Neural ordinary differential equation control of dynamics on graphs

Thomas Asikis[*]
Computational Social Science, ETH Zurich, 8092 Zurich, Switzerland

Lucas Böttcher[†]
Frankfurt School of Finance and Management, Adickesallee 32-34, 60322 Frankfurt am Main, Germany and UCLA, Los Angeles, California 90095, USA

Nino Antulov-Fantulin[‡]
Computational Social Science, ETH Zurich, 8092 Zurich, Switzerland

(Received 12 August 2021; accepted 22 December 2021; published 23 March 2022)

We study the ability of neural networks to calculate feedback control signals that steer trajectories of continuous-time nonlinear dynamical systems on graphs, which we represent with neural ordinary differential equations (neural ODEs). To do so, we present a neural ODE control (NODEC) framework and find that it can learn feedback control signals that drive graph dynamical systems toward desired target states. While we use loss functions that do not constrain the control energy, our results show, in accordance with related work that NODEC produces low energy control signals. Finally, we evaluate the performance and versatility of NODEC against well-known feedback controllers and deep reinforcement learning. We use NODEC to generate feedback controls for more than one thousand coupled, nonlinear ODEs that represent epidemic processes and coupled oscillators.

DOI: 10.1103/PhysRevResearch.4.013221

I. INTRODUCTION

Dynamical processes on complex networks are common tools to model a wide range of real-world phenomena, including opinion dynamics [1,2], epidemic spreading [3–5], synchronization [6,7], and financial distress propagation [8]. Continuous-time dynamics on complex networks can be described by different frameworks including Chapman–Kolmogorov [9], Fokker–Planck [10], stochastic differential [11], and ordinary differential [12–14] equations. The structure of many real-world systems is described by networks with certain common properties including small-world effects [15], heavy-tail degree distributions [16,17], community structure [18], and other features [19,20]. The control of dynamical processes on networks [21,22] is a challenging task with applications in engineering, biology, and the social sciences [23,24]. Optimal control signals can be calculated by solving boundary-value PMP problems [25–27] or computing solutions of the Hamilton–Jacobi–Bellman equation (HJB). Complementing the above approaches, we develop a neural ODE control (NODEC) framework that controls fully observable graph dynamical systems using neural ODEs [28]. Within this framework, feedback control signals are calculated by minimizing a loss function describing differences between the current state and the target state. We perform extensive numerical experiments on coupled high-dimensional and nonlinear dynamical systems to showcase the ability of NODEC to calculate effective control signals.

Mathematically, systems are "controllable" if they can be steered from any initial state $\boldsymbol{x}(t_0)$ to any desired state $\boldsymbol{x}^*$ in finite time T. For linear systems, an analytical condition for controllability of linear time-invariant (LTI) systems was derived by Kalman in the 1960s [29] and is known today as Kalman's rank criterion. In 1969, Popov, Belevitch, and Hautus [30] introduced another controllability test for LTI systems that relies on solutions of an eigenvalue problem of the state matrix. In the 1970s, Lin introduced the framework of structural controllability [31] as a generalization of prior definitions of controllability on graphs. More recently, different large-scale social, technical, and biological networks were analyzed from a network controllability perspective [21,32] building on the framework introduced by Lin [31]. Controlling a complex system becomes more challenging as the number of nodes that can receive a control signal (driver nodes) decreases. Furthermore, Ref. [33] addresses the important issue of quantifying the (control) energy that is needed to control LTI systems.

To solve general nonlinear optimal control problems with energy and driver node constraints, two main approaches are used: (i) Pontryagin's maximum principle (PMP) [25–27]

[*]asikist@ethz.ch
[†]l.boettcher@fs.de
[‡]anino@ethz.ch

and (ii) Bellman's (approximate) dynamic programming [34–38]. Pontryagin's maximum principle [25–27] is based on variational calculus. When applying PMP, the original infinite-dimensional control problem is transformed to a boundary-value problem in a Hamiltonian framework. The downside of this approach is that the resulting boundary-value problems are often very difficult to solve. An alternative to variational methods is provided by Bellman's dynamic programming, which relies on the HJB equation. Given a quadratic loss on the control input, the HJB equation can be transformed into a partial-differential equation (PDE) [35]. Dynamic programming and PMP are connected through the viscosity solutions of the aforementioned PDEs [34]. However, in most cases, the HJB equation is hard to solve [36] and does not admit smooth solutions [39]. Most reinforcement-learning-based controls [37] rely on optimizing the HJB equation and can be viewed as an approximation of the dynamic programming [38] approach.

In this article we follow an alternative approach that utilizes neural ODEs to solve feedback control (FC) problems. We describe and evaluate the ability of NODEC to efficiently control nonlinear continuous-time dynamical systems by calculating feedback control signals. In Sec. II, we discuss related work. Section III summarizes mathematical concepts that are relevant for controlling graph dynamical systems. In Sec. IV, we provide an overview of the basic features of NODEC and formulate conditions for its successful application to solve control problems. In Sec. V, we showcase the ability of NODEC to efficiently control different graph dynamical systems that are described by coupled ODEs. In particular, we use NODEC to calculate feedback controls that synchronize coupled oscillators and contain disease dynamics with a limited number of driver nodes. Interestingly, NODEC achieves low energy controls without sacrificing performance. Section VI concludes our paper.

II. RELATED WORK

Previous works used neural networks (NNs) in control applications [40], in particular, for parameter estimation of model predictive control [41,42]. Extensive applications of NNs are also found in the field of proportional-integral-derivative (PID) controllers [40], where the gain factors are calculated via NNs. Shallow NNs have been trained to interact with and control smaller-scale ODE systems [40], without using neural ODEs or deep architectures. Recently, deep NNs have demonstrated high performance in control tasks, and notably on related work on differentiable physics [43] that often use PMP. Deep reinforcement learning [44] models are also used to calculate control signals and rely on approximations of the HJB equation. Other gradient-based non-NN approaches rely on the usage of adjoint methods [45]. Such model approaches are based on PMP and variational calculus. There also exist different generic approaches to control network dynamics [46,47]. Time-dependent control with NODEC, where the NN is only a function of time t, is extensively studied in Ref. [48]. The current work focuses on feedback control methods where the input of an NN is the state vector $\boldsymbol{x}(t)$. We study nonlinear dynamical systems, where minimum energy (optimal) controls are not always known. In our work, we always choose state-of-the-art control solutions when available, such as feedback control [49] and deep reinforcement learning methods [50,51], so that we can compare NODEC performance with corresponding baselines. The main contributions of this work are: (i) an adaptive efficient feedback control approximation methodology with implicit energy regularization properties that relies on neural ODEs, (ii) detailed numerical experiments involving high-dimensional nonlinear dynamical systems with minimum driver node constraints, and (iii) an extensively tested codebase that can be easily applied to other nonlinear control applications.

III. FEEDBACK CONTROL OF GRAPH DYNAMICAL SYSTEMS

A graph $G(V, E)$ is an ordered pair, where V and E $\subseteq$ V $\times$ V are the corresponding sets of nodes and edges, respectively. We denote the number of nodes $|V|$ by N. Although, in network science [52], it is more common to refer to graphs as networks, in this paper we will use the term "graph" instead of "network" to avoid confusion with NNs. Throughout this paper, we study dynamical systems on graphs described by the adjacency-matrix $\boldsymbol{A}$, which has nonzero elements $\boldsymbol{A}_{i,j}$ if and only if nodes i and j are connected. We describe controlled graph dynamical systems by ODEs of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}_A(t, \boldsymbol{x}(t), \boldsymbol{u}(\boldsymbol{x}(t))), \tag{1}$$

where $\boldsymbol{x}(t) \in \mathbb{R}^N$ denotes the state vector and $\boldsymbol{u}(\boldsymbol{x}(t)) \in \mathbb{R}^M$ $(M \leqslant N)$ an external control signal applied to $M \leqslant N$ (driver) nodes. The adjacency matrix in the subscript of $\boldsymbol{f}$ denotes the graph coupled interactions in the ODE system. For the remainder of the article, we omit the subscript as all systems under evaluation are graph-coupled ODEs that have fixed adjacency matrices over time. We use Newton's dot notation for differentiation $\dot{\boldsymbol{x}}(t)$. The function $\boldsymbol{f}$ in Eq. (1) accounts for both (time-dependent) interactions between nodes and the influence of external control signals on the evolution of $\boldsymbol{x}(t)$. We assume that the system state $\boldsymbol{x}$ is fully observable. In control theory, the control signal $\boldsymbol{u}(\boldsymbol{x}(t))$ is often calculated via two approaches: (i) by using time as input [i.e., $\boldsymbol{u} = \boldsymbol{u}(t)$] [33] or (ii) by using the system's state at time t as input (i.e., $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x}(t))$) [53]. The latter approach is often used in state-feedback control [53], where the control signal is calculated as a function of the difference between the state that the system reached at time t and the control target state $\boldsymbol{u}(\boldsymbol{x}(t) - \boldsymbol{x}^*)$. In the present article, we focus on state-feedback control and denote control signals by $\boldsymbol{u}(\boldsymbol{x}(t))$. The applicability of the current framework on time-dependent control $\boldsymbol{u} = \boldsymbol{u}(t)$ is evaluated in detail in Ref. [48].

For the majority of complex dynamical systems, Eq. (1) cannot be solved analytically. Instead, numerical solvers can be used to calculate approximate solutions of Eq. (1). As a starting point, one may use an explicit Euler method where for a given state $\boldsymbol{x}(t)$ at time t, the state of the system at time $t + \Delta t$ is calculated according to $\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \Delta t\, \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(\boldsymbol{x}(t)))$. Apart from an Euler forward integration scheme, there exist many more numerical methods [54] to solve Eq. (1). We use the expression ODESolve$(\boldsymbol{x}(t), t_0, T, \boldsymbol{f}, \boldsymbol{u}(\boldsymbol{x}(t)))$ to indicate a generic ODE solver that uses the right-hand side of Eq. (1) as an input and computes the state trajectory, or set of state vectors, $\mathrm{X}_{t_0}^T = \{\boldsymbol{x}(t)\}_{t_0 \leqslant t \leqslant T}$. In Sec. V, we employ Dormand–Prince and Runge–Kutta schemes as our ODESolve methods.

A. Driver node selection

Our aim is to showcase the ability of NODEC to produce efficient feedback controls for systems where the number driver nodes approaches the minimum number necessary to achieve control. Thus, we need to identify driver nodes that are able to fully control the underlying dynamics. Usually, we are interested in finding the minimum set of driver nodes, which is equivalent to the graph theoretical problems of maximum matching or minimum edge dominating sets [55,56]. However, for general graphs, finding the maximum matching set is NP-hard [57,58]. In our NODEC framework, we determine driver nodes according to two methods: (i) the maximum matching method [21] for disease dynamics and (ii) from stability criteria in the case of Kuramoto oscillators [59]. We denote the set of driver nodes and its cardinality by $\mathrm{B} \subseteq \mathrm{V}$ and M, respectively. Furthermore, we use $\boldsymbol{B} \in \mathbb{R}^{N\times M}$ to denote the driver matrix. The elements $\boldsymbol{B}_{i,m}$ are equal to 1 if node i is a driver node and if the control signal u_m is applied on node i. Otherwise, the driver-matrix elements $\boldsymbol{B}_{i,m}$ are equal to 0. The driver matrix $\boldsymbol{B}$ connects a control input $u_m(\mathbf{x}(t))$ associated with a driver node m to the corresponding graph node i for nonzero elements $\boldsymbol{B}_{i,m}$. Although NODEC can be used to evaluate shared and/or interacting control signals [60], in the current article we evaluate dynamical systems where each control signal $u_m(\boldsymbol{x}(t))$ is assigned to one and only one graph node i; thus, only one matrix element $\boldsymbol{B}_{i,m}$ is nonzero per row $\boldsymbol{B}_i$. We acknowledge that there are multiple studies on driver node placement on graphs, yet there is considerably fewer work that addresses ways of efficiently finding control inputs for high-dimensional dynamical systems with a limited number of driver nodes.

B. Control energy constraints

In complex systems, it may not always be possible to apply any control signal to a driver node. Consider a disease that spreads between networked communities and a control signal that denotes the intensity of quarantine. Applying a constant control signal with high values indicating blanket lockdown measures may not be acceptable by society. In the given example, our goal would be to contain disease spread as much as possible, while applying appropriate control signals to the selected driver nodes. A widely use metric for the intensity of the control signal [21] is the control energy

$$E(\boldsymbol{u}(\boldsymbol{x}(t))) = \int_{t_0}^{T} \|\boldsymbol{u}(\boldsymbol{x}(t))\|_2^2 \, dt, \tag{2}$$

where $\|\cdot\|_2$ denotes the L2 norm. In our numerical experiments we approximate over Ξ timesteps Eq. (2) by

$$E(T) \approx \sum_{\xi=1}^{\Xi} \|\boldsymbol{u}(\boldsymbol{x}(t_0 + \xi\,\Delta t))\|_2^2 \,\Delta t. \tag{3}$$

In Ref. [48], we show that NODEC may approximate optimal (or minimum energy) control signals without the necessity of explicitly accounting for an integrated energy cost in the underlying loss function. Instead, NODEC appears to implicitly minimizes the control energy via the interplay of an induced gradient descent, neural ODE solver dynamics, and NN initialization. Avoiding the control energy term in a constrained optimization also reduces computational cost of learning compared to solving boundary-value PMP problems [25–27], or computing solutions of the Hamilton–Jacobi–Bellman (HJB) equation [34–38]. In the present article, we provide evidence that NODEC achieves lower energy and higher performance when compared to different FC baselines for large complex systems.

