Neural ODEs. In *Advances in Neural Information Processing Systems*, volume 33, pages 3952–3963. Curran Associates, Inc., 2020.

[60] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1310–1318. PMLR, May 2013.

[61] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[62] Jarrett Revels, Miles Lubin, and Theodore Papamarkou. Forward-Mode Automatic Differentiation in Julia, July 2016. arXiv:1607.07892 [cs].

[63] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1(6):445–466, 1961. ISSN 0006-3495. doi: 10.1016/S0006-3495(61)86902-6.

[64] Jin ichi Nagumo, Suguru Arimoto, and Shuji Yoshizawa. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE*, 50:2061–2070, 1962. doi: 10.1109/JRPROC.1962.288235.

[65] Ryan N. Gutenkunst, Joshua J. Waterfall, Fergal P. Casey, Kevin S. Brown, Christopher R. Myers, and James P. Sethna. Universally Sloppy Parameter Sensitivities in Systems Biology Models. *PLOS Computational Biology*, 3(10):e189, 2007. ISSN 1553-7358. doi: 10.1371/journal.pcbi.0030189.

[66] Ch. Tsitouras. Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Computers & Mathematics with Applications*, 62(2):770–775, July 2011. ISSN 0898-1221. doi: 10.1016/j.camwa.2011.06.002.

[67] Gerd Steinebach. Construction of Rosenbrock–Wanner method Rodas5P and numerical benchmarks within the Julia Differential Equations package. *BIT Numerical Mathematics*, 63(2):27, April 2023. ISSN 1572-9125. doi: 10.1007/s10543-023-00967-x.

[68] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010.

A Additional Results

This section introduces additional results, figures, and tables not included in the main text for brevity. It also contains results from a robustness benchmark on UDEs in Julia, complementing the Python based benchmarks in Sec. 5.3.

A.1 Summary table of benchmark results

This section contains a table summarizing the benchmark results for the NODE and UDE models in Sec. 5.3 in Tabs. A.1–A.4, and for the high-dimensional reaction–diffusion models (Sec. 5.4) in Tab. A.5.

Table A.1: Benchmark results for the Lotka–Volterra UDE (Eq. B.2). Interpolation and extrapolation MSE across all 100 multi-starts are reported as $\mathrm{median} \pm \mathrm{median\ absolute\ deviation}$ for each training strategy. The best interpolation and extrapolation values are shown in bold.

Strategy	Interpolation MSE	Extrapolation MSE
CMS	0.0101 ± 0.00173	$\mathbf{0.0148 \pm 0.00610}$
CL	0.0123 ± 0.00551	0.0785 ± 0.0996
MS	0.00990 ± 0.00235	0.0251 ± 0.0134
Adam	0.0242 ± 0.0226	0.188 ± 0.262
Adam+BFGS	$\mathbf{0.00526 \pm 0.00131}$	1.52 ± 2.23

Table A.2: Benchmark results for the Lotka–Volterra NODE (Eq. B.3). Values are reported as in Tab. A.1.

Strategy	Interpolation MSE	Extrapolation MSE
CMS	0.00913 ± 0.00165	0.0126 ± 0.00476
CL	0.00837 ± 0.000975	$\mathbf{0.0126 \pm 0.00410}$
MS	0.00841 ± 0.00134	0.0128 ± 0.00484
Adam	0.0103 ± 0.00318	0.0157 ± 0.00794
Adam+BFGS	$\mathbf{0.00569 \pm 0.00145}$	0.0180 ± 0.0135

Table A.3: Benchmark results for the SIR (Eq. B.4). Values are reported as in Tab. A.1.

Strategy	Interpolation MSE	Extrapolation MSE
CMS	0.0113 ± 0.000502	0.0606 ± 0.00378
CL	$\mathbf{0.0111 \pm 0.000166}$	$\mathbf{0.0605 \pm 0.00137}$
MS	13.3 ± 15.3	24.9 ± 29.1
Adam	0.0142 ± 0.00437	0.0839 ± 0.0175
Adam+BFGS	0.0115 ± 0.000321	0.0759 ± 0.00373

Table A.4: Benchmark results for the double pendulum model (Eq. B.8). Values are reported as in Tab. A.1.

Strategy	Interpolation MSE	Extrapolation MSE
CMS	0.267 ± 0.127	$\mathbf{2.45 \pm 1.66}$
CL	0.265 ± 0.0687	6.07 ± 5.16
MS	3.19 ± 2.31	17.6 ± 10.1
Adam	0.466 ± 0.204	2.99 ± 1.69
Adam+BFGS	$\mathbf{0.178 \pm 0.0167}$	15.7 ± 16.5

Table A.5: Benchmark results for the FHN reaction–diffusion UDE (Eq. B.10) and NODE (Eq. B.11) on a 64×64 grid (8192 states). Interpolation and extrapolation MSE across all 100 multistarts are reported as $\text{median} \pm \text{median absolute deviation}$ for each training strategy. The best interpolation and extrapolation values within each model are shown in bold. For reference, predicting zero for all states gives an interpolation MSE of 1.00 and an extrapolation MSE of 0.781.

Model	Strategy	Interpolation MSE	Extrapolation MSE
Reaction–diffusion UDE	CMS	0.168 ± 0.105	0.315 ± 0.242
	CL	$\mathbf{0.124 \pm 0.0373}$	$\mathbf{0.197 \pm 0.0856}$
	MS	0.281 ± 0.0950	1.00 ± 0.403
	Adam	0.744 ± 0.238	0.787 ± 0.0545
Reaction–diffusion NODE	CMS	0.0550 ± 0.0271	0.210 ± 0.106
	CL	$\mathbf{0.0454 \pm 0.0150}$	$\mathbf{0.208 \pm 0.0736}$
	MS	3.51 ± 1.49	7.44 ± 6.42
	Adam	0.463 ± 0.459	1.59 ± 1.70

A.2 Additional Figures

Fig. A.1 shows the effect of increasing output regularization strength (λ_O, Eq. 3) on the training/interpolation MSE of the Lotka–Volterra UDE model. Although too strong regularization ultimately degrades training performance for all strategies, curriculum multiple shooting (CMS) is the most robust. This suggests that CMS is better suited to harder optimization objectives, such as those induced by stronger regularization.

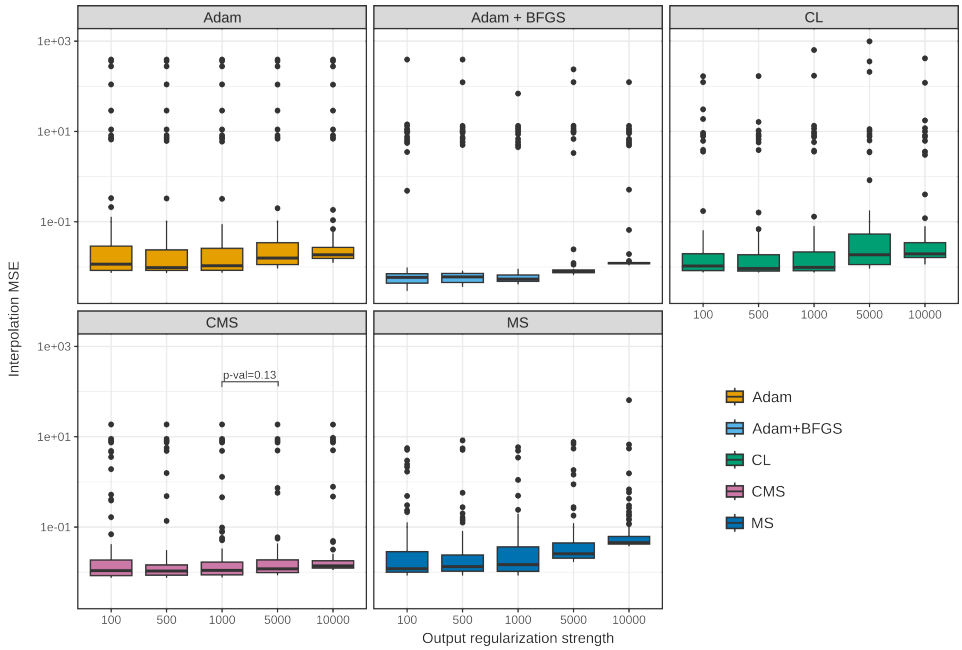

Figure A.1: Effect of increasing output regularization strength (λ_O in Eq. 3) on the interpolation MSE of the Lotka–Volterra UDE model for each training strategy. Results are based on 100 multistarts with random parameter initializations and are shown on a $\log_{10}$ scale. P-values are reported using the Wilcoxon rank-sum test. The tested λ_O values are a subset of those considered during hyperparameter tuning (Tabs. C.3–C.4).

Fig. A.2 provides robustness checks for CMS on the Lotka–Volterra UDE under higher measurement noise and longer simulation horizons. The strong performance of CMS, particularly for longer time spans, further supports its suitability for training on longer time-series data, which have been shown to make NODE and UDE training more difficult [16].

