Control of dynamical systems with neural networks

Lucas Böttcher

July 20, 2025

Abstract Control problems frequently arise in scientific and industrial applications, where the objective is to steer a dynamical system from an initial state to a desired target state. Recent advances in deep learning and automatic differentiation have made applying these methods to control problems increasingly practical. In this paper, we examine the use of neural networks and modern machine-learning libraries to parameterize control inputs across discrete-time and continuous-time systems, as well as deterministic and stochastic dynamics. We highlight applications in multiple domains, including biology, engineering, physics, and medicine. For continuous-time dynamical systems, neural ordinary differential equations (neural ODEs) offer a useful approach to parameterizing control inputs. For discrete-time systems, we show how custom control-input parameterizations can be implemented and optimized using automatic-differentiation methods. Overall, the methods presented provide practical solutions for control tasks that are computationally demanding or analytically intractable, making them valuable for complex real-world applications.

Keywords dynamical systems · control · neural networks · neural ODEs · reinforcement learning · model predictive control · conformal prediction

Lucas Böttcher
Department of Computational Science and Philosophy, Frankfurt School of Finance and Management, Adickesallee 32–34, 60322, Frankfurt am Main, Germany and Laboratory for Systems Medicine, University of Florida, Gainesville, 32610-0225, Florida, United States of America
E-mail: l.boettcher@fs.de

1 Introduction

Control problems frequently arise in scientific and industrial applications, where the objective is to steer a dynamical system from an initial state to a desired target state. The theoretical foundations for addressing such problems are provided by control theory, which is historically connected to neuroscience and machine intelligence through the framework of cybernetics [1,2] and approaches like connectionism [3–7]. More recently, advances in automatic differentiation [8,9] and machine learning [10–26] have further strengthened these connections.

Modern machine-learning approaches are increasingly complementing traditional control methods and have shown potential in controlling complex dynamical systems, such as biomedical systems [25,27–29], inventory systems [19,30], and fusion reactors [31].[1] The challenge of applying standard control methods to complex dynamical systems was already recognized in 1971 by Alexey Ivakhnenko, a pioneer of deep learning [32], in his Polynomial Theory of Complex Systems [33]:

> Modern control theory, based on differential equations, is not an adequate tool for solving the problems of complex control systems. Constructing differential equations to trace input-output paths requires a deductive, deterministic approach. However, this approach is impractical for complex systems due to the difficulty of identifying these paths.

[1] Here, we use the term "complex" to refer to characteristics of a dynamical system that make its optimization and control challenging. These include factors such as high dimensionality of the underlying system, a large action space that defies efficient enumeration, and control inputs that are difficult to parameterize.

> Alexey Ivakhnenko in Polynomial Theory of Complex Systems (1971)

Control methods involving neural networks have been applied to both discrete-time and continuous-time dynamical systems [34]. To maintain tractability in gradient-based parameter updates, early applications of neural network controllers (NNCs) often focused on shallow architectures and linear dynamics. When dealing with high-dimensional, nonlinear systems where direct gradient updates are intractable or computationally demanding, "identifier" neural networks have been used to approximate and replace the underlying system dynamics [34]. With the popularization of neural ordinary differential equations (neural ODEs) within the `PyTorch` framework [35], building on earlier contributions such as Runge–Kutta neural networks [36], the direct application of deep neural network architectures to high-dimensional, nonlinear dynamical systems has become more accessible.

Other common applications of neural networks in control include neural Hamilton–Jacobi–Bellman (HJB) methods and deep reinforcement learning [37–41]. Neural HJB methods have been applied to state-feedback control problems, relying on the existence (or approximability) of a smooth value function for the considered system [37]. Deep reinforcement learning is often used in model-free settings, where the system dynamics are unknown or non-differentiable [42–44]. In this work, we focus on model-based methods, which directly incorporate a neural network into the known system dynamics. These approaches, sometimes referred to as actor-only reinforcement learning, can be more sample-efficient and converge more rapidly to near-optimal solutions than their model-free, actor-critic counterparts.

We examine how the expressive power of neural networks (*i.e.*, their ability to approximate a broad class of functions) can be leveraged to parameterize control inputs across a wide range of control and optimization problems. These problems span both discrete-time and continuous-time systems, as well as deterministic and stochastic dynamics. We will highlight applications across various domains, including biology, engineering, physics, and medicine. For continuous-time dynamical systems, neural ODEs provide a valuable approach to parameterizing control inputs. For discrete-time systems, we will describe how automatic differentiation can be used to implement and optimize custom control-input parameterizations.

This paper proceeds as follows. In Sections 2 and 3, we review selected neural-control methods for discrete- and continuous-time dynamics, complemented by new examples that illustrate key ideas and potential extensions. In Section 4, we compare a neural control approach with model predictive control (MPC) and discuss prior work on neural-MPC frameworks. In Section 5, we integrate conformal prediction into neural control systems to quantify uncertainty. We conclude by summarizing key results and listing relevant code repositories in Section 6.

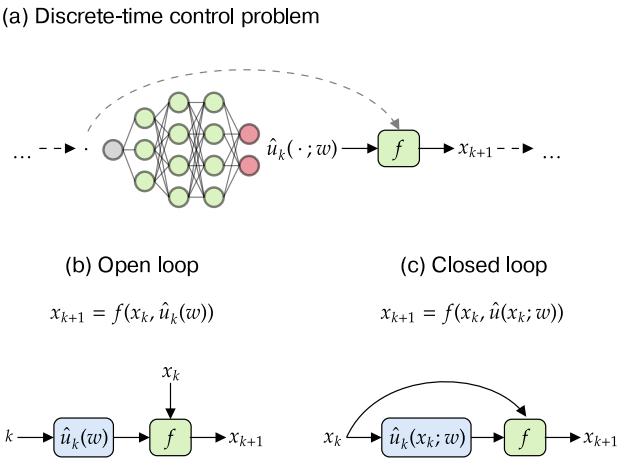

Fig. 1 Schematic of a discrete-time control problem, where a control input $\hat{u}_k(\cdot; w)$ is parameterized by a neural network with parameters $w \in \mathbb{R}^N$. We refer to these control inputs as neural network controllers (NNCs). (a) The overall system structure, in which the NNC generates control inputs that influence the state transition function $f(\cdot)$. When the NNC depends on the system state x_k, a skip connection (dashed grey line) propagates state information directly from the controller input to $f(\cdot)$, creating a ResNet-like unfolding of the underlying dynamics. (b) An open-loop control scenario, where the control input $\hat{u}_k(w)$ depends on the time step k but not on the system state x_k. (c) A closed-loop control scenario, where the control input $\hat{u}_k(x_k; w)$ is computed based on the current state x_k.

2 Discrete-time dynamics

We first consider discrete-time dynamical systems of the form

$$x_{k+1} = f(x_k, u_k)\,, \quad k \in \{0, \dots, T-1\}\,, \tag{1}$$

where $x_k \in \mathbb{R}^n$ and $u_k \in \mathbb{R}^m$ represent the system state and control input at time step k, respectively, and $f\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the state transition function. We examine both open-loop and closed-loop control inputs, $\hat{u}_k(w)$ and $\hat{u}_k(x_k; w)$, parameterized by a neural network with parameters $w \in \mathbb{R}^N$. We refer to these control inputs as neural network controllers (NNCs), whose parameters are trained using standard optimizers such as Adam and RMSProp, as implemented in `PyTorch`.

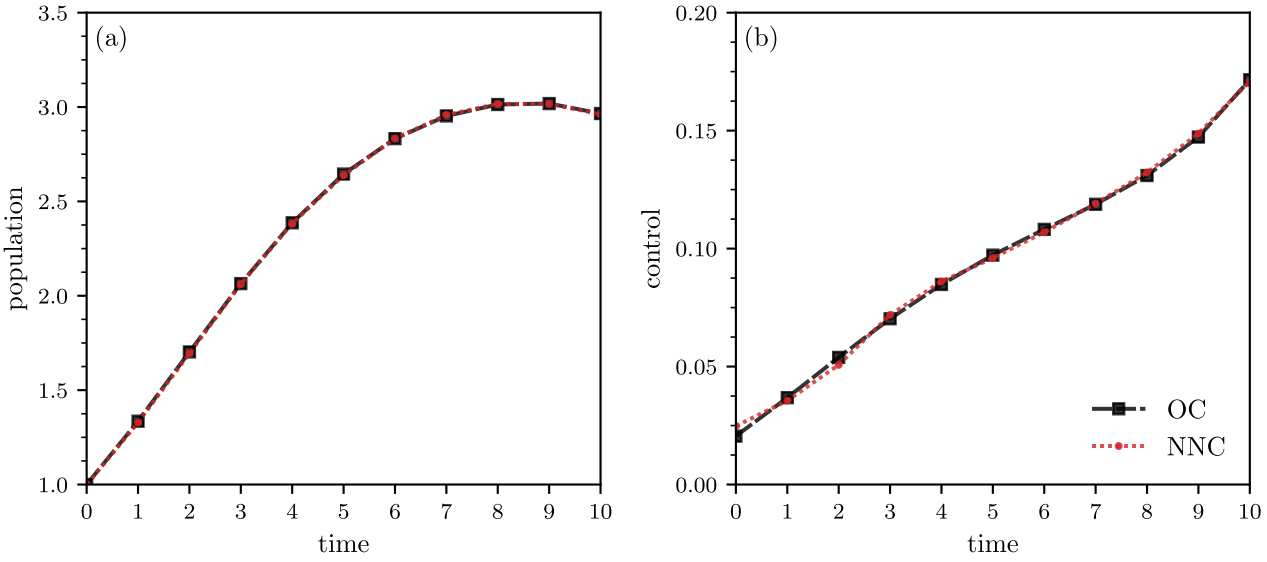

Fig. 2 Comparison of control approaches for the Beverton–Holt model (3) with harvesting over a finite time horizon $T = 10$, using parameters $\gamma = 1$, $a = 0.1$, $c = 10$, and $r = 1.5$. The control objective is to maximize the total discounted net benefit over T, as defined in Eq. (4). (a) Evolution of the population x_k under the optimal control (OC) and the NNC approaches. (b) Corresponding control input over time. The population initially increases due to low control, then stabilizes as the control intensifies. The NNC closely resembles the OC solution.

In Fig. 1, we show a schematic of a discrete-time control problem with an NNC $\hat{u}_k(\cdot; w)$. When the control input depends on the system state x_k, a skip connection [dashed grey line in Fig. 1(a)] propagates state information directly from the controller input to the state transition function $f(\cdot)$, creating a ResNet-like unfolding of the underlying dynamics.[2] In Fig. 1(b), we illustrate an open-loop control scenario, where the control input $\hat{u}_k(w)$ depends only on the time step k. In contrast, Fig. 1(c) depicts a closed-loop control scenario, where the control input $\hat{u}_k(x_k; w)$ depends on the current state x_k, allowing the NNC to adapt its output based on the system's behavior.

To train NNCs, we define and optimize a suitable loss function that reflects the control objective. A common example is the finite-horizon cost functional

$$J[\{x_k\}, \{u_k\}] = \sum_{k=0}^{T-1} L(x_k, u_k, k) + V(x_T)\,, \tag{2}$$

where $L(x_k, u_k, k)$ denotes the running cost at time step k, and $V(x_T)$ is the terminal cost associated with the final state. Replacing u_k with the parameterized control input $\hat{u}_k(\cdot; w)$, we optimize the cost functional $J[\{x_k\}, \{\hat{u}_k\}]$ using automatic differentiation to update the NNC parameters w.

[2] In a residual neural network (ResNet), each layer adds a residual $f(x)$ to its input x, so that the input to the next layer becomes $f(x) + x$. This is implemented via a skip connection that carries the input forward and adds it to the layer's output.

We now present several examples to illustrate the applicability of NNCs to discrete-time dynamical systems. The first is a simple, illustrative example based on a deterministic discrete-time system describing the evolution of a single-species population under harvesting [45] using an open-loop controller. The remaining two examples, which focus on predator-prey interactions [25] and inventory dynamics [19], involve stochasticity and closed-loop controllers.

2.1 Illustrative example

As a basic illustrative example, we consider a discrete-time optimal control problem describing the evolution of a single-species population subject to harvesting. The population dynamics follow a version of the Beverton–Holt model

$$x_{k+1} = \frac{r x_k}{1 + a x_k}(1 - u_k)\,, \tag{3}$$

where $x_k \geq 0$ denotes the population size at time $k \in \{0, \ldots, T-1\}$, $r > 0$ is the intrinsic growth rate, $a > 0$ captures density-dependent effects (*e.g.*, crowding or competition), and $u_k \in [0, 1]$ represents the fraction of the population harvested at time k. The Beverton–Holt model was originally introduced by Beverton and Holt (1957) to describe the growth dynamics of fish populations [45]. Related ideas can be traced back to the early work of Baranov (1918) [46], whose contributions continue to influence modeling approaches in fisheries science [47].

The control objective is to maximize the total discounted net benefit over a finite time horizon T [48], given by

$$J_1[\{x_k\},\{u_k\}] = \sum_{k=0}^{T-1}\left[\gamma^k x_k \frac{r x_k}{1+a x_k} - c u_k^2\right], \tag{4}$$

where $\gamma \geq 0$ is the discount factor and $c \geq 0$ penalizes large harvest rates, *e.g.*, those arising from increasing marginal harvesting costs or ecological impacts.

We aim to determine a control sequence $\{u_k\}_{k=0}^{T-1}$ such that $u_k \in [0,1]$ for all k, in order to maximize $J_1[\{x_k\},\{u_k\}]$ subject to the state dynamics (3).

We use an NNC $\hat{u}_k(w)$ to parameterize the control input with parameters $w \in \mathbb{R}^N$, which we determine by minimizing $-J_1[\{x_k\},\{\hat{u}_k\}]$. The NNC that we employ has five hidden layers with four rectified linear units (ReLUs) each.

As a baseline for the NNC approach, we also apply Pontryagin's maximum principle to the Hamiltonian

$$H = \sum_{k=0}^{T-1}\left[\gamma^k u_k \frac{r x_k}{1+a x_k} - c u_k^2 + \lambda_k\left(x_{k+1} - \frac{r x_k}{1+a x_k}(1-u_k)\right)\right], \tag{5}$$

with the adjoint variables λ_k, as a necessary condition for obtaining the optimal control (OC) solution.

The corresponding control input satisfies

$$u_k = \frac{\gamma^k + \lambda_k}{2c}\frac{r x_k}{1+a x_k} \tag{6}$$

and the adjoint variables evolve according to

$$\lambda_{k-1} = \lambda_k \frac{r}{(1+a x_k)^2}(1-u_k) - \gamma^k u_k \frac{r}{(1+a x_k)^2}. \tag{7}$$

In Fig. 2, we show the evolution of the population as well as the control trajectories obtained from the OC and NNC policies. The simulation is conducted over a finite time horizon of $T = 10$, with parameter values $\gamma = 1$, $a = 0.1$, $c = 10$, and $r = 1.5$. The population initially grows due to weak control and later stabilizes as the control signal increases. The NNC solution closely approximates the OC trajectory, while avoiding the iteration of the coupled control-adjoint system [see Eqs. (6) and (7)].