IV. NEURAL ODE CONTROL

As in Sec. III, we consider a dynamical system Eq. (1) with initial state $\boldsymbol{x}(t_0)$, final reached state $\boldsymbol{x}(T)$, and target state $\boldsymbol{x}^*$. The goal of NODEC is to minimize differences between $\boldsymbol{x}(T)$ and $\boldsymbol{x}^*$ using control inputs $\hat{\boldsymbol{u}}(\boldsymbol{x}(t), \boldsymbol{w})$, where the vector $\boldsymbol{w}$ represents the weights of an underlying NN. We quantify differences between reached and target states with the control loss function $J(\mathrm{X}_{t_0}^T, \boldsymbol{x}^*)$ over the state trajectory $\mathrm{X}_{t_0}^T$. The general NODEC procedure is thus based on finding weights $\boldsymbol{w}$ that minimize a loss function $J(\mathrm{X}_{t_0}^T, \boldsymbol{x}^*)$ under the constraint Eq. (1), using a gradient descent update over a certain number of epochs. That is,

$$\begin{aligned} &\min_{w} J\left(\mathrm{X}_{t_0}^T, \boldsymbol{x}^*; \boldsymbol{w}\right) \\ &\text{s.t.} \quad \dot{\boldsymbol{x}}(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(\boldsymbol{x}(t))), \end{aligned} \tag{4}$$

where the control signal $\boldsymbol{u}(\boldsymbol{x}(t)) = \hat{\boldsymbol{u}}(\boldsymbol{x}(t); \boldsymbol{w})$ is calculated as an NN output and

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \Delta\boldsymbol{w} \quad \text{with} \quad \Delta\boldsymbol{w} = -\eta \nabla_{\boldsymbol{w}} J\left(\mathrm{X}_{t_0}^T, \boldsymbol{x}^*; \boldsymbol{w}\right). \tag{5}$$

Here, $\eta > 0$ denotes the learning rate. Our proposed method relies on the usage of neural ODEs [28], which are a natural choice for the approximation of continuous-time control signals. Using neural ODEs instead of discrete-time controls allows us to approximate a continuous-time interaction and express the control function $\hat{\boldsymbol{u}}(\boldsymbol{x}(t); \boldsymbol{w})$ as a parameterized NN within an ODE solver (see Fig. 1).

We show that NODEC can be used to control nonlinear graph dynamical systems with different loss functions. It is of particular relevance for continuous-time control problems with unknown and intractable optimal control functions. The ability of NODEC to approximate various control functions is established by universal approximation theorems for the approximation of continuous-time control functions with NNs. Similar to reinforcement learning (RL), NODEC is able to learn control inputs directly from the underlying dynamics in an interactive manner. Contrary to other control approaches [25–27,33,35], we do not impose a control energy constraint directly on our optimization loss function, improving the learning efficiency considerably [61].

In Algorithms 1 and 2, we show the two parts of a generic NODEC algorithm that approximates control signals. The main elements of NODEC are: (i) input and target states, (ii) graph coupled dynamics, (iii) NN architecture and initialization, (iv) the parameters of the ODE solver (e.g., step-size),

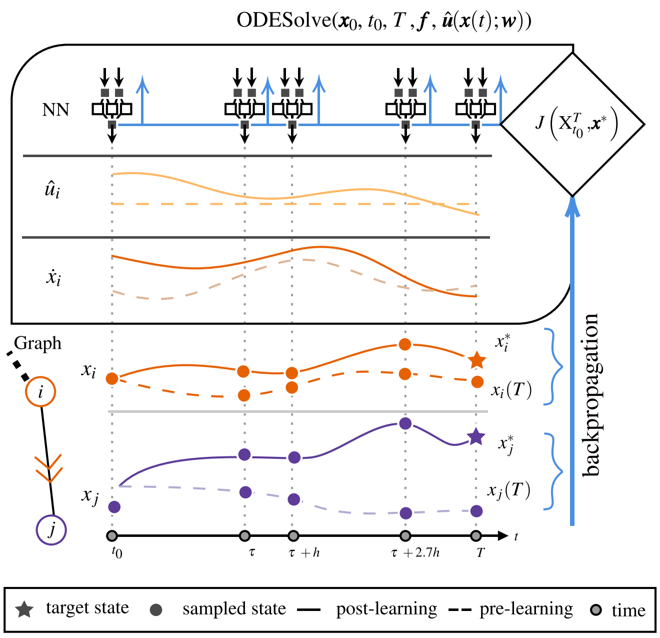

FIG. 1. A schematic that summarizes the training process of NODEC. An NN learns the control within the ODESolve method.

Algorithm 2: A simple ODESolve implementation.

```
Function ODESolve($x_0$, $t_0$, $T$, $f$, $\hat{u}(x(t);w)$):
    // Euler Method
    $t \leftarrow t_0$;
    // State trajectory:a set of state vectors.
    Set $X_{t_0}^T \leftarrow \{x_0\}$;
    while $t \leq T$ do
        // Computational graph is
        // preserved through time
        // gradients flow through $x$
        $\hat{u} \leftarrow \hat{u}(x(t);w)$;
        $x \leftarrow x + \tau f(t, x, \hat{u})$;
        $X_{t_0}^T \leftarrow X_{t_0}^T \cup \{x\}$
        // Step $\tau$ could be adaptive
        $t \leftarrow t + \tau$;
    end
    return $X_{t_0}^T$;
end
```

and (v) the gradient descent algorithm and its hyperparameters, such as the learning rate. Note that Algorithm 2 relies on automatic differentiation methods [62,63], where the gradients "flow" through the underlying NN that is time-unfolded by ODE solvers [54].

A. Neural ODE and NODEC learning settings

Although NODEC utilizes neural ODEs [28], the learning tasks of both frameworks differ significantly. Neural ODEs model dynamics of the hidden state $\boldsymbol{h}(t)$ of an NN according to

$$\dot{\boldsymbol{h}}(t) = \boldsymbol{g}(t, \boldsymbol{h}(t); \boldsymbol{w}), \tag{6}$$

where $\boldsymbol{g}(\boldsymbol{h}(t), \boldsymbol{t}; \boldsymbol{w})$ and $\dot{\boldsymbol{h}}(t)$ represent the NN and hidden-state evolution, respectively. As in Eq. (4), the vector $\boldsymbol{w}$ denotes the neural-network weights. Previously, neural ODEs were mainly applied in supervised learning tasks [64] and in normalizing flows [28]. For NODEC, we use an NN as a parameterized function to approximate the control term $\boldsymbol{u}(\boldsymbol{x}(t))$ in graph dynamical systems (1). Contrary to supervised applications of neural ODEs [28], our proposed framework numerically solves control problems in an interactive manner, similar to reinforcement learning.

Algorithm 1: A generic algorithm that describes the parameter learning of NODEC.

```
Result: $w$
Init:: $t_0$,$x_0$,$w$, $f(\cdot)$, ODESolve$(\cdot)$,$J(\cdot)$,$x^*$;
Params:: $\eta$, epochs;
epoch $\leftarrow$ 0;
while epoch < epochs do
    // Generate a trajectory based on NODEC.
    $X_{t_0}^T \leftarrow$ ODESolve($x_0$,$t_0$,$T$,$f$,$\hat{u}(x(t);w)$);
    // gradient descent update
    $w \leftarrow w - \eta \nabla_w J(X_{t_0}^T, x^*)$;
    // or Quasi-Newton with Hessian:
    // $w \leftarrow w - \eta H^{-1} \nabla_w J(X_{t_0}^T, x^*)$
end
```

B. Learnability of control with neural networks

As reachability of a target state $\boldsymbol{x}^*$ from an initial state $\boldsymbol{x}(t_0)$ implies the existence of a control function $\boldsymbol{u}(\boldsymbol{x}(t))$, we now focus on the ability to approximate (i.e., learn) $\boldsymbol{u}(\boldsymbol{x}(t))$ for reachable target states with an NN.

Given that (i) a target state $\boldsymbol{x}^*$ is reachable with continuous-time dynamics (1) and (ii) the control function $\boldsymbol{u}(\boldsymbol{x}(t))$ that reaches the target state $\boldsymbol{x}^*$ is continuous or Lebesque integrable in its domain, then a corresponding universal approximation (UA) theorem applies for an NN that can approximate a control function $\hat{\boldsymbol{u}}(\boldsymbol{x}(t); \boldsymbol{w}) \to \boldsymbol{u}(\boldsymbol{x}(t))$ by learning parameters $\boldsymbol{w}$.

The above proposition holds when both an appropriate UA theorem [65–69] and reachability [70] requirements are satisfied by the underlying dynamics and the NN controller. Related work indicates UA properties for neural ODEs [71] that can be leveraged to calculate approximate solutions to the control problem Eq. (4). The ability of an NN to learn control signals has also been covered in the literature outside of the domain of neural ODEs [41,72–74]. In the current work, we choose to compare our proposed model to an analytical feedback control baseline [59] and state-of-the-art reinforcement learning [51] for nonlinear dynamical systems describing Kuramoto oscillators and disease spread.

C. Learning loss and control goals

To apply NODEC to control tasks, we have to translate a control goal into an adequate learning loss. The choice of the control goal depends on the underlying dynamics, graph structure, and objectives of the control designer. A very common goal in literature [75] is "microscopic" control where each node i has to reach a predetermined state value within time T, i.e., $x_i(T) = x_i^*$. Such a control goal is often applied in industrial applications and may be used to steer electric and mechanical systems toward desired target states [75]. This control goal is achieved by minimizing a metric that quantifies the distance between the target and reached states $\boldsymbol{x}(T)$ and $\boldsymbol{x}^*$. One possible choice of such a metric is the mean squared error (MSE) $J(\boldsymbol{x}(T), \boldsymbol{x}^*) = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i(T) - \boldsymbol{x}_i{}^*)^2$. When the MSE is used, corresponding optimal control problems may be expressed as convex optimization problems [76]. For more details on the application of NODEC to microscopic loss functions, see Ref. [48].

We focus on control goals that do not require a specific target state value for each node, but instead require that constraints over aggregate values or statistical properties of the system's states are satisfied. For the control of certain complex systems, it is useful to consider such "macroscopic" constraints [23,77]. Often such goals lack exact optimal control solutions, thus offering many opportunities for novel control applications of NODEC.

A common macroscopic control goal is that nodes in the target state are required to be synchronized, i.e., the nodes' states are required to have the same value or constant phase shifts. Such synchronization conditions are often considered in the context of controlling oscillator systems [45,78]. When synchronizing oscillators, reaching the target state at time T may not satisfy the control goal completely, as we may require the system to preserve the state properties that satisfy the synchronization goal for a time period $[t, T]$. In the current work we showcase that NODEC is able to optimize such control problems.

We also consider more complex control goals, where the system evolution includes coupled ODEs with more than one state variable. In the context of epidemic models, the state x_i of a node i is represented by a vector that consists of multiple state variables. For susceptible-infected-recovered (SIR) models, three state variables, $S_i(t)$, $I_i(t)$, and $R_i(t)$, are used to model the part of a population on node i at time t that is susceptible, infected, and recovered, respectively. A relevant control goal for controlling epidemics is the "flattening" of the curve, or reducing the maximum infected population that occurs at time $t^* \in [t_0, T]$.

V. EXPERIMENTAL EVALUATION

In this section, we evaluate the ability of NODEC to (i) reach target states efficiently with a limited number of driver nodes, (ii) control different dynamics and losses, and (iii) calculate low energy control signals. We first evaluate the performance of NODEC for two nonlinear systems with very different control tasks to showcase its applicability and versatility in computationally challenging settings for which analytical solutions or approximate control schemes may not exist. We describe the experimental setup by defining the dynamical systems, initial state, control goal, and NN hyperparameters used for training. The choice of NN hyperparameters focuses mainly on the network architecture, inputs, optimizers, and training procedures. For the sake of brevity, we omit technical details in the main text and provide further information in the Appendix and in our code [79,80] and data repositories [81].

A. Coupled oscillator dynamics

Here we study the ability of NODEC to control a network of coupled oscillators via feedback control. Such systems are used to model power grids and brain networks [82,83]. One common control goal for oscillator systems is to reach a fully synchronized target state and stabilize the system over time. This introduces two main challenges: (i) a target state that satisfies this goal needs to be reached and preserved and (ii) the trained model needs to be able to achieve synchronization stability for initial states not seen in training. For continuous-time linear time invariant systems and systems that can be linearized, there exist optimal feedback control methods [84]. Continuous-time oscillatory dynamics may not always be linearizable [59] and exhibit chaotic behavior [85,86], which cannot be observed in (finite-dimensional) LTI systems. NODEC does not require linearization and could potentially control systems that are costly or intractable to linearize.

In a graph of N coupled oscillators, a possible mathematical description of the evolution of phase x_i of oscillator i with natural frequency ω_i is

$$\dot{x}_i = \omega_i + \sum_m \boldsymbol{B}_{i,m} u_m(\boldsymbol{x}(t)) + K \sum_j \boldsymbol{A}_{i,j} \hbar(x_j - x_i), \quad (7)$$

where $\boldsymbol{A}$ is the interaction matrix, K the coupling constant, and $\hbar$ a 2π-periodic function [59]. By setting $\hbar(\boldsymbol{x}) = \sin(\boldsymbol{x})$, we recover Kuramoto dynamics [87]

$$\dot{x}_i = \omega_i + \sum_m \boldsymbol{B}_{i,m} u_m(\boldsymbol{x}(t)) + K \sum_j \boldsymbol{A}_{i,j} \sin(x_j - x_i). \quad (8)$$

In accordance with Ref. [59], we consider a target state in which all oscillators are tightly packed in a synchronized cluster where $|x_j - x_i| \approx 0$ for all i, j. Linearization of the uncontrolled version of Eq. (8) within the rotating reference frame yields the target state

$$\boldsymbol{x}^\diamond = K^{-1}\boldsymbol{L}^\dagger \boldsymbol{\omega}, \quad (9)$$

where $\boldsymbol{L}^\dagger$ is the pseudoinverse of the graph Laplacian $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ and $\omega = [\omega_1, \dots, \omega_N]$ is the vector of natural frequencies [59]. For the derivation of Eq. (9), one uses that the mean natural frequency is zero. The quantity $\boldsymbol{D}$ is the degree diagonal matrix, with each element of the diagonal $\boldsymbol{D}_{i,i} = d_i$ being equal to the degree d_i of the corresponding node i.

1. Learning loss function

We measure the degree of synchronization of Kuramoto oscillators in terms of the order parameter [78]

$$r(t) = \frac{1}{N}\sqrt{\sum_{i,j} \cos\left(x_i(t) - x_j(t)\right)} = \frac{1}{N}\sqrt{\sum_{i,j} e^{\iota(x_i - x_j)}}. \quad (10)$$

We denote with ι the imaginary unit.