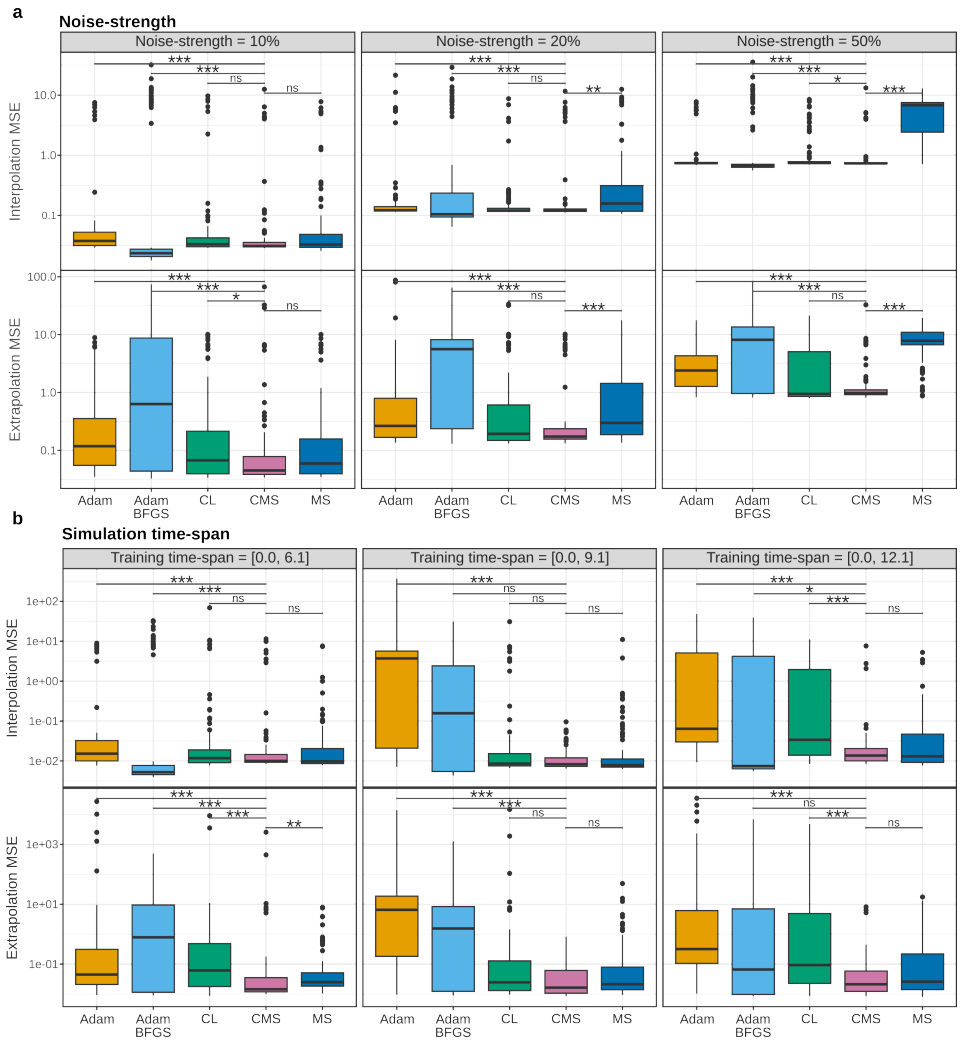

Figure A.2: Effect of increasing (a) measurement noise and (b) simulation time-span on the Lotka–Volterra UDE model for each training strategy, evaluated using interpolation MSE (top row) and extrapolation MSE (bottom row). Results are based on 100 multistarts with random parameter initializations and are shown on a $\log_{10}$ scale. Holm-corrected p-values per model are reported using the Wilcoxon rank-sum test, with ns ($p \geq 0.05$), * ($p < 0.05$), ** ($p < 0.01$), and *** ($p < 0.001$). Due to computational cost, the same hyperparameter values as in the benchmark in Fig. 1 were used.

Fig. A.3 investigates the sensitivity of multiple shooting (MS) and CMS to hyperparameters on the two models where MS performed well. In particular, for the hard-to-tune continuity penalty λ_{cont} in Eq. 6, CMS is substantially more robust, achieving lower interpolation MSE over a wider range of values.

Fig. A.4 shows an example where the Adam+BFGS training strategy overfits the Lotka–Volterra UDE model, achieving low interpolation MSE but high extrapolation MSE.

Fig. A.5 provides additional loss-landscape visualizations. Overall, these plots show that in later curriculum stages, where the objective becomes harder to optimize, CMS is typically already positioned in a favourable region of the loss landscape. This is reflected by the small displacement between the starting point (white square/triangles) and endpoint (yellow star) within later stages.

Finally, Fig. A.6 shows gradient ℓ_2 norm traces for each model and training strategy.

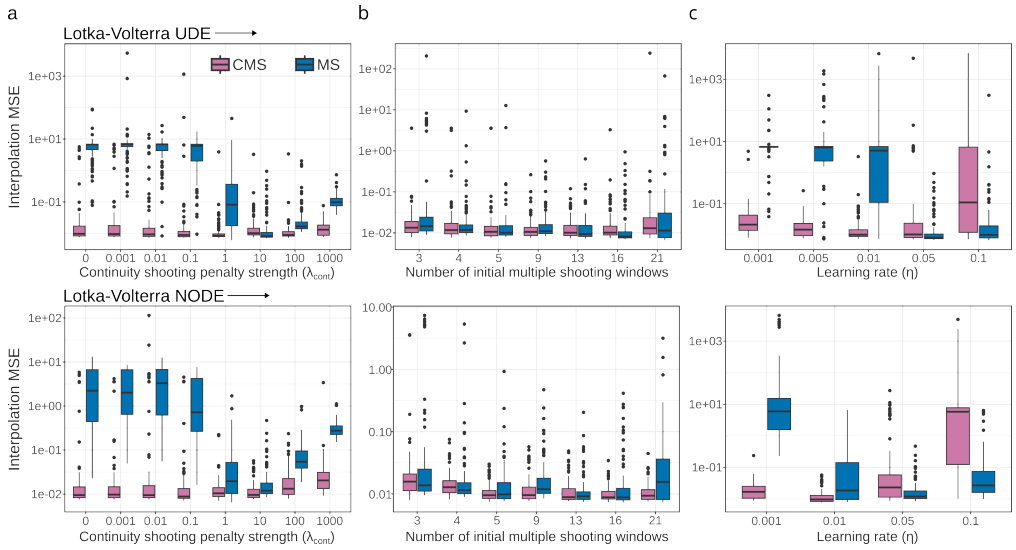

Figure A.3: Sensitivity of interpolation MSE to the hyperparameters (a) continuity penalty (λ_{cont} in Eq. 6), (b) number of initial multiple shooting windows shooting windows, and (c) learning rate for the multiple shooting (MS) and curriculum multiple shooting (CMS) training strategies on the Lotka–Volterra UDE (top row) and Lotka–Volterra NODE (bottom row) models. Results are based on 100 multistarts with random parameter initializations and are shown on a $\log_{10}$ scale. The tested values are those considered during hyperparameter tuning Tabs. C.3– C.4.

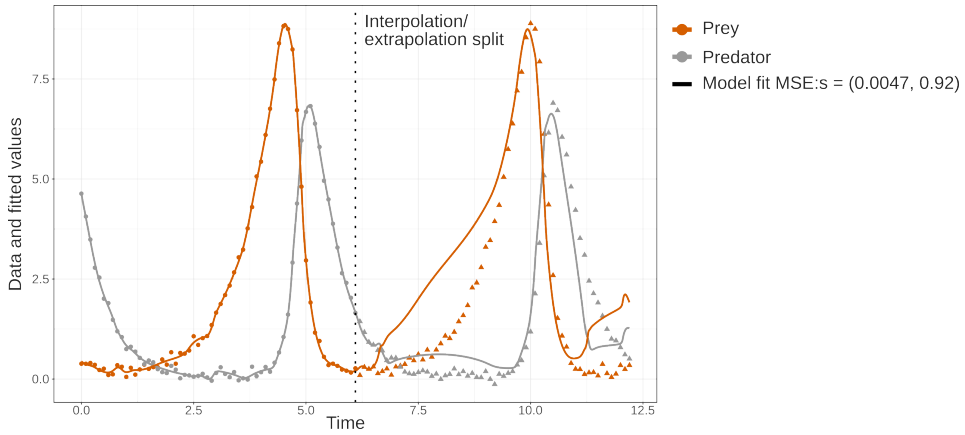

Figure A.4: Adam+BFGS overfits the Lotka–Volterra UDE training data for the prey and predator observables. Lines shows a representative model fit with low interpolation/training error but high extrapolation/validation error. MSE values are reported in the legend as (interpolation, extrapolation).

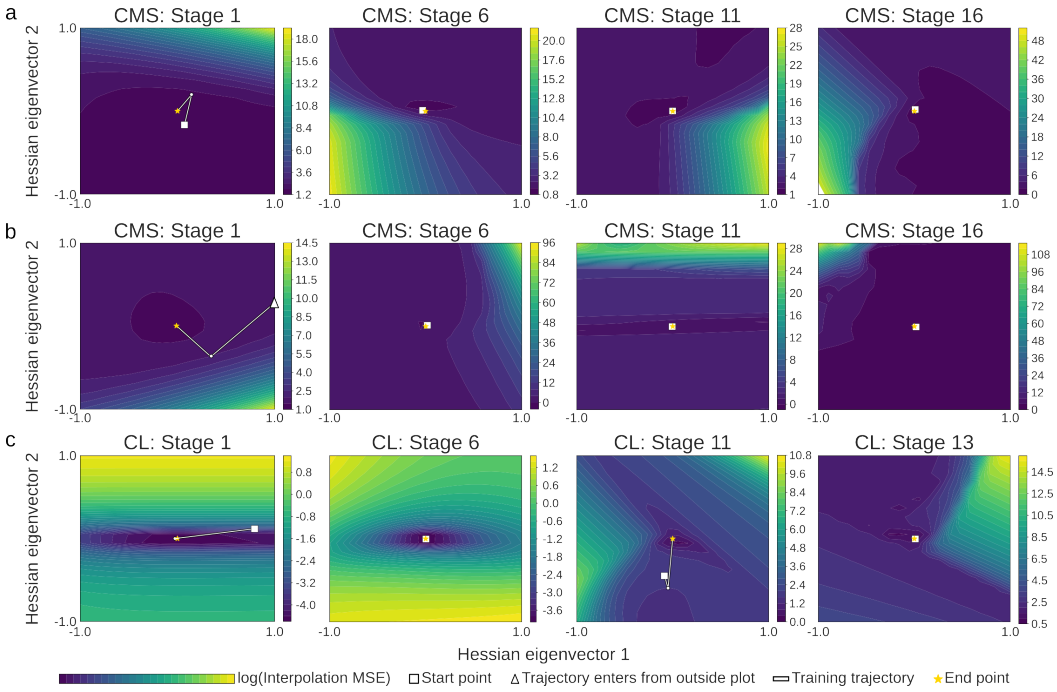

Figure A.5: Loss-landscape visualisations for representative multistarts of the Lotka–Volterra UDE with interpolation MSE < 0.02, shown along the top two Hessian eigenvectors. Each panel shows one representative multistart for (a–b) curriculum multiple shooting and (c) curriculum learning. Landscapes are evaluated at the final epoch of the corresponding curriculum stages.