2.2 Predatory-prey dynamics

As a more involved example, we consider a predator-prey agent-based model (ABM) with three species A, B, and C [25, 49–52]. We denote the population sizes of species A, B, and C at time step k by a_k, b_k, and c_k, respectively. In an ecological context, this model can represent interactions such as those between grass, sheep, and wolves, or between plankton, forage fish, and predatory fish. Similar models also appear in systems biology. For example, in models of Aspergillosis, a common lung infection caused by the fungus *Aspergillus*, the three species may correspond to iron (as a nutrient source), *Aspergillus* (as prey), and macrophages (as predators) [53, 54]. Generalized predator-prey models with even more species have found applications in studies of microbial communities [55], whose continuous-time description will be the subject of Section 3.2.

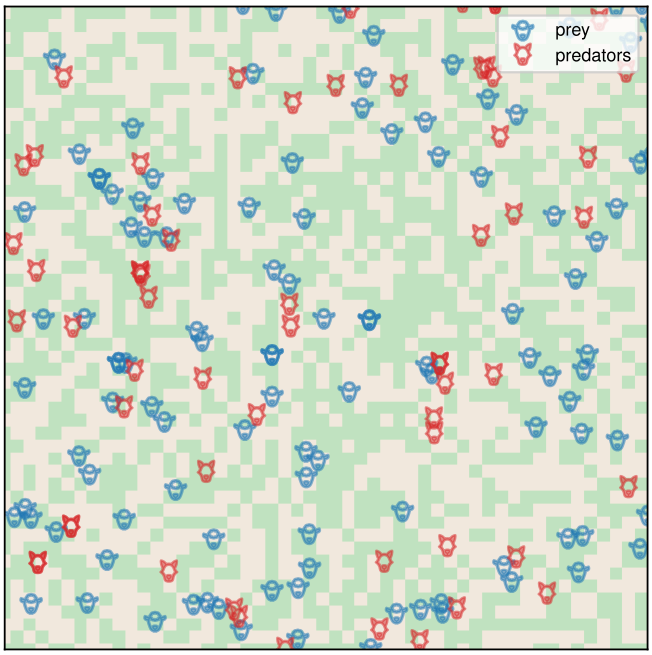

Fig. 3 Predator-prey ABM. Snapshot of a three-species predator-prey ABM simulation on a 51×51 grid. Grid cells colored green and light brown indicate nutrient-rich and nutrient-poor regions, respectively.

We simulate the three-species predator-prey dynamics on an $L \times L$ periodic grid using an ABM defined by the following rules. Each grid cell can be in one of two states: (i) nutrient-rich or (ii) nutrient-poor. Prey move randomly with a directional bias toward the positive x-direction and rely on nutrient consumption to survive. The energy gain per unit of nutrient is κ_1. When a prey encounters a nearby nutrient-rich cell, it consumes the nutrients, causing the cell to switch to a nutrient-poor state. Nutrients in that cell regenerate after τ time steps.

Predators also perform a random walk with the same directional bias as the prey and consume prey when they occupy the same grid cell. The energy gained per consumed prey is κ_2. In each time step, both predators and prey lose one unit of energy to sustain their

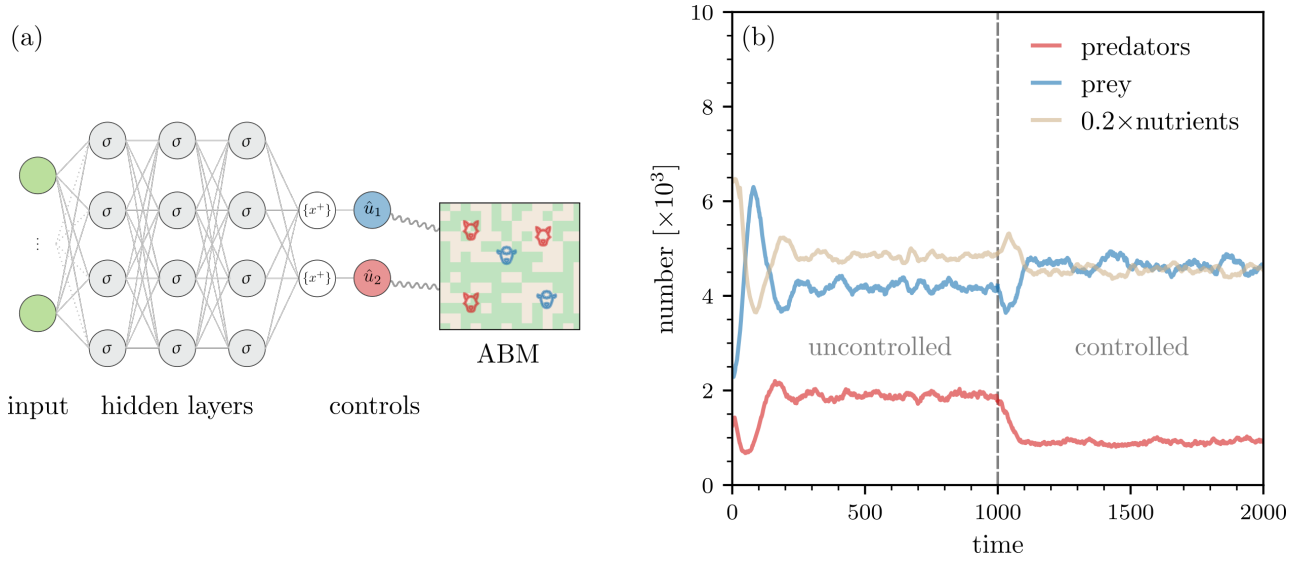

Fig. 4 Control of a predator-prey ABM using an NNC. (a) To control the predator-prey ABM, we first define appropriate inputs and outputs for the NNC. Potential inputs include the population sizes a_k, b_k, and c_k of species A, B, and C at time step k. Since we aim to directly control predator and prey populations, the ANN produces two outputs, $\hat{u}_1$ and $\hat{u}_2$. A problem-specific straight-through estimator is used to obtain integer-valued control actions by removing the fractional part $\{\max\{0,\cdot\}\}$ from the positive hidden-layer outputs. We denote the hidden-layer activations by σ, and the straight-through estimator by $\{x^+\}$. (b) Evolution of predators, prey, and nutrient-right lattice sites in a single realization of the predator-prey ABM. The vertical dashed gray line marks the time when the NNC is activated. The controller is designed to increase the mean number of prey by 10% and reduce the mean number of predators by 50%. Dashed blue and red lines indicate the target prey and predator levels, respectively (*i.e.*, $\bar{b}^* = 4575$ and $\bar{c}^* = 948$). The simulation uses a 255×255 grid with initial conditions $b_0 = 2500$, $c_0 = 1250$, and parameters $\alpha_1 = 4.0$, $\alpha_2 = 5.0$, $\kappa_1 = 4.0$, $\kappa_2 = 20.0$, and $\tau = 30$ [51]. Initially, 50% of the lattice sites are nutrient-rich.

metabolism. Individuals die if their energy is smaller than 0. Predators and prey reproduce at rates α_1 and α_2, respectively.

In Fig. 3, we show a snapshot of a three-species predator-prey ABM simulation on a 51×51 grid. Green and light brown cells indicate nutrient-rich and nutrient-poor regions, respectively. For additional details on this model, see [51].

To control the predator-prey ABM, we define suitable inputs and outputs for an NNC. Potential inputs are the population sizes a_k, b_k, and c_k of species A, B, and C at time step k. As an example, we aim to shift the system toward a new steady state by directly adjusting the numbers of predators and prey. The NNC has two integer-valued outputs, $\hat{u}_1$ and $\hat{u}_2$ [see Fig. 4(a)]. If $\hat{u}_1 > 0$ ($\hat{u}_2 > 0$), a new prey (predator) is added at a randomly selected grid cell. If $\hat{u}_1 < 0$ ($\hat{u}_2 < 0$), a prey (predator) is removed from a randomly selected location. The underlying ABM used in our simulations consists of a 255×255 grid.

Before applying control, we allow the ABM to evolve freely for 1000 time steps to estimate steady-state population levels. For our chosen parameters, the mean numbers of predators and prey over the final 100 steps are approximately 1896 and 4159, respectively. We then run the controlled dynamics for an additional 1000 steps, giving a total time horizon $T = 2000$.

Our control objective is to increase the mean number of prey by 10% and simultaneously reduce the mean number of predators by 50% over the final T' time steps ($k \in \{T - T' + 1, \ldots, T\}$) [see Fig. 4(b)]. Since such a substantial drop in predator numbers naturally leads to a rise in prey, the controller must suppress both populations relative to the original steady state. This goal can be achieved using a two-node NNC, with each output regulating the population of predators and prey, respectively. To train the NNC, we use the quadratic loss function

$$J_2(w) = \left(\bar{b}(w) - \bar{b}^*\right)^2 + \left(\bar{c}(w) - \bar{c}^*\right)^2, \tag{8}$$

where $w \in \mathbb{R}^N$ are the parameters of the NNC, and $\bar{b}^*$ and $\bar{c}^*$ are the target values, corresponding to the desired mean numbers of prey and predators over the final T' time steps. In this steady-state control example, we use $N = 2$ NNC parameters, $w = (w_1, w_2)^\top$, with target values $\bar{b}^* = 4575$, $\bar{c}^* = 948$, and $T' = 100$. The quantities

$$\bar{b}(w) = \frac{1}{T'} \sum_{k=1}^{T'} b_{(T-T'+k)}(w) \tag{9}$$

and

$$\bar{c}(w) = \frac{1}{T'}\sum_{k=1}^{T'} c_{(T-T'+k)}(w) \tag{10}$$

are the corresponding reached states.

We parameterize the integer-valued control input $\hat{u}(b_k, c_k; w)$ according to

$$\hat{u}(b_k, c_k; w) = \begin{pmatrix} -(\max\{0, b_k w_1\} - \{\max\{0, b_k w_1\}\}) \\ -(\max\{0, c_k w_2\} - \{\max\{0, c_k w_2\}\}) \end{pmatrix}. \tag{11}$$

Here, $\{x\}$ denotes the fractional part of x, defined as $\{x\} = x - \lfloor x \rfloor$ for $x > 0$, where $\lfloor \cdot \rfloor$ is the floor function. Training the NNC with the control input defined in Eq. (11) is based on a problem-specific straight-through estimator [19, 56–58], which enables backpropagation with integer-valued outputs.

We use the two control inputs

$$\hat{u}_1(b_k; w_1) = -(\max\{0, b_k w_1\} - \{\max\{0, b_k w_1\}\}) \tag{12}$$

and

$$\hat{u}_2(c_k; w_2) = -(\max\{0, c_k w_2\} - \{\max\{0, c_k w_2\}\}) \tag{13}$$

to adjust the population sizes of prey and predators, respectively. These control inputs are set up such that they output negative integer-valued controls, meaning that a certain number of prey and predators will be removed from the ABM at each time step. For further details on the training of this NNC and an application of a multilayer NNC to transient control, see [25].

The lowest training loss, $J_1(w) \approx 74.09$, is achieved for parameters $w_1 = 0.0083$ and $w_2 = 0.0047$. The corresponding mean populations are approximately $\bar{b}(w) \approx 4573$ prey and $\bar{c}(w) \approx 956$ predators.

The learned NNC parameters $(0.0083, 0.0047)$ are close to the optimal ones $(0.0083, 0.0045)$, which have been determined by performing a grid search over the underlying parameter space [59]. To examine the uncertainty in the target quantities (*i.e.*, the numbers of prey and predators), we tested the steady-state NNC on 50 previously unseen ABM instances. The resulting mean population sizes were 4587 (± 71) for prey and 950 (± 40) for predators, both closely aligned with the target values of $\bar{b}^* = 4575$ and $\bar{c}^* = 948$. (Values in parentheses indicate the unbiased sample standard deviation.) These results show that the controller performs reliably on new data.

Control solutions obtained using the described NNC approach have been compared with those derived from several surrogate models. The results show that surrogate-based control solutions deviate more from the optimum than those produced by the NNC [59].

2.3 Inventory dynamics

The following examples focus on neural network–controlled inventory management problems [60], which arise across various industries, including manufacturing, retail, warehousing, and energy. The primary objective in these problems is to determine the optimal ordering policy, which may involve one or more suppliers, that minimizes total costs. These costs usually include ordering expenses, holding costs for excess inventory, and penalties associated with stockouts under stochastic demand.

The sourcing problems we consider are generally formulated as infinite-horizon models aimed at minimizing the expected cost per period under stationary stochastic demand. When training NNCs, we optimize costs across multiple demand trajectories [19]. This enables us to handle not only non-stationary demand but also both finite-horizon and infinite-horizon discounted settings. Unlike traditional model-free reinforcement learning methods [61], the approach pursued here leverages knowledge of the system dynamics, enabling more efficient training and more accurate solutions.

A fundamental yet analytically intractable problem in inventory management is dual sourcing [62–64], where decisions must be made about ordering from either a low-cost, regular supplier or a higher-cost, expedited supplier. In contrast, single sourcing [65, 66] is analytically tractable and often serves as a baseline for evaluating policies in more complex multi-sourcing scenarios.

In the following two sections, we apply NNCs to learn ordering policies for single- and dual-sourcing problems. Our implementation is provided in the Python library `idinn`, which supports both traditional and neural policies for these sourcing models.[3]

2.3.1 Single-sourcing problems

In single-sourcing dynamics, the net inventory evolves according to

$$I_{k+1} = I_k + q_{k-l} - D_k\,, \tag{14}$$

where q_{k-l} is the replenishment order placed l periods earlier (*i.e.*, the order arriving in period $k \in \{0, 1, 2, \dots\}$, l is the lead time, and D_k denotes the stochastic demand in period k.[4]

The cost incurred in period k is

$$c_k = h \max\{0, I_{k+1}\} + b \max\{0, -I_{k+1}\}\,, \tag{15}$$

[3] Documentation and examples are available at https://inventory-optimization.readthedocs.io/en/latest/.

[4] We use the term "period" rather than "time step" in the context of inventory dynamics, as it aligns more closely with the standard terminology in that field.

