IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 April 2023, accepted 27 May 2023, date of publication 2 June 2023, date of current version 7 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3282435

RESEARCH ARTICLE

Learning Parameterized ODEs From Data

QING LI[1], STEINAR EVJE[1], AND JIAHUI GENG[2], (Graduate Student Member, IEEE)
[1]Department of Energy and Petroleum Engineering, University of Stavanger, 4021 Stavanger, Norway
[2]Department of Electrical Engineering and Computer Science, University of Stavanger, 4021 Stavanger, Norway

Corresponding author: Qing Li (qing.li@uis.no)

This work was supported by the Department of Energy and Petroleum Engineering at the University of Stavanger, under Grant IN-12570.

ABSTRACT In contemporary research, neural networks are being used to derive Ordinary Differential Equations (ODEs) from observations. However, parameterized ODEs pose a more significant challenge than non-parameterized ODEs since the networks are required to understand the roles of the parameters, i.e., the structure of the equations. This paper proposes a novel approach by combining Symbolic Neural Network (S-Net) with ODE Solver to solve this issue. First, S-Net learns the structure of the parameterized ODEs and then predicts the dynamics based on the new parameters with the new initial states. To assess its performance, we compare our approach with a widely used Ordinary Neural Network (O-Net) that directly learns and predicts ODEs. Our numerical experiments demonstrate that our approach outperforms O-Net when applied to the Lotka-Volterra and Lorenz equations.

INDEX TERMS Parameterized ordinary differential equations, neural networks, ODE solver.

I. INTRODUCTION

The combination of deep learning and differential equations is a highly promising research direction. Deep learning can help uncover the underlying differential equations governing the behavior of many systems, such as electromagnetism, aerodynamics, weather prediction, and geophysics. Differential equation-guided network design can significantly improve model interpretability and generalization performance. The potential impact of this approach on scientific discovery and innovation is immense.

Chen et al. [1] introduced Neural Ordinary Differential Equations (NODEs), a new family of deep networks. They used a neural network to parameterize the derivative of the hidden state, serialized the neural network layers and parameters, and used the adjoint ODE method to optimize the neural network instead of back-propagation, which saved memory. Their work inspired research on variants of NODE methods, such as [2] and [3]. Other studies have also explored the potential of neural networks to learn and solve differential equations. For example, Chen et al. [4] proposed the Symplectic Recurrent Neural Network (SRNN) to capture the dynamics of physical systems from regularly observed data. Raissi et al. [5] introduced physics-informed neural networks (PINNs) to solve data-driven solutions and data-driven discovery problems of partial differential equations (PDEs). Long et al. [6], [7] proposed PDE-Net, a feed-forward deep network that predicts the dynamics of complex systems and uncovers the underlying hidden PDE models. PDE-Net implemented Symbolic Neural Network (S-Net) to learn the structure of PDEs and demonstrated powerful learning ability. Similarly, [8], [9] also proved the competitive generalization capability of S-Net. These works show the potential of deep learning in solving PDEs and discovering hidden models.

Parameterized ODEs are a class of ODEs with solutions that vary with parameters, representing multiple dynamics specified by input parameter instances. They have been extensively studied in computational science and engineering domains, such as fluid dynamics and the ideal pendulum system. In critical situations, computing high-fidelity solutions of parameterized ODEs is necessary, either for numerous input parameter instances or initial states. Data-driven methods have been used in recent years to estimate the evolution of dynamical systems over time, including different initial conditions [10] and system parameters [11]. To learn the latent dynamics of complex dynamical processes in computational physics, Lee and Parish [11] proposed encoder-decoder parameterized NODEs (PNODEs). Shimizu and Parish [12] presented the windowed space-time least-squares

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

Petrov-Galerkin method (WST-LSPG) for model reduction of nonlinear parameterized dynamical systems. WST-LSPG divides the time simulation into several windows and sequentially minimizes the discrete-in-time residual within its own unique low-dimensional space-time subspace. Lee and Trask [13] introduced POUNODEs, a new variant of NODEs with evolving model parameters. They modeled the evolution using partition-of-unity networks, allowing for greater flexibility in capturing the dynamics of complex systems.

To the best of our knowledge, the majority of existing approaches for solving parameterized ODEs rely on black-box methods, which employ neural networks directly to simulate its behavior and make predictions. Our proposed approach, on the other hand, aims to gain a deeper understanding of the underlying mechanics of the dynamic system by first learning the structure of the ODEs. This understanding enables us to predict the behavior of the dynamical system more accurately and efficiently for new parameters and initial states compared to black-box methods. The contributions of this work are summarized as follows:

- Our proposed framework combines S-Net with an ODE solver to learn parameterized ODEs. This novel approach enables accurate and efficient modeling of the dynamics of complex systems, even when the properties of the governing equations vary across multiple input parameters.
- To evaluate our approach, we compare its performance with that of a baseline model, O-Net, which represents a black-box approach. We empirically demonstrate the advantages of our proposed framework in terms of both accuracy and interpretability.

The remainder of this paper is organized as follows: Section II presents related works. The considered parameterized ODEs problem and the ODE Solver are briefly introduced in Section III. Section IV illustrates the proposed approach. We evaluate the performance of our method on two case studies: the Lotka-Volterra Equation in Section V and the Lorenz Lotka-Volterra Equation in Section VI. Finally, in Section VII, we summarize our findings and discuss potential avenues for future research.

II. RELATED WORK

Several studies have explored the use of Gaussian process regression to develop tailored functional representations for a given linear operator [14], [15], [16]. However, the local linearization of nonlinear terms in time and prior assumptions of Gaussian process regression limit the representation capacity of the model. Sparse regression, discussed in [17], [18], [19], and [20], overcomes this limitation by developing a dictionary of basic functions and partial derivatives that can accurately represent the data using sparsity-promoting techniques. However, the predictive and expressive capabilities of the dictionary are restricted since the sparse regression method necessitates predefining specific numerical approximations for spatial differentiation.

Mesh-based simulations have recently shown significant progress [21], [22], surpassing grid-based convolutional neural networks (CNNs) in terms of runtime and exhibiting greater adaptivity to the simulation domain. While several methods, such as AntisymmetricRNN [23] and the continuous-time Gated Recurrent Unit with a Bayesian update network [24], have leveraged the stability of underlying differential equations to capture long-term dependencies, they did not address the challenge of learning the expression of the equation to gain a deeper understanding of the underlying mechanism behind the observed data. In another work [25], the authors learned the unknown parameters of the ODE system by constructing certain time-related features, but this method did not address the expressiveness of the equation.

III. KNOWLEDGE

In this section, we'll define parameterized ODEs and differentiate them from non-parameterized ODEs. Additionally, we'll introduce the ODE Solver, a crucial component in our approach.