The control loss may also aggregate the order parameter over time, when the control goals take stability into account.

In such a case, one might consider the mean order parameter over time,

$$\bar{r}(t) = \frac{1}{T}\int_0^T r(t)\,dt, \tag{11}$$

which approaches zero if the oscillators are incoherent. By discretizing T into Ξ intervals, we can also discretize Eq. (11) using

$$\bar{r}(t) = \frac{1}{\Xi}\sum_{\xi=0}^{\Xi} r(\xi\tau)\tau, \quad \Xi\tau = T. \tag{12}$$

For the numerical calculations we omit τ. Equation (12) can be used as a loss function,

$$J\big(\mathrm{X}_\tau^T\big) = -\bar{r}(t) = -\frac{1}{\Xi}\sum_{\xi=1}^{\Xi} r(\xi\tau), \tag{13}$$

to achieve stable synchronization of coupled oscillators. Loss functions that represent periodically and randomly changing oscillator states can be also used as an input for learning control inputs. Instead of using loss function (13) that is associated with maximizing the degree of synchronization in the whole network, one may wish to achieve different degrees of synchronization in different time windows, which can be modeled by using time-dependent synchronization losses. Furthermore, forcing parts of a network into different oscillatory states may be possible by minimizing differences between observed and desired oscillator phases in certain subgraphs.

The loss function (13) introduces two challenges with respect to the classical MSE loss [76]: (i) it is a macroscopic loss as we do not require to reach a specific state vector $\boldsymbol{x}^*$ to minimize [88] Eq. (13) and (ii) the loss is calculated over a time interval $[\tau, T]$ [89]. The initial time is omitted [$\xi = \{1, \ldots, \Xi\}$ in Eq. (13)], since we assume that no control is applied prior to reaching the initial state. In our numerical experiments we observed that using such a loss affects numerical stability, especially for long time intervals, e.g., when $\Xi = 100$. Averaging over $r(\xi t)$ may smooth out temporal variations of $r(t)$, especially for very high values of Ξ. When such drops occur in sampled training trajectories, NODEC learns to achieve high synchronicity only temporarily. NODEC learns controls that yield highly synchronized stable trajectories similar to the FC baselines, when we extend Eq. (13) by subtracting the minimum order parameter value $\min_{t\in[\tau,T]} r(t)$ over time:

$$J\big(\mathrm{X}_\tau^T\big) = -\Big(\bar{r}(t) + \min_{t\in[\tau,T]} r(t)\Big). \tag{14}$$

Introducing the minimum order parameter term increases the stability of the learned control as the loss creates higher gradients for controls that cause loss of synchronization. We train NODEC on trajectories that may at maximum reach a total time of $T = 40$, and we test and evaluate the performance of the trained neural network on random initial states for $T = 150$. As we report in Sec. V A 5, the described training protocol is able to achieve good performance for these testing parameters.

2. *Control baselines*

A FC baseline for Kuramoto dynamics is presented in Ref. [59]. First, the feedback control gain vector $\boldsymbol{b}^{(\mathrm{FC})}$ is defined for the control baseline. In accordance with Ref. [59], we use a control gain vector $\boldsymbol{b}^{(\mathrm{FC})}$ instead of a driver matrix. An element $\boldsymbol{b}_i^{(\mathrm{FC})}$ of the gain control vector is assigned to a graph node i and it needs to satisfy

$$b_i^{(\mathrm{FC})} \geqslant \sum_{j\neq i}\{|K\boldsymbol{A}_{i,j}\cos(x_i^\diamond - x_j^\diamond) - \epsilon| \\ - [K\boldsymbol{A}_{i,j}\cos(x_j^\diamond - x_i^\diamond) - \epsilon]\}. \tag{15}$$

If $\cos(x_i^\diamond - x_j^\diamond) \geqslant 0$, the corresponding term in the summation vanishes. We take the equality of the constraint in Eq. (15) to calculate the control gain vector elements $b_i^{(\mathrm{FC})}$ based on Ref. [59]. The error margin buffer ϵ is implemented as suggested in related work [59] by setting $\epsilon \geqslant 0$ when selecting driver nodes in Eq. (15). For $\epsilon = 0$, the driver node selection might be insufficient and it may not be possible to drive the system to a desired target state [59]. Using an error margin buffer increases the driver node selection tolerance and, thus, selects more diver nodes, which can steer the system to a desired target state. The nonzero values of the driver matrix can be chosen arbitrarily, as long as the constraint in Eq. (15) is satisfied. Nonzero values $b_i^{(\mathrm{FC})}$ determine the driver nodes. The baseline control signal u_i for a node i is calculated as

$$u_i(x_i(t)) = \zeta b_i^{(\mathrm{FC})}\sin(x_i{}^* - x_i(t)). \tag{16}$$

In driver matrix notation, we iterate over all nodes and select a node i as the mth driver node by setting the driver matrix element $\boldsymbol{B}_{i,m} = b_i^{(\mathrm{FC})}$ if $b_i^{(\mathrm{FC})} \neq 0$. We set the target state in Eq. (16) to $x_i{}^* = 0$.

We require that feedback control reaches comparable performance to NODEC in terms of $r(t)$; thus we multiply the vector $\boldsymbol{b}^{(\mathrm{FC})}$ with a positive scalar value [90] $\zeta = 10$. Higher absolute values of $\zeta|b_i^{(\mathrm{FC})}|$ may create control signals that reach the target state in less time at the expense of a higher control energy. As the driver matrix is calculated based on an approximation of the graph Laplacian pseudoinverse $\boldsymbol{L}^\dagger$ of a singular system, optimal control guarantees for minimum energy may not always hold.

3. *Numerical simulation parameters*

To evaluate the system synchronicity, we calculate the order parameter [see Eq. (10)], which reaches the maximum value $r(t) = 1$ if all oscillators are fully synchronized.

For our numerical experiments, we create an Erdős–Rényi graph $G(N, p)$ with $N = 1024$ nodes, expected mean degree $\bar{d} = 6$, and link probability $p = \bar{d}/(N-1)$. We generate the driver matrix as in Sec. V A 2, and select the nonzero elements as driver nodes. To reduce approximation errors due calculation of the Laplacian pseudoinverse matrix, we set a buffer margin of $\epsilon = 0.1$ in Eq. (15), when selecting driver nodes. Control signal energy is evaluated with Eq. (3). Moreover, we set the coupling constant to $K = 0.4$ and sample the natural frequencies ω_i from a uniform distribution $\mathcal{U}(-\sqrt{3}, \sqrt{3})$ [49]. This setting results in approximately 70% of the nodes being assigned as driver nodes.

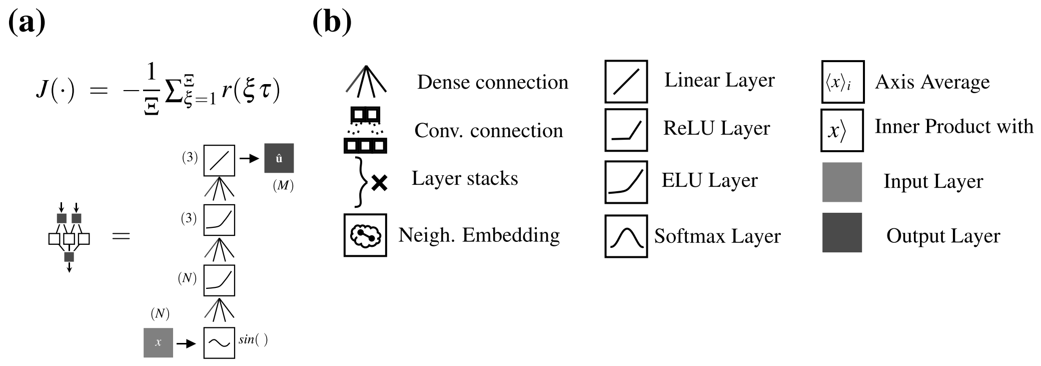

FIG. 2. (a) NN architecture for controlling Kuramoto oscillators and (b) symbol legend.

4. NODEC hyperparameters

Only the current system state $\boldsymbol{x}(t)$ is provided as an input for the NN, similar to the baseline described in Sec. V A 2. We use a fully connected architecture as illustrated in Fig. 2(a). Finally, to calculate the binary driver matrix $\boldsymbol{B}$ for the NN in Eq. (8), we iterate all nodes of G, and for each node i we assign $\boldsymbol{B}_{i,m} = 1$ and set it as the m-th driver node if $b_i^{(\mathrm{FC})} \neq 0$ and $\boldsymbol{B}_{i,m} = 0$ otherwise. The driver index m increments by one each time a driver node is assigned. We use a binary driver matrix instead of the gain matrix used in FC, as we require the network to learn the control signals per driver node without prior knowledge of the exact control gains.

The current control goal is to stabilize Kuramoto oscillators in a synchronized state over a period of time. The loss of synchronization may occur at any point during this period. To avoid loss of synchronization over a period of time, we train NODEC (see Appendix Algorithm 3) in a curriculum learning procedure [91], where NODEC is initially trained on trajectories sampled for low values of T. The value of T increases gradually as training proceeds. The learning process in the beginning of the curriculum, when T is very low, allows NODEC to learn controls that steer the oscillators through the transient state between synchronicity and non-synchronicity. As T increases, the network also learns controls that preserve the network in the synchronized state.

In feedback control, the target is often to synchronize the system for different initial states [92]. To train the system for more than one initial state, we use a mini-batch-training procedure that samples 8 random initial states per epoch for training. We observed that randomly sampling an initial state from a uniform distribution in $[0, 2\pi]$ does not improve training performance and fails to learn synchronization. It has been reported in the literature [93] that normally distributed layer inputs (with zero mean and unit variance) can help NNs converge faster. Therefore, we decided to sample initial states from a normal distribution with zero mean and unit variance. Our results confirm that learning and convergence improve. Sampling initial states enables us to use mini-batches to speed up and stabilize training as well. In the Kuramoto example we use the Adam optimizer [94] for parameter optimization. The complete training scheme is also illustrated in Appendix Algorithm 3.

5. Results

To test the control performance of NODEC, we first sample an unobserved initial state close to the synchronized steady state in accordance with Ref. [59]. The initial state values for single sample evaluation [see Figs. 3(a) and 3(b)] are uniformly sampled within -10% of the synchronized steady state values, i.e., $x_i \in [0.9x_i^\diamond, x_i^\diamond]$, to be close to the synchronized steady state as proposed in Ref. [59]. We observe that the NN achieves a target state with larger order parameter values [see Fig. 3(b)] and requires lower energy [see Fig. 3(a)] than the FC baseline. We also observe that NODEC requires higher energy and slightly more time to synchronize the system but less to preserve it, compared to the FC baseline [see Fig. 3(b) and Appendix Fig. 7] .

To determine whether NODEC can achieve synchronization stability regardless of the initial state choice [see Fig. 3(c)] and its proximity to the synchronized steady state, we test the trained model on 100 initial states, with values uniformly sampled in $[0, 1]$. In Fig. 3(c) the vertical axis represents the relative total energy difference between NODEC and FC for the same initialization $(E_{\mathrm{NODEC}}(T) - E_{\mathrm{FC}}(T))/E_{\mathrm{FC}}(T)$. The horizontal axis represents the mean relative order parameter difference calculated as $[r_{\mathrm{NODEC}}(T) - r_{\mathrm{FC}}(T)]/r_{\mathrm{FC}}(T)$. NODEC achieves around 1% higher order parameter values and almost 86% less total control energy for all samples. More sophisticated strategies of adapting the constant term ζ in Eq. (15) could be applied to adapt the driver matrix values in feedback control. This is, however, outside the scope of this paper. Our results show that NODEC can be adapted to achieve highly synchronized states in Kuramoto dynamics on an Erdős–Rényi graph via feedback control.

B. Epidemic spreading and targeted interventions

Designing targeted intervention and immunization strategies [5,95] is important to contain the spread of epidemics. To study the performance of NODEC in such containment tasks, we will use the susceptible-infected-recovered-type (SIR-type) model [96] that extends the classical SIR model by accounting for quarantine interventions and other preventive or reactive measures for disease containment. In our formulation of SIR-type dynamics, we also account for control inputs and network structure. The “R” compartment in our model is used to describe (i) recovered individuals that were infected and acquired immunity and (ii) removed individuals (i.e.,

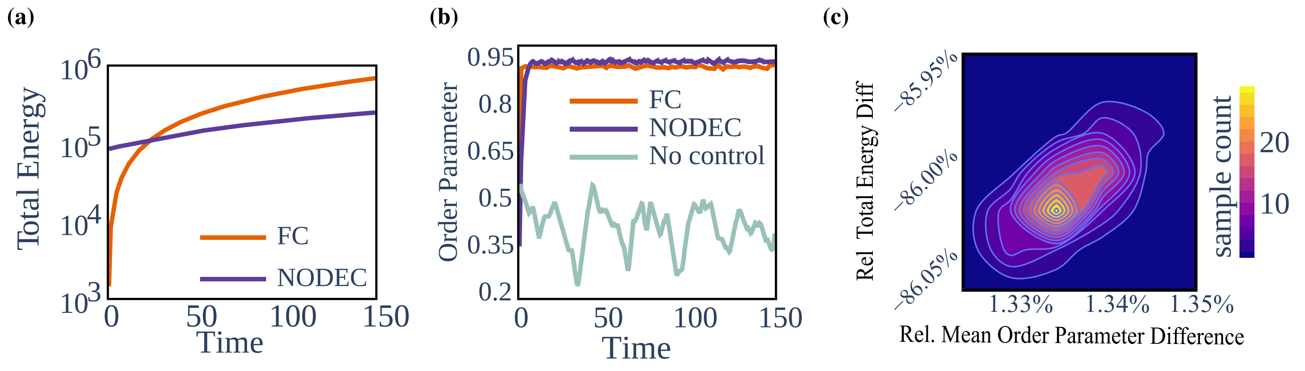

FIG. 3. Comparison of NODEC and FC in terms of (a) energy, (b) synchronization stability, and (c) relative performance for randomly sampled unobserved initial states.