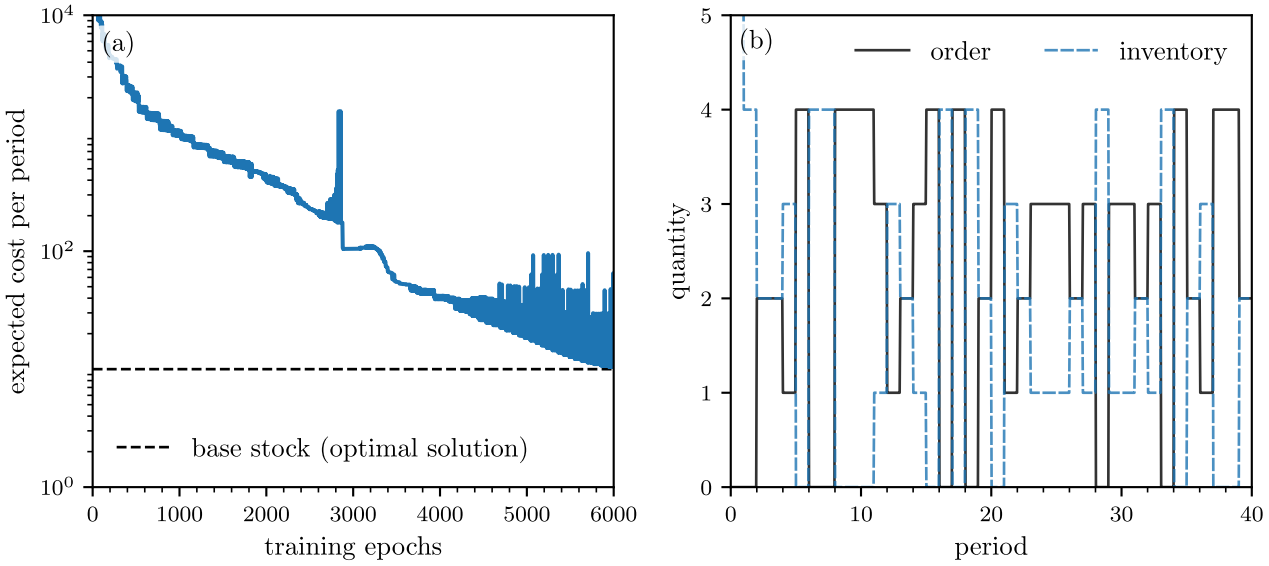

Fig. 5 Controlling single-sourcing dynamics using an NNC. (a) Training performance of the learned ordering policy, showing the expected cost per period over training epochs for single-sourcing dynamics with lead time $l = 0$, unit holding cost $h = 5$, unit shortage cost $b = 495$, and demand distribution $\mathcal{U}\{0, 4\}$. The dashed black line indicates the optimal cost of 10 achieved by the base-stock policy. The simulation time was approximately 2 minutes on a standard laptop. (b) Example trajectory of the learned policy, illustrating order quantities (solid black line) and inventory levels (dashed blue line) over time.

where h and b denote the unit holding and shortage costs, respectively. The term $\max\{0, I_{k+1}\}$ represents excess inventory (on-hand stock), while $\max\{0, -I_{k+1}\}$ captures inventory shortages (backorders). The objective is to minimize the total expected cost accumulated over time.

To mathematically describe the optimal ordering policy in single-sourcing problems [65, 66], we let z denote the target inventory position (*i.e.*, the desired net inventory level plus all outstanding orders). The inventory position at time t under single-sourcing dynamics, $\tilde{I}_t$, is

$$\tilde{I}_k = \begin{cases} I_k\,, & \text{if } l = 0 \\ I_k + \sum_{i=1}^{l} q_{t-i}\,, & \text{if } l > 0\,. \end{cases} \tag{16}$$

The optimal target inventory level [65] is given by the critical fractile

$$z^* = \Phi^{-1}\left(\frac{b}{b+h}\right)\,, \tag{17}$$

where $\Phi(x) = \Pr(D \leq x)$ is the cumulative distribution function of demand D over $l + 1$ periods. If the inventory position in period k falls below z^*, a replenishment order $q_k = z^* - \tilde{I}_k$ is placed to bring the inventory position back to the optimal target level. The optimal single-sourcing policy, often referred to as the "base-stock policy", is

$$q_k = \max\{0, z^* - \tilde{I}_k\}\,, \tag{18}$$

that is, the positive part of $z^* - \tilde{I}_k$. This quantity depends on the optimal inventory position z^*, the current net inventory, and the sum of past orders q_{k-i} for $i \in \{1, \ldots, l\}$, where $l > 0$.

Notice that the mathematical structure of the base-stock policy resembles that of a rectified linear unit (ReLU). The ReLU function returns the positive part of a real number. That is, $\mathrm{ReLU}(x) = \max\{0, x\}$.

To build an NNC that learns replenishment orders $\hat{q}_k$, we use $l + 1$ inputs representing the current net inventory and previous orders (*i.e.*, the system state). We also include a bias term in the input layer to capture the unknown optimal target inventory level z^*. These inputs are passed through a ReLU-type activation function that generalizes the expression in Eq. (18).

However, during training with backpropagation, a ReLU unit can become inactive and consistently output values near zero, often due to a large negative bias term [67]. Once this happens, the corresponding gradients vanish, making it difficult for gradient descent to update the weights. As a result, the output remains stuck near zero, a phenomenon known as the "dead ReLU" problem. To avoid this issue, we instead use a continuously differentiable exponential linear unit

$$\mathrm{CELU}(x; \alpha) = \max\{0, x\} - \max\{0, \alpha\,(1 - \exp(x/\alpha))\}\,, \tag{19}$$

which approaches $\mathrm{ReLU}(x) = \max\{0, x\}$ in the limit $\alpha \to 0^+$ [68]. CELUs offer an advantage over ReLUs because their smooth, continuous derivatives make gra-

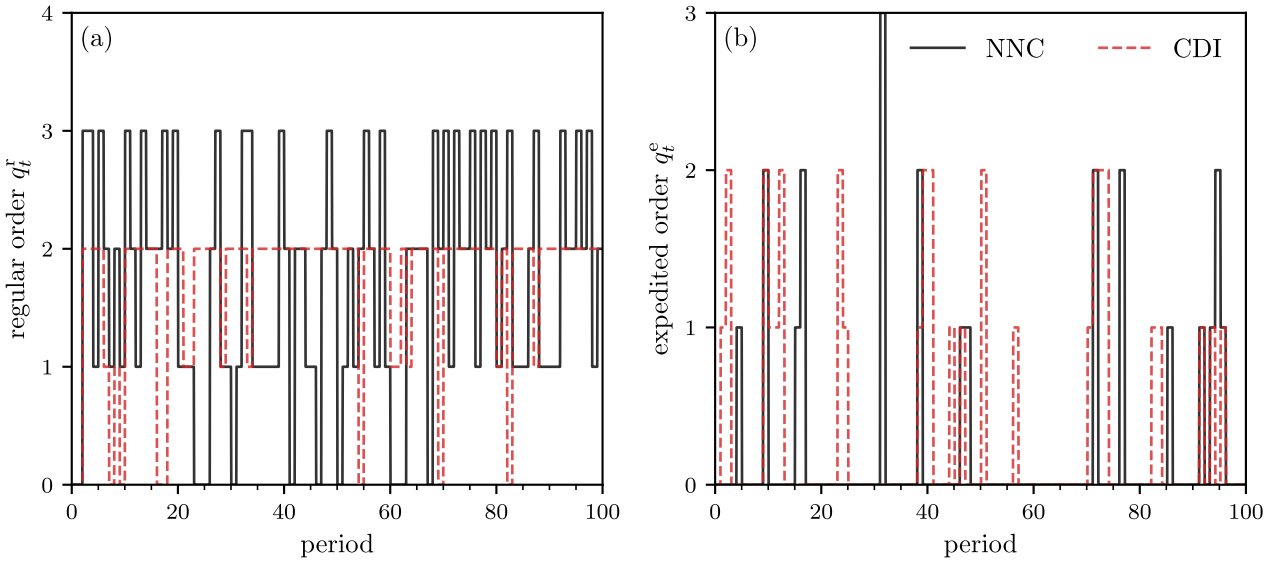

Fig. 6 Evolution of regular orders (a) and expedited orders (b) under NNC and CDI policies. The underlying dual-sourcing problem is defined by parameters $h = 5$, $b = 495$, $c_r = 0$, $c_e = 20$, $l_r = 2$, and $l_e = 0$. Demand follows a discrete uniform distribution $\mathcal{U}\{0, 4\}$.

dient calculations more stable during neural network training.

As in the predator-prey example discussed in Section 2.2, we employ a straight-through estimator to produce integer-valued order quantities while enabling backpropagation of real-valued gradients [see Eq. (11)]. The loss function we optimize is the expected cost per period

$$J_3[\{c_k^{(j)}\}] = \frac{1}{T}\sum_{k=0}^{T-1}\gamma^k \frac{1}{M}\sum_{j=1}^{M} c_k^{(j)}\,, \tag{20}$$

where γ is a discount factor (set to 1 in our case), M is the number of realizations of the sourcing dynamics, and $c_k^{(j)}$ is the cost in period k for the j-th realization [see Eq. (15)].

As an example, we consider a single-sourcing problem with lead time $l = 0$, unit holding cost $h = 5$, unit shortage cost $b = 495$, and discrete uniform demand distribution $\mathcal{U}\{0, 4\}$. For these parameters, the optimal order-up-to level is $z^* = 4$, and the corresponding optimal expected cost per period is $h(z^* - \bar{D}) = 10$, where $\bar{D} = 2$ is the mean demand per period.

For training the NNC, we set $T = 50$ and use $M = 128$ realizations. We observe that the NNC approaches the expected cost level of the optimal base-stock policy after approximately 6000 training epochs [see Fig. 5(a)]. The total training time is approximately 2 minutes on a standard laptop. As the NNC converges toward the optimal solution, small changes in the neural network weights can lead to large fluctuations in the expected cost, resulting in the onset of oscillatory behavior after around 4500 epochs.

By extracting the NNC parameters corresponding to the lowest observed cost, we can examine the evolution of inventory and order quantities in a representative trajectory [see Fig. 5(b)]. We find that the NNC successfully learns the optimal base-stock policy, placing orders that consistently reach the optimal order-up-to level $z^* = 4$. This is also reflected in the learned NNC parameters, which align with those of a base-stock policy.

2.3.2 Dual-sourcing problems

Building on the previous example of single-sourcing dynamics, we now turn to dual-sourcing problems, which are generally analytically intractable. In such problems, the first sourcing option is a "regular" supplier, R, which delivers goods with a (non-negative) integer lead time $l_r > 0$ at a cost c_r. A second option is an "emergency" supplier, E, which provides goods with a shorter lead time $l_e < l_r$ but at a higher cost $c_e > c_r$. The premium paid for expedited delivery via supplier E is thus defined as $c := c_e - c_r > 0$.

As in the single-sourcing setting, we denote by I_k the net inventory at the beginning of period k. The replenishment orders placed in period k to suppliers E and R are denoted by q_k^e and q_k^r, respectively. In each period k, stochastic demand D_t is realized.

Using these definitions, the sequence of events in each period of the dual-sourcing model is as follows:

1. At the beginning of period k, the inventory manager places replenishment orders q_k^r and q_k^e, based on the current net inventory I_k and the outstanding orders

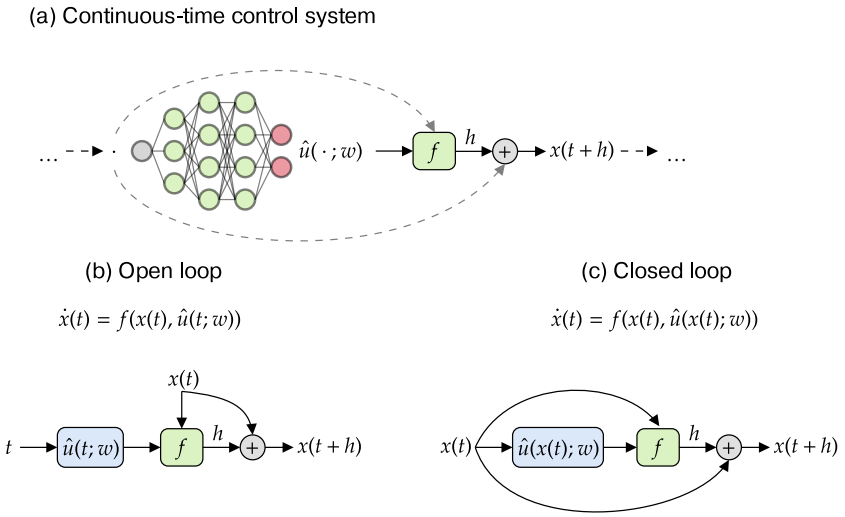

Fig. 7 Schematic of a continuous-time control problem, where a control input $\hat{u}(\cdot; w)$ is parameterized by a neural network with parameters $w \in \mathbb{R}^N$. We refer to these control inputs as neural ODE controllers (NODECs). (a) The overall system structure, in which a NODEC generates control inputs that influence the vector field $f(\cdot)$. (b) An open-loop control scenario, where the control input $\hat{u}(t; w)$ depends on the time t but not on the system state $x(t)$. (c) A closed-loop control scenario, where the control input $\hat{u}(x(t); w)$ is computed based on the current state $x(t)$. In both control scenarios, the future state $x(t+h)$ is computed by integrating the system dynamics over the time step h. For example, using an explicit Euler scheme, the function $f(\cdot)$ is used to determine the rate of change $\dot{x}(t)$, which is then multiplied by h and added to the current state $x(t)$ to obtain the next state (*i.e.*, $x(t+h) = x(t) + hf(x(t), \hat{u}(\cdot; w))$).

that have not yet arrived: $Q_k^{\mathrm{r}} = (q_{k-l_{\mathrm{r}}}^{\mathrm{r}}, \dots, q_{k-1}^{\mathrm{r}})$ and $Q_k^{\mathrm{e}} = (q_{k-l_{\mathrm{e}}}^{\mathrm{e}}, \dots, q_{k-1}^{\mathrm{e}})$.

2. Orders $q_{k-l_{\mathrm{r}}}^{\mathrm{r}}$ and $q_{k-l_{\mathrm{e}}}^{\mathrm{e}}$ arrive and are added to the inventory.
3. Demand D_k is realized and subtracted from the current inventory.

The dual-sourcing problem is a Markov decision process with state $(I_k, Q_k^{\mathrm{r}}, Q_k^{\mathrm{e}})$ and action $(q_k^{\mathrm{r}}, q_k^{\mathrm{e}})$, where the inventory level (including backlogged excess demand) evolves according to

$$I_{k+1} = I_k + q_{k-l_{\mathrm{r}}}^{\mathrm{r}} + q_{k-l_{\mathrm{e}}}^{\mathrm{e}} - D_k\,. \tag{21}$$

The corresponding total cost in period k is

$$\begin{aligned} c_k =& c_{\mathrm{r}} q_k^{\mathrm{r}} + c_{\mathrm{e}} q_k^{\mathrm{e}} + h \max\{0, I_k + q_{k-l_{\mathrm{r}}}^{\mathrm{r}} + q_{k-l_{\mathrm{e}}}^{\mathrm{e}} - D_k\} \\ &+ b \max\{0, D_k - I_k - q_{k-l_{\mathrm{r}}}^{\mathrm{r}} - q_{k-l_{\mathrm{e}}}^{\mathrm{e}}\}\,. \end{aligned} \tag{22}$$

A stationary optimal policy exists for minimizing the expected cost per period when the demand distribution has finite support over the considered time horizon [69]. However, a general analytical characterization of the optimal policy in dual-sourcing problems remains elusive.