A. NON-PARAMETERIZED AND PARAMETERIZED ODES

Assume that ODEs take the following generic form, which is usually used to describe some physical dynamics with different functions $f(s, p, r, \alpha, \beta, \gamma)$, $g(s, p, r, \alpha, \beta, \gamma)$ and $h(s, p, r, \alpha, \beta, \gamma)$,

$$\begin{cases} \dfrac{ds}{dt} = f(s, p, r, \alpha, \beta, \gamma), \\ \dfrac{dp}{dt} = g(s, p, r, \alpha, \beta, \gamma), \\ \dfrac{dr}{dt} = h(s, p, r, \alpha, \beta, \gamma). \end{cases} \tag{1}$$

Here, we consider the nonlinear ODE system with independent variables s, p, and r, and initial states s_0, p_0, and r_0 at time t_0. The system also involves parameters α, β, and γ, where $t \in (0, T]$ represents the time interval of interest.

In non-parameterized ODEs, parameters such as α, β, and γ remain fixed during both model training and prediction, whether they are known or unknown. Thus, we assume that $\alpha = \alpha_1$, $\beta = \beta_1$, and $\gamma = \gamma_1$, and denote the observations as:

$$\begin{aligned} Z^{\text{fixed}} = \Big\{ \Big(& s(t_i; s_0^j, p_0^j, r_0^j, \alpha_1, \beta_1, \gamma_1), p(t_i; s_0^j, p_0^j, r_0^j, \alpha_1, \beta_1, \\ & \times \gamma_1), r(t_i; s_0^j, p_0^j, r_0^j, \alpha_1, \beta_1, \gamma_1) \Big) \Big| i = 1, \ldots, N_{\text{obs}}, \\ & \times j = 1, \ldots, M \Big\}. \end{aligned} \tag{2}$$

The observations are denoted by a set of $N_{\text{obs}} > 0$ time points, and the number of initial states is denoted by $M > 0$. If $(s_0^*, p_0^*, r_0^*) \in \{(s_0^j, p_0^j, r_0^j) | j = 1, \ldots M\}$, we predict $\big(s(t_*; s_0^*, p_0^*, r_0^*, \alpha_1, \beta_1, \gamma_1), p(t_*; s_0^*, p_0^*, r_0^*, \alpha_1, \beta_1, \gamma_1), r(t_*; s_0^*, p_0^*, r_0^*, \alpha_1, \beta_1, \gamma_1)\big)$ at $t_* > t_{N_{\text{obs}}}$. Otherwise, we predict the dynamics at $t_* \in (0, T]$. The non-parameterized ODEs problem has been widely studied in [2], [3], and [4].

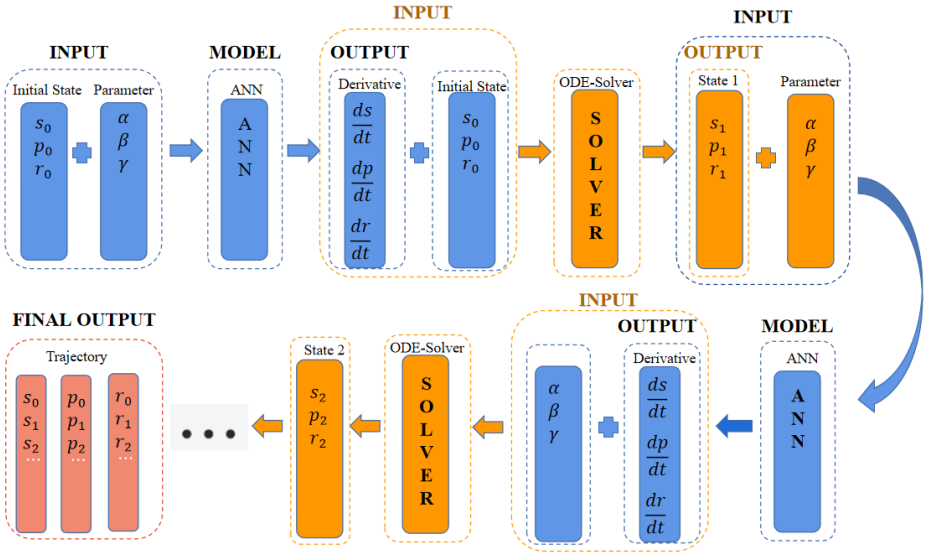


FIGURE 1. Schematic diagram of the framework. Initial states s_0, p_0, r_0 and parameters α, β, γ are fed into the ANN model to predict the derivatives of s, p, and r with respect to time. These derivatives combined with initial states s_0, p_0, r_0 will be further fed into the ODE Solver to obtain the next state s_1, p_1, r_1, and so on. Finally we will get the trajectories of $S = \{s_0, s_1, \ldots s_T\}$, $P = \{p_0, p_1, \ldots, p_T\}$ and $R = \{r_0, r_1, \ldots, r_T\}$.

Concerning the parameterized ODEs problem, the observation data

$$Z^{\text{variable}} = \Big\{\Big(s(t_i; s_0^j, p_0^j, r_0^j, \alpha_k, \beta_k, \gamma_k), p(t_i; s_0^j, p_0^j, r_0^j, \times \alpha_k, \beta_k, \gamma_k), r(t_i; s_0^j, p_0^j, r_0^j, \alpha_k, \beta_k, \gamma_k)\Big) \Big| i = 1, \times \ldots, N_{\text{obs}}, j = 1, \ldots, M, k = 1, \ldots, K\Big\}, \tag{3}$$

is generated by different parameters $(\alpha_k, \beta_k, \gamma_k)$ at some specified time t_i on different initial states (s_0^j, p_0^j, r_0^j). Here, we define $K > 0$ as the number of parameters, $N_{\text{obs}} > 0$ as the number of observation time points, and $M > 0$ as the number of initial states.

In the scenario of parameterized ODEs, if $(s_0^*, p_0^*, r_0^*) \in \{(s_0^j, p_0^j, r_0^j) | j = 1, \ldots M\}$ and $(\alpha_*, \beta_*, \gamma_*) \in \{(\alpha_k, \beta_k, \gamma_k) | k = 1, \ldots K\}$, we predict $\big(s(t_*; s_0^*, p_0^*, r_0^*, \alpha_*, \beta_*, \gamma_*), p(t_*; s_0^*, p_0^*, r_0^*, \alpha_*, \beta_*, \gamma_*), r(t_*; s_0^*, p_0^*, r_0^*, \alpha_*, \beta_*, \gamma_*)\big)$ at $t_* > t_{N_{\text{obs}}}$. Otherwise, we predict the dynamics at $t_* \in (0, T]$.

Parameterized ODEs find applications in various scenarios. For instance, in [26], different parameters associated with the ODEs represent distinct strategies for regulating cancer tumor progression behavior. Given patient data describing the evolution of a tumor, it would be advantageous to obtain the functional form of the ODE system underlying this complex system, both for the state variables and parameters. Subsequently, once we have constructed the functional form of the ODE system using a neural network, we can directly compute solutions of the ODE for new initial states and parameter sets. Traffic flow problems represent another common application. Changes in traffic flow over time at two specific locations x_1, x_2 in a city can be modeled as

$$\begin{cases} \dfrac{ds}{dt} = f(s, p, \boldsymbol{w}), \\ \dfrac{dp}{dt} = g(s, p, \boldsymbol{w}). \end{cases} \tag{4}$$

In this context, the number of cars at positions x_1 and x_2 are represented by s and p, respectively, and the traffic flow changes over time at these locations are modeled using functions f and g. Here, $t \in (0, 24]$ denotes a day for a cycle, and the vector $\boldsymbol{w}$ represents external factors that influence traffic conditions, such as weather, temperature, and so on. We assume that the external factors $\boldsymbol{w}$ change daily, but the expressions for f and g remain constant as the mechanism by which each factor affects traffic flow remains the same. Once we obtain the correct analytical formulas, we can accurately predict the system.