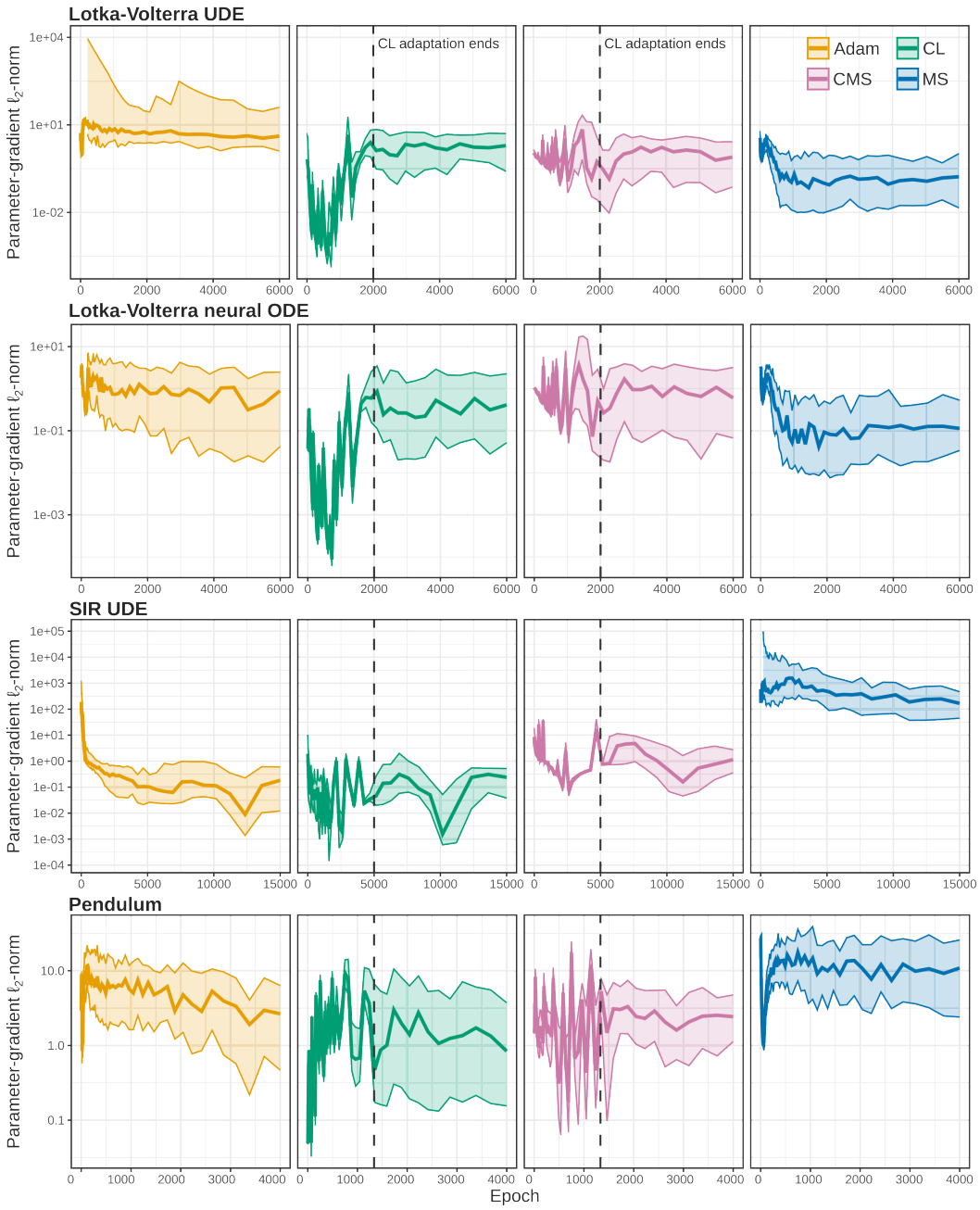

Figure A.6: ℓ_2 norm of the parameter (both neural and mechanistic) gradient of the training objective (e.g. for MS this corresponds to the multiple-shooting objective in Eq. 5) for the Lotka–Volterra UDE (top), Lotka–Volterra NODE (upper middle), SIR UDE (lower middle), and pendulum UDE (bottom) models. Results are based on 100 multistarts with random parameter initializations and are shown on a $\log_{10}$ scale. Solid lines denote the median, and shaded intervals denote the 25th–75th percentiles across multistarts. Dashed vertical lines indicate the end of curriculum adaptation for the relevant strategies.

A.3 Additional analysis on UDE/NODE training traces

As CL performed on par with CMS for the SIR and Lotka–Volterra NODE models (Fig. 2), we next compared their single-shooting (Eq. 2) interpolation-MSE training curves over epochs (Fig. A.7). CMS generally converged fastest, reaching a low MSE within roughly one-third of the training epochs,

whereas CL typically required at least two-thirds. The exception was the SIR model (Fig. A.7c), where CL reached a low MSE within two-thirds of the epochs, whereas CMS required the full 15 000 epochs. SIR was also the only model for which the CL MSE decreased monotonically and, importantly, the model dynamics are non-oscillatory. For the other models, the CL interpolation MSE spiked during early curriculum stages (Fig. A.7). This suggested that CL can overfit when early curriculum stages are fitted to data subsets that are not representative of the full time-series data, for example when they cover only part of an oscillation period. By contrast, because CMS is trained on the full time series, it can explain why it often converged faster.

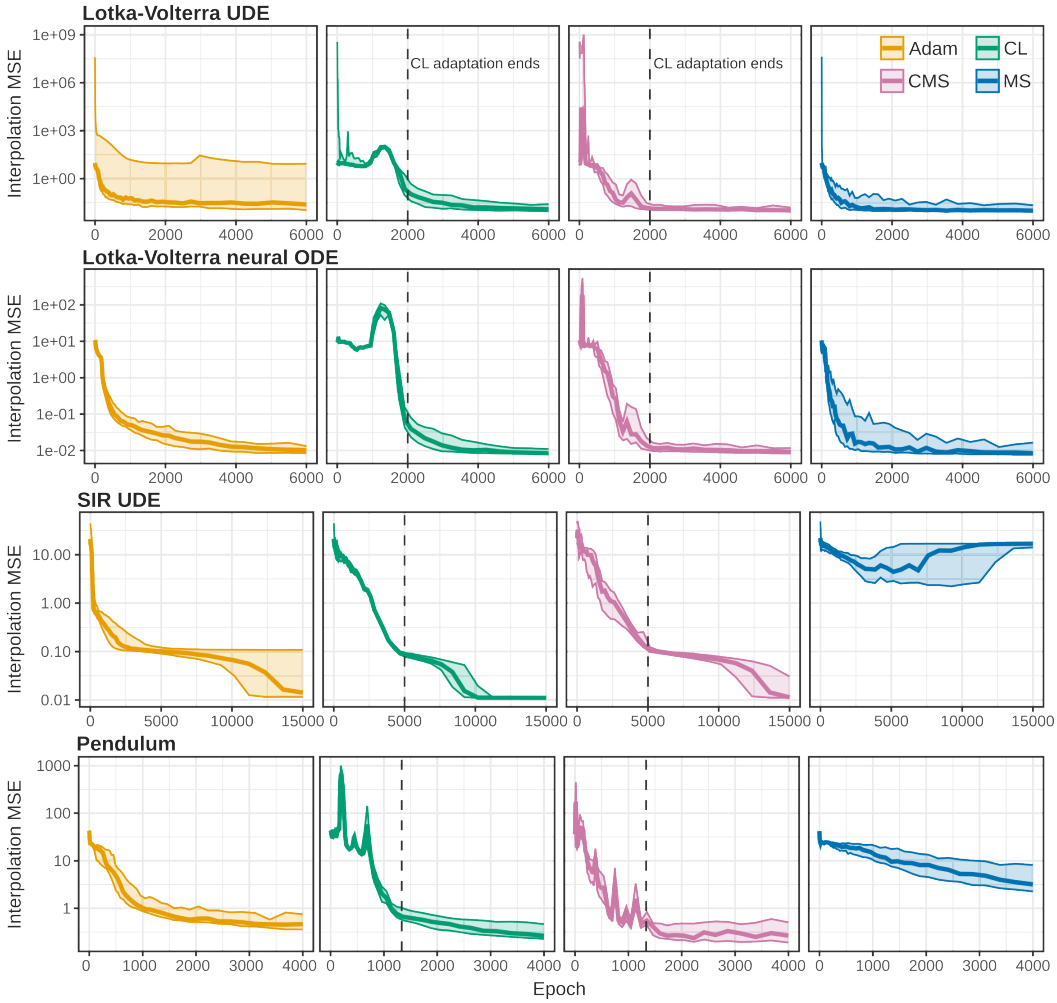

Figure A.7: Interpolation-MSE traces over training epochs for the Lotka–Volterra UDE (top), Lotka–Volterra NODE (upper middle), SIR UDE (lower middle), and pendulum UDE (bottom) models. Results are based on 100 multistarts with random parameter initializations shown on a $\log_{10}$ scale. Solid lines denote the median and shaded intervals the 25th–75th percentiles across multistarts. Dashed vertical lines indicate the end of curriculum adaptation for the relevant strategies. Adam+BFGS is excluded because we could not extract training traces from the BFGS `Fides` optimizer.

A.4 Additional details on mechanistic benchmarks

This section provides additional discussion of curriculum learning for the mechanistic ODE benchmarks.

Since CL is competitive with CMS on a subset of both mechanistic ODE benchmarks and UDE/NODE models (Figs. 2–4), we asked under which conditions CL performs well. Although no single definitive pattern emerges, three models where CL performs strongly (Boehm, Fiedler, and Sneyd; Fig. 4b) share a common feature: each curriculum stage contains all observables (measured model outputs) and all simulation conditions, i.e. experiments in which the model is simulated under different control parameters. This suggests that CL works well in cases where the number of curriculum stages can be chosen such that early curriculum stages are representative of the full problem. To further test this hypothesis, we considered an additional benchmark model, the Bachmann model, which has multiple simulation conditions, many of which appear only at late time points and therefore only in later CL stages. In this case, CL performed poorly (Fig. A.8).

While representativeness provides useful intuition for when CL performs well, it does not determine how to chose the number of curriculum stages. Adding more total representative stages improves performance for some mechanistic models (Boehm, Fiedler, and Sneyd), whereas others (Okuonghae and Schwen) perform best with only two stages (Fig. A.8). Similarly, for the NODE/UDE benchmarks, using more representative curriculum stages did not consistently improve training performance for 2/4 models (Tab. C.2). These results suggest that CL is most effective when the problem changes gradually across stages, but that the optimal number of stages remains problem-dependent.

In summary, as CL has fewer hyperparameters and is faster than CMS, it remains attractive when it performs well. Our results suggest that this occurs when early curriculum stages are representative of the full time series, for example when all observables are present in early stages or when the dynamics are relatively simple (e.g. non-oscillatory). Otherwise, CMS performs better while remaining competitive on easier problems. A relevant future direction is to develop practical guidance on when CL is sufficient.