As an example, we consider a dual-sourcing problem with parameters $h = 5$, $b = 495$, $c_{\mathrm{r}} = 0$, $c_{\mathrm{e}} = 20$, $l_{\mathrm{r}} = 2$, and $l_{\mathrm{e}} = 0$. The demand follows a discrete uniform distribution $\mathcal{U}\{0, 4\}$. For this instance, the best-performing available heuristic, the capped dual index (CDI) policy [70], achieves an expected cost per period of 23.26, while the optimal expected cost per period, determined via dynamic programming, is 23.07.

For the NNC architecture, we use seven hidden layers with 128, 64, 32, 16, 8, 4, and 2 CELU neurons, respectively [see Eq. (19) with $\alpha = 1$]. We train the NNC using the loss function (20), based on the dual-sourcing cost (22). We consider a time horizon of $T = 1000$ and use $M = 500$. The NNC policy achieves an expected cost per period of 23.13. Extensive testing of NNC policies over a large set of instances has demonstrated that it performs as well as or better than CDI [19].

In Fig. 6, we show the evolution of regular and expedited orders under the CDI and NNC policies. A key advantage of NNC over CDI is its ability to tailor regular order decisions to the current inventory level and past order placements. As a result, the need for expedited orders is reduced.

3 Continuous-time dynamics

We now turn our focus to continuous-time dynamical systems of the form

$$\dot{x} = f(x, u)\,, \tag{23}$$

where $x \equiv x(t) \in \mathbb{R}^n$ and $u \equiv u(t) \in \mathbb{R}^m$ represent the system state and control input at time t, respectively. The vector field is $f\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. We consider both

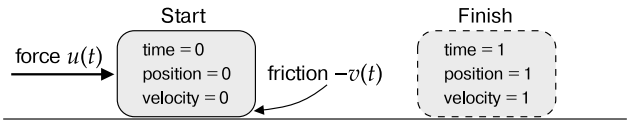

Fig. 8 Schematic of the block-move example.

open-loop and closed-loop control inputs, $\hat{u}(t;w)$ and $\hat{u}(x(t);w)$, parameterized by a neural network with parameters $w \in \mathbb{R}^N$. We refer to these controllers as neural ODE controllers (NODECs). As in the discrete-time case, we train these controllers using standard optimizers such as Adam and RMSProp, as implemented in `PyTorch`.

In Fig. 7, we show a schematic of a continuous-time control problem with a NODEC $\hat{u}(\cdot;w)$. In Fig. 7(b), we illustrate an open-loop control scenario, where the control input $\hat{u}(t;w)$ depends on time t but not on the system state $x(t)$. In contrast, Fig. 7(c) depicts a closed-loop control scenario, where the control input $\hat{u}(x(t);w)$ depends on the current state $x(t)$, allowing the NODEC to adapt its output based on the system's behavior.

To train NODECs, we define and optimize a suitable loss function that reflects the control objective. Analogous to the discrete-time finite-horizon cost functional in Eq. (2), a commonly used continuous-time counterpart is

$$J[x(t),u(t)] = \int_0^T L(x(t),u(t))\,\mathrm{d}t + V(x(t))\,, \tag{24}$$

where $L(x(t),u(t))$ denotes the running cost at time t, and $V(x(t))$ is the terminal cost associated with the final state. By replacing $u(\cdot)$ with the parameterized control input $\hat{u}(\cdot;w)$, we optimize the cost functional $J[x(t),\hat{u}(\cdot)]$ using automatic differentiation to update the NODEC parameters w.

We focus on several examples to demonstrate the applicability of NODEC to continuous-time dynamical systems. In the first example, we examine a basic control problem involving the movement of a block subject to friction, where the objective is to minimize work [14, 71]. Next, we apply NODEC to Lotka–Volterra-type predator-prey dynamics, which arise, for instance, in biomedical contexts such as modeling microbiome interactions [55]. Finally, we consider the control of systems composed of coupled oscillators [15].

3.1 Illustrative example

As an illustrative continuous-time control problem, we consider the task of moving a block in the presence of friction (see Fig. 8). The objective is to minimize the total work

$$W[u,v] = \int_0^T u(t)v(t)\,\mathrm{d}t\,, \tag{25}$$

subject to the dynamics and constraints

$$\begin{aligned} \dot{x}(t) &= v(t)\,,\\ \dot{v}(t) &= -v(t)+u(t)\,,\\ v(t) &\geq 0\,,\\ 0 \leq u(t) &\leq 2\,,\\ (x(0),v(0)) &= (0,1)\,,\\ (x(T),v(T)) &= (1,1)\,, \end{aligned} \tag{26}$$

where the time horizon is $T=1$ [71]. The control input $u(t)$ represents the force applied to the block, and in this case, the optimal control is $u^*(t)=1$. Although the solution is analytically straightforward, some numerical methods have difficulty accurately approximating the constant control input [71].

To solve this control problem using NODEC, we represent $u(t)$ with a neural network $\hat{u}(t;w)$ consisting of eight hidden layers, each containing six ELU neurons. The controller is trained for 100 epochs using the Adam optimizer with a learning rate of 5×10^{-3}, minimizing the loss

$$J_4(z(T)) = \|z(T)-z^*\|_2^2\,, \tag{27}$$

where $z(T) = (x(T),v(T))^\top$ denotes the final system state, and $z^* = (1,1)^\top$ is the desired target state. That is, we do not explicitly minimize the work $W[u,v]$.

In Fig. 9(a), we show the evolution of $x(t)$ and $u(t)$. The optimal control (OC) and the NODEC solutions are shown as grey and red lines, respectively. We observe that NODEC has learned a control input that closely resembles the optimal one.

We now study how the structure of the neural network and the choice of activation function affect the performance of NODEC. To this end, we fix the number of neurons per layer to six and vary the number of hidden layers from 2 to 64, initializing all weights and biases to 10^{-2}. Controllers are trained for 100 epochs using the Adam optimizer with a learning rate of 5×10^{-3}, and the best-performing model is selected based on the loss value. In Fig. 9(b), we compare results for both ELU and ReLU activations in terms of the final loss value. For networks with at least 8 hidden layers, the loss consistently drops below 10^{-7}. In contrast, single-layer networks, regardless of width, fail to approximate the optimal control solution. Overall, performance is nearly identical for both ELU and ReLU activation functions.

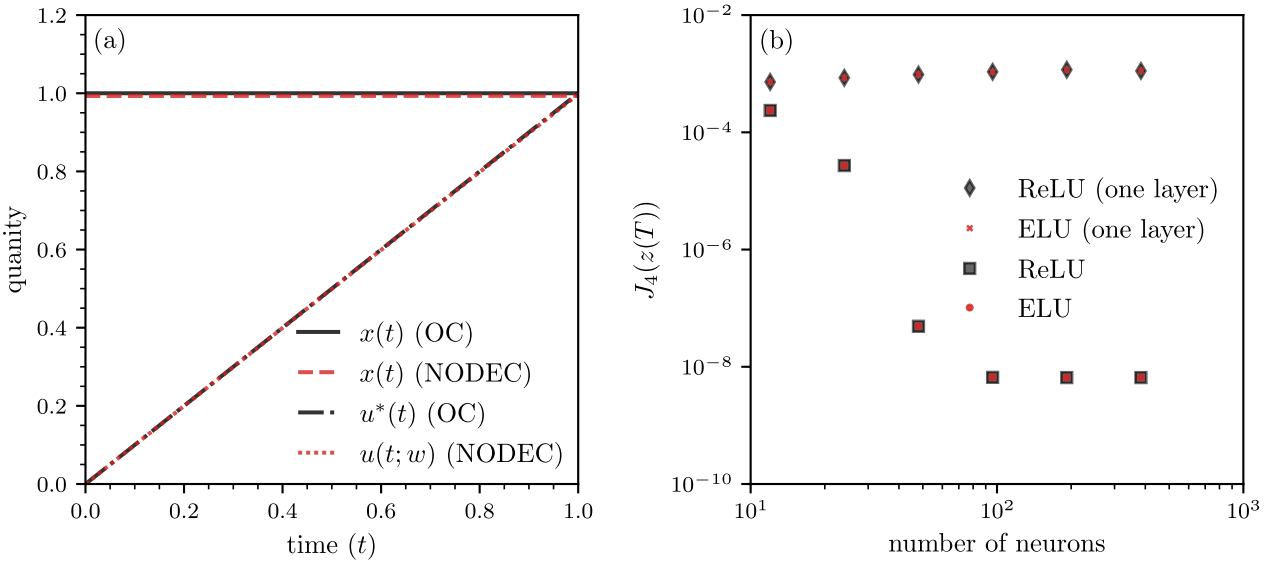

Fig. 9 Control of a moving block. (a) Evolution of the state $x(t)$ and control input $u(t)$. Optimal control (OC) and NODEC-based solutions are shown in black and red, respectively. (b) The loss $J_4(z(T)) = \|z(T) - z^*\|_2^2$ for different activation functions and architectures, plotted as a function of the number of neurons. Diamonds and crosses indicate solutions based on single-layer architectures. For square and disk markers, the number of hidden layers is 2, 4, 8, 16, or 32, with 6 neurons per layer.

In addition to the previous numerical experiment, we now explicitly include the additional loss term $W[u, v]$ and optimize the objective

$$J_5[u, v] = \|z(T) - z^*\|_2^2 + \mu W[u, v]\,, \tag{28}$$

where μ is a Lagrange multiplier that determines the influence of $W[\cdot]$ on the total loss. To initialize the weights, we use the Kaiming uniform initializer [72], and we set all biases initially to a value of 10^{-2}.

The neural network that we use to optimize $J_5[u, v]$ consists of 8 hidden layers, each containing 6 ELU neurons. We train the network using the Adam optimizer with a learning rate of $\eta = 10^{-1}$ for 100 epochs, and we evaluate the best-performing model.

The use of uniform weight initialization and a relatively large learning rate represents a deliberately non-optimized hyperparameter configuration that requires tuning of the multiplier μ.

As shown in [14], the loss term $\|z(T) - z^*\|_2^2$ reaches a minimum for $\mu \approx 2 \times 10^{-3}$, and the corresponding value of $W[u, v]$ is reasonably close to the optimal solution. However, the results in [14] also demonstrate that implicit regularization by excluding $W[u, v]$ from the loss [see Eq. (27)] can achieve lower overall loss values while still producing similar values of $W[u, v]$.

Furthermore, when analyzing solutions obtained for different values of the multiplier μ, we observe that while the state trajectories $x(t)$ remain largely aligned with the optimal control solution, variations in μ lead to noticeable deviations in the control input $\hat{u}(t; w)$ relative to the optimal control $u^*(t)$.

3.2 Population dynamics

We continue our discussion of NODEC by applying it to models of population dynamics, which are useful for studying species interactions in both ecological and biomedical contexts. While we examined discrete-time formulations for single- and multi-species dynamics in Sections 2.1 and 2.2, we now shift our focus to a continuous-time framework and consider the generalized Lotka–Volterra (gLV) equations. This model has been employed, for instance, in microbial ecology [55], where species frequently engage in inhibitory and facilitative interactions. In this context, a commonly used form of the gLV equations is

$$\dot{x}_i(t) = x_i(t)\left(b_i + \sum_{j=1}^{n} m_{ij} x_j(t) + \epsilon_i u(t)\right)\,, \tag{29}$$

where $x_i(t)$ denotes the abundance of microbial species i at time t, b_i is its intrinsic growth rate, and m_{ij} represents the interaction coefficient quantifying the effect of species j on species i. The term $\epsilon_i u(t)$ models the effect of an external antibiotic treatment $u(t)$, where ϵ_i indicates the antibiotic susceptibility of species i. A negative ϵ_i corresponds to inhibition by the antibiotic, while a positive value indicates the opposite.

While gLV models are widely used to study microbial systems, a key limitation is their assumption of direct interactions between microbial species, which overlooks indirect effects mediated by competition for shared nutrients.

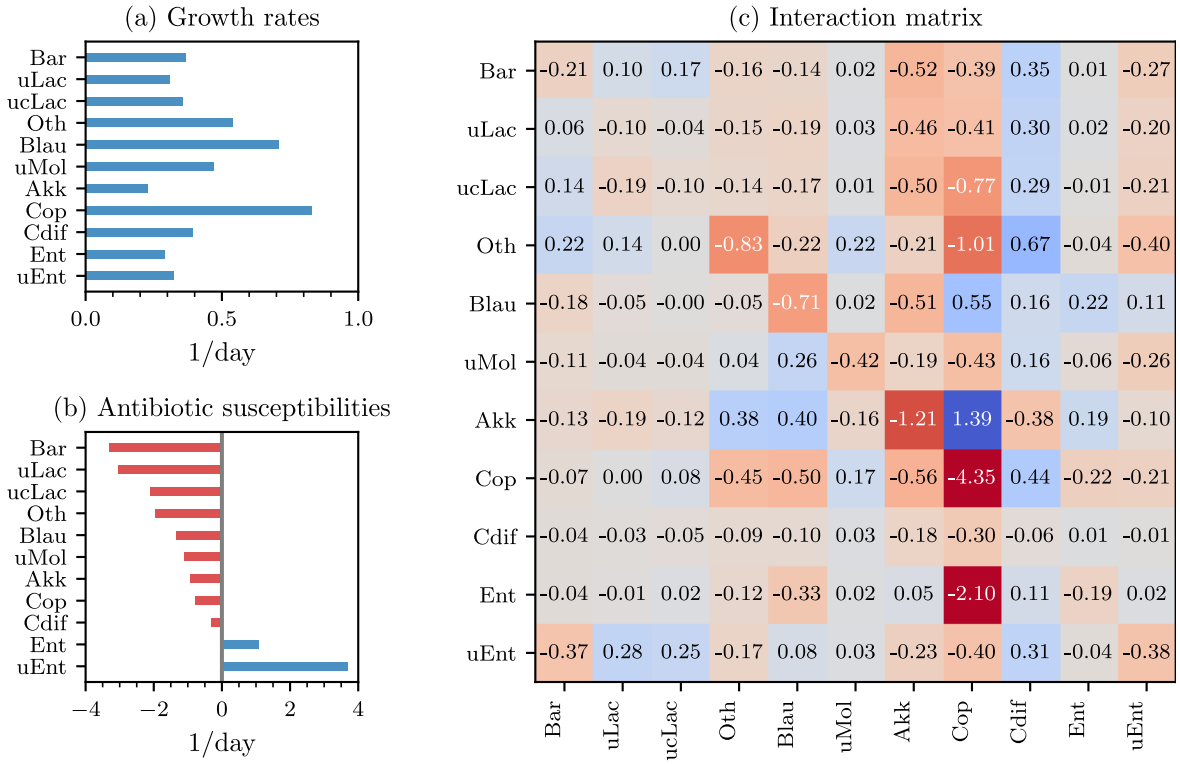

Fig. 10 Microbial dynamics under antibiotic perturbation. (a) Species-specific growth rates (1/day). (b) Antibiotic susceptibilities (1/day) in response to clindamycin, with negative values (red) indicating inhibition and positive values (blue) indicating facilitation. (c) Interaction matrix quantifying pairwise effects between species (element m_{ij} shows effect of species j on species i), with negative values (red) indicating inhibition and positive values (blue) indicating facilitation. Species are ordered by antibiotic susceptibility from most inhibited to most promoted. Species abbreviations: Bar (*Barnesiella*), uLac (undefined Lachnospiraceae), ucLac (unclassified Lachnospiraceae), Oth (Other), Blau (*Blautia*), uMol (unclassified Mollicutes), Akk (*Akkermansia*), Cop (*Coprobacillus*), Cdif (*C. difficile*), Ent (*Enterococcus*), uEnt (undefined Enterobacteriaceae). Data is based on [73,74].