This approach is effective for discrete chaotic maps, using the Logistic map as an example. The Logistic map is a straightforward, one-dimensional discrete-time dynamical system characterized by the following equation:

$$x_{n+1} = \gamma x_n (1 - x_n). \tag{5}$$

Here, x is the state variable at time step n, and γ is a parameter. We can employ the proposed method to learn the expression on the right side of (5), which means that the input of the S-Net consists of both γ and x.

B. ODE SOLVER

The ODE Solver, also known as a numerical integrator, iterates a numerical scheme to obtain improved approximations of the solution. There exist works dedicated to designing numerical integrators that produce more accurate solutions [27], [28]. Typically, the ODE Solver is employed to

approximate the true solution of the form $\frac{dz}{dt} = \boldsymbol{F}(z, t)$ based on the initial state z_0, where $\boldsymbol{F}(z, t)$ is a vector function. Considering the parameters are variable, parameters α, β, and γ can not be embedded in $\boldsymbol{F}$ in our problem. So $\boldsymbol{F}$ can be expressed in the form of $\boldsymbol{F}(z, t, \alpha, \beta, \gamma) = (f(z, t, \alpha, \beta, \gamma), g(z, t, \alpha, \beta, \gamma), h(z, t, \alpha, \beta, \gamma))$ and $z(t; z_0, \alpha, \beta, \gamma) = \{s(t; z_0, \alpha, \beta, \gamma), p(t; z_0, \alpha, \beta, \gamma), r(t; z_0, \alpha, \beta, \gamma)\}$.

In this paper, we begin with Euler integrator [29], a popular choice due to its simplicity, as demonstrated in previous works [1], [2], [13]. Euler method also facilitates a fair comparison with black-box methods. Given the state z_n at time point $t_n = t_0 + n\Delta t$, the state at the next time point can be computed using the following formula:

$$z_{n+1} = z_n + \Delta t \boldsymbol{F}(z_n, t_n, \alpha, \beta, \gamma). \tag{6}$$

The time step size, denoted by Δt, is a crucial parameter in numerical methods for solving ODEs. Specifically, Euler method can generate unstable solutions for stiff ODE systems unless a very small Δt is employed, as noted in [30]. To maintain solution stability in our experiments, we meticulously select an appropriate value for Δt. In Appendix A, we present a rigorous proof of the convergence of Euler method, which is essential for understanding the accuracy of numerical solutions. Our approach can also be adapted to learn unknown functions using semi-implicit solvers with appropriate modifications. In Appendix B, we provide a brief explanation of how to make the necessary adjustments to fit semi-implicit solvers.

IV. OUR APPROACH

A. PIPELINE

We use initial states $Z_0 = \left\{z_0^j | j = 1, \ldots, M\right\}$ and parameters $W = \{(\alpha_k, \beta_k, \gamma_k) | k = 1, \ldots, K\}$ for the training process. Following the work [1], we let the right side of ODEs be a parametric function $\boldsymbol{F}_{\boldsymbol{\theta}}(z, \alpha, \beta, \gamma)$, where $z = (s, p, r)$ and $\boldsymbol{\theta}$ is the vector of parameters of the neural network. After training, trajectories $\hat{Z} = \left\{\hat{z}_{\boldsymbol{\theta}}(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k)\right\}_{i=1}^{N_{\text{obs}}}$ are generated based on initial state $z_0^j \in Z_0$, parameters $(\alpha_k, \beta_k, \gamma_k) \in W$ and $\boldsymbol{\theta}$, i.e.,

$$\begin{aligned}\hat{z}_{\boldsymbol{\theta}}(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k) = &\hat{z}_{\boldsymbol{\theta}}(t_{i-1}; z_0^j, \alpha_k, \beta_k, \gamma_k) \\ &+ \Delta t \boldsymbol{F}_{\boldsymbol{\theta}}\left(\hat{z}_{\boldsymbol{\theta}}(t_{i-1}; z_0^j, \alpha_k, \beta_k, \gamma_k),\right. \\ &\left.\times \alpha_k, \beta_k, \gamma_k\right), \end{aligned} \tag{7}$$

where $i = 1, \ldots, N_{\text{obs}}$.

Given a series of observations $Z = \left\{z(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k) | i \in \{1, \ldots, N_{\text{obs}}\}, j \in \{1, \ldots, M\}, k \in \{1, \ldots, K\}\right\}$, we estimate the parameter $\boldsymbol{\theta}$ by minimizing the error between the observed trajectories Z and predicted trajectories $\hat{Z}$, denoted as

$$L^{\text{data}} = \sum_{i=1}^{N_{\text{obs}}} \sum_{j=1}^{M} \sum_{k=1}^{K} \|\hat{z}_{\boldsymbol{\theta}}(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k)\}$$

Algorithm 1 Algorithm for Solving the Parameterized ODEs Problem

Input: N_{obs}: the number of observation time points; Integrator: ODE Solver; Z_0: initial state set; W: parameter set; $\boldsymbol{F}_{\boldsymbol{\theta}}(s, p, r, \alpha, \beta, \gamma)$: neural network; $\boldsymbol{\theta}_0$: initial parameters of neural network; Z: observed trajectories of initial state Z_0 and parameters W; n_{epochs}: the number of epoch; Δt: time step; L: loss function

Output: $\hat{Z}$: estimated trajectories of initial state Z_0 and parameters W; $\boldsymbol{F}_{\boldsymbol{\theta}^*}$: the trained neural network

```
θ = θ_0
for i = 1, ..., n_epochs do
    Ẑ = []
    for j = 1, ..., M do
        Select one element (s_0^j, p_0^j, r_0^j) from Z_0
        for k = 1, ..., K do
            Select one element (α_k, β_k, γ_k) from W
            for i = 1, ..., N_obs do
                (ds/dt, dp/dt, dr/dt) =
                  F_θ(s_{i-1}^j, p_{i-1}^j, r_{i-1}^j, α_k, β_k, γ_k)
                s_i^j = integrator(s_{i-1}^j, Δt, ds/dt)
                p_i^j = integrator(p_{i-1}^j, Δt, dp/dt)
                r_i^j = integrator(r_{i-1}^j, Δt, dr/dt)
                Add (s_i^j, p_i^j, r_i^j) to Ẑ.
            end
        end
    end
    Loss = L(Ẑ, Z)
    Update parameters θ based on Loss
end
```

$$- z(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k)\}\|_2. \tag{8}$$

Inspired by [7], we also add the regularization term $L^{\text{S-Net}}$ to the loss function to avoid overfitting and enhance the generalization capability of the model. $L^{\text{S-Net}}$ is defined as (9),

$$L^{\text{S-Net}} = \sum_{p \in \theta_{L^{\text{S-Net}}}} l_1^s(p), \tag{9}$$

where $l_1^s(x) = \begin{cases} |x| - \frac{s}{2}, & \text{if } |x| > s \\ \frac{1}{2s}x^2, & \text{otherwise} \end{cases}$ and $s = 0.001$.