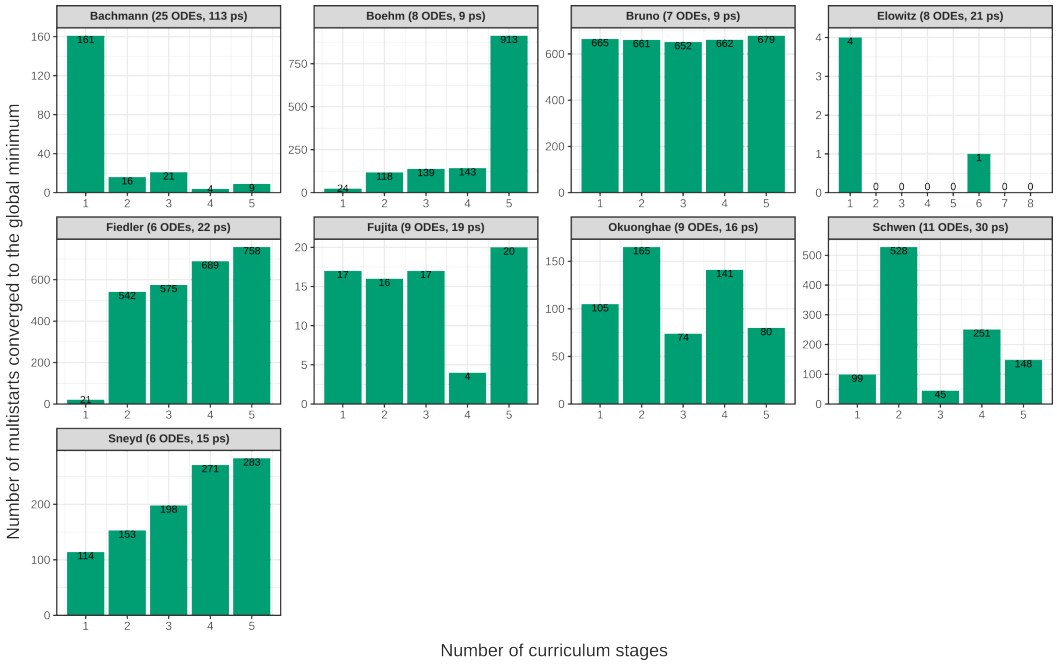

Figure A.8: Number of runs converging to the global minimum out of 1000 multistarts for curriculum learning with different numbers of curriculum stages. For example, for the Boehm model, 24 runs converge with one curriculum stage, corresponding to standard single-shooting training, whereas 913 runs converge with five stages. The models are the same as in Fig. 4 and Tab. B.1, with the addition of the Bachmann model. For each model, the information within parenthesis denotes the number of model states and number of parameters (ps) to estimate.

A.5 Complementary UDE/NODE benchmarks in Julia

To assess the robustness of CMS to the software ecosystem and gradient-computation method, we complemented the main (Sec. 5) Python UDE/NODE benchmarks with experiments in the Julia SciML ecosystem [5], using `PEtab.jl` [49]. As described in App. C, with `PEtab.jl` we used forward-mode automatic differentiation via `ForwardDiff.jl` [62] to compute gradients. In this setting, sensitivities are computed alongside the discretized ODE solution, so ODE solver error control also acts on the dual components carrying gradient information. This can therefore be considered an optimize-then-discretize gradient-computation scheme. This contrasts with the main Python `diffrax`-based benchmarks in Sec. 5 (Figs. 1–2), where gradients were computed using `RecursiveCheckpointAdjoint`, corresponding to a discretize-then-optimize scheme.

To avoid the computational cost of re-tuning all training strategies, we used the same hyperparameters as in Sec. 5 (Tabs. C.1–C.4) for single-shooting Adam, curriculum learning (CL), multiple shooting (MS), and curriculum multiple shooting (CMS). We did not consider the Adam+BFGS training strategy. As in the main experiments, we ran 100 multistarts. We considered the Lotka–Volterra UDE (Eq. B.2), Lotka–Volterra NODE (Eq. B.3), and SIR UDE (Eq. B.4). We also attempted the double-pendulum UDE (Eq. B.8), but excluded it because a single multistart required more than five days and exceeded our compute budget. This substantial runtime is because forward-mode automatic differentiation scales poorly with the number of estimated parameters [39], and the double pendulum has the largest neural network. All models were trained for the same number of epochs as in the main experiments (App. C). Noticeably, one implementation detail differs from the main CMS experiments. In the main text, we penalize the full overlap between adjacent windows. In the `PEtab.jl` implementation, CMS is implemented at the abstraction level of the PEtab parameter-estimation standard [48], which only allowed us to penalize the first overlapping point, i.e. the MS window boundaries.

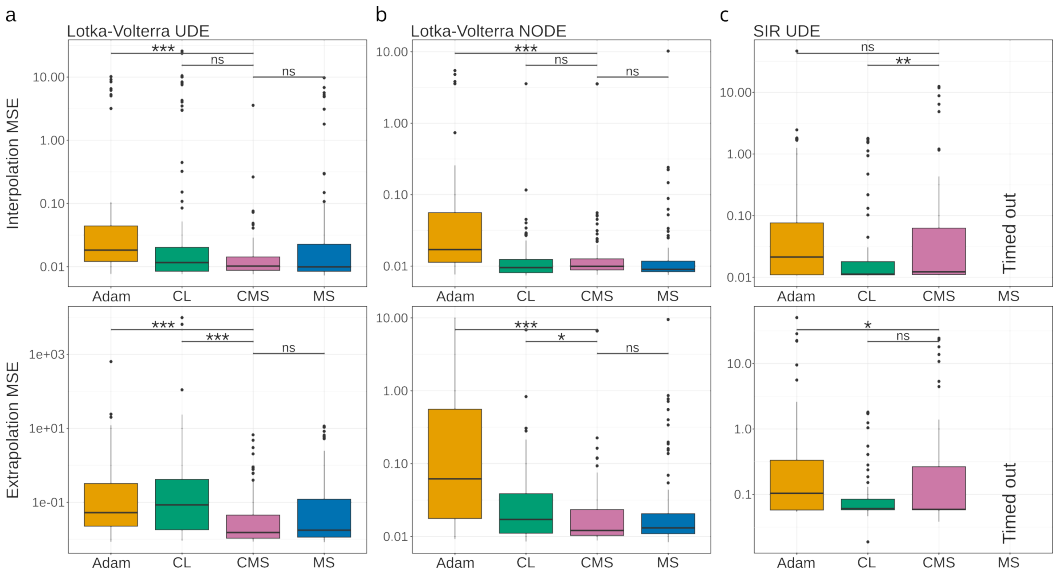

Figure A.9: Complementary UDE/NODE benchmark results from 100 multistarts with random parameter initializations using the Julia SciML ecosystem. Interpolation MSE (top) and extrapolation MSE (bottom) are shown on a $\log_{10}$ scale for the (a) Lotka–Volterra UDE, (b) Lotka–Volterra NODE, and (c) SIR UDE models. P-values were computed using the Wilcoxon rank-sum test and Holm-corrected within each model. Significance levels are denoted by ns ($p \geq 0.05$), * ($p < 0.05$), ** ($p < 0.01$), and *** ($p < 0.001$).

As in the main experiments (Sec. 5), CMS was consistently top-performing in terms of median extrapolation MSE (Fig. A.9 and Tab. A.6). For the Lotka–Volterra UDE (Fig. A.9a), CMS performed similarly to the second-best method, multiple shooting (0.0152 vs. 0.0176; $p = 0.44$), and significantly outperformed CL and Adam in extrapolation. For the Lotka–Volterra NODE (Fig. A.9b), CMS again matched the second best MS strategy (0.0121 vs. 0.0131; $p = 0.56$). We note that plain single-shooting Adam differed in performance from the main Python benchmark, with interpolation MSE

increasing from 0.0103 to 0.0171 and extrapolation MSE from 0.016 to 0.062 (Tabs. A.2 and A.6). This highlights that the software ecosystem and gradient-computation method can affect training even when using the same hyperparameters and ODE-solver algorithm. Finally, for the SIR UDE (Fig. A.9c), CMS performed again similarly to the second-best method, CL (0.0589 vs. 0.0605; p = 0.69), while having slightly worse interpolation MSE (0.0123 vs. 0.0113; $p = 1.6 \times 10^{-3}$). The MS strategy timed out for the SIR UDE, as no multistart completed within the 48-hour compute budget. This reflects poor MS training on this model, which inspection of training traces confirmed.

In summary, CMS remains robust when changing both the software ecosystem and gradient-computation method. This robustness holds despite using a different continuity-penalty scheme, where re-tuning could potentially improve performance. The differences in plain Adam performance between the Julia and Python benchmarks further show that implementation details can affect training performance and the optimal hyperparameter configuration.

Table A.6: Benchmark results for the Lotka–Volterra UDE, Lotka–Volterra NODE, and SIR UDE models in the Julia SciML ecosystem benchmark. Interpolation and extrapolation MSE are reported as $\mathrm{median} \pm \mathrm{median\ absolute\ deviation}$ for each training strategy. The best interpolation and extrapolation values within each model are shown in bold.

Model	Strategy	Interpolation MSE	Extrapolation MSE
Lotka–Volterra UDE	Adam	0.0183 ± 0.0122	0.0525 ± 0.0611
	CL	0.0116 ± 0.00504	0.0851 ± 0.111
	CMS	0.0102 ± 0.00260	$\mathbf{0.0152 \pm 0.00805}$
	MS	$\mathbf{0.00995 \pm 0.00303}$	0.0176 ± 0.0123
Lotka–Volterra NODE	Adam	0.0171 ± 0.0116	0.0619 ± 0.0754
	CL	0.00954 ± 0.00243	0.0171 ± 0.0107
	CMS	0.00994 ± 0.00199	$\mathbf{0.0121 \pm 0.00361}$
	MS	$\mathbf{0.00901 \pm 0.00131}$	0.0131 ± 0.00559
SIR UDE	Adam	0.0213 ± 0.0156	0.104 ± 0.0700
	CL	$\mathbf{0.0113 \pm 0.000667}$	0.0605 ± 0.00563
	CMS	0.0123 ± 0.00202	$\mathbf{0.0589 \pm 0.0169}$

B Benchmark models and data

This section provides additional details for each benchmark model and its associated time-series dataset.

B.1 Lotka–Volterra System

The Lotka–Volterra system is a simplified model of predator (x) and prey (y) population dynamics given by

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \alpha x - \beta xy, \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= -\delta y + \gamma xy, \end{aligned} \tag{B.1}$$

where α is the prey birth rate, δ is the predator death rate, and (β, γ) describe the interaction strength.