susceptible individuals under quarantine who do not interact with anyone else). The complete state of the epidemic model is now described by a matrix $\boldsymbol{X}(t) \in \mathbb{R}^{4\times N}$, where each row represents the state vector of the corresponding node [97]. The proportion of susceptible, infected, recovered, and quarantined individuals at node i is $\boldsymbol{X}_{1,i} = S_i$, $\boldsymbol{X}_{2,i} = I_i$, $\boldsymbol{X}_{3,i} = R_i$, $\boldsymbol{X}_{4,i} = Y_i$, respectively. The corresponding generalized SIR-type dynamics of node i is described by a set of rate equations:

$$\dot{S}_i(t) = -\beta S_i(t)\sum_j \boldsymbol{A}_{i,j} I_j(t) - \sum_m \boldsymbol{B}_{i,m} u_m(\boldsymbol{X}(t)) S_i(t), \tag{17a}$$

$$\dot{I}_i(t) = \beta S_i(t)\sum_j \boldsymbol{A}_{i,j} I_j(t) - \gamma I_i(t) - \sum_m \boldsymbol{B}_{i,m} u_m(\boldsymbol{X}(t)) I_i(t), \tag{17b}$$

$$\dot{R}_i(t) = \gamma I_i(t) + \sum_m \boldsymbol{B}_{i,m} u_m(\boldsymbol{X}(t)) S_i(t), \tag{17c}$$

$$\dot{Y}_i(t) = \sum_m \boldsymbol{B}_{i,m} u_m(\boldsymbol{X}(t)) I_i(t), \tag{17d}$$

subject to the conditions that (i) the total population is conserved and (ii) the control budget is bounded from above by $\mathcal{b}$:

$$\sum_i (S_i + I_i + R_i + Y_i) = N, \tag{18a}$$

$$\sum_{m,i} \boldsymbol{B}_{i,m} u_m(\boldsymbol{X}(t)) \leqslant \mathcal{b}. \tag{18b}$$

The driver nodes $B_{i,m} = 1$ can be selected via different methods, e.g., the nodes/communities that are willing to apply proactive and reactive measures. In our simulations, driver nodes are selected with the maximum matching method [55,56]. Furthermore, we assume that the epidemic originates from a localized part in the graph and we minimize the proposed epidemic loss in Eq. (19) for a different part of the graph. The parameters β and γ are the infection and recovery rates, and $u_m(\boldsymbol{X}(t))$ describes the effect of containment interventions (e.g., quarantine, mask-usage and distancing). When an NN controller (NODEC or RL) is used, we set $u_m(\boldsymbol{X}(t)) = \hat{u}_m(\boldsymbol{X}(t))$. These terms are used to model preventive and reactive measures, respectively. For example, susceptible individuals may isolate themselves and completely avoid infection ($S \to R$) until the pandemic passes (preventive) or infected individuals are quarantined and put to intensive care to contain the spread and help infected patients recover ($I \to Y$) (reactive measure). The described control problem is complicated by several factors. First, the budget constraint [see Eq. (18b)] does not allow assignment of high control values across all nodes. Second, we need to distribute limited intervention resources dynamically across structurally similar nodes. Unlike in networks with community structure, where isolating single nodes can effectively control epidemic spreading, the regularity of the square lattice does not admit such a control approach.

Another possible formulation of time-dependent control targets in epidemic modeling is to weigh the number of new infections with a discount factor to prioritize minimizing current infections over minimizing future infections. For network epidemic models, degree-based approximations can be used in conjunction with optimal control theory to derive interventions for such loss functions. For more information, see, e.g., Ref. [98].

1. Learning loss function

The control goal is to "flatten" the curve, i.e., to delay and minimize the mean infected fraction over nodes in the subgraph G^*, which has no overlap with the part of the graph containing the initial spreading seed. Based on these control goals, we formulate the following loss function:

$$J\big(\mathrm{X}_{t_0}^T, \boldsymbol{X}^*\big) = \Big(\max_{t_0 \leqslant t \leqslant T} \bar{I}_{G^*}(t)\Big)^2, \tag{19}$$

where $\bar{I}_{G^*}$ denotes the mean fraction of infected individuals in G^*. This control goal is macroscopic, as we do not know the exact feasible state $\boldsymbol{X}^*$ for which $I^*(t^*) = \mathrm{argmin}_{I(t)} J(I(t))$ that minimizes such a loss. Furthermore, the exact time t^* at which the minimum loss is achieved is not known, and therefore we need to evaluate samples from the state trajectory $\mathrm{X}_{t_0}^T$ to determine t^*. Similar to Eq. (14), the current control goal requires loss calculations over a time interval. Moreover, this loss is not calculated over the whole state matrix $\boldsymbol{X}$ but only on the infected state I_{G^*} of the target subgraph. Intuitively, one would trivially achieve the proposed goal if there are no further constraints. If nodes that connect the subgraph G^* to the rest of the graph cannot be controlled efficiently, then achieving the control goal becomes nontrivial. Tackling the outlined epidemic control problem allows us to evaluate

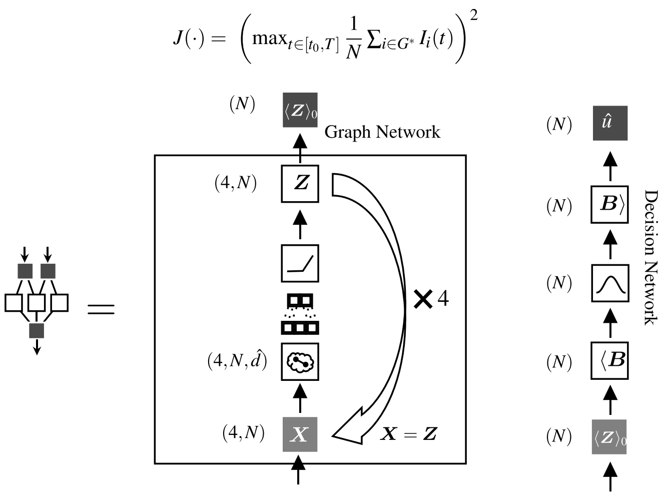

FIG. 4. NODEC architecture for controlling SIR-type dynamics.

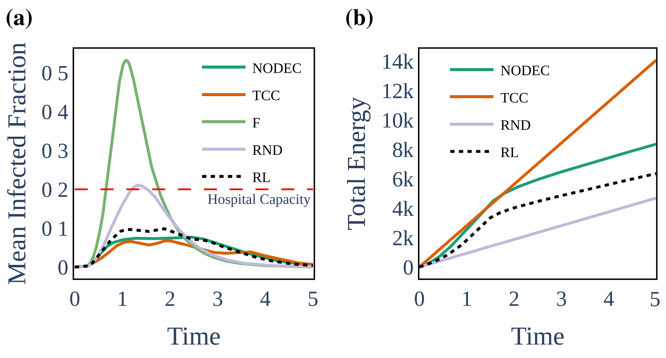

FIG. 5. SIR-type control evaluation in terms of (a) proportion of infected individuals and (b) total energy. NODEC versus baselines: reinforcement learning (RL), targeted constant control (TCC), random constant control (RND), and free dynamics with no control (F).

NODEC on a complex control task (see Sec. V B 5) with applications in disease control.

2. *Control baselines*

A baseline that takes structural node properties (e.g., node degree or centrality) into the account, may be a good baseline for graphs with structural heterogeneity, but not for regular structures like lattices. Clearly, a weak baseline is random control (RND), where we assign random control inputs to driver nodes with $u_m(t) = \mathcal{b} c_m / \sum_{m'=0}^{M} c_{m'}$, $c_m \sim \mathcal{U}(0,1)$. However, a targeted constant control baseline (TCC), which in the presence of an "oracle" assigns constant control inputs $u_m(t) = \mathcal{b}/M$ to every driver node in G^*, is a strong baseline for constant control. As TCC is a static control, it already protects the driver nodes from $t = 0$ on, so TCC-controlled nodes will be infected very slowly. Assigning all budget to all driver nodes of interest also minimizes wasted "containment" budget. Still, distributing more budget to a smaller number of nodes increases the L2 norm of the control, making controls very expensive when considering quadratic energy costs. To have a control with less energy, it is important to distribute the budget to more nodes, therefore enabling more global containment and less constant containment on the target subgraph.

We also study the performance of neural dynamic control baselines, such as continuous-action RL, with fully connected NNs and our variant (see Fig. 4) as policy architectures (see Sec. V B 4 and Appendix B 1). Only one of the three evaluated training routines of RL provided high-performance results. We tested: SAC [50], TD3 [51], and A2C [99], but we report only the results of TD3 which were more competitive with respect to NODEC. To allow RL to tackle the SIR-type control problem, we first implement SIR-type dynamics as an RL environment. The input of RL is the tensor of all SIR-type states at time t. We consider an observation space, which includes continuous values in [0,1] and has dimension $4 \times N$. RL actions $a_m(t) \in \mathbb{R}$ are continuous values for each driver node and correspond to control signals. Once the actions are passed to the environment, a preprocessing operation takes place to convert the RL action into valid control signals (see decision network of Fig. 4). RL is allowed to provide (change) the control signals to (interact with) the environment in a fixed discrete time interaction interval $\Delta t = 10^{-2}$ during training. Lower interaction intervals were also considered, but required longer training and did not seem to improve performance. For RL, we need to express the control goal as a reward function which is used for the approximation of action value function within the RL framework. We tested several reward designs and we describe this process in Appendix B 2, but we simulate best performance with the following reward function:

$$\rho(t) = \begin{cases} 0, & \text{if } \bar{I}_{G^*}(t) \leqslant \max_{\tau<t}(\bar{I}_{G^*}(\tau)), \\ -\bar{I}^2_{G^*}(t) + (\max_{\tau<t} \bar{I}_{G^*}(\tau))^2, & \text{otherwise.} \end{cases} \tag{20}$$

3. *Numerical simulation*

To determine the target time T, we observe the SIR-type dynamics ($\beta = 6$ and $\gamma = 1.8$) on a 32×32 lattice without control and set its value to the time at which the mean infection over all nodes is approximately zero. Initially, the epidemic starts from a deterministic selection of nodes in the upper-right quadrant. For all control strategies, the budget (maximal number of control interventions) is $\mathcal{b} = 600$. Given that RL takes considerably longer to converge and that we were required to perform a much more extensive hyper parameter search, we showcase our experiments only on the lattice graph and a single initial state. Our control goal is to contain epidemic outbreaks (i.e., "flattening" the infection curve) in the subgraph G^*, which is located in the bottom-left quadrant (see Fig. 6). All baselines are compared with timestep $\Delta t = 10^{-3}$.

4. *NODEC hyperparameters*

From a technical perspective, the SIR-type dynamics introduce extra state variables. Therefore, fully connected layers will require one to estimate considerably more parameters. We observe that neither NODEC nor RL converged to a high-performance solution when using fully connected layers, and we thus omit these results. Furthermore, the control task requires the network to optimize a loss that is not calculated over whole graph, but rather on a specific subgraph. NODEC has no direct information on which nodes are part of subgraph G^*. The information is provided via the minimization of the learning loss-function in Eq. (19). Back-propagation happens

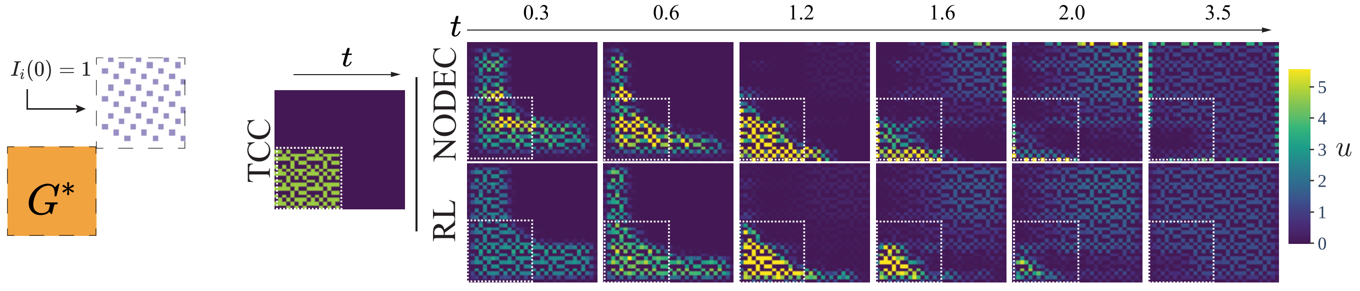

FIG. 6. Initial infection, target subgraph, and control trajectories for SIR-type dynamics. Colorscale plots represent 99.5% of the presented values for dynamics with NODEC controls.

at time $t^* = \text{argmax}_{t\leqslant T} J(I_{G^*}(t))$. This time is approximated by preserving a sample of states when using the ODESolve, and picking the maximum observed peak infection from that sample.

As the existing neural architectures discussed in Sec. V A 4 did not perform well, we switch to an architecture that includes the graph structure. To leverage the information of the graph structure and generate efficient control signals that "flatten" the curve, we decide to design a more specialized NN architecture that includes the information of the graph structure within its layers. For that reason we use a Graph NN (GNN) architecture (see Fig. 4 and Appendix B 1). We use a learning rate $\eta = 0.07$ and the Adam optimizer. The same GNN architecture is implemented in the RL baselines as the policy network. The GNN approach encountered fewer numerical instabilities during training and allowed for efficient learning without curriculum procedures. We use a training procedure for SIR-type control as shown in Appendix Algorithm 4 that preserves the best performing model in terms of loss. The hidden state matrix $\boldsymbol{Z}$ is calculated from the GNN and then provided as an input to the decision NN (see Fig. 4 right side). The decision network contains operations that enforce the budget and driver constraints by applying a softmax activation function and calculating control signal outputs for the driver nodes. The decision network contains no learned parameters and is included inside the NODEC architecture and RL environment. Transfer learning [100] between NODEC and RL can be achieved by pretraining the GNN network with NODEC and then using it as an RL policy. RL achieves the same performance as NODEC when transfer learning is tested. Further fine tuning of the pretrained policy with RL does not improve performance of NODEC in this setting, but transfer learning indicates a possible future extension of combining model-based training with real-world model-free fine tuning.