In [74], gLV parameters were inferred using mouse data from a study [73] that examined the effect of the antibiotic clindamycin on intestinal colonization by the spore-forming pathogen *C. difficile*. The dataset includes a total of $n = 11$ species. In Fig. 10, we show the estimated growth rates b_i, clindamycin susceptibilities ϵ_i, and elements of the interaction matrix m_{ij}. All growth rates are positive, while the diagonal elements of the interaction matrix, m_{ii}, are negative. These negative values indicate that each species can reach its carrying capacity even in the absence of other species. The inferred clindamycin susceptibilities suggest that the antibiotic inhibits all microbial species except *Enterococcus* and an undefined group of *Enterobacteriaceae*. *C. difficile* itself appears to be only mildly inhibited by clindamycin.

We now consider a control problem focused on treating a *C. difficile* infection following the administration of clindamycin [75–77].

We use the growth rates, interaction coefficients, and clindamycin susceptibilities from [73,74] (see Fig. 10) and initialize the model with initial condition 5 from [74, 75]. To simulate infection onset, we introduce a small initial perturbation of 10^{-10} (in nondimensional units) to the *C. difficile* compartment and apply a unit dose of clindamycin on the first day. This treatment protocol is consistent with the constant dosing schedule considered in [75].

In Fig. 11(a), we show the corresponding evolution of microbial species. The results indicate that the initial antibiotic intervention, in combination with the *C. difficile* perturbation, leads to a substantial infection after approximately 90–100 days. In the absence of both perturbation and treatment, the system evolves as in Fig. 3(a) of [75].

C. difficile infections are, for instance, treated with antibiotics such as vancomycin or metronidazole [78]. Following the approach in [75], we now consider a hypothetical targeted antibiotic that is highly effective against *C. difficile*. The treatment begins on day 100 and lasts for 10 days. In our model, we set the antibiotic susceptibility of *C. difficile* to -1, while the susceptibilities of all other microbial species are set to 0.

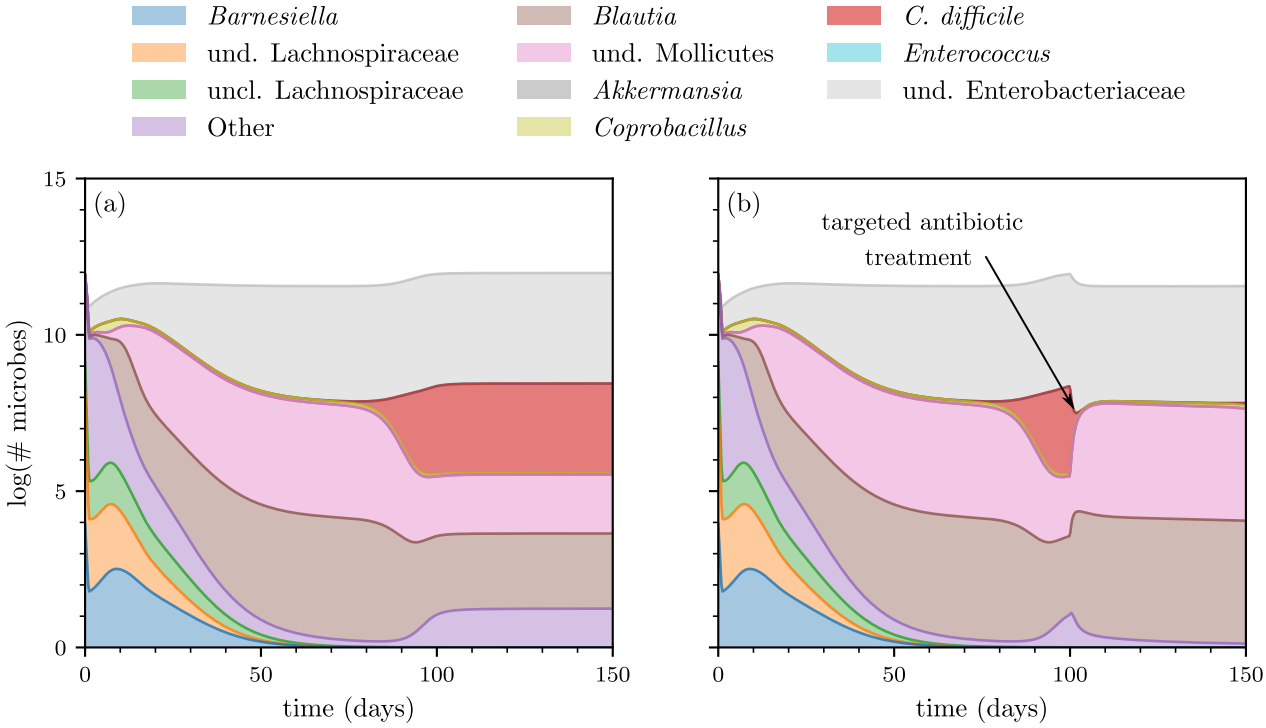

Fig. 11 Simulated microbial dynamics under antibiotic interventions. (a) Without targeted treatment, the initial administration of clindamycin promotes the outgrowth of *C. difficile*, leading to a persistent infection. (b) NODEC-based targeted antibiotic treatment starting on day 100 (black arrow), effectively suppressing *C. difficile*. Colors represent different microbial groups as indicated in the legend.

We train a NODEC to minimize the loss function

$$J_6[x,\hat{u}] = \frac{1}{50}\int_{100}^{150} x_9(t)\,\mathrm{d}t + \mu\frac{1}{10}\int_{100}^{110}\hat{u}^2(t;w)\,\mathrm{d}t\,, \tag{30}$$

which penalizes the abundance of *C. difficile* (modeled by compartment x_9) over time, while also promoting prudent use of the targeted antibiotic. Time is measured in days. In our simulations, we set $\mu = 0.01$.

In Fig. 11(b), we show the simulation results obtained using the trained controller. We observe that the targeted antibiotic treatment successfully suppresses the *C. difficile* infection. The NODEC that we employ consists of five hidden layers, each with four ELU neurons. Training was performed for 200 epochs using the Adam optimizer with a learning rate of 10^{-3}, yielding a minimum loss of 0.061. As a baseline for comparison, we simulated a naive treatment strategy that administered a constant unit dose per day over the same 10-day period. This approach resulted in a loss more than ten times higher and failed to eliminate the infection.

3.3 Oscillator dynamics

We conclude this section on continuous-time dynamics by examining control problems associated with the Kuramoto model [79], which describes a system of coupled oscillators. Each oscillator is characterized by a phase θ_i and an intrinsic (natural) frequency ω_i, where $i \in \{1,\dots,n\}$. The system dynamics are given by

$$\begin{aligned}\dot{\Theta}(t) &= \Omega + f(\Theta(t),u(t))\\ \Theta(0) &= \Theta_0\,,\end{aligned} \tag{31}$$

where $\Theta = (\theta_1,\dots,\theta_n)^\top$ and $\Omega = (\omega_1,\dots,\omega_n)^\top$ [79]. We sample the natural frequencies ω_i and initial phases $\theta_i(0)$ from a normal distribution with mean 0 and standard deviation 0.2.

The interactions among oscillators, as well as the effect of control inputs $u_i(t)$ on oscillator i, are modeled by the function

$$f_i(\Theta(t),u(t)) = \frac{K u_i(t)}{n}\sum_{j=1}^{n} a_{ij}\sin(\theta_j(t)-\theta_i(t))\,, \tag{32}$$

where K is the coupling strength, and a_{ij} are the adjacency-matrix elements representing the underlying undirected network. To evaluate the level of synchronization at the

final time T, we use the complete synchronization condition

$$|\dot{\theta}_i(T) - \dot{\theta}_j(T)| = 0 \text{ for } (i,j) \in E\,, \tag{33}$$

where E denotes the set of edges in the network [80,81]. When condition (33) is met, all connected oscillators exhibit constant phase differences.

In the special case where all control inputs are set to 1 (*i.e.*, $u_i(t) = 1\ \forall i$) the system described by Eq. (31) possesses a unique and stable synchronized state, provided that the coupling strength K exceeds the threshold

$$K^* = \left\|L^{\dagger}\Omega\right\|_{E,\infty}\,, \tag{34}$$

where $L^{\dagger}$ denotes the Moore–Penrose pseudo-inverse of the combinatorial graph Laplacian and $\|x\|_{E,\infty} = \max_{(i,j)\in E}|x_i - x_j|$ is the maximum distance between elements in $x = (x_1,\ldots,x_n)^{\top}$ that are connected via an edge in E [82]. In our simulations, we use a subcritical coupling strength $K = 0.1K^*$, such that synchronization requires that some $u_i(t)$ must exceed 1 to achieve it.

For a global control input $u(t)$ (*i.e.*, $u_i(t) = u(t)\ \forall i$), there exists an optimal control $u^*(t)$ that minimizes the cost functional

$$J_7[\Theta(T), u] = \frac{1}{2}\sum_{i,j} a_{ij}\sin^2(\theta_j(T) - \theta_i(T)) + \frac{\mu}{2}E[u]\,, \tag{35}$$

where the parameter μ determines the relative weight of the energy regularization term $E[u] = \int_0^T \|u(t)\|_2^2\,\mathrm{d}t$ in the cost function. Minimizing $J_7[\Theta(T), u]$ aligns with the synchronization objective defined in Eq. (33) [81].

The optimal control for this problem can be computed using the adjoint-gradient method (AGM), which combines Pontryagin's maximum principle with gradient descent on u [81]. Specifically, the control is updated according to

$$u^{(\ell+1)} = u^{(\ell)} - \tilde{\eta}\left[\mu u^{(\ell)} + \frac{K}{n}\sum_{i=1}^{n}\lambda_i\sum_{j=1}^{n}a_{ij}\sin(\theta_j - \theta_i)\right]\,, \tag{36}$$

where $\tilde{\eta}$ denotes the AGM learning rate, and the quantity $\lambda = (\lambda_1,\ldots,\lambda_n)^{\top}$ is the solution to the adjoint system

$$\begin{aligned}-\dot{\lambda}_i = &-\frac{Ku\lambda_i}{n}\sum_{i\neq j}a_{ij}\cos(\theta_j - \theta_i)\\ &+\frac{Ku}{n}\sum_{i\neq j}a_{ij}\lambda_j\cos(\theta_j - \theta_i)\,,\end{aligned} \tag{37}$$

with terminal condition $\lambda_i(T) = 1/2\sum_{i\neq j}a_{ij}\sin(2\theta_i(T) - 2\theta_j(T))$.

We compare the control performance of NODEC applied to Eq. (31) with that of the AGM. The neural controller we employ learns $\hat{u}(t;w)$ based on the loss function (35) with a gradient descent in w and without energy regularization term $\mu E[u]/2$. We denote this loss function by

$$J_8(\Theta(T)) = \frac{1}{2}\sum_{i,j}a_{ij}\sin^2(\theta_j(T) - \theta_i(T))\,. \tag{38}$$

Since both NODEC and the AGM rely on different optimization procedures with distinct learning rates, we chose learning rates for which the corresponding order parameter values are approximately equal. As shown in [15], a high degree of synchronization can be achieved by controlling only a fraction of the nodes. That work also demonstrates how a maximum matching approach [83] can be used to identify driver nodes for controlling linear dynamics involving over 1000 nodes.

A commonly used measure of the degree of synchronization is the order parameter

$$r(t) = \frac{1}{n}\sqrt{\sum_{i,j}\cos\left[\theta_j(t) - \theta_i(t)\right]}. \tag{39}$$

This expression follows from the fact that the squared magnitude of the complex order parameter $z = re^{i\psi(t)} = \frac{1}{n}\sum_{j=1}^{n}e^{i\theta_j(t)}$ [79] can be rewritten as

$$r(t)^2 = |z|^2 = \frac{1}{n^2}\sum_{i,j}\cos\left[\theta_j(t) - \theta_i(t)\right]\,. \tag{40}$$

A value of $r(t) = 1$ indicates perfect synchronization, where all oscillators share the same phase.

We now apply NODEC to control oscillator systems on a square lattice with periodic boundaries and with $n = 2500$ nodes, and compare it with the AGM, setting $T = 0.5$. We find that the control energy and order parameter ratios are $E^{\mathrm{NODEC}}[u]/E^{\mathrm{AGM}}[u] \approx 1.0045$ and $r^{\mathrm{NODEC}}(T)/r^{\mathrm{AGM}}(T) \approx 0.9999$, respectively. NODEC and the AGM achieve similar values for both the order parameter and control energy at time $T = 0.5$, indicating that both methods effectively control the considered oscillator system. In [15], NODEC has also been applied to directed networks, and its robustness to noise has been analyzed.