Model architectures. We consider two model classes: a UDE and a black-box NODE. Following Rackauckas et al. [5], for the UDE we use a structure that incorporates known birth and death terms, while learning the interaction terms:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \alpha x(t) + \mathbf{NN}_1\big(x(t), y(t); \boldsymbol{\theta}_n\big), \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= -\delta y(t) + \mathbf{NN}_2\big(x(t), y(t); \boldsymbol{\theta}_n\big). \end{aligned} \tag{B.2}$$

For the black-box NODE, the full right-hand side (RHS) is represented by the neural network:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= \mathbf{NN}_1\big(x(t), y(t); \boldsymbol{\theta}_n\big), \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \mathbf{NN}_2\big(x(t), y(t); \boldsymbol{\theta}_n\big). \end{aligned} \tag{B.3}$$

For both models, $\mathbf{NN}(x, y; \boldsymbol{\theta}_n)$ is a feed-forward neural network that takes (x, y) as input and has layer widths $[2, 10, 10, 2]$. Each layer includes a bias term; swish activations are used in all hidden layers, and the identity function is used in the output layer.

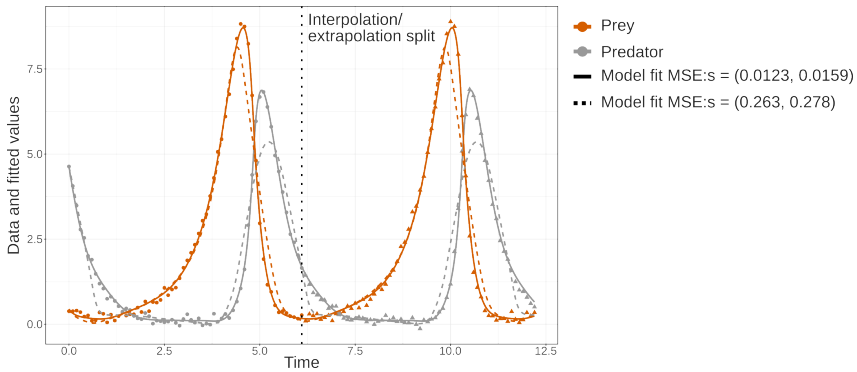

Figure B.1: Lotka–Volterra UDE and NODE training and validation data for the prey and predator observables. Lines show two representative model fits with different interpolation/training and extrapolation/validation errors. MSE values are reported in the legend as (interpolation, extrapolation).

For training, we follow [16] and assume known initial values $(x_0, y_0) = (0.44249296, 4.6280594)$. For the UDE, we estimate the mechanistic parameters (α, δ) and the neural-network parameters $\boldsymbol{\theta}_n$, while for the NODE we estimate only $\boldsymbol{\theta}_n$.

Training and validation data. For both Lotka–Volterra models, we use simulated data. Following [16], for the main experiments in Figs. 1 and 2, we simulate Eq. B.1 on the interval $[0.0, 12.2]$ with sampling interval $\Delta t = 0.1$, initial values $(x_0, y_0) = (0.44249296, 4.6280594)$, and parameter values $(\alpha, \beta, \gamma, \delta) = (1.3, 0.9, 0.8, 1.8)$. We noise-corrupt the measurements with Gaussian noise with zero mean and standard deviation equal to 5% of the mean of each trajectory. The data are split into training and validation using a 50/50, so we train on measurements in $t \in [0.0, 6.1]$ (62×2 measurements) and validate on measurements in $t \in [6.2, 12.2]$ (62×2 measurements). The data, together with representative fits for different MSE values, are shown in Fig. B.1.

For the noise-robustness experiments (Fig. A.2), we generate data as above but with noise levels of 10%, 20%, and 50%. For the simulation-horizon experiments, we instead simulate the system on $[0.0, 18.2]$ and $[0.0, 22.2]$, using the same sampling interval $\Delta t = 0.1$. For both robustness experiments, we apply a 50/50 train-validation split.

B.2 SIR UK UDE System (SIR UDE)

The SIR (susceptible, infected, recovered) UDE model was introduced in [12], where Dandekar et al. augmented a standard SIR model to better capture the early dynamics of the COVID-19 outbreak.

Model architecture. We use essentially the same model structure as in [12], with dynamics

$$\begin{aligned}
\frac{\mathrm{d}S}{\mathrm{d}t} &= -\frac{\beta S(t) I(t)}{N}, \\
\frac{\mathrm{d}I}{\mathrm{d}t} &= \frac{\beta S(t) I(t)}{N} - \gamma I(t) - \mathbf{NN}_1\big(S(t)/N, I(t)/N, R(t)/N; \boldsymbol{\theta}_n\big) \cdot I(t), \\
\frac{\mathrm{d}R}{\mathrm{d}t} &= \gamma I(t) + \delta T(t), \\
\frac{\mathrm{d}T}{\mathrm{d}t} &= \mathbf{NN}_1\big(S(t)/N, I(t)/N, R(t)/N; \boldsymbol{\theta}_n\big) \cdot I(t) - \delta T(t),
\end{aligned} \tag{B.4}$$

where T denotes the quarantined population, (β, γ, δ) are unknown mechanistic parameters, $N = S + I + R$ is the total population, and $\mathbf{NN}$ is a neural network parameterized by $\boldsymbol{\theta}_n$ that represents a time-varying quarantine strength. Following [12], $\mathbf{NN}$ is a feed-forward network with layer widths $[3, 10, 1]$, bias terms in each layer, swish activations in the hidden layers, and the identity function in the output layer. Our implementation differs from Dandekar et al. [12] in one respect: we use normalized inputs $(S/N, I/N, R/N)$ rather than the unnormalized states. This choice is motivated by the observation that unnormalized inputs are highly multi-scale (e.g. $S \propto 10^7$ while $R \propto 10^3$), leading to large gradients only in the first layer and vanishing gradients in subsequent layers.

For model training, we use data from the United Kingdom (UK) (see below) and take the initial time to be the day on which approximately 500 infected cases were observed, corresponding to initial values $(S, I, R, T) = (66000000, 762, 26, 10)$. Because the system is only partially observed, with no direct measurements of S or T, we match model outputs to the observables

$$\begin{aligned}
obs_1(t) &= I(t) + T(t), \\
obs_2(t) &= R(t).
\end{aligned} \tag{B.5}$$

We estimate the mechanistic parameters (β, γ, δ) and the neural-network parameters $\boldsymbol{\theta}_n$.

Training and validation data. We use the same UK dataset as in [12], originally obtained from the CSSE at Johns Hopkins University. The dataset contains 79 daily measurements of infected cases (I_{data}) and recovered cases (R_{data}). As in [12], we use a MSE loss comparing measurements on log-scale:

$$\mathcal{L} = \frac{1}{N_{\mathrm{obs}}} \sum_j \Big(\big(\log(obs_1(t_j)) - \log(I_{\mathrm{data}}(t_j)) \big)^2 + \big(\log(obs_2(t_j)) - \log(R_{\mathrm{data}}(t_j)) \big)^2 \Big), \tag{B.6}$$

where N_{obs} denotes the total number of observed measurements across both observables. Note, the I_{data} is assumed to include both the infected (I) and quarantine (T) population from the model. We use a 70/30 split, thus training on measurements in $t \in [0, 63]$ (64×2 measurements) and validate on measurements in $t \in [64, 79]$ (32×2 measurements), with sampling interval $\Delta t = 1.0$. The data with fits for different MSE values are shown in Fig. B.2.

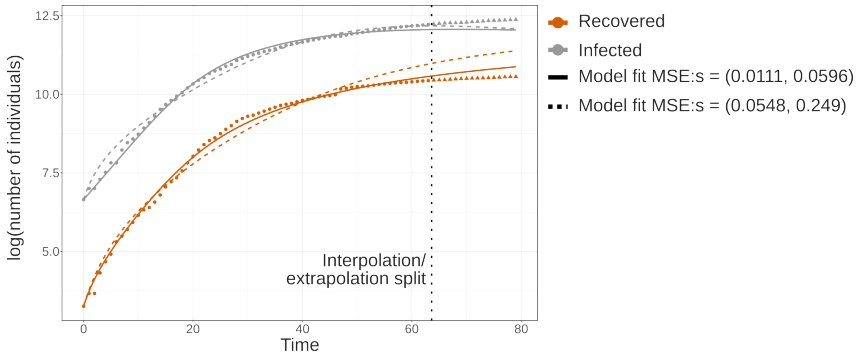

Figure B.2: SIR UDE training and validation data for the infected and recovered observables (Eq. B.5). Lines show two representative model fits with different interpolation/training and extrapolation/validation errors. MSE values are reported in the legend as (interpolation, extrapolation).

B.3 Double Pendulum System

The double pendulum is a canonical system in classical mechanics. It is governed by four states $(\theta_1, \theta_2, \omega_1, \omega_2)$, describing the angles and angular velocities of the two pendulums relative to the vertical. Its dynamics are naturally second order, but can be written in first-order form as

$$\begin{aligned} \frac{\mathrm{d}\theta_i}{\mathrm{d}t} &= \omega_i(t), \qquad i = 1, 2, \\ \frac{\mathrm{d}\omega_i}{\mathrm{d}t} &= f_i\big(\omega_1(t), \omega_2(t), \theta_1(t), \theta_2(t)\big), \qquad i = 1, 2, \end{aligned} \tag{B.7}$$

where f_1 and f_2 can be derived from the Lagrangian formulation of classical mechanics; the full expressions are given in [16]. The double pendulum is a challenging oscillatory system to learn and therefore provides a strong stress test for our CMS method.

Model architecture. Following [16], we use a physics-informed UDE structure of the form

$$\begin{aligned} \frac{\mathrm{d}\theta_i}{\mathrm{d}t} &= \omega_i, \qquad i = 1, 2, \\ \frac{\mathrm{d}\omega_i}{\mathrm{d}t} &= \mathbf{NN}_i\big(\omega_1(t), \omega_2(t), \theta_1(t), \theta_2(t); \boldsymbol{\theta}_n\big), \qquad i = 1, 2, \end{aligned} \tag{B.8}$$

where $\mathbf{NN}$ is a feed-forward network with layer widths $[4, 50, 50, 2]$, bias terms in each layer, swish activations in the hidden layers, and the identity function in the output layer. The data are fully observed, so we use initial values $(\theta_1, \theta_2, \omega_1, \omega_2) = (-1.13, -1.30, -0.53, -6.15)$. For training, we estimate the neural-network parameters $\boldsymbol{\theta}_n$.

Training and validation data. We use the same real double-pendulum dataset as in Ko et al. [16], originally generated by Schmidt and Lipson [51]. While the double pendulum can exhibit chaotic behaviour, we follow Ko et al. [16] and evaluate on the same short trajectory segment. This short-horizon setting limits the extent to which long-term chaotic divergence affects the comparison of training strategies. We use a 70/30 split, training on measurements in $t \in [0.0, 1.71]$ (160×4 measurements) and validating on measurements in $t \in [1.72, 2.14]$ (40×4 measurements). The data with representative fits are shown in Fig. B.3.