L^{data} and $L^{\text{S-Net}}$ constitute the loss function L, that is

$$L = L^{\text{data}} + L^{\text{S-Net}}. \tag{10}$$

Figure 1 provides a detailed illustration of how to generate a trajectory based on the initial state $z_0 = (s_0, p_0, r_0) \in Z_0$ and parameters $(\alpha, \beta, \gamma) \in W$. Herein, we denote $s_i = s(t_i; z_0, \alpha, \beta, \gamma)$, $p_i = p(t_i; z_0, \alpha, \beta, \gamma)$ and $r_i = r(t_i; z_0, \alpha, \beta, \gamma)$ to represent the states of s, p and r at time t_i. Algorithm 1 illustrates the pipeline for solving the parameterized ODE problem. We back-propagate the loss and use it to update the parameter $\boldsymbol{\theta}$, which allows us to obtain the best $\boldsymbol{\theta}^*$. This value represents the ODE system and enables us to predict the dynamics based on new initial states and parameters.

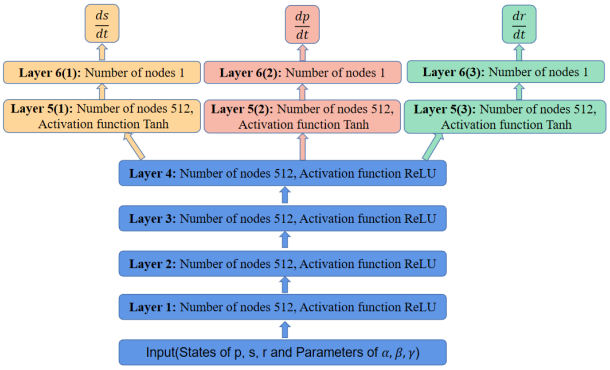


FIGURE 2. Schematic diagram of the O-Net. States of s, p, and r and parameters of α, β, γ are the input vector. The input vector goes through a four-layer network structure. Each layer has 512 units and a relu function. There are three parts for $\frac{ds}{dt}$, $\frac{dp}{dt}$ and $\frac{dr}{dt}$ respectively after four layers. Each part has two layers: the first layer has 512 units with the tanh activation function, and the second layer outputs the derivatives of $\frac{ds}{dt}$, $\frac{dp}{dt}$ and $\frac{dr}{dt}$.

Algorithm 2 O-Net

Input: $s, p, r, \alpha, \beta, \gamma \in \mathbb{R}$
Output: $\frac{ds}{dt}, \frac{dp}{dt}, \frac{dr}{dt}$

$y_1 = \text{relu}(\boldsymbol{w}_1^T(s, p, r, \alpha, \beta, \gamma)^T + \boldsymbol{b}_1)$,
$\boldsymbol{w}_1 \in \mathbb{R}^{6\times512}, \boldsymbol{b}_1 \in \mathbb{R}^{512\times1}$;
$y_2 = \text{relu}(\boldsymbol{w}_2^T y_1 + \boldsymbol{b}_2)$,
$\boldsymbol{w}_2 \in \mathbb{R}^{512\times512}, \boldsymbol{b}_2 \in \mathbb{R}^{512\times1}$;
$y_3 = \text{relu}(\boldsymbol{w}_3^T y_2 + \boldsymbol{b}_3)$,
$\boldsymbol{w}_3 \in \mathbb{R}^{512\times512}, \boldsymbol{b}_3 \in \mathbb{R}^{512\times1}$;
$y_4 = \text{relu}(\boldsymbol{w}_4^T y_3 + \boldsymbol{b}_4)$,
$\boldsymbol{w}_4 \in \mathbb{R}^{512\times512}, \boldsymbol{b}_4 \in \mathbb{R}^{512\times1}$;
$y_{51} = \tanh(\boldsymbol{w}_{51}^T y_4 + \boldsymbol{b}_{51})$,
$\boldsymbol{w}_{51} \in \mathbb{R}^{512\times512}, \boldsymbol{b}_{51} \in \mathbb{R}^{512\times1}$;
$y_{52} = \tanh(\boldsymbol{w}_{52}^T y_4 + \boldsymbol{b}_{52})$,
$\boldsymbol{w}_{52} \in \mathbb{R}^{512\times512}, \boldsymbol{b}_{52} \in \mathbb{R}^{512\times1}$;
$y_{53} = \tanh(\boldsymbol{w}_{53}^T y_4 + \boldsymbol{b}_{53})$,
$\boldsymbol{w}_{53} \in \mathbb{R}^{512\times512}, \boldsymbol{b}_{53} \in \mathbb{R}^{512\times1}$;
$\frac{ds}{dt} = \boldsymbol{w}_{61}^T y_{51} + \boldsymbol{b}_{61}, \boldsymbol{w}_{61} \in \mathbb{R}^{512\times1}, \boldsymbol{b}_{61} \in \mathbb{R}$;
$\frac{dp}{dt} = \boldsymbol{w}_{62}^T y_{52} + \boldsymbol{b}_{62}, \boldsymbol{w}_{62} \in \mathbb{R}^{512\times1}, \boldsymbol{b}_{62} \in \mathbb{R}$;
$\frac{dr}{dt} = \boldsymbol{w}_{63}^T y_{53} + \boldsymbol{b}_{63}, \boldsymbol{w}_{63} \in \mathbb{R}^{512\times1}, \boldsymbol{b}_{63} \in \mathbb{R}$;

B. ARTIFICIAL NEURAL NETWORKS:O-NET AND S-NET

In Algorithm 1, we can use any form of the neural network, but we adopt the O-Net as our baseline, which is commonly used for regression tasks without exploring the underlying mechanism connecting inputs and outputs. However, to learn the combinations of different variables and parameters, we design S-Net, inspired by [6], [7], [8], and [9]. Unlike O-Net, S-Net first learns the analytic expression of ODEs and then predicts the dynamics. Notably, S-Net has no activation functions, and the most significant difference between O-Net and S-Net is the mapping between units in the layers.

1) O-NET

O-Net uses fully connected layers to learn the mapping from low to high-dimensional space and uses activation functions to learn the nonlinear relationship. Consider an O-Net with four shared layers and two unique layers for $\frac{ds}{dt}$, $\frac{dp}{dt}$ and $\frac{dr}{dt}$, as illustrated in Figure 2. To better understand O-Net, which is usually used for regression, we present a mathematical description in Algorithm 2 showing how O-Net is constructed.