5. Results

Our main results are summarized in Fig. 5 and Table I and indicate similar superior performance of TCC and NODEC compared to the other control strategies, but with lower energy costs for NODEC. In Fig. 5, we observe that NODEC is providing strong protection with total energy costs that are not as high as TCC (see Table I). If we assume that the proposed system will reach maximum hospital capacity at 20% of the infected fraction in the target subgraph, we observe that TCC, RL, and NODEC are sustainable control strategies. In Fig. 5 and Table I, NODEC underperforms TCC with approximately 1% higher maximum infection fraction, but requires almost 41% less control energy. The effectiveness of the control can be attributed to the adaptive nature of NODEC. The other adaptive baseline, RL requires around 54% less energy than TCC but allows for 2.1% higher peak infection compared to NODEC. The effectiveness of targeted adaptive controls in time can be used to model and examine the effectiveness of proposed real-world long-term pandemic control strategies, such as rolling lockdowns [101] and/or vaccine allocation [98,102].

NODEC achieves better performance at the cost of higher energy compared to RL. Reinforcement learning is often described as "model-free" and addresses the (i) prediction problem and (ii) control problem [37]. We note that RL approaches may suffer from credit assignment challenges, where a reward signal is uninformative regarding the specific actions (especially in terms of time) that help reach the goal [103]. However, even after testing different reward designs and parameters settings, no RL framework managed to perform better than our baselines. It may be possible that extensive reward engineering, and other model upgrades may lead to a better performance. In contrast to RL, the proposed NODEC framework is not model-free and the underlying gradient descent is directly calculated from the loss function. Therefore, we do not need to consider value prediction and credit assignment. It is possible to design a model-free NODEC by learning the underlying system dynamics simultaneously with control similar to Ref. [43], which could be an interesting future extension of our work.

The spread of the epidemic, target subgraph, and controls of the main baselines are illustrated in Fig. 6. RL and NODEC calculate control signals that change over time and slowly fade out as $t \to T$. We also observe that controls persist in some driver nodes even then the infection wave is over (see also

TABLE I. Total energy $E(T)$ and peak infection $\max_t(\bar{I}(t))$ achieved by different epidemic spreading control methods.

Control	Peak Infection	Total Energy
TCC	0.068	14062.6
NODEC	0.078	8356.6
RL	0.099	6358.0
RND	0.210	4688.9
F	0.532	0.0

Appendix Fig. 10). This behavior is also observed in other baselines that satisfy the equality of the constraint Eq. (18b) (RND and TCC). The budget constraint Eq. (18b) allows control signals to sum up to the budget value $\mathscr{b}$. The implemented NN architecture calculates controls by multiplying the budget with a softmax activation function output over a hidden state output from the learned GNN architecture (see Fig. 4 right side). The output of the softmax activation function is nonzero by definition [104] and thus the NN always calculates nonzero control signals. Once the infection wave has traversed the graph, both RL and NODEC controllers distribute the control over several nodes, thus decreasing required control energy [105]. This outcome is an artifact of the softmax activation function, but it may also indicate the implicit energy regularization properties of NODEC [48]. On the contrary, the higher energy costs of TCC keep increasing, as high control signals remain in place after the infection wave has passed.

VI. DISCUSSION AND CONCLUSION

Neural ODE control is able to effectively control dynamical systems based on observations of their state evolution. Contrary to Ref. [28] that parameterizes the derivative of hidden states using NNs, our neural ODE systems describe controlled dynamical systems on graphs. In general, NNs are able to approximate any control input as long as they satisfy corresponding universal approximation theorems. However, in practice, NODEC needs to deal with different numerical hurdles such as large losses and stiffness problems of the underlying ODE systems. By testing NODEC on various graph structures and dynamical systems, we provide evidence that neural ODE-based control approaches are useful in feedback control and that numerical hurdles can be overcome with appropriate choices of both hyperparameters and ODE solvers.

Future studies may study the effectiveness of NODEC under additional constraints such as partial observability and delayed and noisy controls.

ACKNOWLEDGMENTS

L.B. acknowledges financial support from the SNF (Grant No. P2EZP2_191888). N.A.-F. has been funded by the European Program scheme "INFRAIA-01- 2018-2019: Research and Innovation action," Grant Agreement No. 871042, "SoBigData++: European Integrated Infrastructure for Social Mining and Big Data Analytics." T.A. received financial support from the LCM–K2 Center within the framework of the Austrian COMET-K2 program.

APPENDIX A: KURAMOTO OSCILLATORS

1. Curriculum learning

A curriculum learning procedure is used to train Kuramoto models. The algorithm is described below in Algorithm 3.

2. Synchronization loss before convergence

In this section, we describe one of the results presented in Fig. 3(b) in more detail. We observe that NODEC takes more time to converge to a synchronized state in the example illustrated in Fig. 7. We also observe that NODEC requires a higher amount of control energy before reaching the synchronized state [see Fig. 3(a)]. Once synchronicity is reached, the NN can adapt and produce lower energy controls. This might not be the case for feedback control, which has a constant term ζ multiplied by the driver matrix values.

Algorithm 3: Curriculum training process of NODEC. A procedure that gradually increases total time is introduced in this algorithm. Here we present a stochastic procedure, but a deterministic procedure is also possible.

```
Result: $\boldsymbol{w}$
Init:: $x_0$, $\boldsymbol{w}$, $\boldsymbol{f}(\cdot)$, ODESolve$(\cdot)$, Optimizer$(\cdot)$, $J(\cdot)$, $\boldsymbol{x}^*$;
Params:: $\eta$, epochs, stepSize ;
epoch $\leftarrow 0$;
$T \leftarrow 0$;
while epoch $<$ epochs do
    $t \leftarrow 0$ ;
    $c \sim \mathcal{U}(0,1)$ ;
    $T \leftarrow T + 2 \cdot c$ ;
    $\boldsymbol{x} \sim \mathcal{N}_N(0_1^N, 1_1^N)$;
    meanLoss $\leftarrow$ List;
    minLoss $\leftarrow \infty$ ;
    while $t < T$ do
        $\boldsymbol{x}(t)$, hasNumInstability $\leftarrow$
          ODESolve$(\boldsymbol{x}, t, t+$stepSize$, \boldsymbol{f}, \hat{\boldsymbol{u}}(\boldsymbol{x};\boldsymbol{w}))$;
        if Not hasNumInstability then
            meanLoss $\leftarrow$ (stepSize$/T) \cdot J(\boldsymbol{x}(t), \boldsymbol{x}^*)$);
            if minLoss $> J(\boldsymbol{x}_t, \boldsymbol{x}^*)$ then
                minLoss $\leftarrow J(\boldsymbol{x}_t, \boldsymbol{x}^*)$ ;
            end
        end
        $t \leftarrow t$ + stepSize ;
    end
    Optimizer.update($\boldsymbol{w}$, meanLoss + minLoss);
end
```

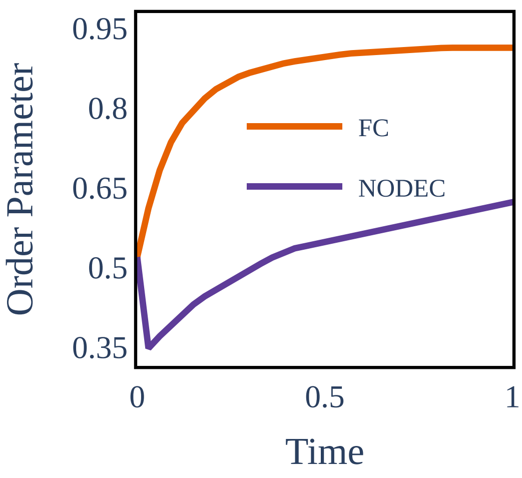

FIG. 7. Early order parameter values based on Fig. 3(b).

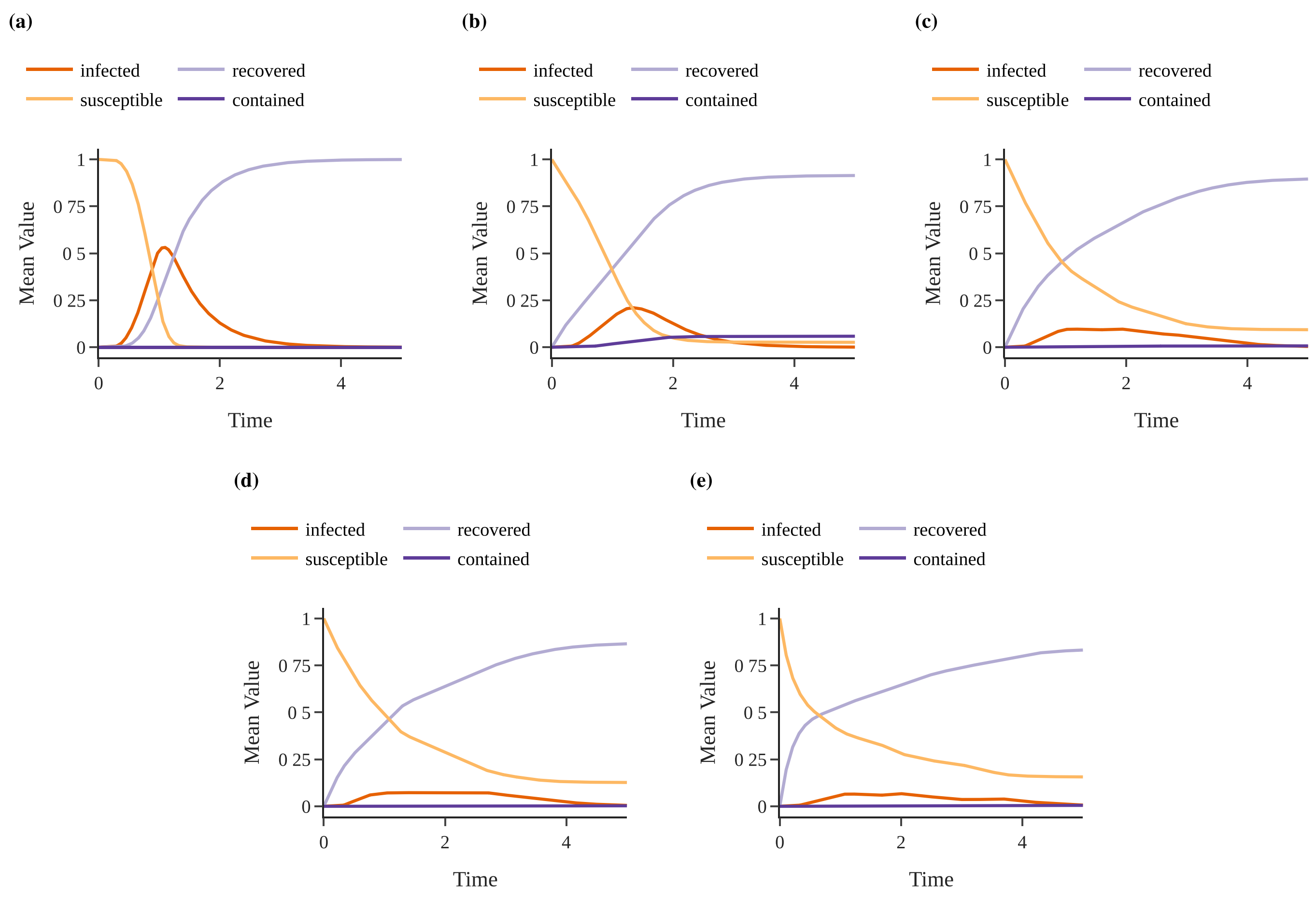

FIG. 8. Evolution of the proportion of susceptible, infected, recovered, and contained individuals for all baselines in the target subgraph G^*. We report the performance for (a) no control (F) baseline, (b) random control (RND) baseline, (c) reinforcement learning control (RL) baseline, (d) NN control (NODEC) baseline, and (e) targeted constant control (TCC) baseline.

APPENDIX B: EPIDEMIC MODEL

1. Neural network architecture

Here, we provide some technical details and an overview of the GNN architecture presented in Fig. 4. The final output of an NN is a control vector $\hat{\boldsymbol{u}}(\boldsymbol{X}(t))$. The input of the GNN is a tensor $\boldsymbol{\Psi} \in \mathbb{R}^{4\times N\times\hat{d}}$, where $\hat{d}$ is the maximum degree of the graph. An element $\boldsymbol{\Psi}_{k,j,i}$ of the tensor represents the kth state of the jth neighbor of node i. The jth neighbor of node i is fixed via any permutation of neighbors prior to training. The operation that constructs a tensor $\boldsymbol{\Psi}$ from the input state matrix $\boldsymbol{X}(t)$ is referred to as "neighborhood embedding." GNN applies an operation for each node that aggregates the state values over all neighboring nodes and produces a hidden state tensor $\boldsymbol{H}(\boldsymbol{\Psi})$. This hidden state is provided to the consecutive layers, and a hidden state matrix (or embedding) $\boldsymbol{Z} \in \mathbb{R}^{4\times N}$ is calculated, with same dimensions as the input state matrix $\boldsymbol{X}$. This matrix $\boldsymbol{Z}$ is provided again as an input to the GNN structure described above (see left side of Fig. 4) and a new tensor $\boldsymbol{\Psi}$ is calculated based on the neighborhood embedding procedure. Providing the calculated hidden state matrix $\boldsymbol{Z}$ as an input to the GNN is termed "message passing" [106]. This is a typical procedure when training GNNs. Message passing essentially allows the NN to calculate a hidden state representation for each node i but also leverage information of nonadjacent neighbors for the calculation after the first repetition. We observe that allowing the message passing process to repeat four times maximizes the performance of the network for the current control task. For example, in the second iteration of the above procedure, the input tensor $\boldsymbol{\Psi}$ of the GNN contains a representation calculated by a functional on an aggregation over all states $x_j(t)$ of all next-nearest

TABLE II. Tested and evaluated hyperparameters for the TD3 reinforcement learning baseline.