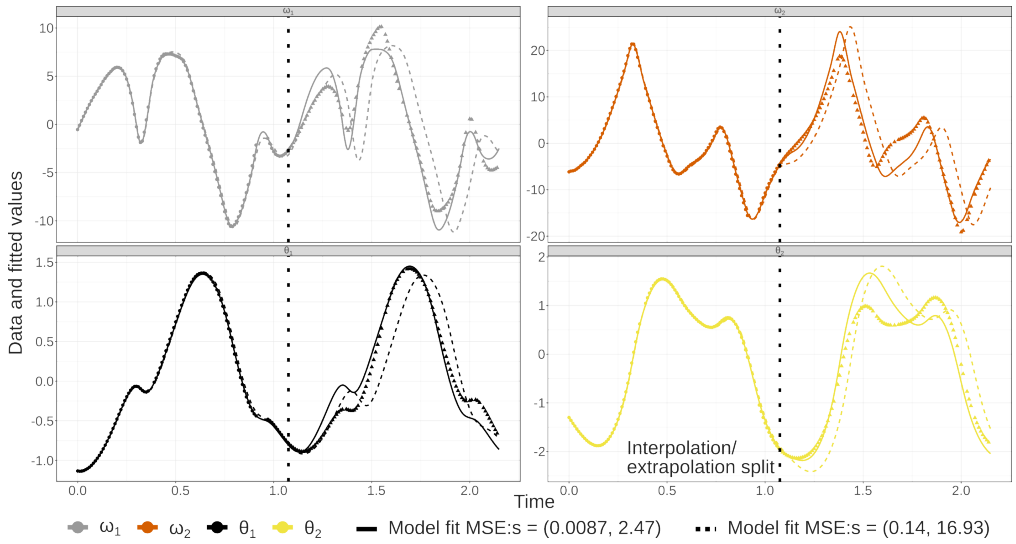

Figure B.3: Pendulum training and validation data for the $(\omega_1, \omega_2, \theta_1, \theta_2)$ observables. Lines show two representative model fits with different interpolation/training and extrapolation/validation errors. MSE values are reported in the legend as (interpolation, extrapolation).

B.4 Reaction–Diffusion System

To test scaling to high-dimensional dynamics, we used the FitzHugh–Nagumo reaction–diffusion system [63, 64] for an activator $u(\mathbf{x}, t)$ and an inhibitor $v(\mathbf{x}, t)$ on a two-dimensional domain,

$$\begin{aligned} \frac{\partial u}{\partial t} &= D_u \nabla^2 u + u - \frac{u^3}{3} - v, \\ \frac{\partial v}{\partial t} &= D_v \nabla^2 v + \varepsilon\,(u + a - bv), \end{aligned} \tag{B.9}$$

with $(D_u, D_v) = (1, 1)$, $(a, b) = (0.1, 1)$ [52], and time-scale separation $\varepsilon = 0.005$. For these parameters, the homogeneous fixed point $(u^*, v^*) \approx (-0.669, -0.569)$ lies on the unstable middle branch of the cubic nullcline, so the medium is oscillatory. We discretized Eq. B.9 on a 64×64 grid over a square domain of side length 100 ($\Delta x = 1.5625$) using a five-point Laplacian with Neumann (zero-flux) boundary conditions, giving $2 \times 64^2 = 8192$ states. The initial condition varies u linearly along one spatial axis and v along the other, each spanning the limit cycle around the fixed point, so that a single phase defect at the domain centre seeds a rotating spiral wave. The spiral rotation period is approximately 258 time units.

Model architectures. Our model structure is adapted from APHYNITY [53]. The UDE keeps the diffusion operator mechanistic with unknown coefficients and learns the reaction terms:

$$\frac{\partial}{\partial t}\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} D_u \nabla^2 u \\ D_v \nabla^2 v \end{pmatrix} + \mathbf{NN}\big(u, v; \boldsymbol{\theta}_n\big). \tag{B.10}$$

For the black-box NODE, the neural network represents the full right-hand side:

$$\frac{\partial}{\partial t}\begin{pmatrix} u \\ v \end{pmatrix} = \mathbf{NN}\big(u, v; \boldsymbol{\theta}_n\big). \tag{B.11}$$

For both models, $\mathbf{NN}$ is a three-layer convolutional network with 3×3 kernels, channel widths $[2, 16, 16, 2]$, bias terms, and ReLU activations in the hidden layers. Its padding matches the Neumann boundary condition of the diffusion operator. Unlike APHYNITY, we omit batch normalization, which is ill-suited to an ODE right-hand side evaluated by an adaptive solver. For the UDE, we

estimate the diffusion coefficients (D_u, D_v) and $\boldsymbol{\theta}_n$; for the NODE, only $\boldsymbol{\theta}_n$. The initial state is set to the observed field at $t = 0$ and held fixed during training; since the data are noise-free and fully observed, this equals the true initial condition. For MS and CMS, the initial states of all subsequent windows are estimated (Sec. 3.3).

Training and validation data. We simulated Eq. B.9 with `Tsit5` and $(\mathrm{atol}, \mathrm{rtol}) = (10^{-8}, 10^{-8})$ and sampled both species on the full grid with $\Delta t \approx 14.75$. The data were not noise-corrupted. We train on two spiral rotations, $t \in [0, 516.4]$ (36 time points $\times$ 8192 measurements), and validate on the following rotation, $t \in [531.2, 782.0]$ (18×8192 measurements), i.e. a 2:1 split. Because the two species vary on different scales (standard deviations of 1.65 and 0.375), each species is standardized by its training-data standard deviation in the MSE loss, so that both contribute equally. All reported MSE values use this standardized loss.

B.5 Mechanistic ODE Models

The mechanistic ODE models are drawn from the PEtab benchmark collection, which contains a diverse set of models with real time-series data [27, 48]. The collection primarily comprises models from cell biology (e.g. cell signaling and gene regulation) and epidemiology (e.g. COVID-19 progression). From a numerical perspective, these models present a range of challenging training features. Most exhibit stiff dynamics arising from disparate time scales (e.g. fast and slow chemical reactions) [49], state variables spanning multiple orders of magnitude, and non-identifiability leading to sloppy parameter landscapes [27, 65]. Most models are also only partially observed, with observables typically given as functions of multiple state variables (e.g. $obs_1 = x + y$). Measurements are noisy and often irregularly sampled, with different observables measured at different time points. A subset of models also includes multiple experimental conditions, requiring simulations under different control parameter values for different measurements. We selected eight representative models of varying size from this collection, whose key properties are summarized in Tab. B.1. Overall, they provide a robust stress test for our training strategy.

Table B.1: Summary of the eight mechanistic ODE benchmark models fitted to real data. For each model, the number of states, parameters to estimate (Params.), measurements (Meas.), observables (Obs.), experimental conditions (Cond.), as well as the application domain is reported.

Model	States	Params.	Meas.	Obs.	Cond.	Domain
Boehm	8	9	48	3	1	Biology
Bruno	7	13	77	5	6	Biology
Elowitz	8	21	58	1	1	Biology
Fiedler	6	22	72	2	3	Biology
Fujita	9	19	144	3	6	Biology
Okuonghae	2	9	92	1	1	Epidemiology
Schwen	11	30	286	4	19	Biology
Sneyd	6	15	135	1	9	Biology

C Additional experiment details

This section provides additional experimental details, including the software and ODE solvers used, parameter initialization procedures, implementation details for the training strategies evaluated in Sec. 5, and the hyperparameter optimization and selection procedures.

C.1 Software implementation

All training strategies considered in this study were implemented by us. The main NODE and UDE benchmarks were implemented in Python using `diffrax` and `equinox` [38]. Julia was used for the mechanistic ODE benchmarks, as well as for complementary robustness experiments on the UDE benchmarks, using `PEtab.jl` [49]. Code for all benchmarks is available at `https://github.com/frohlich-lab/CurriculumMultipleShooting_NeurIPS_2026`.

For the Python `diffrax`-based benchmarks, we used the explicit `Tsit5` ODE solver [66], with tolerances $(\mathrm{atol}, \mathrm{rtol}) = (10^{-8}, 10^{-8})$. Gradients were computed using `RecursiveCheckpointAdjoint`, corresponding to a discretize-then-optimize approach in which gradients are obtained by differentiating through the numerical solver.

For the Julia `PEtab.jl`-based benchmarks, we used `Tsit5` with $(\mathrm{atol}, \mathrm{rtol}) = (10^{-8}, 10^{-8})$ for the UDE robustness experiments, and the stiff `Rodas5P` solver [67], also with $(\mathrm{atol}, \mathrm{rtol}) = (10^{-8}, 10^{-8})$, for the mechanistic ODE benchmarks. This choice was motivated by previous results showing that models from the PEtab benchmark collection are generally stiff [49]. For gradient computation, we used forward-mode automatic differentiation via `ForwardDiff.jl` [62], where derivatives are propagated through the numerical solve using dual numbers. In this setting, sensitivities are computed alongside the discretized ODE solution, so solver error control also acts on the dual components carrying gradient information. This has been argued to yield more reliable gradients [39], although other work suggests that discretize-then-optimize approaches can produce more accurate gradients [36].

For the reaction–diffusion benchmark, we additionally capped the solver step size at the explicit stability limit of the mechanistic diffusion operator at the current (D_u, D_v), and set the number of `RecursiveCheckpointAdjoint` checkpoints to approximately the number of solver steps per solve (4096), which reduced runtime by about $3\times$ relative to the default.

C.2 Computational resources

All benchmarks were run on CPUs on the NEMO cluster at the Francis Crick Institute. The average time for a Lotka-Volterra training run was between 3 and 15 minutes depending on the training approach, and the requested amount of memory was 16GB. The SIR model trained in under 15 minutes on average also with a requested memory of 16GB. The double pendulum model trained in under 10 minutes on average, with the exception of the Adam+BFGS training approach, where some runs timed out completely at a cut-off of 2 hours. The requested memory for pendulum training runs was 32GB.

The reaction–diffusion benchmark was substantially more expensive. Each training run used 8 CPU cores and 48GB of memory, and took between approximately 9 hours (NODE, Adam) and 40 hours (CMS) on average. The final 800 runs (2 models $\times$ 4 strategies $\times$ 100 multistarts) required approximately 1.5×10^5 CPU-hours.