For a runtime performance comparison, we measure the learning (or wall-clock) time associated with controlling the system. To this end, we determine the runtime of 50 control realizations for both the AGM and NODEC. The mean runtimes are 74 s and 1.03 s for the

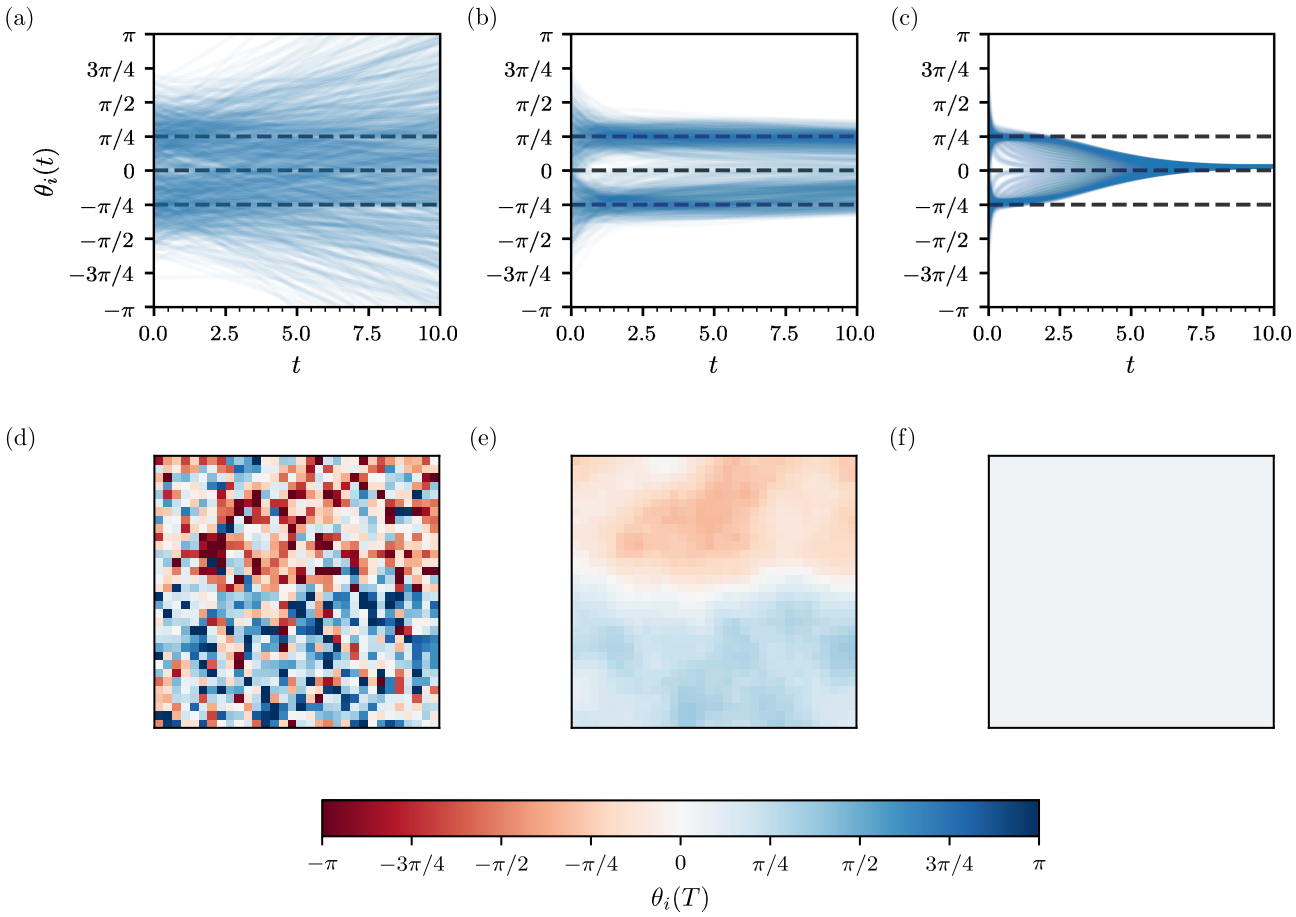

Fig. 12 Kuramoto dynamics and distinct target states. We simulate the phase evolution $\theta_i(t)$ of $n = 1024$ coupled Kuramoto oscillators ($i \in \{1, \dots, n\}$), arranged on a 32×32 square lattice without periodic boundary conditions [see Eqs. (31) and (32)]. A subcritical coupling strength of $K = 0.01K^*$ is used, and the control horizon is set to $T = 10$. Initially, the oscillator phases follow a bimodal Gaussian distribution with means $-\pi/4$ (top half of the lattice) and $\pi/4$ (bottom half), and variance 0.5. The top panels show the evolution of $\theta_i(t)$, while the bottom panels display the spatial phase distribution $\theta_i(T)$ at the final time $T = 10$. Each pixel in the bottom panels corresponds to the phase of a specific oscillator in the lattice. (a,d) With the control input fixed at $u_i(t) = 1$ for all i (uncontrolled dynamics), the phase differences grow over time. (b,e) NODEC successfully drives the system toward a state in which the oscillator phases approach $-\pi/4$ and $\pi/4$ by minimizing the loss function $J_9(\Theta(T))$ [see Eq. (41)]. (c,f) NODEC achieves global synchronization by minimizing $J_8(\Theta(T))$ [see Eq. (38)]. The dashed black lines in panels (a–c) indicate reference phases of $-\pi/4$, 0, and $\pi/4$. The learning rate is set to 15 in panels (b,e) and 0.12 in panels (c,f).

AGM and NODEC, respectively. For the considered oscillator system, NODEC's training time is thus approximately two orders of magnitude shorter than that of the AGM. In [15], runtime differences between NODEC and the AGM have been analyzed in more detail. The main bottleneck that has been identified in the AGM is that the adjoint system solver requires small step sizes to accurately capture the interaction between the adjoint dynamics [see Eq. (37)] and the gradient descent updates [see Eq. (36)] applied to the control inputs.

We now consider a control problem with a target state that differs from full synchronization. Specifically, we aim to steer each oscillator toward either $-\pi/4$ or $\pi/4$. This control objective is captured by the loss function

$$J_9(\Theta(T)) = \frac{1}{2} \sum_{i=1}^{n} \left(|\theta_i(T)| - \frac{\pi}{4} \right)^2 . \tag{41}$$

Figure 12 shows that NODEC, when trained using the loss function $J_9(\Theta(T))$, can successfully steer a system of $n = 1024$ coupled Kuramoto oscillators, arranged on a square lattice with subcritical coupling strength $K = 0.01K^*$, towards a target state consisting of two distinct spatial regions. In this configuration, the oscillators converge to phase values $\theta_i(T)$ close to either $-\pi/4$ or $\pi/4$.

In Figs. 12(a,d), no control is applied, and we observe increasing phase dispersion over time. In contrast, Figs. 12(b,e) show that NODEC, trained with

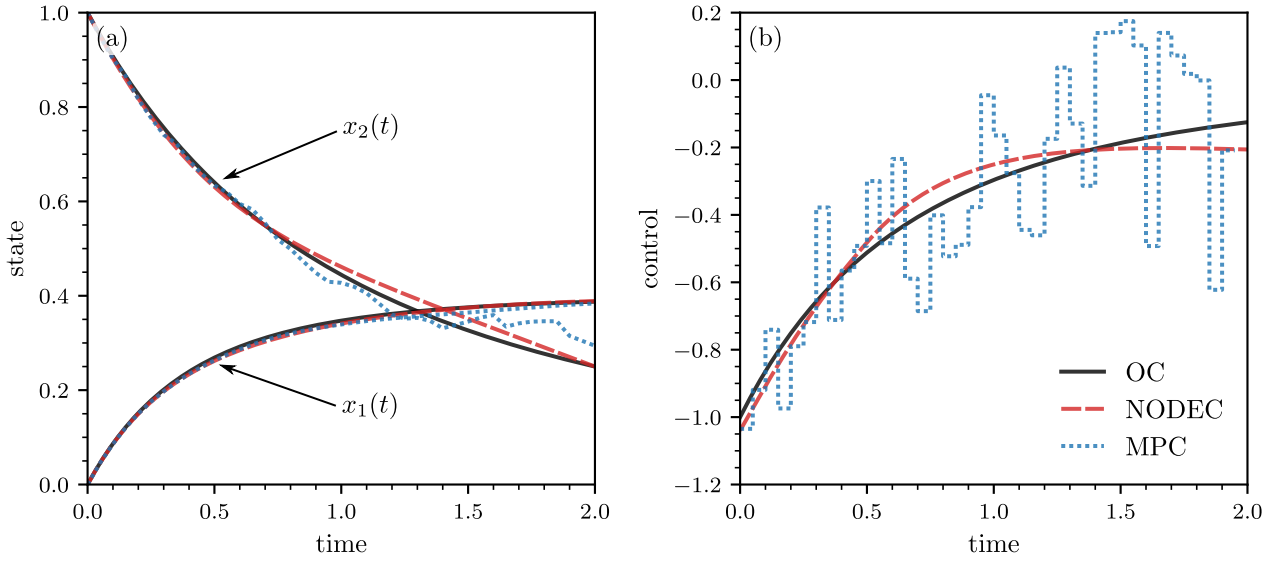

Fig. 13 Comparison of control approaches applied to the nonlinear control problem from Eqs. (42) and (43). (a) Evolution of the state variables $x_1(t)$ and $x_2(t)$ under OC (solid black line), NODEC (dashed red line), and MPC (dotted blue line). (b) Corresponding control inputs. NODEC approximates OC more closely than MPC, which exhibits higher variability.

$J_9(\Theta(T))$, drives the system toward a target state in which oscillators in the upper half of the lattice converge to phase values near $-\pi/4$ (indicated by light orange), while those in the lower half converge to values around $\pi/4$ (indicated by light blue).

For comparison, we also use NODEC with the synchronization loss function $J_8(\Theta(T))$, which results in complete phase alignment across the lattice as illustrated in Figs. 12(c,f).

In the studied examples, NODEC has two key advantages over adjoint-based control methods. First, approximate optimal control trajectories can be obtained without deriving and solving the adjoint system (see Section 2.1 for a corresponding example in discrete time). The only inputs necessary are (i) the dynamical system, (ii) its initial state, and (iii) the desired target state. Second, the runtime of NODEC may be substantially faster than that of adjoint-gradient methods.

4 Comparison with model predictive control

Model predictive control (MPC) represents a control paradigm closely related to the neural controllers discussed in previous sections. In MPC, one optimizes a loss function over a finite time horizon at each time step, applies only the first control input, and repeats this process until the final time. This requires solving an optimization problem for each updated system state, which can be computationally demanding, especially in nonlinear settings. However, MPC has the advantage of naturally handling constraints, such as bounds on control inputs, and of adapting in real time to disturbances or model errors. In contrast, NODEC, as employed in the earlier sections, does not solve an optimization problem during execution. Instead, it learns a parameterized control policy by constructing a computational graph over the entire system evolution and minimizing a loss function offline. This approach can be computationally efficient, but enforcing bounds on control inputs or adapting to unexpected conditions would require extensions beyond the approach considered in previous sections.

To provide insights into the different optimization paradigms underlying NODEC and MPC, we adapt a nonlinear optimal control problem from [71] as our test case. The goal is to minimize

$$J_{10}[u] = 4\int_0^2 u^2(t),\mathrm{d}t\,, \tag{42}$$

subject to the nonlinear dynamics and boundary conditions

$$\begin{aligned} \dot{x}_1(t) &= x_2^3(t)\,,\\ \dot{x}_2(t) &= u(t)\,,\\ (x_1(0), x_2(0)) &= (0,1)\,,\\ (x_1(2), x_2(2)) &= (0.3875, 0.25)\,. \end{aligned} \tag{43}$$

This control problem admits the analytical solution

$$u^*(t) = -\frac{8}{(2+t)^3}\,, \tag{44}$$

$$x_1^*(t) = \frac{2}{5} - \frac{64}{5(2+t)^5}\,, \tag{45}$$

$$x_2^*(t) = \frac{4}{(2+t)^2}\,. \tag{46}$$

To solve the described control problem with NODEC, we use a parameterized control input $\hat{u}(t; w)$, represented by a neural network with two hidden layers, each containing four ELU activations. We trained NODEC using the Adam optimizer with a learning rate of 0.1.

Training was performed in two stages. In the first stage, we minimized the mean squared error (MSE) between the predicted and target values of $x_1(t)$. In the second stage, we jointly minimized the MSE with respect to both $x_1(t)$ and $x_2(t)$. Directly optimizing the full loss, including the control objective J_{10}, did not lead to satisfactory solutions. This observation is consistent with related findings in [15].

For comparison with NODEC, we consider a receding-horizon MPC approach. At each time step $t_k = k\Delta t$ of the discretized dynamics (43), we solve the finite-horizon optimal control problem

$$\begin{aligned}\min_{\{u_k\}} \quad & 100 \left\| z_{N_\mathrm{p}} - z^* \right\|^2 + 10 \sum_{k=1}^{N_\mathrm{p}-1} (u_k - u_{k-1})^2 \\ & + \Delta t \sum_{k=0}^{N_\mathrm{p}-1} u_k^2 \,,\end{aligned} \tag{47}$$

where N_p is the number of steps in the prediction horizon, $z^* = (0.3875, 0.25)^\top$ is the desired terminal state, and z_{N_p} denotes the predicted state at the end of the horizon. The multipliers in Eq. (47) have been chosen to obtain a solution that is aligned with the optimal one.

The state trajectory evolves according to the discretized dynamics

$$z_{k+1} = f_\mathrm{d}(z_k, u_k), \quad z_0 = z(t_k), \tag{48}$$

where $f_\mathrm{d} \colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the discrete-time approximation of the continuous-time system (43) obtained via numerical integration.

After solving the optimization problem, only the first control input u_0 is applied to the system. At the next time step, the procedure is repeated using the updated system state. For each time step, we use the Adam optimizer with a learning rate of 0.5 to optimize the control sequence over the finite prediction horizon.

In Fig. 13, we show a comparison of the evolution of both the system state and control inputs obtained using OC [see Eqs. (44)–(46)], NODEC, and MPC. NODEC closely tracks the optimal trajectory, whereas MPC exhibits larger deviations and higher variability in the control signal due to its locally optimal receding-horizon updates. NODEC reaches a state of $(0.39, 0.25)$, which is closer to the target state of approximately $(0.39, 0.25)$ than MPC's $(0.38, 0.29)$. Furthermore, NODEC achieves a loss value of 1.56, nearly matching the optimal 1.55, while MPC yields a higher loss of 1.73.

The example presented here is intended to illustrate the different optimization procedures underlying NODEC and MPC. (A related comparison between MPC and a neural control approach using experimental data of CartPole and F1TENTH Race Car systems has been studied in [84].) Despite these differences between neural controllers and MPC, there is significant potential in integrating neural approximators into MPC frameworks. For example, neural ODEs have been employed to model unknown dynamical systems [85, 86], or to augment physics-based models in real-world systems such as aerial robots [16]. Additionally, input convex neural networks [87] have been used to model systems while preserving convexity properties, enabling efficient optimization in MPC [88]. A recent application of such an approach focuses on brain stimulation in Parkinson's disease [29].

Beyond augmenting models with neural approximators, other works have studied the integration of learned dynamics into MPC approaches more broadly. In low-data regimes where interpretability and online adaptability are important, the sparse identification of nonlinear dynamics (SINDY) framework [89] provides a promising alternative to standard neural approximators in MPC tasks. In another work, researchers have proposed a two-stage framework that first learns the dynamics of networked systems offline using graph neural networks, and then employs MPC online using the learned model to compute control inputs [22]. This approach has been applied to complex systems including an agent-based model, a networked epidemic model, and the Kuramoto model.

5 Uncertainty quantification with conformal prediction

When applying neural control models to real-world systems, it is essential to account for uncertainty due to noise. In this context, conformal prediction [90,91] provides a model-agnostic and computationally efficient approach for uncertainty quantification in machine learning. The core idea is to begin with a point-prediction method and define a nonconformity score that quantifies the distance between a new sample and one from the calibration set. The conformal-prediction algorithm then transforms these scores into prediction regions with guaranteed coverage.