O-Net is ineffective in solving variable parameters ODEs problems as it fails to capture the role of parameters. Although both parameters and variables are treated as input features, they behave differently. Specifically, at each time point, the input vector is $(z, \alpha, \beta, \gamma)$, where z changes over time while (α, β, γ) do not. As a result, it is challenging for α, β, γ to find appropriate weights in a fully connected neural network, making it difficult for O-Net to effectively learn the underlying dynamics of the system.

2) S-NET

S-Net possesses the ability to learn analytical expressions that can generalize to new domains effectively. The primary distinction between O-Net and S-Net lies in their layer-unit mapping, where S-Net is designed to capture the interaction between various variables and parameters efficiently. This is accomplished through two types of transformations: identity and linear combination maps, which enable S-Net to learn the underlying dynamics of the system accurately.

As an example, consider a two-layer S-Net that aims to learn the function $f \in \boldsymbol{F}$ in (1), as illustrated in Figure 3. With the appropriate number of layers, S-Net can represent all polynomials of the variables $(s, p, r, \alpha, \beta, \gamma)$. Herein, we only use addition and multiplication operators in S-Net. If necessary, we can add more operations to the S-Net to increase the capacity of the network. To better understand S-Net, we present an example in Algorithm 3 showing how S-Net is constructed. Particularly, we illustrate the learning process of $\alpha s - \beta r$ using S-Net in (11), (12), (13), (14) , and (15).

$$(\delta_1, \varepsilon_1)^T = \boldsymbol{w}_1 \times \left[s, p, r, \alpha, \beta, \gamma\right]^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} s \\ p \\ r \\ \alpha \\ \beta \\ \gamma \end{bmatrix}, \tag{11}$$

$$f_1 = \delta_1 \varepsilon_1 = \alpha s, \tag{12}$$

$$(\delta_2, \varepsilon_2)^T = \boldsymbol{w}_2 \times \left[s, p, r, \alpha, \beta, \gamma, \alpha s\right]^T = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} s \\ p \\ r \\ \alpha \\ \beta \\ \gamma \\ \alpha s \end{bmatrix}, \tag{13}$$

$$f_2 = \delta_2 \varepsilon_2 = \beta r, \tag{14}$$

$$\boldsymbol{w}_3 \times \left[s, p, r, \alpha, \beta, \gamma, \alpha s, \beta r\right]^T$$

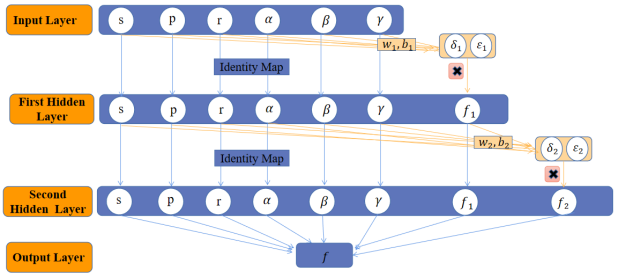


FIGURE 3. Schematic diagram of the S-Net. The identity map directly transfers $s, p, r, \alpha, \beta, \gamma$ from input layer to the first hidden layer. The linear combination chooses two elements remarked by δ_1 and ε_1 from the input vector using w_1 and b_1. The expressions $f_1 = \delta_1\varepsilon_1$ and $f_1 = \frac{\delta_1}{\varepsilon_1}$ correspond to the multiplication and division of two elements, respectively. We only implement the multiplication operation in this work. Apart from s, p, r, α, β and γ gotten by the identity map, f_1 will also be input to the second hidden layer. Similar to the first hidden layer, we get the further combination $f_2 = \delta_2\varepsilon_2$ by w_2 and b_2. Finally, we obtain the analytic expression of function f. It is necessary to enforce the sparsity of S-Net since it helps reduce overfitting and enables more robust predictions.

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} s \\ p \\ r \\ \alpha \\ \beta \\ \gamma \\ \alpha s \\ \beta r \end{bmatrix}$$

$$= \alpha s - \beta r. \tag{15}$$

Our analysis of $\alpha s - \beta r$ reveals that the sparsity of S-Net is a critical factor. Therefore, we introduced a regularization term into the loss function to promote model sparsity, as discussed in section IV-A.

V. NUMERICAL STUDIES: LOTKA-VOLTERRA EQUATION

The Lotka-Volterra model is frequently used to describe the dynamics of ecological systems where two species interact. [31] shows that cannibalism has both positive and negative effects on the stability of the Lotka-Volterra predator-prey model. It depends on the dynamic behaviors of the original system.

In this section, we consider Lotka-Volterra Equation (16), where different ecological systems have different parameters of α, β, δ and γ and initial states of x_0 and y_0,

$$\begin{cases} \dfrac{dx}{dt} = \alpha x - \beta xy, \\ \dfrac{dy}{dt} = -\delta y + \gamma xy. \end{cases} \tag{16}$$

The Euler integrator simulates ground truth trajectories in both training and testing stages with the time step $\Delta t = 0.1$, which empirically meets the stability requirements. The training and testing data consist of 150 and 33 trajectories, respectively, each of which starts from random initial state (x_0, y_0) in the interval $[0.6, 1.4]$ and parameters of $(\alpha, \beta, \delta, \gamma)$ in the interval $([1.0, 2.0], [0.5, 1.5], [2.5, 3.5], [0.5, 1.5])$ respectively. There is no overlap between the training and test data.

Algorithm 3 S-Net

Input: $s, p, r, \alpha, \beta, \gamma \in \mathbb{R}$
Output: F

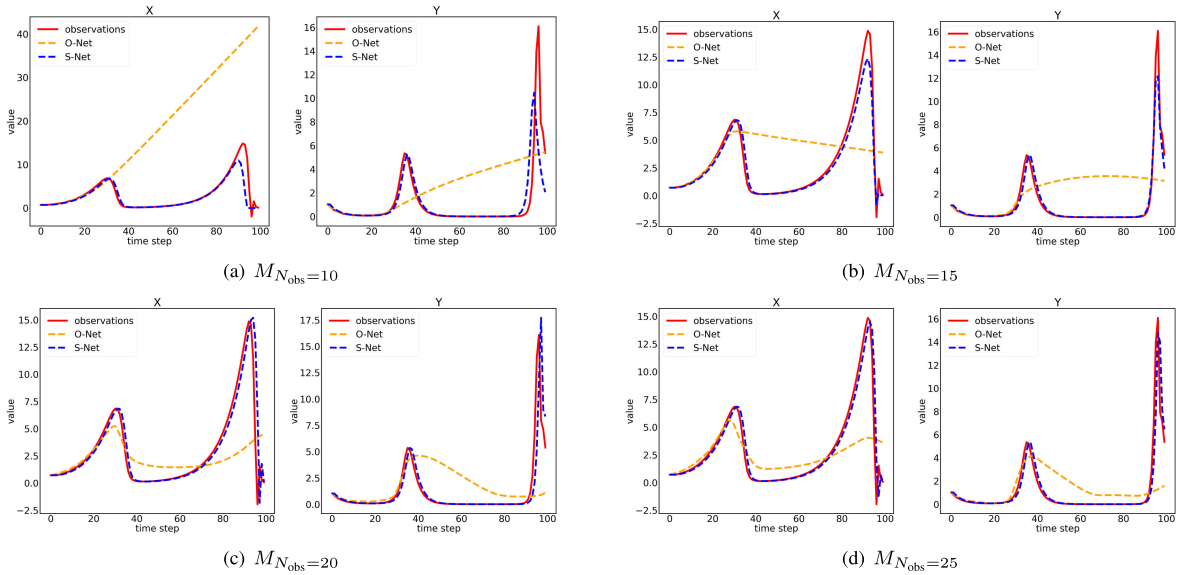
$(\delta_1, \varepsilon_1)^T = \boldsymbol{w}_1(s, p, r, \alpha, \beta, \gamma)^T + \boldsymbol{b}_1$,
$\boldsymbol{w}_1 \in \mathbb{R}^{2\times 6}, \boldsymbol{b}_1 \in \mathbb{R}^{2\times 1}$;
$f_1 = \delta_1\varepsilon_1$;
$(\delta_2, \varepsilon_2)^T = \boldsymbol{w}_2(s, p, r, \alpha, \beta, \gamma, f_1)^T + \boldsymbol{b}_2$,
$\boldsymbol{w}_2 \in \mathbb{R}^{2\times 7}, \boldsymbol{b}_2 \in \mathbb{R}^{2\times 1}$;
$f_2 = \delta_2\varepsilon_2$;
$F = \boldsymbol{w}_3(s, p, r, \alpha, \beta, \gamma, f_1, f_2)^T + b_3$,
$\boldsymbol{w}_3 \in \mathbb{R}^{1\times 8}, b_3 \in \mathbb{R}$;