Hyperparameter	Value	Tested values
Actor learning rate	0.0003	0.0003, 0.003, 0.03
Actor architecture	GNN	GNN, FC
Critics learning rate	0.0001	0.0001, 0.001, 0.01
Critics architecture	FC	FC
τ (Polyak update parameter)	0.005	0.005, 0.05
γ (discount factor)	0.99	0.5, 0.8, 0.99, 1
Exploration Gaussian noise mean	0.01	0, 0.01. 0.1
Update frequency of actor parameters	4 epochs	1–4 epochs
Policy noise	0.001	0.001. 0.01, 0.1
Noise clip	0.5	0.5, 0.2
Reward normalization	True	True, False

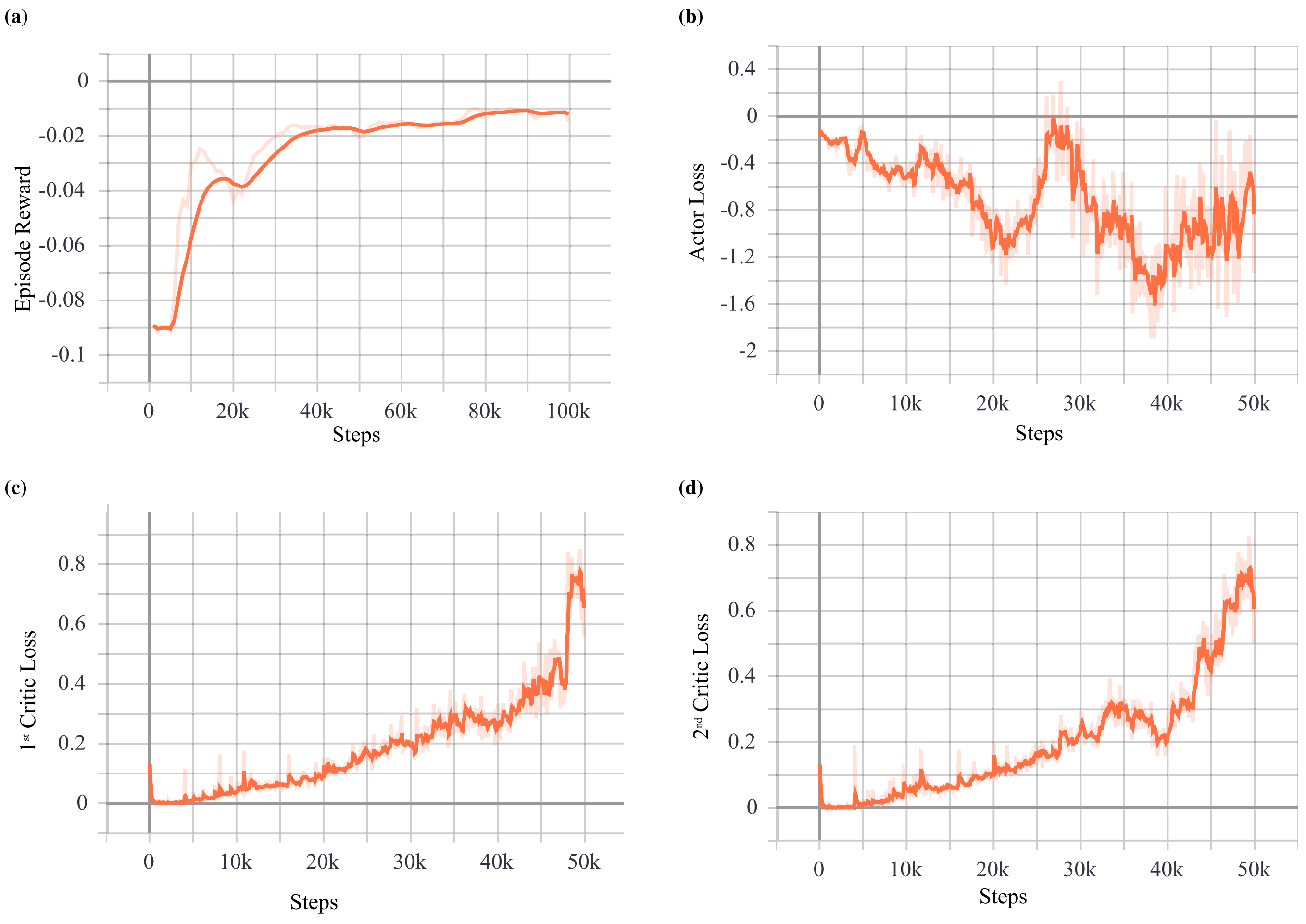

FIG. 9. RL learning performance evaluation plots using Tensorboard with 0.8 smoothing for (a) total episode reward as TD3 trains, (b) actor loss as TD3 trains, (c) first critic loss, and (d) second critic loss.

neighbors j of node i. In conclusion, the GNN architecture aims to learn a state representation $\boldsymbol{Z}$ that can be used to produce efficient control signals that take into account the states of nonneighboring nodes of each driver. After the last message propagation is finished, the mean over the channels is calculated over the hidden state matrix $\langle \mathbf{Z} \rangle_0$ generating a hidden state vector $\mathbf{z} \in \mathbb{R}^N$.

2. Reinforcement learning

In this section, we focus on the technical details of the RL baseline we used in the main paper. Reinforcement learning is often described as “model-free” and addresses the (i) prediction problem and (ii) control problem [37]. We note that RL approaches may suffer from credit assignment challenges, where a reward signal is uninformative regarding the specific actions (especially in terms of time) that help reach the goal [103]. In contrast to RL, the proposed NODEC is not model-free and the underlying gradient descent is directly calculated from the loss function. Therefore, we do not need to consider value prediction and credit assignment. It is possible to design a model-free NODEC by learning the underlying system dynamics simultaneously with control, which could be an interesting future extension of our work. Note that a direct performance comparison between RL and NODEC in terms of target loss may be considered unfair especially toward RL methods, unless extensive hyperparameter optimization is performed beforehand.

We first implement SIR-type dynamics as an RL environment. The softmax activation function and budget assignment discussed in Sec. V B 4 take place in the environment and RL computes the softmax logit values over all nodes. Reinforcement learning is allowed to interact with the environment in a fixed interaction interval $\Delta t = 10^{-2}$, similar to NODEC. A2C and SAC implementations are taken from StableBaselines3 [107]. Both implementations were tested for different parameter sets and trained for at least 50000 steps. Unfortunately, no implementation was able to “flatten the curve” considerably better than random control. Next, we use the TD3 implementation from Tianshu [108], which currently showcases high-speed benchmarks and allows more customization of policy/critic architectures. The corresponding RL training takes around 17 s per epoch, whereas NODEC takes approximately 5.5 s per epoch. Neither TD3 or NODEC fully utilized the GPU in terms of computing and memory resources, often staying below 50% of usage, while memory utilization usually was below 10GB per method.

We show an overview of the hyperparameters that we use to train TD3 in Table II. For more detailed explanations of these hyperparameters, see Ref. [51] and the Tianshu documentation [108]. Several baseline architectures in RL frameworks are often fully connected multilayer perceptrons. Still, we observe that the graph NN presented in Fig. 4 was more efficient in converging rewards in less computation time. We trained all models for 100 epochs and stored and evaluated the best model. In SAC and A2C, one training environment was used, whereas TD3 was sampling from two independent environments simultaneously due to its computational speed.

In terms of parameters both the TD3 policy network and NODEC GNN have exactly the same learning parameters (weights), but training is very different, as the gradient flows described in Fig. 1 and Algorithms 1 and 2 cannot happen. The value function is now used for the calculation of similar gradients by predicting the cumulative reward signal. We studied several possible reward designs, and in the end we rigorously tested the following rewards:

The first reward signal we tested is calculated based on the mean number of infected nodes belonging to the target subgraph $\bar{I}_{G^*}(t)$ at time t:

$$\rho_1(t) = -[\bar{I}_{G^*}(t)]^2 \Delta t. \tag{B1}$$

Although this reward seemingly provides direct feedback for an action, it also leads to several challenges. First, it does not necessarily flatten the curve, but it minimizes the overall infection through time. Such a reward could, for instance, potentially reinforce actions that lead to "steep" peaks instead of a flattened infection curve, as in practice it minimizes the area under the $I(t)$ curve. Furthermore, as current containment controls may have effect if applied consistently and in the long term, such reward design suffers from temporal credit assignment, since the reward value depends on a long and varying sequence of actions. Finally, any actions that happen after the peak infection occurrence will still be rewarded negatively, although such actions do not contribute to the goal minimization.

The next reward

$$\rho_2(t) = \begin{cases} 0 & \text{, if } t < T, \\ -(\max_{t\leqslant T} \bar{I}_{G^*}(t))^2 & \text{, otherwise,} \end{cases} \tag{B2}$$

is designed to overcome the aforementioned shortcomings. This reward signal is sparse through time, as it is nonzero only at the last step of the control when the infection peak is known. The main property of interest of Eq. (B2) is that it has the same value as the loss that we used to train NODEC (see Eq. (19). This reward signal also suffers from credit assignment problems. As the reward is assigned at a fixed time and not as a direct result of the actions that caused it, the corresponding reward dynamics is non-Markovian [109]. To address challenges caused by rewards with non-Markovian properties, reward shaping [110] and recurrent value estimators [111] can be used. Furthermore, n-step methods or eligibility traces can be evaluated if we expect the reward signal to be Markovian but with long and/or varying time dependencies.

The final reward $\rho_3(t)$ that we evaluated and used in the presented results is designed with two principles in mind:

$$\sum_t \rho_3(t) \underset{\sim}{\propto} \left(\max_{t\leqslant T} \bar{I}_{G^*}(t)\right)^2, \tag{B3a}$$

$$\text{argmin}_{t\leqslant T} \sum_t \rho_3(t) = \left(\underset{t\leqslant T}{\text{argmax}}\, \bar{I}_{G^*}(t)\right) \tag{B3b}$$

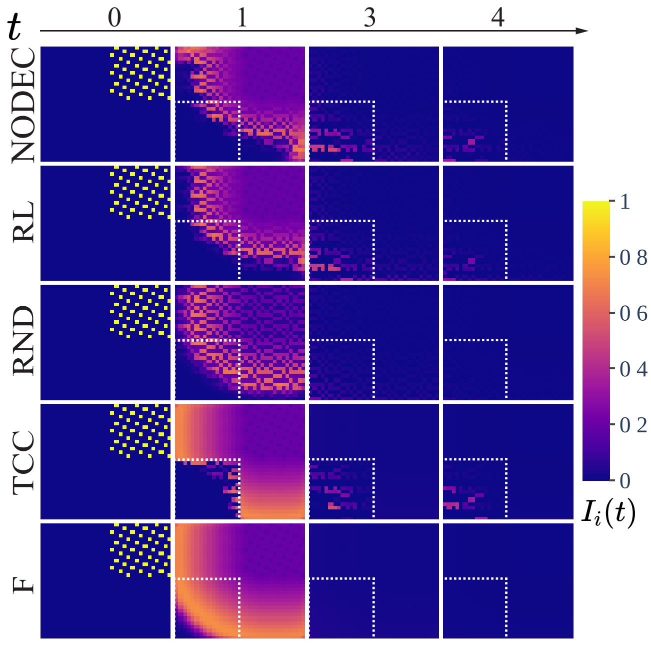

FIG. 10. Spread of infection for all baselines.

Following those principles, the reward signal is approximately proportional to and provides information about the value of the infection peak used in the NODEC loss calculation. The reward sum minimizes exactly at the time when peak infection occurs. This property is expected to reduce effects of temporal credit assignment. When aiming to replace the proportionality in Eq. (B3a) with an equality, we obtain

$$\rho_3(t) = \begin{cases} 0, & \text{if } \bar{I}_{G^*}(t) \leqslant \max_{\tau<t}(\bar{I}_{G^*}(\tau)), \\ -\bar{I}^2_{G^*}(t) + (\max_{\tau<t} \bar{I}_{G^*}(\tau))^2, & \text{otherwise,} \end{cases} \tag{B4}$$

which is equivalent to the reward function of Eq. (20) in the main paper. It is straightforward to show that Eq. 20 indeed satisfies $\sum_t \rho_3(t) = \max_{t\leqslant T}(\bar{I}_{G^*}(t))^2$ and Eq. (B3b). This reward greatly improved performance without resorting to recurrent value estimators or further reward shaping. Still, after all proposed reward design and hyperparameter optimization, NODEC has a higher performance (see Fig. 8), although TD3 performs better than random control.

In Figs. 6 and 10 the dynamic controls of both RL and NODEC seem to focus on protecting the target subgraph by containing the infection as it spreads. In contrast to targeted constant control, they succeed in doing so by protecting driver nodes outside the target subgraph. When comparing the dynamic control patterns, the budget allocation of NODEC seems to be much more concentrated on specific nodes, and it creates more often contiguous areas of containment.