We apply split conformal prediction to quantify uncertainty in the dynamical system (43) controlled using NODEC. In our simulations, we incorporate both uncertainty in the initial condition (modeled as Gaussian noise with a standard deviation of 0.02) and additive process noise (Gaussian with a standard devia-

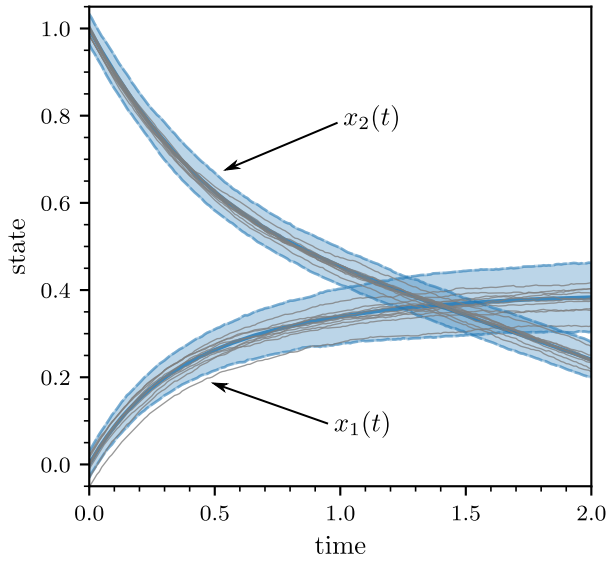

Fig. 14 Conformal-prediction intervals for NODEC-controlled trajectories of the nonlinear system in Eqs. (42)–(43), under process and initial state noise. The solid blue line indicates the mean trajectory and blue shaded regions show the 90% prediction intervals. Solid grey lines are sample trajectories from the test set.

tion of 0.1). We simulate multiple realizations at each time step. These samples are drawn independently from the same distribution and are therefore exchangeable, which is a key assumption in conformal prediction [90].

We implement split conformal prediction using the `PUNCC` library [92]. The key steps are as follows:

1. We first compute a predictor $x_{(i,k)}$ for each state element $i \in \{1,2\}$ and time step $k \in \{0,\dots,N_T\}$, defined as the mean over M realizations.
2. Next, we compute nonconformity scores for each time step $k \in \{0,\dots,N_T\}$ and calibration trajectory $j \in \{1,\dots,\widetilde{M}\}$. Specifically, for each state element $i \in \{1,2\}$, the nonconformity scores are

$$s^{(j)}_{(i,k)}\left(x_{(i,k)}, \tilde{x}^{(j)}_{(i,k)}\right) = \left|x_{(i,k)} - \tilde{x}^{(j)}_{(i,k)}\right| , \tag{49}$$

   where $x_{(i,k)}$ denotes the i-th element of the mean trajectory at time step k, and $\tilde{x}^{(j)}_{(i,k)}$ is the corresponding element of the j-th noisy calibration trajectory.
3. For a new trajectory element $\tilde{x}^{(\widetilde{M}+1)}_{(i,k)}$, we compute the conformal prediction interval from the nonconformity scores as

$$C\left(\tilde{x}^{(\widetilde{M}+1)}_{(i,k)}\right) = \left[C_-\left(\tilde{x}^{(\widetilde{M}+1)}_{(i,k)}\right), C_+\left(\tilde{x}^{(\widetilde{M}+1)}_{(i,k)}\right)\right] . \tag{50}$$

The corresponding interval bounds are

$$C_\pm\left(\tilde{x}^{(\widetilde{M}+1)}_{(i,k)}\right) = x_{(i,k)} \pm Q_{1-\alpha}\left(\sum_{j=1}^{\widetilde{M}} \frac{1}{\widetilde{M}+1}\cdot\delta_{s^{(j)}_{(i,k)}} + \frac{1}{\widetilde{M}+1}\cdot\delta_{+\infty}\right), \tag{51}$$

where $Q_{1-\alpha}$ is the $(1-\alpha)$-quantile of the weighted empirical distribution, and δ_x is a point mass at x. The probability that the new trajectory element $\tilde{x}(i,k)^{(\widetilde{M}+1)}$ lies within the prediction interval satisfies

$$\Pr\left(\tilde{x}^{(\widetilde{M}+1)}_{(i,k)} \in C\left(\tilde{x}^{(\widetilde{M}+1)}_{(i,k)}\right)\right) \geq 1-\alpha\,. \tag{52}$$

(See Theorem 1 in [93] and Proposition 1 in [94].)

Figure 14 shows the mean predicted trajectory (solid blue line) based on $M = 100$ realizations, as well as the 90% conformal prediction intervals (blue shaded regions), computed using $\widetilde{M} = 100$ calibration trajectories. Sample test trajectories are shown as solid grey lines. Most test trajectories lie well within the predicted intervals, demonstrating the effectiveness of the conformal prediction approach in quantifying uncertainty when applying NODEC in noisy environments.

If multiple process realizations are not available in a given problem (*i.e.*, if only a single time series is observed), the exchangeability assumption underlying standard conformal prediction becomes particularly restrictive, as time-series data are typically temporally correlated. To address this limitation, researchers have proposed a class of methods based on weighted quantiles [93], which relax the exchangeability requirement by assigning lower weights to observations that are further back in time.

Instead of assigning uniform weights $1/(\widetilde{M}+1)$ in Eq. (51), one can, for instance, use geometrically decaying weights such as

$$w_{(i,k)} = \rho^{N_T-k+1} \tag{53}$$

with $\rho \in (0,1)$ and normalize them according to

$$\tilde{w}_{(i,k)} = \frac{w_{(i,k)}}{W}\,, \quad \text{where } W = \sum_{k=0}^{N_T} w_{(i,k)} + 1\,. \tag{54}$$

Here, $\tilde{w}_{N_T+1} = W^{-1}$ is the weight of a future (unseen) observation. This weighting scheme is consistent with approaches in [21, 92, 93]. These weights are then used to compute a prediction interval for a future time step.

In addition to the example that we considered in this section, conformal prediction has been integrated

into MPC frameworks where neural networks are used to generate prediction regions [95]. It has also been employed in MPC tasks where neural networks model unknown system dynamics [16,21]. In the context of nonlinear models of biological systems, conformal prediction has enabled uncertainty quantification up to two orders of magnitude faster than a Bayesian method, under the assumption of homoscedastic noise (*i.e.*, constant variance of measurement errors relative to the signal across the time horizon) [96]. Related work has also applied conformal prediction to provide uncertainty quantification for neural surrogate models of partial differential equations [97].

6 Conclusions

As deep-learning and automatic-differentiation frameworks continue to improve, their application to control and optimization problems is becoming increasingly practical.

In this paper, we first reviewed selected neural-control approaches for discrete- and continuous-time dynamical systems, in both deterministic and stochastic settings, complemented by new examples that illustrate key concepts and highlight possible extensions for future research. We focused on applications across various domains such as biology, engineering, physics, and medicine. For continuous-time dynamical systems, neural ordinary differential equations (neural ODEs) provide a flexible framework for control-input parameterization. For discrete-time systems, we showed how custom neural control-input parameterizations can be implemented and optimized via automatic differentiation.

We then compared the differing optimization paradigms underlying model predictive control (MPC) and neural ODE control (NODEC), and discussed related approaches that incorporate neural networks into MPC. While NODEC typically relies on a fixed integration scheme during training, further research is needed to understand how changes in time step size or solver configuration affect generalization and performance at deployment [98,99]. Finally, we integrated conformal prediction into noisy, neural-controlled dynamical systems to generate prediction intervals with guaranteed statistical coverage. We summarize code repositories associated with our work and related examples in Table 1.

In several of the systems we studied, such as the block-move example, oscillator control, and the nonlinear system analyzed in the NODEC-MPC comparison, we observed that minimizing the distance to the terminal state can also implicitly reduce the running cost. Recent work [100] presents an example demonstrating improved convergence of neural controllers that leverage a control Lyapunov function, compared to those focused solely on terminal-cost minimization.

We also highlighted the utility of a straight-through estimator for obtaining integer-valued control inputs in both the predator-prey agent-based model and the inventory dynamics example. Similar techniques have been applied in other domains, including recommender system design [57] and the calibration of financial agent-based models [58]. However, broader adoption of this approach will likely require advances in differentiating through more complex dynamics, particularly those involving discrete or stochastic elements (*e.g.*, personalized medical digital twins [101]). In this context, stochastic automatic differentiation techniques show considerable promise [102, 103]. Connections to differentiable variants of the Gillespie algorithm [104] also warrant further exploration. Alternatively, gradient-free methods such as ensemble Kalman inversion [17] may provide a viable path forward, especially in settings where gradients are ill-defined or intractable.

Acknowledgements This work was supported by hessian.AI. I am grateful to Thomas Asikis, Nino Antulov-Fantulin, Ioannis Fragkos, Jiawei Li, Luis L. Fonseca, Marcos Matabuena, and Reinhard C. Laubenbacher for valuable discussions. I also thank Daniel Garza for organizing the School of Artificial Intelligence Applied to Microbiomes at AgroParisTech.

Table 1 Relevant code repositories related to this work.

Topic	Repository link
Application of NNC to two agent-based models	`https://gitlab.com/ComputationalScience/abm-control`
Application of NNC to inventory dynamics	`https://gitlab.com/ComputationalScience/idinn`
Application of NODEC to several dynamical systems	`https://github.com/asikist/nnc`
Additional NODEC examples	`https://gitlab.com/ComputationalScience/near-optimal-control`
Additional examples presented in this paper	`https://gitlab.com/ComputationalScience/neural-control`
Application of NODEC to microbiome dynamics	`https://github.com/danielriosgarza/AiSchool/tree/main/content/LucasBottcher`
Conformal prediction	`https://github.com/deel-ai/puncc`

References

1. Arturo Rosenblueth, Norbert Wiener, and Julian Bigelow. Behavior, purpose and teleology. *Philosophy of Science*, 10(1):18–24, 1943.
2. Norbert Wiener. *Cybernetics or control and communication in the animal and the machine*. MIT Press, Boston, MA, USA, 1948.
3. Edward Thorndike. *Fundamentals of learning*. Teachers College, Columbia University, New York City, NY, USA, 1932.
4. Patricia S. Churchland and Terrence J. Sejnowski. *The computational brain*. MIT Press, Cambridge, MA, USA, 1992.
5. David E. Rumelhart and James L. McClelland. *Parallel distributed processing, Volume 1: Explorations in the microstructure of cognition: Foundations*. MIT Press, Cambridge, MA, USA, 1986.
6. David E. Rumelhart and James L. McClelland. *Parallel distributed processing, Volume 2: Explorations in the microstructure of cognition: Psychological and biological models*. MIT Press, Cambridge, MA, USA, 1986.
7. W Thomas Miller, Paul J Werbos, and Richard S Sutton. *Neural networks for control*. MIT Press, Cambridge, MA, 1995.
8. Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. *NeurIPS Workshop on Autodiff*, 2017.
9. James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
10. Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian networks: Using physics as model prior for deep learning. *arXiv preprint arXiv:1907.04490*, 2019.
11. Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019.
12. Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
13. Thomas Asikis, Lucas Böttcher, and Nino Antulov-Fantulin. Neural ordinary differential equation control of dynamics on graphs. *Physical Review Research*, 4(1):013221, 2022.
14. Lucas Böttcher and Thomas Asikis. Near-optimal control of dynamical systems with neural ordinary differential equations. *Machine Learning: Science and Technology*, 3(4):045004, 2022.
15. Lucas Böttcher, Nino Antulov-Fantulin, and Thomas Asikis. AI Pontryagin or how artificial neural networks learn to control dynamical systems. *Nature Communications*, 13(1):333, 2022.
16. Kong Yao Chee, Tom Z Jiahao, and M Ani Hsieh. KNODE-MPC: A knowledge-based data-driven predictive control framework for aerial robots. *IEEE Robotics and Automation Letters*, 7(2):2819–2826, 2022.
17. Lucas Böttcher. Gradient-free training of neural ODEs for system identification and control using ensemble Kalman inversion. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems, Honolulu, HI, USA, 2023*, 2023.
18. Saviz Mowlavi and Saleh Nabi. Optimal control of PDEs using physics-informed neural networks. *Journal of Computational Physics*, 473:111731, 2023.
19. Lucas Böttcher, Thomas Asikis, and Ioannis Fragkos. Control of dual-sourcing inventory systems using recurrent neural networks. *INFORMS Journal on Computing*, 35(6):1308–1328, 2023.
20. Truong X Nghiem, Ján Drgoňa, Colin Jones, Zoltan Nagy, Roland Schwan, Biswadip Dey, Ankush Chakrabarty, Stefano Di Cairano, Joel A Paulson, Andrea Carron, et al. Physics-informed machine learning for modeling and control of dynamical systems. In *2023 American Control Conference (ACC)*, pages 3735–3750, 2023.
21. Kong Yao Chee, M Ani Hsieh, and George J Pappas. Uncertainty quantification for learning-based MPC using weighted conformal prediction. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 342–349, 2023.
22. Muyun Mou, Yu Guo, Fanming Luo, Yang Yu, and Jiang Zhang. Model predictive complex system control from observational and interventional data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 34(9), 2024.
23. Song Chen, Jiaxu Liu, Pengkai Wang, Chao Xu, Shengze Cai, and Jian Chu. Accelerated optimization in deep learning with a proportional-integral-derivative controller. *Nature Communications*, 15(1):10263, 2024.
24. Emmanuel Delaleau, Cédric Join, and Michel Fliess. Synchronization of kuramoto oscillators via HEOL, and a discussion on AI. *IFAC-PapersOnLine*, 59(1):229–234, 2025. 11th Vienna International Conference on Mathematical Modelling MATHMOD 2025.
25. Lucas Böttcher, Luis L Fonseca, and Reinhard C Laubenbacher. Control of medical digital twins with artificial neural networks. *Philosophical Transactions A*, 383(2292):20240228, 2025.
26. Pengkai Wang, Song Chen, Jiaxu Liu, Shengze Cai, and Chao Xu. PIDNODEs: Neural ordinary differential equations inspired by a proportional–integral–derivative controller. *Neurocomputing*, 614:128769, 2025.
27. Aniruddh Raghu, Matthieu Komorowski, Imran Ahmed, Leo Celi, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. *arXiv preprint arXiv:1711.09602*, 2017.
28. Yue Wen, Jennie Si, Andrea Brandt, Xiang Gao, and He Helen Huang. Online reinforcement learning control for the personalization of a robotic knee prosthesis. *IEEE Transactions on Cybernetics*, 50(6):2346–2356, 2019.
29. Sebastian Steffen and Mark Cannon. Deep learning model predictive control for deep brain stimulation in Parkinson's disease. *arXiv preprint arXiv:2504.00618*, 2025.
30. Yuming Deng, Xinhui Zhang, Tong Wang, Lin Wang, Yidong Zhang, Xiaoqing Wang, Su Zhao, Yunwei Qi, Guangyao Yang, and Xuezheng Peng. Alibaba realizes millions in cost savings through integrated demand forecasting, inventory management, price optimization, and product recommendations. *INFORMS Journal on Applied Analytics*, 53(1):32–46, 2023.
31. Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo

Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

32. Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
33. Alexey Grigorevich Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):364–378, 1971.
34. FW Lewis, Suresh Jagannathan, and Aydin Yesildirak. *Neural network control of robot manipulators and nonlinear systems*. CRC Press, Boca Raton, FL, 2020.
35. Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.
36. Yi-Jen Wang and Chin-Teng Lin. Runge-Kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks*, 9(2):294–307, 1998.
37. Murad Abu-Khalaf and Frank L Lewis. Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach. *Automatica*, 41(5):779–791, 2005.
38. Guido Novati, L. Mahadevan, and Petros Koumoutsakos. Controlled gliding and perching through deep-reinforcement-learning. *Physical Review Fluids*, 4:093902, 2019.
39. Agostino De Marco, Paolo Maria D'Onza, and Sabato Manfredi. A deep reinforcement learning control approach for high-performance aircraft. *Nonlinear Dynamics*, 111(18):17037–17077, 2023.
40. Zhiyang Gu, Chengli Fan, Dengxiu Yu, and Zhen Wang. Optimal synchronized control of nonlinear coupled harmonic oscillators based on actor–critic reinforcement learning. *Nonlinear Dynamics*, 111(22):21051–21064, 2023.
41. Xiaolong Wang, Jianfu Cao, Ye Cao, and Feng Zou. Energy-efficient trajectory planning for a class of industrial robots using parallel deep reinforcement learning. *Nonlinear Dynamics*, 113(8):8491–8511, 2025.
42. Eiji Mizutani and Stuart E Dreyfus. Two stochastic dynamic programming problems by model-free actor-critic recurrent-network learning in non-Markovian settings. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, volume 2, pages 1079–1084. IEEE, 2004.
43. Chi Jin, Zeyuan Allen-Zhu, Sébastien Bubeck, and Michael I. Jordan. Is Q-learning provably efficient? In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4868–4878, 2018.
44. Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 10674–10681. AAAI Press, 2021.
45. R. J. H. Beverton and S. J. Holt. *On the Dynamics of Exploited Fish Populations*, volume 19 of *Fisheries Investigations Series II*. Ministry of Agriculture, Fisheries and Food, London, UK, 1957.
46. F. I. Baranov. On the question of the biological basis of fisheries. *Izvestiya Otdella Rybolovstva i Nauchno-Promyslovykh Issledovanii*, 1:81–128, 1918. In Russian.
47. Trevor J Kenchington. Baranov's contributions to the Beverton–Holt model. *ICES Journal of Marine Science*, 78(6):2166–2172, 2021.
48. Andrew Whittle. Discrete time mathematical models in ecology. University of Tennessee, Department of Mathematics.
49. Robert M May and Warren J Leonard. Nonlinear aspects of competition between three species. *SIAM Journal on Applied Mathematics*, 29(2):243–253, 1975.
50. Andrzej Pekalski and Dietrich Stauffer. Three species Lotka–Volterra model. *International Journal of Modern Physics C*, 9(05):777–783, 1998.
51. Uri Wilensky. NetLogo Wolf Sheep Predation Model. *Center for Connected Learning and Computer-Based Modeling, Northwestern University*, 1997.
52. Uri Wilensky. NetLogo. http://ccl.northwestern.edu/netlogo/, 1999.
53. Matthew Oremland, Kathryn R Michels, Alexandra M Bettina, Chris Lawrence, Borna Mehrad, and Reinhard Laubenbacher. A computational model of invasive aspergillosis in the lung and the role of iron. *BMC Systems Biology*, 10(1):1–14, 2016.
54. Henrique AL Ribeiro, Luis Sordo Vieira, Yogesh Scindia, Bandita Adhikari, Matthew Wheeler, Adam Knapp, William Schroeder, Borna Mehrad, and Reinhard Laubenbacher. Multi-scale mechanistic modelling of the host defence in invasive aspergillosis reveals leucocyte activation and iron acquisition as drivers of infection outcome. *Journal of the Royal Society Interface*, 19(189):20210806, 2022.
55. Karoline Faust and Jeroen Raes. Microbial interactions: from networks to models. *Nature Reviews Microbiology*, 10(8):538–550, 2012.
56. Sukhan Lee and Jun Park. Dual-mode dynamics neural network (D2NN) for knapsack packing problem. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 3, pages 2425–2428, 1993.
57. Thomas Asikis. Towards recommendations for value sensitive sustainable consumption. In *NeurIPS 2023 Workshop on Tackling Climate Change with Machine Learning: Blending New and Existing Knowledge Systems*, 2023.
58. Joel Dyer, Arnau Quera-Bofarull, Ayush Chopra, J. Doyne Farmer, Anisoara Calinescu, and Michael J. Wooldridge. Gradient-assisted calibration for financial agent-based models. In *4th ACM International Conference on AI in Finance, ICAIF 2023, Brooklyn, NY, USA, November 27-29, 2023*, pages 288–296. ACM, 2023.
59. Luis L Fonseca, Lucas Böttcher, Borna Mehrad, and Reinhard C Laubenbacher. Optimal control of agent-based models via surrogate modeling. *PLOS Computational Biology*, 21(1):e1012138, 2025.

60. Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis Zhang. AI in inventory management: The disruptive era of DRL and beyond. *Available at SSRN*, 2025.
61. Joren Gijsbrechts, Robert N Boute, Jan A Van Mieghem, and Dennis J Zhang. Can deep reinforcement learning improve inventory management? performance on lost sales, dual-sourcing, and multi-echelon problems. *Manufacturing & Service Operations Management*, 24(3):1349–1368, 2022.
62. Edward Barankin. A delivery-lag inventory model with an emergency provision. *Naval Research Logistics Quarterly*, 8:285–311, 1961.
63. Yasunari Fukuda. Optimal policies for the inventory problem with negotiable leadtime. *Management Science*, 10(4):690–708, 1964.
64. Linwei Xin and Jan A Van Mieghem. Dual-sourcing, dual-mode dynamic stochastic inventory models. In *Research Handbook on Inventory Management*, pages 165–190. Edward Elgar Publishing, Cheltenham, UK, 2023.
65. Kenneth J. Arrow, Theodore Harris, and Jacob Marschak. Optimal inventory policy. *Econometrica*, 19(3):250–272, 1951.
66. Herbert Scarf and Samuel Karlin. Inventory models of the Arrow-Harris-Marschak type with time lag. In Kenneth J. Arrow, Samuel Karlin, and Herbert E. Scarf, editors, *Studies in the Mathematical Theory of Inventory and Production*. Stanford University Press, Stanford, CA, 1958.
67. Scott C. Douglas and Jiutian Yu. Why RELU units sometimes die: Analysis of single-unit error backpropagation in neural networks. In Michael B. Matthews, editor, *52nd Asilomar Conference on Signals, Systems, and Computers, ACSSC 2018, Pacific Grove, CA, USA, October 28-31, 2018*, pages 864–868. IEEE, 2018.
68. Jonathan T Barron. Continuously differentiable exponential linear units. *arXiv preprint arXiv:1704.07483*, 2017.
69. Zhongsheng Hua, Yimin Yu, Wei Zhang, and Xiaoyan Xu. Structural properties of the optimal policy for dual-sourcing systems with general lead times. *IIE Transactions*, 47(8):841–850, 2015.
70. Jiankun Sun and Jan A Van Mieghem. Robust dual sourcing inventory management: Optimality of capped dual index policies and smoothing. *Manufacturing & Service Operations Management*, 21(4):912–931, 2019.
71. Qi Gong, Wei Kang, and I Michael Ross. A pseudospectral method for the optimal control of constrained feedback linearizable systems. *IEEE Transactions on Automatic Control*, 51(7):1115–1129, 2006.
72. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.
73. Charlie G Buffie, Irene Jarchum, Michele Equinda, Lauren Lipuma, Asia Gobourne, Agnes Viale, Carles Ubeda, Joao Xavier, and Eric G Pamer. Profound alterations of intestinal microbiota following a single dose of clindamycin results in sustained susceptibility to *Clostridium difficile*-induced colitis. *Infection and Immunity*, 80(1):62–73, 2012.
74. Richard R Stein, Vanni Bucci, Nora C Toussaint, Charlie G Buffie, Gunnar Rätsch, Eric G Pamer, Chris Sander, and Joao B Xavier. Ecological modeling from time-series inference: insight into dynamics and stability of intestinal microbiota. *PLOS Computational Biology*, 9(12):e1003388, 2013.
75. Eric W Jones and Jean M Carlson. *In silico* analysis of antibiotic-induced *Clostridium difficile* infection: Remediation techniques and biological adaptations. *PLOS Computational Biology*, 14(2):e1006001, 2018.
76. E. W. Jones, P. Shankin-Clarke, and J. M. Carlson. Navigation and control of outcomes in a generalized Lotka–Volterra model of the microbiome. In J. Kotas, editor, *Advances in Nonlinear Biological Systems: Modeling and Optimal Control*, volume 11 of *AIMS Series on Applied Mathematics*, pages 97–120. American Institute of Mathematical Sciences, Springfield, MO, USA, 2020.
77. Bernard Bonnard, Jérémy Rouot, and Cristiana J Silva. Geometric optimal control of the generalized lotka–volterra model of the intestinal microbiome. *Optimal Control Applications and Methods*, 45(2):544–574, 2024.
78. Lynne V McFarland, Gary W Elmer, and Christina M Surawicz. Breaking the cycle: treatment strategies for 163 cases of recurrentclostridium difficiledisease. *Official Journal of the American College of Gastroenterology—ACG*, 97(7):1769–1775, 2002.
79. Yoshiki Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In *International Symposium on Mathematical Problems in Theoretical Physics*, pages 420–422. Springer, 1975.
80. Seung-Yeal Ha, Hwa Kil Kim, and Sang Woo Ryoo. Emergence of phase-locked states for the Kuramoto model in a large coupling regime. *Communications in Mathematical Sciences*, 14(4):1073–1091, 2016.
81. Umberto Biccari and Enrique Zuazua. A stochastic approach to the synchronization of coupled oscillators. *Frontiers in Energy Research*, 8(115), 2020.
82. Florian Dörfler, Michael Chertkov, and Francesco Bullo. Synchronization in complex oscillator networks and smart grids. *Proceedings of the National Academy of Sciences USA*, 110(6):2005–2010, 2013.
83. Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Controllability of complex networks. *Nature*, 473(7346):167–173, 2011.
84. Marcin Paluch, Florian Bolli, Xiang Deng, Antonio Rios Navarro, Chang Gao, and Tobi Delbruck. Hardware neural control of CartPole and F1TENTH race car. *arXiv preprint arXiv:2407.08681*, 2024.
85. Jean Saint-Donat, Naveen Bhat, and Thomas J McAvoy. Neural net based model predictive control. *International Journal of Control*, 54(6):1453–1468, 1991.
86. Andreas Draeger, Sebastian Engell, and Horst Ranke. Model predictive control using neural networks. *IEEE Control Systems Magazine*, 15(5):61–66, 1995.
87. Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 146–155. PMLR, 2017.
88. Felix Bünning, Adrian Schalbetter, Ahmed Aboudonia, Mathias Hudoba de Badyn, Philipp Heer, and John Lygeros. Input convex neural networks for building MPC. In Ali Jadbabaie, John Lygeros, George J. Pappas, Pablo A. Parrilo, Benjamin Recht, Claire J. Tomlin, and Melanie N. Zeilinger, editors, *Proceedings of the 3rd Annual Conference on Learning for Dynamics and Control, L4DC 2021, 7-8 June 2021, Virtual*

Event, Switzerland, volume 144 of *Proceedings of Machine Learning Research*, pages 251–262. PMLR, 2021.
89. Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A*, 474(2219):20180335, 2018.
90. Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*, volume 29. Springer, New York, NY, USA, 2005.
91. Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.
92. Mouhcine Mendil, Luca Mossina, and David Vigouroux. PUNCC: a python library for predictive uncertainty calibration and conformalization. In Harris Papadopoulos, Khuong An Nguyen, Henrik Boström, and Lars Carlsson, editors, *Conformal and Probabilistic Prediction with Applications, 13-15 September 2023, Limassol, Cyprus*, volume 204 of *Proceedings of Machine Learning Research*, pages 582–601. PMLR, 2023.
93. Rina Foygel Barber, Emmanuel J Candes, Aaditya Ramdas, and Ryan J Tibshirani. Conformal prediction beyond exchangeability. *The Annals of Statistics*, 51(2):816–845, 2023.
94. Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive confidence machines for regression. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning: ECML 2002, 13th European Conference on Machine Learning, Helsinki, Finland, August 19-23, 2002, Proceedings*, volume 2430 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2002.
95. Lars Lindemann, Matthew Cleaveland, Gihyun Shim, and George J Pappas. Safe planning in dynamic environments using conformal prediction. *IEEE Robotics and Automation Letters*, 8(8):5116–5123, 2023.
96. Alberto Portela, Julio R Banga, and Marcos Matabuena. Conformal prediction for uncertainty quantification in dynamic biological systems. *PLOS Computational Biology*, 21(5):e1013098, 2025.
97. Vignesh Gopakumar, Ander Gray, Joel Oskarsson, Lorenzo Zanisi, Stanislas Pamela, Daniel Giles, Matt Kusner, and Marc Peter Deisenroth. Uncertainty quantification of surrogate models using conformal prediction. *arXiv preprint arXiv:2408.09881*, 2024.
98. Arvind Mohan, Ashesh Chattopadhyay, and Jonah Miller. What you see is not what you get: Neural partial differential equations and the illusion of learning. *arXiv preprint arXiv:2411.15101*, 2024.
99. Ashish S Nair, Shivam Barwey, Pinaki Pal, Jonathan F MacArt, Troy Arcomano, and Romit Maulik. Understanding latent timescales in neural ordinary differential equation models of advection-dominated dynamical systems. *Physica D: Nonlinear Phenomena*, 476:134650, 2025.
100. Keyan Miao, Liqun Zhao, Han Wang, Konstantinos Gatsis, and Antonis Papachristodoulou. Opt-ODENet: A neural ODE framework with differentiable QP layers for safe and stable control design (longer version). *arXiv preprint arXiv:2504.17139*, 2025.
101. Adam Knapp, Daniel A. Cruz, Borna Mehrad, and Reinhard C. Laubenbacher. Personalizing computational models to construct medical digital twins. *Journal of the Royal Society Interface*, 22(20250055), 2025.
102. Gaurav Arya, Moritz Schauer, Frank Schäfer, and Christopher Rackauckas. Automatic differentiation of programs with discrete randomness. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
103. Ayush Chopra, Alexander Rodríguez, Jayakumar Subramanian, Arnau Quera-Bofarull, Balaji Krishnamurthy, B. Aditya Prakash, and Ramesh Raskar. Differentiable agent-based epidemiology. In Noa Agmon, Bo An, Alessandro Ricci, and William Yeoh, editors, *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, pages 1848–1857. ACM, 2023.
104. Krishna Rijal and Pankaj Mehta. A differentiable Gillespie algorithm for simulating chemical kinetics, parameter estimation, and designing synthetic biological circuits. *ELife*, 14:RP103877, 2025.