The performance of O-Net and S-Net could get improved with a reasonable increase in the number of time points, i.e., N_{obs}, in the training data. We set four group experiments on training data with $N_{\text{obs}} = 10, 15, 20$, and 25. If the number of observation points is too large, it will also hurt the effectiveness of the model. Once there are too many observations, the accumulated error will be too large for one trajectory, which is not conducive to model training. We train S-Net with the L-BFGS optimizer [32] and use maximum iterations of 30000, a batch size of 150. We train O-Net with ADAM optimizer [33] for 5000 epochs using a batch size of 50, and a learning rate of 5e-3. We predict trajectories based on various initial states and parameters on $t \in (0, 10]$.

A. RESULTS AND DISCUSSIONS

Our study shows that S-Net is able to effectively recover the ODEs for the unknown variables of x and y, as well as the parameters α, β, δ, and γ. We use the notation $M_{N_{\text{obs}}}$ to represent the model trained on training data with N_{obs} time points. The results, summarized in Table 1, demonstrate that we are able to accurately recover the terms of (16). We observe that increasing the number of time points within a specific range results in higher model accuracy. Specifically, we find that when $M_{N_{\text{obs}}} = 25$, the coefficients of αx, βxy, δy, and γxy match those of the true equation. Additionally, we find that the terms not included in (16) have relatively small coefficients.

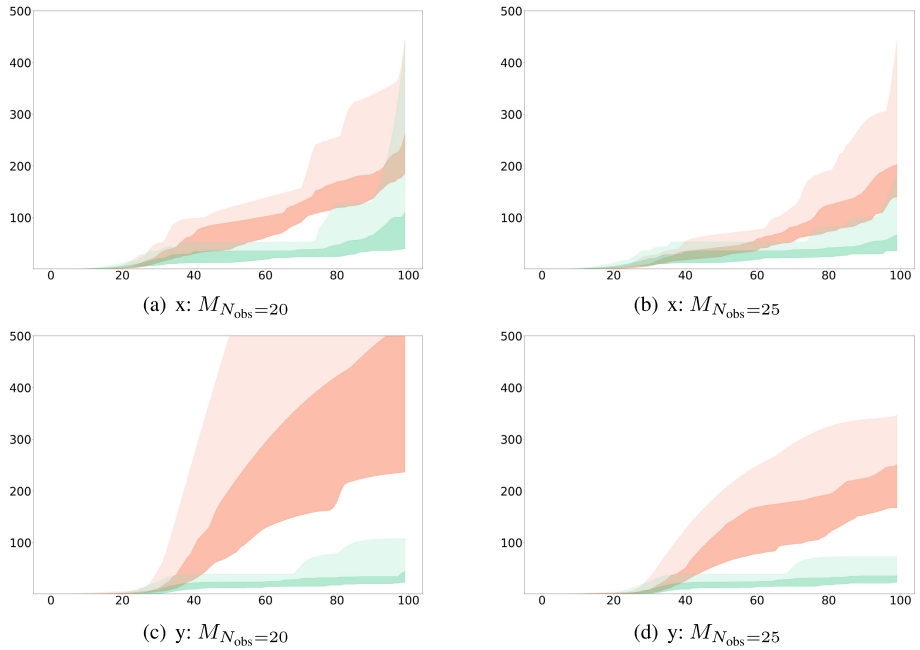
To evaluate the ability of the models to generate correct trajectories for new initial states and parameters, we feed 33 testing trajectories into the well-trained models and obtained predicted trajectories. Figure 4 shows the results of S-Net and O-Net with $N_{\text{obs}} = 10, 15, 20, 25$ for a specific test example $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (0.71468, 1.01860, 1.10023, 1.17939, 2.78173, 1.18328)$. We find that S-Net can accurately predict the trajectory in the period of $t \in [0, 80\Delta t] = (0, 8]$, but its accuracy decreases when $t \in (8, 10]$. However, we observe that increasing the number of time points used in the training process improves the prediction of the S-Net of long-time dynamics. Specifically, S-Net trained on $M_{N_{\text{obs}}} = 25$ predicts with higher accuracy for $t \in (8, 10]$ than S-Net trained on $M_{N_{\text{obs}}} = 10$. On the other hand, O-Net performs poorly in all four cases, even though it performs well in the early stages



(a) $M_{N_{\text{obs}}=10}$ (b) $M_{N_{\text{obs}}=15}$

(c) $M_{N_{\text{obs}}=20}$ (d) $M_{N_{\text{obs}}=25}$

FIGURE 4. Lotka-Volterra: Testing results on $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (0.71468, 1.01860, 1.10023, 1.17939, 2.78173, 1.18328)$ by S-Net and O-Net with time points 10, 15, 20 and 25 for the x-component (left) and y-component (right). In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 10]$, and the vertical axis shows the values. The solid red line is the ground truth. The blue and orange dashed lines show the O-Net and S-Net results, respectively.



(a) x: $M_{N_{\text{obs}}=20}$ (b) x: $M_{N_{\text{obs}}=25}$

(c) y: $M_{N_{\text{obs}}=20}$ (d) y: $M_{N_{\text{obs}}=25}$

FIGURE 5. Lotka-Volterra: Prediction errors of the O-Net(orange) and S-Net(green) with different training time points 20 and 25. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 10]$, and the vertical axis shows the errors. The banded curves indicate the 25% – 100% percentile of the relative errors among 33 test samples. The dark regions indicate the 25% – 75% percentile of the relative error, which shows that S-Net performs significantly better than O-Net. The upper row is for variable x and the bottom low is for variable y.

for a short period, its predictions often deviate significantly from the actual trajectory. In some cases, O-Net even predicts trends that are opposite to the actual situation. Therefore, we conclude that S-Net has a stronger generalization ability than O-Net.