C.3 Initialization of estimated parameters

For both the mechanistic and NODE/UDE models, we performed multiple training runs from randomly sampled initial parameter values for all training strategies. For the mechanistic models, parameters were initialized using Latin hypercube sampling, which has been shown to outperform uniform random sampling in this setting [29]. All mechanistic models provide parameter bounds, and sampling was performed within those bounds.

For the UDE models with unknown mechanistic parameters, namely the SIR and Lotka–Volterra UDEs, mechanistic parameters were Latin hypercube sampled on the $\log_{10}$ scale within two orders of magnitude of a reference parameter vector $\boldsymbol{\theta}_{\mathrm{ref}}$. For the Lotka–Volterra models, $\boldsymbol{\theta}_{\mathrm{ref}}$ was the true

parameter vector; for the SIR model, it was the initial parameter vector used for training in [12]. Neural-network parameters were initialized with zero biases and Glorot-uniform weights [68], using a gain of 0.1. This gain is smaller than the default, as NODEs often benefit from initialization at smaller parameter values [38].

For the reaction–diffusion UDE, the diffusion coefficients (D_u, D_v) were Latin hypercube sampled on the $\log_{10}$ scale within one order of magnitude of the true values. The convolutional networks used the default `equinox` initialization with the weights and bias of the final layer scaled by 0.1.

C.4 Training budgets

NODE and UDE benchmarks. Following Ko et al. [16], we trained the Lotka–Volterra UDE and NODE models for 6000 epochs, and the double-pendulum UDE for 4000 epochs. We trained the SIR UDE for $15\,000$ epochs. This budget was chosen to allow the training strategies to converge, while remaining substantially smaller than the $60\,000$ epochs used in the original publication [12]. To ensure a fair comparison, all training strategies were run for the same number of epochs for each model. For the reaction–diffusion models, we trained for 500 epochs, owing to their computational cost.

Mechanistic ODE benchmarks. For the mechanistic ODE benchmarks, we used a Newton trust-region optimizer with adaptive step lengths and automatic termination criteria. Therefore, different methods did not necessarily run for the same number of iterations. As is standard for these benchmarks [49], we allowed a maximum of 1000 optimizer iterations per run.

C.5 Training strategy implementation details

This section provides additional implementation details for the training strategies evaluated in Sec. 5, as well as the output regularization used in the UDE benchmarks.

C.5.1 Output regularization

Neural network output regularization was introduced by Philipps et al. [21] to prevent the neural network component of a UDE from dominating the dynamics and effectively overriding the mechanistic component, for example by driving mechanistic parameters toward zero. The regularization term is defined in Eq .3, which is repeated here for completness:

$$\lambda_O \left(\int_{t_0}^{T} \|\mathbf{NN}(\mathbf{u}; \boldsymbol{\theta})\|_2 \, \mathrm{d}t \right)^2,$$

where $[t_0, T]$ is the simulation interval, $\|\cdot\|_2$ is the ℓ_2 norm, $\boldsymbol{\theta}$ denotes the neural network parameters, and λ_O is a hyperparameter controlling the strength of the regularization. This term is added to the loss function in Eq. 2. In practice, the integral is computed by augmenting the UDE with an additional state $\phi(t)$,

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = \|\mathbf{NN}(\mathbf{u}; \boldsymbol{\theta})\|_2, \qquad \phi(t_0) = 0,$$

and evaluating $\lambda_O \phi(T)^2$ at the final time.

C.5.2 Curriculum learning implementation details

Our curriculum learning strategy is described in Sec. 3.2. It introduces three tunable parameters: the number of curriculum stages n_c, the number of epochs per stage, and the stage boundaries t_i. Among these, we treated only the number of stages n_c as a tunable hyperparameter; its tuning procedure in App. C.6.

For the other two parameters, we used default settings that performed well across benchmarks. For the number of epochs per stage, given a fixed total training budget, we found it effective to allocate equal numbers of epochs to stages $i = 1, \ldots, n_c - 1$ over the first third of training, followed by training on the full time series for the remaining two thirds. For the stage boundaries t_i, we chose

them so that each stage contained approximately the same number of measurements, with at most a one-measurement difference between stages. As discussed in Sec. 5, the boundaries should also be chosen so that the optimization does not change too abruptly between stages, for which distributing measurements approximately evenly is a reasonable heuristic.

For the reaction–diffusion benchmark, we instead allocated an equal number of epochs to every stage, including the final full-data stage. With the tuned values, this gives 458 curriculum epochs for CL ($n_c = 12$) and 400 for CMS ($n_w = 5$), i.e. the final single-shooting stage spans the last 42 and 100 of the 500 epochs, respectively.

C.5.3 Multiple shooting details

Our multiple shooting strategy is described in Sec. 3.3 and closely follows the implementation in the `DiffEqFlux.jl` Julia package [25]. Other variants are possible [47, 24], such as treating and solving the multiple shooting objective as an equality-constrained optimization problem [24]. We focus on the approach of Rackauckas et al. [25], as this is the variant extended by our curriculum multiple shooting method. Multiple shooting introduces three tuning parameters: the number of windows n_w, the continuity penalty strength λ_{cont}, and the window boundaries t_i. Among these, we treat the first two as tunable hyperparameters; their tuning procedure can be found in App. C.6.

For the window boundaries, we followed the same principle as for curriculum learning and chose them so that each window contained approximately the same number of measurements. Importantly, we placed splits only at times t_i for which measurements were available. To ensure that each window over $[t_{i-1}, t_i]$ included all measurements in that interval, we duplicated measurements at the boundaries when needed. It is also possible to split at times without measurements, but for a window to be informative it should contain at least one measurement.

C.5.4 Curriculum multiple shooting details

Our curriculum multiple shooting approach is described in detail in Sec. 4. Here, for completeness, we note that we set the number of epochs per stage as in curriculum learning and the initial multiple-shooting window boundaries t_i as in multiple shooting.

C.5.5 Adam + BFGS details

For the Adam+BFGS strategy, we first trained the model with Adam and then refined the resulting parameters using the `Fides` Newton trust-region optimizer with a BFGS Hessian approximation [57]. For example, for the Lotka–Volterra UDE benchmark (Fig. 1), each Adam-trained run was subsequently optimized with `Fides`. We chose `Fides` because it has been shown to perform well for mechanistic ODE models [57, 49]. As a trust-region method, it adapts the step size based on the local accuracy of the trust-region approximation.

C.5.6 Mechanistic ODE model details

For the mechanistic ODE models, we used the training strategies described above, but replaced Adam with the `Fides` Newton trust-region optimizer because `Fides` has been shown to perform well for mechanistic ODEs [57]. As Fides has adaptive step-length, no learning-rate tuning was required. One tuning parameter required by `Fides` is the Hessian approximation. Here, we used a Gauss–Newton Hessian approximation, which has been shown to outperform BFGS approximations for the mechanistic ODE models considered here [49].

C.6 Hyperparameter tuning and selection

We report the hyperparameter ranges explored for each method together with the final selected values, including those used for the mechanistic models.

C.6.1 UDE and NODE benchmarks

For the UDE/NODE benchmark models (Lotka–Volterra UDE, Lotka–Volterra NODE, SIR UDE, and pendulum UDE), hyperparameters for each training strategy were tuned using a two-step linear sweep. We first tuned the learning rate, as poor learning-rate choices led to poor performance regardless of

the other hyperparameters. After fixing the learning rate, we swept the remaining hyperparameters around central values and selected one value at a time. Configurations were compared using the training/interpolation loss across 100 random multistarts, summarized by the median. We generally selected the value with the lowest interpolation median loss, except for output regularization (Eq. 3). Because output regularization is intended to prevent the neural network from dominating the dynamics and absorbing mechanistic components, we selected the largest λ_O whose training loss was not significantly worse than the lowest-median configuration according to a Wilcoxon rank-sum test. We did not perform a full grid search due to the prohibitive computational cost. The explored values and selected settings are reported in Tabs. C.1–C.4.

Notably, output regularization (Eq. 3) was generally tuned to be large only for the Lotka–Volterra UDE (Tabs. C.1–C.4). For the Lotka–Volterra NODE and pendulum models without mechanistic parameters, the neural network must represent the full dynamics, so penalizing its output may hinder learning. In the SIR UDE, the neural network is structurally restricted to a specific quarantine-related component (Eq. B.4), making it less likely to replace the full mechanistic dynamics. In the SIR UDE, the neural network is structurally restricted to a specific quarantine-related component (Eq. B.4); strong output regularization may therefore suppress the mechanism it is intended to learn, rather than prevent it from replacing the full dynamics. In contrast, for the Lotka–Volterra UDE, the neural network can in principle explain most of the right-hand side, for example by driving the mechanistic parameters toward zero.

For the reaction–diffusion models, the cost of a run made tuning on 100 multistarts infeasible. We therefore used the same one-at-a-time sweeps around central values on 5 multistarts, and re-checked the learning rate, continuity penalty, window length, and number of curriculum stages on 10 multistarts (all except the MS NODE learning rate). Values were selected by median interpolation MSE, together with the number of multistarts improving on the current value. Where the two disagreed (the learning rates of the UDE with Adam and with CMS), we kept the previously selected value. All results in Fig. 3 and Tab. A.5 use 100 further multistarts that were not used for tuning, so they are not biased toward the selected configurations. Because the training grid has a fixed number of time points, we tuned the shooting-window length, which determines the number of windows $n_w \in \{3, 5, 7, 12\}$. For CMS, n_w also fixes the number of curriculum stages, whereas for CL we tuned n_c independently. The output regularization for the UDE was fixed a priori at $\lambda_O = 10^4$, which increased the median interpolation MSE of CL by approximately 7% relative to $\lambda_O = 10$ (5 multistarts). Since the training loss tends to favour the weakest regularization, it cannot be used to select λ_O.

Table C.1: Adam training strategy hyperparameter tuning. Shown are the linear search spaces and selected values used for the Adam strategy. Tuned hyperparameters are the learning rate η and the output regularization strength λ_O (Eq. 3).

Model	Parameter	Search space	Selected value
LV-UDE	η	[0.001, 0.01, 0.05, 0.1]	0.05
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 3000.0, 5000.0, 10 000.0, 30 000, 50 000.0]	1000.0
LV-NODE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.01
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 5000.0, 10 000.0, 50 000.0]	0.0
SIR UDE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.005
	λ_O	[0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]	0.001
Double pendulum	η	[0.001, 0.005, 0.01, 0.02 0.05, 0.1]	0.02
	λ_O	[0.0, 1.0, 500.0, 500.0]	0.0
FHN-UDE	η	[10^{-5}, 3×10^{-5}, 10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, 10^{-2}]	10^{-4}
	λ_O	[0.0, 1.0, 10.0, 100.0, 1000.0, 10 000.0, 100 000.0]	10 000.0
FHN-NODE	η	[10^{-5}, 3×10^{-5}, 10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, 10^{-2}]	3×10^{-4}
	λ_O	[0.0]	0.0

Table C.2: Curriculum learning (CL) training strategy hyperparameter tuning. Shown are the linear search spaces and selected values used for the CL strategy. Tuned hyperparameters are the learning rate η, number of curriculum stages n_c and the output regularization strength λ_O (Eq. 3).