In Fig. 8, we also show the evolution of the proportion of susceptible $S(t)$, infected $I(t)$, recovered $R(t)$, and contained individuals $Y(t)$. We observe that TCC and NODEC show clear signs of flattening the curve by preserving the highest susceptibility fraction and lowest recovery fraction at time T, which can be interpreted as less susceptible nodes becoming infected and needing to recover. The random method outperforms the other frameworks in terms of effective containment

TABLE III. Nomenclature Part I for Sec. IV.

t_0	Initial time for control of a dynamical process. Often we may also use $t = 0$ without loss of generality.
T	Terminal time for control of a dynamical system.
Δt	Timestep
$G(\mathrm{V}, \mathrm{E})$	Graph represented as an ordered pair of a set of nodes V and a set of edges E.
N	Number of nodes in a graph.
$\boldsymbol{A}$	Adjacency matrix that represents a graph G. It has non zero elements $\boldsymbol{A}_{i,j} \neq 0$ if and only if nodes i, j are connected.
$\boldsymbol{x}(t)$	Vector $\boldsymbol{x}(t) \in \mathbb{R}^N$, which denotes the state of a dynamical system at time t.
$\boldsymbol{x}^*$	Vector that denotes the target state of a dynamical system.
$\dot{\boldsymbol{x}}(t)$	Newton's dot notation for differentiation of the system state.
M	Number of driver nodes, i.e., nodes that can be controlled in a graph. As the driver nodes is a subset of all the nodes we have $M \leqslant N$.
$\boldsymbol{f}(t, \boldsymbol{x}(t), u(\boldsymbol{x}(t)))$	System evolution function that denotes the dynamic interactions between nodes and drivers when calculating the state derivative.
$\boldsymbol{u}(\boldsymbol{x}(t))$	Feedback control signal function $\boldsymbol{u}(\boldsymbol{x}(t)) : \mathbb{R}^N \to \mathbb{R}^M$ calculated based on the system state at time t.
$\boldsymbol{B}$	Driver matrix $\boldsymbol{B} \in \mathbb{R}^{N\times M}$, where $\boldsymbol{B}_{i,m} = 1$ if node i is the m-th driver node and receives a control signal $u_m(t)$.
$E(\boldsymbol{u}(\boldsymbol{x}(t)))$	Total energy value of a control signal calculated from time t_0 until time t.
$\hat{\boldsymbol{u}}(\boldsymbol{x}(t))$	Control signal value calculated by NODEC.
$\boldsymbol{w}$	Vector with NN parameters for NODEC.
$\mathrm{X}_{t_0}^T$	State trajectory between t_0 and T. An ordered set of state vectors $\boldsymbol{x}(t), t \in [t_0, T]$.
$J(\mathrm{X}_{t_0}^T, \boldsymbol{x}^*; \boldsymbol{w})$	Learning and control objective function for NODEC. In the current work, we evaluate control goals $\boldsymbol{x}^*$ that are achieved over a state trajectory $\mathrm{X}_{t_0}^T$.
$\Delta \boldsymbol{w}$	Gradient descent update for NN parameters.
η	Learning rate hyperparameter for gradient descent.
$\boldsymbol{h}(t)$	Hidden state evolution function used in the neural ODE paper [28].
$\mathrm{ODESolve}(\mathbf{x}(t), t, T, \boldsymbol{f}, \boldsymbol{u})$	Function that denotes a numerical ODE solving scheme.

fractions, as random control assignments at each time step let the disease spread such that higher proportions of infected individuals $I(t)$ are reached in the target subgraph and therefore drivers with high infection fractions are effectively contained when controlled. Although low energy effective containment might seem favorable at first sight, it is not optimal in terms of flattening the curve with restricted budget, as it allows high infection fractions to occur within an area of interest. Budget restrictions often do not allow to fully constrain the spread in all infected nodes.

In Fig. 9, we observe that although RL does not converge in terms of critic and actor loss, it still converges to a higher reward. This confirms that RL is capable of controlling continuous dynamics with arbitrary targets, but it requires significant parametrization and training effort to have good stable value estimates.

Finally, we tried to examine transfer learning capabilities from NODEC to RL. A closer look at Fig. 4 reveals that the parameterized graph neural architecture used for NODEC and RL can be the same, i.e., there are no weights in the decision

TABLE IV. Nomenclature Part II (Coupled Oscillators) for Sec. V A.

ω_i	Natural frequency of oscillator (node) i.
K	Coupling constant.
$\hbar(x_i - x_j)$	2π-periodic function function that couples oscillators. Often the sine function is used, s.t. $\hbar(\cdot) = \sin(\cdot)$.
$\boldsymbol{x}^\diamond$	Synchronized steady state of coupled oscillator system.
$\boldsymbol{D}$	Graph degree diagonal matrix $\boldsymbol{D}$ of G, where all off-diagonal elements are 0 and diagonal elements are equal to the degree $\boldsymbol{D}_{i,i} = d_i$ of the corresponding node.
$\boldsymbol{L}$	Graph Laplacian matrix $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$ of G.
$\boldsymbol{L}^\dagger$	Pseudoinverse of the graph Laplacian matrix $\boldsymbol{L}$ of G.
$\mathbf{b}^{(\mathrm{FC})}$	Feedback control gain vector of the FC baseline.
$r(t)$	Order parameter, which denotes the synchronization of coupled oscillators.
ζ	Scaling parameter for feedback control baseline.
τ	Timestep.
Ξ	Number of timesteps for discretizing the time period $[0, T]$.
ξ	Timestep index used to calculate discretized approximations of continuous-time quantities.
$r_{\mathrm{NODEC}(t)}$	Order parameter value achieved under NODEC control at time t.
$E_{\mathrm{NODEC}(t)}$	Total energy value achieved under NODEC control at time t.
$r_{\mathrm{FC}(t)}$	Order parameter value achieved under feedback control baseline at time t.
$E_{\mathrm{FC}(t)}$	Total energy value achieved under feedback control baseline control at time t.

TABLE V. Nomenclature Part III (Disease Spreading) for Sec. V B.

$S_i(t)$	Susceptible fraction of individuals at node i at time t.
$I_i(t)$	Infected fraction of individuals at node i at time t.
$R_i(t)$	Recovered fraction of individuals at node i at time t.
$Y_i(t)$	Contained fraction of individuals at node i at time t.
$\boldsymbol{X}(t)$	Matrix representation of the state, where the vectors **S**, **I**, **R**, **Y** are rows. Each column represent the state of a certain node.
G^*	Target subgraph, i.e., the subset of nodes that we are interested to reduce the peak infection.
β	Infection rate.
γ	Recovery rate.
c_j	Number sampled from a uniform distribution $c_j \sim \mathcal{U}(0, 1)$ to calculate random control.
$\mathscr{b}$	Control budget. A linear constraint on maximum total control that can be applied at time t.
$\rho(t)$	Reward signal for reinforcement learning techniques.
d_i	Degree of a node i.
$\hat{d}$	Maximum degree.
$\boldsymbol{\Psi}$	Input tensor for convolutional NN of the GNN.
$\boldsymbol{Z}$	Output of hidden layers to be used for message propagation in the graph NN.

network layers of Fig. 4. This means that the architectures trained with NODEC can be used as the "logit" action policy in RL, showcasing an effective use of transfer learning. In the given example, the RL policy network starting with trained NODEC parameters, is further trained for 100 episodes. After training, RL had a similar performance as NODEC since both methods flatten the curve at approximately $\bar{I}_{G^*} = 0.0788$. This means that RL did not improve the solution generated by NODEC. This example can be used to illustrate the interplay between NODEC and RL and how they can be used in synergy, e.g., when back-propagating through continuous dynamics is too expensive for high number of epochs. Reinforcement learning can be used as a metaheuristic on top of NODEC, and the latter can be treated as an alternative to imitation learning.

APPENDIX C: OTHER NOTES

1. Hardware and code

Our experiments were mainly conducted on a dedicated server that was equipped with a NVIDIA TITAN RTX GPU, 64GB of RAM, and an Intel I9 9900KF 8-core processor. Partial code tests with assertions were conducted to examine (i) stiffness, (ii) numerical errors or bugs, and (iii) validity and similarity of the same dynamics controlled by different models. For the majority of the experiments seeds are fixed and initial states parameters are stored in data files to enable reproducibility. ODEsolve and sample experiments may be affected by stochasticity on different machines. Based on statistical testing, we observe that with a good initialization and NN hyperparameter optimization, NODEC performs close to the reported values. Future works under provided repository, may perform extensive hyperparameter studies dedicated to specific dynamics and graphs. The average training time of NODEC per task is between 5 and 10 min, depending on the complexity of the task. Baseline methods calculations and parametrizations would also take minutes, making time performance comparable.

The project code can be found on GitHub [112] under MIT license. Numerical experiments are stored in the experiment folder (please check github README for more details).

2. ODE solvers and stiffness

We used the Dormand–Prince solver [54] for the majority of our numerical experiments (in particular, for training). For evaluating our results, we use a specific method, which allows the controller to change the control signal at constant time intervals. This choice allows us to compare control errors and energy costs without considering interaction frequency bias that occurs when one method outperforms another method because the solver allowed it to interact more often with the system and produce more tailored control signals. Adaptive step length helps the NN to learn controls for variable interaction intervals and approximate continuous control better. We performed small-scale unit tests with VODE [54] against Dormand–Prince, Runge–Kutta, and implicit Adams

Algorithm 4: Adaptive Learning rate training process of NODEC.

```
Result: $\boldsymbol{w}$
Init:: $\boldsymbol{x}_0$, $\boldsymbol{w}$, $(f)(\cdot)$, ODESolve$(\cdot)$, Optimizer$(\cdot)$, $J(\cdot)$, $\boldsymbol{x}^*$;
Params:: $\eta$, epochs, $q$, tolRatio;
epoch $\leftarrow$ 0;
bestLoss $\leftarrow \infty$;
bestParams $\leftarrow$ copy($\boldsymbol{w}$);
previousLoss;
while epoch < epochs do
    $t \leftarrow 0$ ;
    $\boldsymbol{x} \leftarrow \boldsymbol{x}_0$;
    $\mathrm{X}_{t_0}^T$, hasNumInstability $\leftarrow$ ODESolve($\boldsymbol{x}, 0, T, \boldsymbol{f}, \hat{\boldsymbol{u}}(\boldsymbol{x}(t); \boldsymbol{w})$);
    if $J(\mathrm{X}_{t_0}^T, \boldsymbol{x}^*)$ > tolRatio $\cdot$ previousLoss$\vee$ hasNumInstability
      then
        $\boldsymbol{w} \leftarrow$ bestParams;
        $\eta \leftarrow \eta q$;
        Optimizer.reset();
        Optimizer.learningRate $\leftarrow \eta$;
    end
    else
        if $J(\boldsymbol{x}, \boldsymbol{x}^*)$ < bestLoss then
            bestParams $\leftarrow$ copy($\boldsymbol{w}$);
            bestLoss $\leftarrow J(\boldsymbol{x}, \boldsymbol{x}^*)$;
        end
        previousLoss $\leftarrow J(\boldsymbol{x}, \boldsymbol{x}^*)$;
        Optimizer.update($\boldsymbol{w}$, $J(\boldsymbol{x}, \boldsymbol{x}^*)$);
    end
end
```

implementations, and we noticed that for most systems numerical errors were negligible.

The goal of this paper is to evaluate the ability of NODEC to learn controls within a solver. In future works that aim at controlling large-scale systems, different ODE solvers may be chosen according to both the system's stiffness and performance requirements of the application. Whenever dynamics and training had high VRAM requirements, the adjoint method was used (mainly the implementations from Refs. [28,113]).

3. Adaptive learning rate training

Learning rate plays an important role on reaching a low energy control. To determine the optimal learning rate values we propose the adaptive learning rate scheme found in Algorithm 4.

APPENDIX D: NOMENCLATURE

The notation used in this article is summarized in Tables III–V.

[1] L. Böttcher, H. J. Herrmann, and H. Gersbach, Clout, activists and budget: The road to presidency, PLoS One **13**, e0193199 (2018).

[2] M. Hoferer, L. Böttcher, H. J. Herrmann, and H. Gersbach, The impact of technologies in political campaigns, Physica A **538**, 122795 (2020).

[3] D. Brockmann and D. Helbing, The hidden geometry of complex, network-driven contagion phenomena, Science **342**, 1337 (2013).

[4] N. Antulov-Fantulin, A. Lančić, T. Šmuc, H. Štefančić, and M. Šikić, Identification of Patient Zero in Static and Temporal Networks: Robustness and Limitations, Phys. Rev. Lett. **114**, 248701 (2015).

[5] L. Böttcher, J. Andrade, and H. J. Herrmann, Targeted recovery as an effective strategy against epidemic spreading, Sci. Rep. **7**, 1 (2017).

[6] F. A. Rodrigues, T. K. D. Peron, P. Ji, and J. Kurths, The Kuramoto model in complex networks, Phys. Rep. **610**, 1 (2016).

[7] M. K. Stephen Yeung and S. H. Strogatz, Time Delay in the Kuramoto Model of Coupled Oscillators, Phys. Rev. Lett. **82**, 648 (1999).

[8] D. Delpini, S. Battiston, M. Riccaboni, G. Gabbi, F. Pammolli, and G. Caldarelli, Evolution of controllability in interbank networks, Sci. Rep. **3**, 01626 (2013).

[9] N. G. Van Kampen, *Stochastic Processes in Physics and Chemistry* (Elsevier, Amsterdam, 1992), Vol. 1.

[10] J. E. Moyal, Stochastic processes and statistical physics, J. R. Stat. Soc. Ser. B (Methodol.) **11**, 150 (1949).

[11] H. Andersson and T. Britton, *Stochastic Epidemic Models and Their Statistical Analysis* (Springer Science and Business Media, Berlin, 2012), Vol. 151.

[12] C. Castellano and R. Pastor-Satorras, Thresholds for Epidemic Spreading in Networks, Phys. Rev. Lett. **105**, 218701 (2010).

[13] J. P. Gleeson, Binary-State Dynamics on Complex Networks: Pair Approximation and Beyond, Phys. Rev. X **3**, 021004 (2013).

[14] B. Barzel and A.-L. Barabási, Universality in network dynamics, Nat. Phys. **9**, 673 (2013).

[15] D. J. Watts and S. H. Strogatz, Collective dynamics of "small-world" networks, Nature (London) **393**, 440 (1998).

[16] D. J. D. S. Price, Networks of scientific papers, Science **149**, 510 (1965).

[17] A.-L. Barabási and R. Albert, Emergence of scaling in random networks, Science **286**, 509 (1999).

[18] M. Girvan and M. E. Newman, Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA **99**, 7821 (2002).

[19] S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes, Critical phenomena in complex networks, Rev. Mod. Phys. **80**, 1275 (2008).

[20] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, Kronecker graphs: An approach to modeling networks, J. Mach. Learn. Res. **11**, 985 (2010).

[21] Y.-Y. Liu, J.-J. Slotine, and A.-L. Barabási, Controllability of complex networks, Nature (London) **473**, 167 (2011).

[22] Y.-Y. Liu and A.-L. Barabási, Control principles of complex systems, Rev. Mod. Phys. **88**, 035006 (2016).

[23] A. Barrat, M. Barthelemy, and A. Vespignani, *Dynamical Processes on Complex Networks* (Cambridge University Press, Cambridge, UK, 2008).

[24] R. Pastor-Satorras, C. Castellano, P. Van Mieghem, and A. Vespignani, Epidemic processes in complex networks, Rev. Mod. Phys. **87**, 925 (2015).

[25] O. L. Mangasarian, Sufficient conditions for the optimal control of nonlinear systems, SIAM J. Control **4**, 139 (1966).

[26] M. I. Kamien and N. L. Schwartz, Sufficient conditions in optimal control theory, J. Econ. Theory **3**, 207 (1971).