We define the error between observations Z and predictions $\hat{Z}$ as $\epsilon = \sqrt{||Z - \hat{Z}||^2}$. The error plots are shown in Figure 5 for two different time points, 20 and 25. In both cases, the error of O-Net is significantly larger than that of S-Net. For example, in Figure 5(c), we can observe that the error of O-Net sharply increases over time and soon reaches 500. In contrast, the error of S-Net increases at a relatively slower rate and reaches a maximum of approximately 100. Specifically, Table 2 shows the error of S-Net and O-Net for different models. The error of O-Net can be five or six times

TABLE 1. Lotka-Volterra identification with different time points.

True ODEs	$\frac{dx}{dt} = \alpha x - \beta xy,$ $\frac{dy}{dt} = -\delta y + \gamma xy,$
$M_{N_{\text{obs}}=10}$	$\frac{dx}{dt} = 0.996\alpha x - 0.988\beta xy + 0.00174x^2 - 0.00116\beta\delta y - 0.00105\beta x$ $\frac{dy}{dt} = -0.997\delta y + 0.987\gamma xy - 0.00101xy + 0.000967\gamma y + 0.000939x^2 y$
$M_{N_{\text{obs}}=15}$	$\frac{dx}{dt} = 0.998\alpha x - 0.989\beta xy - 0.00165\beta x^2 y - 0.00115\beta x - 0.000996xy$ $\frac{dy}{dt} = -0.997\delta y + 0.987\gamma xy - 0.00101xy + 0.000967\gamma y + 0.000939x^2 y$
$M_{N_{\text{obs}}=20}$	$\frac{dx}{dt} = 0.999\alpha x - 0.996\beta xy - 0.00110xy - 0.000948\alpha\beta xy - 0.000707\beta x$ $\frac{dy}{dt} = -0.998\delta y + 1.0\gamma xy - 0.000621\delta\gamma y + 0.000530y + 0.000442\alpha y$
$M_{N_{\text{obs}}=25}$	$\frac{dx}{dt} = 1.0\alpha x - 1.0\beta xy + 0.000764\alpha - 0.000683\beta y - 0.000652\beta x$ $\frac{dy}{dt} = -1.0\delta y + 1.0\gamma xy - 0.0377\gamma y + 0.0203y + 0.0112\gamma^2 y$

TABLE 2. MSE of S-Net and O-Net for equations of different models.

	Lotka–Volterra			Lorenz	
Model	O-Net	S-Net	Model	O-Net	S-Net
$M_{N_{\text{obs}}=10}$	302.70	58.99	$M_{N_{\text{obs}}=4}$	3119.18	415.95
$M_{N_{\text{obs}}=15}$	68.16	56.18	$M_{N_{\text{obs}}=6}$	402.89	8.87
$M_{N_{\text{obs}}=20}$	32.81	10.35	$M_{N_{\text{obs}}=10}$	417.71	2.29
$M_{N_{\text{obs}}=25}$	12.95	2.20	$M_{N_{\text{obs}}=20}$	1031.48	1.98

that of S-Net when $M_{N_{\text{obs}}}$ is 10 or 25, respectively. Clearly, S-Net performs significantly better than O-Net, suggesting that proper discretization is crucial when the ODE structure is unknown. Appendix C presents the findings of three other cases using different models obtained by $M_{N_{\text{obs}}} = 10$, 15, 20, and 25, respectively.

VI. NUMERICAL STUDIES: LORENZ EQUATION

In this section, we consider Lorenz Equations (17) where different ecological systems have different parameters σ, ρ and β and initial states x_0, y_0 and z_0.

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x), \\ \frac{dy}{dt} = x(\rho - z) - y, \\ \frac{dz}{dt} = xy - \beta z. \end{cases} \tag{17}$$

Both the training and testing data consist of 150 and 60 trajectories, respectively, simulated by the Euler integrator with a time step of $\Delta t = 0.01$. The initial states and parameters are randomly selected from the intervals [0.8, 1.2], [0, 0.2], [0, 0.3], [5, 15], [23, 33], and [2, 3], respectively, without any overlap between the two sets. We conduct four group experiments on the training data with different numbers of time points: 4, 6, 10, and 20. For training S-Net, we use the L-BFGS optimizer with a maximum of 50,000 iterations and a batch size of 150. For training O-Net, we use the ADAM optimizer with a learning rate of 5e-3, a batch size of 100, and 5,000 epochs. We predict trajectories for various initial states and parameters on the time interval $t \in (0, 1]$.

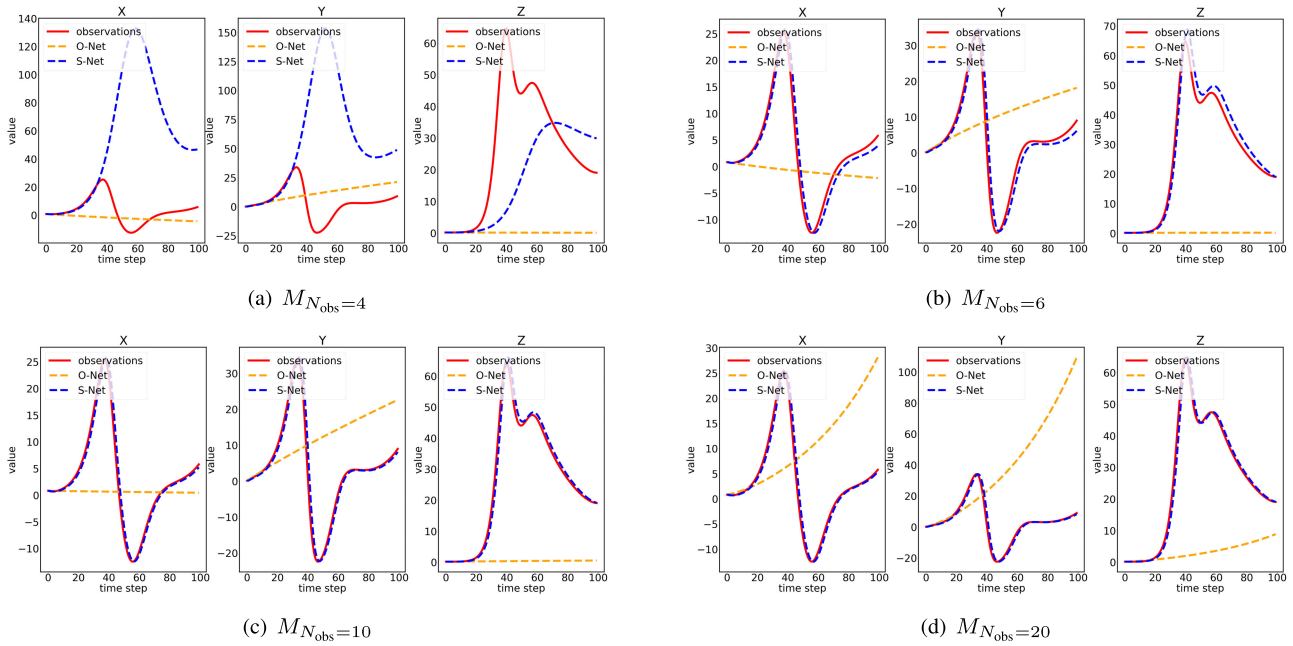
TABLE 3. Lorenz identification with different models.