Model	Parameter	Search space	Selected value
LV-UDE	η	[0.001, 0.01, 0.05, 0.1]	0.01
	n_c	[3, 5, 9, 13, 16, 21]	13
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 3000.0, 5000.0, 10 000.0, 30 000, 50 000.0]	1000.0
LV-NODE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.01
	n_c	[3, 5, 9, 13, 16, 21]	21
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 5000.0, 10 000.0, 50 000.0]	0.0
SIR UDE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.005
	n_c	[3, 5, 9, 13, 16, 21]	21
	λ_O	[0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]	0.0
Double pendulum	η	[0.001, 0.005, 0.01, 0.02 0.05, 0.1]	0.005
	n_c	[3, 5, 9, 13, 16, 21]	16
	λ_O	[0.0, 1.0, 500.0, 500.0]	0.0
FHN-UDE	η	[10^{-5}, 3×10^{-5}, 10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, 10^{-2}]	10^{-4}
	n_c	[3, 5, 7, 12, 18]	12
	λ_O	[0.0, 1.0, 10.0, 100.0, 1000.0, 10 000.0, 100 000.0, 1 000 000.0]	10 000.0
FHN-NODE	η	[10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, 10^{-2}]	10^{-3}
	n_c	[3, 5, 7, 12, 18]	12
	λ_O	[0.0]	0.0

Table C.3: Multiple shooting (MS) training strategy hyperparameter tuning. Shown are the linear search spaces and selected values used for the MS strategy. Tuned hyperparameters are the learning rate η, number of multiple shooting windows (n_w), continuity penalty (λ_{cont}) and the output regularization strength λ_O (Eq. 3).

Model	Parameter	Search space	Selected value
LV-UDE	η	[0.001, 0.01, 0.05, 0.1]	0.05
	n_w	[3, 4, 5, 9, 13, 16, 21]	16
	λ_{cont}	[0.0, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]	10.0
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 3000.0, 5000.0, 10 000.0, 30 000, 50 000.0]	1000.0
LV-NODE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.05
	n_w	[3, 4, 5, 9, 13, 16, 21]	16
	λ_{cont}	[0.0, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]	10.0
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 5000.0, 10 000.0, 50 000.0]	0.0
SIR UDE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.1
	n_w	[4, 5, 6, 9, 16]	4
	λ_{cont}	[0.0, 0.1, 1.0, 10.0, 100.0, 1000.0, 10 000.0]	10 000.0
	λ_O	[0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]	1.0
Double pendulum	η	[0.001, 0.005, 0.01, 0.02 0.05, 0.1]	0.05
	n_w	[6, 8, 13, 27, 32, 40, 54]	13
	λ_{cont}	[0.0, 0.1, 100.0, 500.0]	100.0
	λ_O	[0.0, 1.0, 100.0, 500.0]	0.0
FHN-UDE	η	$[10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, 10^{-2}]$	10^{-3}
	n_w	[3, 5, 7, 12]	5
	λ_{cont}	[0.0, 0.01, 0.1, 1.0, 10.0]	0.1
	λ_O	[0.0, 1.0, 10.0, 100.0, 1000.0, 10 000.0, 100 000.0]	10 000.0
FHN-NODE	η	$[10^{-4}, 3 \times 10^{-4}, 10^{-3}, 3 \times 10^{-3}, 10^{-2}]$	10^{-2}
	n_w	[3, 5, 7, 12]	5
	λ_{cont}	[0.0, 0.01, 0.1, 1.0, 10.0]	1.0
	λ_O	[0.0]	0.0

Table C.4: Curriculum multiple shooting (CMS) training strategy hyperparameter tuning. Shown are the linear search spaces and selected values used for the CMS strategy. Tuned hyperparameters are the learning rate η, number of initial multiple shooting windows stages n_w, continuity penalty λ_{cont} and the output regularization strength λ_O (Eq. 3).

Model	Parameter	Search space	Selected value
LV-UDE	η	[0.001, 0.01, 0.05, 0.1]	0.01
	n_w	[3, 4, 5, 9, 13, 16, 21]	16
	λ_{cont}	[0.0, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]	1.0
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 3000.0, 5000.0, 10 000.0, 30 000, 50 000.0]	5000.0
LV-NODE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.01
	n_w	[3, 4, 5, 9, 13, 16, 21]	16
	λ_{cont}	[0.0, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]	1.0
	λ_O	[0.0, 0.1, 1.0, 10.0, 100.0, 500.0, 1000.0, 5000.0, 10 000.0, 50 000.0]	0.0
SIR UDE	η	[0.001, 0.005, 0.01, 0.05, 0.1]	0.005
	n_w	[4, 5, 6, 9, 16]	16
	λ_{cont}	[0.0, 0.1, 1.0, 10.0, 100.0, 1000.0]	0.0
	λ_O	[0.0, 0.0001, 0.001, 0.01, 0.1, 1.0]	0.0
Double pendulum	η	[0.001, 0.005, 0.01, 0.02 0.05, 0.1]	0.05
	n_w	[6, 8, 13, 27, 32, 40, 54]	8
	λ_{cont}	[0.0, 0.1, 100.0, 500.0]	0.1
	λ_O	[0.0, 1.0, 100.0, 500.0]	0.0
FHN-UDE	η	[3×10^{-5}, 10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, 10^{-2}]	3×10^{-4}
	n_w	[3, 5, 7, 12]	5
	λ_{cont}	[0.0, 0.01, 0.1, 1.0, 10.0]	0.0
	λ_O	[0.0, 1.0, 10.0, 100.0, 1000.0, 10 000.0, 100 000.0]	10 000.0
FHN-NODE	η	[10^{-4}, 3×10^{-4}, 10^{-3}, 3×10^{-3}, 10^{-2}]	10^{-3}
	n_w	[3, 5, 7, 12]	5
	λ_{cont}	[0.0, 0.01, 0.1, 1.0, 10.0]	0.01
	λ_O	[0.0]	0.0

C.6.2 Mechanistic ODE benchmarks

As described in Sec. 5, the mechanistic ODE models were primarily included to assess the robustness of the training strategies and their performance on mechanistic models. For these benchmarks, we therefore performed a small grid search over hyperparameter values and report results for the best-performing configuration, measured by the number of runs converging to the global optimum across 1000 multistarts (Fig. 4). For curriculum learning (CL), we tested 2, 3, 4, and 5 curriculum stages (n_c). For multiple shooting (MS) and curriculum multiple shooting (CMS), we tested 2, 3, 4, and 5 initial windows (n_w), together with continuity-penalty values of 0.0, 1.0, 10.0, and 100.0 for λ_{cont}. The only exception was the hard-to-train Elowitz model, for which we tested 2–9 windows and curriculum stages, and the Fujita and Zheng models, for which we tested only 2, 3, and 4 windows and curriculum stages because these models contain few measurements at unique time points. The best-performing values, used for Fig. 4, are reported in Tab. C.5.

Table C.5: Hyperparameters selected for each training strategy for the mechanistic ODE models. For CL, n_c denotes the number of curriculum stages. For MS and CMS, n_w denotes the number of initial multiple-shooting windows and λ_{cont} the continuity penalty weight.

	CL	MS		CMS	
Model	n_c	n_w	λ_{cont}	n_w	λ_{cont}
Boehm	5	5	0.1	5	0.1
Bruno	5	4	100	4	100
Elowitz	6	3	1	6	0
Fiedler	5	4	100	5	10
Fujita	5	2	0.1	2	0
Okuonghae	2	2	0.1	2	0.1
Schwen	2	5	100	5	0.1
Sneyd	5	4	100	5	100

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The main claims and the corresponding sections where those claims are presented are as follows:

   - We introduce curriculum multiple shooting, a general-purpose training strategy for fitting ordinary differential equation models (Sec. 4)
   - Curriculum multiple shooting outperforms state-of-the-art training strategies and generalises well (Sec. 5)
   - We contextualise these results and gain intuition for why CMS often improves training using Hessian-trace, training/interpolation MSE traces, loss-landscape and gradient ℓ_2-norm magnitude traces analyses (Sec. 5.6)

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations are explicitly discussed in Sec. 6.

   Guidelines:

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

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper does not include theoretical results.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: The primary contribution is a novel training strategy applied to ODE-based models. The approach is presented in Sec. 4 which includes an algorithm. Related strategies are presented in Sec. 3 for background. App. B and App. C provide in depth details to facilitate reproducibility.

   Guidelines:

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

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: The code for all benchmarks is included in `https://github.com/frohlich-lab/CurriculumMultipleShooting_NeurIPS_2026`. The models and datasets used for our benchmarks are introduced in Sec. 5.1 and described in more detail in App. B. One dataset is simulated, the other two are real datasets for which references are provided in App. B.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: The models and datasets used in the experiments are presented in 5.1. Training and test details including data splits, evaluation metrics, tuning approach and optimizer choice, are presented in Sec. 5.2. Further details of the implementation and hyperparameter tuning approach can be found in App. C.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Figures of the results of the experiments include indicators of the p-value and p-values are quoted in the text for specific results (Sec. 5).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Sec. 5 notes that all experiments were run on CPUs with a single model fit taking up to 15 minutes. App. C.2 provides more detail on average runtimes and memory limits for each of the benchmark models.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The work does not involve human subjects. The benchmarks use simulated or open-source real world datasets. The work does not raise any concerns of societal impact and impact mitigation measures have been taken where applicable (e.g. dataset and model details, software licensing).

Guidelines:

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

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: The work presents a novel training strategy for ODE-based models, which has no societal impact.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: The paper does not present work that has a high risk of misuse.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: The papers which originally produced the datasets used for benchmarks are cited (Sec. 5.1). Existing software packages used in the work are cited (Sec. 5).

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: The code for all benchmarks presented in the paper is available and documented at `https://github.com/frohlich-lab/CurriculumMultipleShooting_NeurIPS_2026`.

    Guidelines:

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

14. **Crowdsourcing and research with human subjects**

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

    Answer: [N/A]

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

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

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

    Answer: [N/A]

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

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: The core method development in this paper does not involve LLMs.

    Guidelines:

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