[27] E. McShane, The calculus of variations from the beginning through optimal control theory, SIAM J. Control Optim. **27**, 916 (1989).

[28] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, Neural ordinary differential equations, Adv. Neural Inf. Process Syst. **31** (2018).

[29] R. E. Kalman, Contributions to the theory of optimal control, Boletín de la Sociedad Matemática Mexicana **5**, 102 (1960).

[30] M. L. Hautus, Controllability and observability conditions of linear autonomous systems, in *Proceedings of the Indagationes Mathematicae)*, Vol. 72 (North-Holland Publishing Company, Amsterdam Netherlands, 1969), pp. 443–448.

[31] C.-T. Lin, Structural controllability, IEEE Trans. Autom. Control **19**, 201 (1974).

[32] J. Ruths and D. Ruths, Control profiles of complex networks, Science **343**, 1373 (2014).

[33] G. Yan, J. Ren, Y.-C. Lai, C.-H. Lai, and B. Li, Controlling Complex Networks: How Much Energy is Needed? Phys. Rev. Lett. **108**, 218703 (2012).

[34] X. Zhou, Maximum principle, dynamic programming, and their connection in deterministic control, J. Optim. Theory Appl. **65**, 363 (1990).

[35] W. H. Fleming and H. M. Soner, *Controlled Markov Processes and Viscosity Solutions* (Springer Science & Business Media, Berlin, 2006), Vol. 25.

[36] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming* (Princeton University Press, Prtinceton, NJ, 2015).

[37] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (MIT Press, Cambridge, MA, 2018).

[38] B. Kiumarsi, K. G. Vamvoudakis, H. Modares, and F. L. Lewis, Optimal and autonomous control using reinforcement learning: A survey, IEEE Trans. Neural Networks Learn. Syst. **29**, 2042 (2017).

[39] H. Frankowska, Nonsmooth solutions of Hamilton-Jacobi-Bellman equation, in *Modeling and Control of Systems* (Springer-Verlag, Berlin, 1989), pp. 131–147.

[40] F. Lewis, S. Jagannathan, and A. Yesildirak, *Neural Network Control of Robot Manipulators and Nonlinear Systems* (CRC Press, Boca Raton, FL, 2020).

[41] S. J. Yoo, J. B. Park, and Y. H. Choi, Stable predictive control of chaotic systems using self-recurrent wavelet neural network, Int. J. Control Autom. Syst. **3**, 43 (2005).

[42] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, Differentiable mpc for end-to-end planning and control, in *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, NY, 2018), pp. 8289–8300.

[43] P. Holl, V. Koltun, and N. Thuerey, Learning to control PDEs with differentiable physics, in Proceedings of the International Conference on Learning Representations (2020).

[44] B. Pang, Z.-P. Jiang, and I. Mareels, Reinforcement learning for adaptive optimal control of continuous-time linear periodic systems, Automatica **118**, 109035 (2020).

[45] U. Biccari and E. Zuazua, A stochastic approach to the synchronization of coupled oscillators, Front. Energy Res. **8**, 115 (2020).

[46] A. W. Wijayanto and T. Murata, Flow-aware vertex protection strategy on large social networks, in *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)* (IEEE, Piscataway, NJ, 2017), pp. 58–63.

[47] S. P. Cornelius, W. L. Kath, and A. E. Motter, Realistic control of network dynamics, Nat. Commun. **4**, 1 (2013).

[48] L. Böttcher, N. Antulov-Fantulin, and T. Asikis, AI Pontryagin or how artificial neural networks learn to control dynamical systems, Nat. Commun. **13**, 333 (2022).

[49] P. S. Skardal and A. Arenas, On controlling networks of limit-cycle oscillators, Chaos: An Interdisciplinary Journal of Nonlinear Science **26**, 094812 (2016).

[50] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in *Proceedings of the International Conference on Machine Learning* (PMLR, Stockholmsmässan, Stockholm Sweden, 2018), pp. 1861–1870.

[51] S. Fujimoto, H. Hoof, and D. Meger, Addressing function approximation error in actor-critic methods, in *Proceedings of the International Conference on Machine Learning* (PMLR, Stockholmsmässan, Stockholm Sweden, 2018), pp. 1587–1596.

[52] M. Newman, *Networks* (Oxford University Press, Oxford, UK, 2018).

[53] O. Mayr, The origins of feedback control, Sci. Am. **223**, 110 (1970).

[54] L. F. Shampine, *Numerical Solution of Ordinary Differential Equations* (Routledge, Oxford, UK, 2018).

[55] C. Commault, J.-M. Dion, and J. W. van der Woude, Characterization of generic properties of linear structured systems for efficient computations, Kybernetika **38**, 503 (2002).

[56] T. Yamada and L. R. Foulds, A graph-theoretic approach to investigate structural and qualitative properties of systems: A survey, Networks **20**, 427 (1990).

[57] M. R. Garey and D. S. Johnson, *Computers and Intractability* (Freeman, San Francisco, CA, 1979), Vol. 174.

[58] A. Olshevsky, Minimal controllability problems, IEEE Trans. Control Network Syst. **1**, 249 (2014).

[59] P. S. Skardal and A. Arenas, Control of coupled oscillator networks with application to microgrid technologies, Sci. Adv. **1**, e1500339 (2015).

[60] In shared control, the same control signal $u_m[x(t)]$ is applied to multiple nodes, while in interacting control multiple control signals are applied to the same node i.

[61] Imposing an energy constraint would require collecting and back-propagating the norm of all control inputs at each time step during training. Using such a back-propagation scheme would increase training times considerably because of the potentially large number of control inputs in large-scale graph dynamical systems.

[62] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: A survey, J. Mach. Learn. Res. **18**, 1 (2018).

[63] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, NY, 2019), pp. 8026–8037.

[64] P. Kidger, J. Morrill, J. Foster, and T. Lyons, Neural controlled differential equations for irregular time series, in *Advances in Neural Information Processing Systems* (Curran Associates, Red Hook, NY, 2020).

[65] M. H. Stone, The generalized Weierstrass approximation theorem, Math. Mag. **21**, 237 (1948).

[66] D.-X. Zhou, Universality of deep convolutional neural networks, Appl. Comput. Harmon. Anal. **48**, 787 (2020).

[67] A. M. Schäfer and H. G. Zimmermann, Recurrent neural networks are universal approximators, in *Proceedings of the International Conference on Artificial Neural Networks* (Springer, Berlin, 2006), pp. 632–640.

[68] E. D. Sontag and H. Siegelmann, On the computational power of neural nets, J. Comput. Syst. Sci. **50**, 132 (1995).

[69] B. Hanin and M. Sellke, Approximating continuous functions by relu nets of minimal width, arXiv:1710.11278 (2017).

[70] J. Lygeros, On reachability and minimum cost optimal control, Automatica **40**, 917 (2004).

[71] T. Teshima, K. Tojo, M. Ikeda, I. Ishikawa, and K. Oono, Universal approximation property of neural ordinary differential equations, arXiv preprint arXiv:2012.02414 (2020).

[72] M. A. Bucci, O. Semeraro, A. Allauzen, G. Wisniewski, L. Cordier, and L. Mathelin, Control of chaotic systems by

deep reinforcement learning, Proc. R. Soc. A **475**, 20190351, (2019).

[73] C. Hua and X. Guan, Adaptive control for chaotic systems, Chaos, Solitons Fractals **22**, 55 (2004).

[74] S. Bhasin, R. Kamalapurkar, M. Johnson, K. G. Vamvoudakis, F. L. Lewis, and W. E. Dixon, A novel actor–critic–identifier architecture for approximate optimal control of uncertain nonlinear systems, Automatica **49**, 82 (2013).

[75] B. G. Liptak, *Instrument Engineers' Handbook, Volume Two: Process Control and Optimization* (CRC Press, Boca raton, FL, 2018).

[76] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, The explicit linear quadratic regulator for constrained systems, Automatica **38**, 3 (2002).

[77] J. Sethna *et al.*, *Statistical Mechanics: Entropy, Order Parameters, and Complexity* (Oxford University Press, Oxford, UK, 2006), Vol. 14.

[78] M. Brede, Locals vs. global synchronization in networks of nonidentical Kuramoto oscillators, Eur. Phys. J. B **62**, 87 (2008).

[79] T. Asikis, L. Böttcher, and N. Antulov-Fantulin, Github repository for neural network control (2020), https://github.com/asikist/nnc.

[80] T. Asikis, L. Böttcher, and N. Antulov-Fantulin, Code ocean capsule for reproducing all nodec experiments https://codeocean.com/capsule/1934600/tree/v1 (2021).

[81] T. Asikis, L. Böttcher, and N. Antulov-Fantulin, Data repository for all nodec experiments, https://ieee-dataport.org/documents/neural-ordinary-differential-equation-control-dynamics-graphs (2020).

[82] D. Cumin and C. Unsworth, Generalising the Kuramoto model for the study of neuronal synchronisation in the brain, Physica D **226**, 181 (2007).

[83] F. Dorfler and F. Bullo, Synchronization and transient stability in power networks and nonuniform Kuramoto oscillators, SIAM J. Control Optim. **50**, 1616 (2012).

[84] D. Schoenwald and U. Ozguner, Optimal control of feedback linearizable systems, in *Proceedings of the 31st IEEE Conference on Decision and Control* (IEEE, Piscataway, NJ, 1992), pp. 2033–2034.

[85] C. Bick, M. J. Panaggio, and E. A. Martens, Chaos in Kuramoto oscillator networks, Chaos **28**, 071102 (2018).

[86] Y. L. Maistrenko, O. V. Popovych, and P. A. Tass, Chaotic attractor in the Kuramoto model, Int. J. Bifurcat. Chaos **15**, 3457 (2005).

[87] Y. Kuramoto, Self-entrainment of a population of coupled nonlinear oscillators, in *Proceedings of the International Symposium on Mathematical Problems in Theoretical Physics* (Springer, Berlin, 1975), pp. 420–422.

[88] The target states that satisfy this control goal are not unique and not necessarily known, but satisfy $x^* = \text{argmax}_x r(x)$. Since there is no specific dependence on a target state vector, we omit the quantity x^* in the loss function.

[89] The initial time is omitted [$\xi = \{1, \ldots, \Xi\}$ in Eq. (13)], since we assume that no control is applied prior to reaching the initial state.

[90] We tested several other values before selecting the specific value. Smaller values would lead to a lower degree of synchronization than that achieved by NODEC, but they would require less energy. Higher values would either completely fail to synchronize the system or require very high amounts of energy to achieve similar results.

[91] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, Curriculum learning, in *Proceedings of the International Conference on Machine Learning* (Association for Computing Machinery, Montreal, Quebec, Canada, 2009), pp. 41–48.

[92] K. J. Åström and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers* (Princeton Univeristy Press, Princeton, NJ, 2010).

[93] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *Proceedings of the International Conference on Machine Learning* (JMLR.org, Lille, France, 2015), pp. 448–456.

[94] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in *Proceedings of the International Conference on Learning Representations (ICLR), San Diego, CA, May 7–9*, edited by Y. Bengio and Y. LeCun (2015).

[95] M. Salathé and J. H. Jones, Dynamics and control of diseases in networks with community structure, PLoS Comput. Biol. **6**, e1000736 (2010).

[96] B. F. Maier and D. Brockmann, Effective containment explains subexponential growth in recent confirmed covid-19 cases in China, Science **368**, 742 (2020).

[97] We note that here we use capital letters for the SIR-type variables, to follow the common notation in related literature.

[98] M. Xia, L. Böttcher, and T. Chou, Controlling epidemics through optimal allocation of test kits and vaccine doses across networks (2021), arXiv:2107.13709.

[99] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, Asynchronous methods for deep reinforcement learning, in *Proceedings of the International Conference on Machine Learning* (2016), pp. 1928–1937.

[100] S. J. Pan and Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. **22**, 1345 (2009).

[101] D. Acemoglu, V. Chernozhukov, I. Werning, and M. D. Whinston, Optimal targeted lockdowns in a multi-group sir model, NBER Working Paper **27102** (2020).

[102] V. M. Preciado, M. Zargham, C. Enyioha, A. Jadbabaie, and G. Pappas, Optimal vaccine allocation to control epidemic outbreaks in arbitrary networks, in *Proceedings of the IEEE Conference on Decision and Control* (IEEE, Piscataway, NJ, 2013), pp. 7486–7491.

[103] R. S. Sutton, Temporal credit assignment in reinforcement learning, Ph.D. dissertation, University of Massachusetts Amherst, 1985.

[104] In practice control signals may approach 0 due to floating point errors.

[105] Looking at the control energy Eq. (2), we observe that low absolute value control signals assigned over many driver nodes may produce lower energy values compared to very high absolute value control signals applied to fewer driver nodes.

[106] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, How powerful are graph neural networks? in *Proceedings of the International Conference on Learning Representations* (ICLR, New Orleans, USA, 2019).

[107] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, Stable-Baselines3: Reliable Reinforcement Learning Implementations, Journal of Machine Learning Research, **11**, 1 (2021).

[108] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, H. Su, J. Zhu, Tianshou: A highly modularized deep reinforcement learning library, arXiv:2107.14171, 2021.

[109] S. Thiébaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza, Decision-theoretic planning with non-Markovian rewards, J. Artif. Intell. Res. **25**, 17 (2006).

[110] A. Camacho, O. Chen, S. Sanner, and S. A. McIlraith, Non-Markovian rewards expressed in ltl: Guiding search via reward shaping, in *Proceedings of the Annual Symposium on Combinatorial Search* (2017).

[111] E. Mizutani and S. E. Dreyfus, Two stochastic dynamic programming problems by model-free actor-critic recurrent-network learning in non-Markovian settings, in *Proceedings of the IEEE International Joint Conference on Neural Networks* (IEEE, Piscataway, NJ, 2004), Vol. 2, pp. 1079–1084.

[112] https://github.com/asikist/nnc.

[113] P. Kidger, R. T. Q. Chen, and T. Lyons, "Hey, that's not an ODE": Faster ODE Adjoints via Seminorms, edited by M. Meila and T. Zhang, in *Proceedings of the 38th International Conference on Machine Learning*, Proceedings of Machine Learning Research Vol. 139 (PMLR, 2021), pp. 5443–5452.