True ODEs	$\frac{dx}{dt} = \sigma y - \sigma x,$ $\frac{dy}{dt} = x\rho - xz - y,$ $\frac{dz}{dt} = xy - \beta z,$
$M_{N_{\text{obs}}=4}$	$\frac{dx}{dt} = 0.999\sigma y - 0.999\sigma x - 0.003\sigma + 0.000403\sigma z - 0.0004\beta$ $\frac{dy}{dt} = 1.0x\rho - 0.845xz - 0.579y - 0.0928xy - 0.0119\rho y$ $\frac{dz}{dt} = -2.5z + 0.595y + 0.0614x - 0.0126\beta + 0.0122\sigma$
$M_{N_{\text{obs}}=6}$	$\frac{dx}{dt} = 0.999\sigma y - 0.999\sigma x - 0.00183\sigma - 0.000529\beta + 0.000296\sigma z$ $\frac{dy}{dt} = 1.0x\rho - 0.929xz - 0.904y - 0.0183xy - 0.00282\rho y$ $\frac{dz}{dt} = -0.966\beta z + 0.365\beta xy + 0.358\beta y + 0.172\beta x - 0.130\beta^2 y$
$M_{N_{\text{obs}}=10}$	$\frac{dx}{dt} = 0.997\sigma y - 0.996\sigma x - 0.00158\sigma - 0.000392\sigma x^2 - 0.000362x$ $\frac{dy}{dt} = 1.0x\rho - 0.979xz - 0.980y - 0.00403xy - 0.00124\beta x$ $\frac{dz}{dt} = -0.993\beta z + 0.988xy + 0.00173\beta xy + 0.00101y + 0.000909y^2$
$M_{N_{\text{obs}}=20}$	$\frac{dx}{dt} = 1.0\sigma y - 1.0\sigma x - 0.000905y - 0.000392 - 0.000282xy$ $\frac{dy}{dt} = 1.0x\rho - 0.998xz - 0.997y - 0.000986x - 0.000708xy$ $\frac{dz}{dt} = 0.999xy - 0.997\beta z + 0.000818x^2 + 0.000562yz - 0.000405z$

A. RESULTS AND DISCUSSIONS

Table 3 presents the capability of the trained S-Net to identify the underlying ODE model, showing the top five terms of coefficient weights recovered by S-Net with certain accuracy. Notably, when $M_{N_{\text{obs}}} = 20$, the coefficients of σy, σx, and $x\rho$ match those of the true equation with high precision. The coefficients of the remaining four terms, namely xz, y, xy, and βz, deviate only slightly from the true values, with a maximum difference of 0.01. These results demonstrate the effectiveness of S-Net in recovering the underlying ODE model.

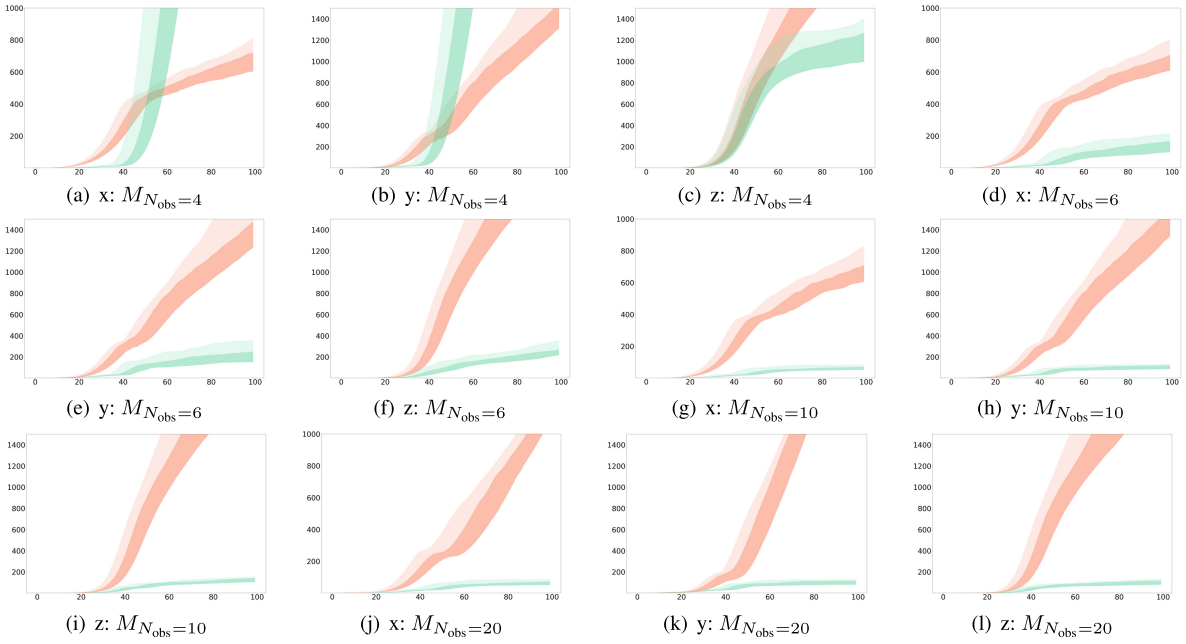
We evaluate the predictive performance of S-Net and O-Net on (17) using 60 testing trajectories and obtain corresponding predictions from the trained models. Figure 6 shows the results of S-Net and O-Net with $N_{\text{obs}} = 4, 6, 10, 20$ for a specific test example $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.82841, 0.14785, 0.17099, 12.39551, 32.03720, 2.67205)$. We predict the trajectories of x, y and z on $t \in (0, 100\Delta t] = (0, 1]$ using the learned models. When $M_{N_{\text{obs}}} = 4$, both O-Net and S-Net predictions are very poor. As the number



(a) $M_{N_{\text{obs}}=4}$ (b) $M_{N_{\text{obs}}=6}$

(c) $M_{N_{\text{obs}}=10}$ (d) $M_{N_{\text{obs}}=20}$

FIGURE 6. Lorenz: Testing results on $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.82841, 0.14785, 0.17099, 12.39551, 32.03720, 2.67205)$ by S-Net and O-Net with time points 4, 6, 10 and 20 for the x-component (left), y-component (middle) and z-component (right). In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 1]$, and the vertical axis shows the values. The solid red line is the ground truth. The blue and orange dashed lines show the O-Net and S-Net results, respectively.



(a) x: $M_{N_{\text{obs}}=4}$ (b) y: $M_{N_{\text{obs}}=4}$ (c) z: $M_{N_{\text{obs}}=4}$ (d) x: $M_{N_{\text{obs}}=6}$

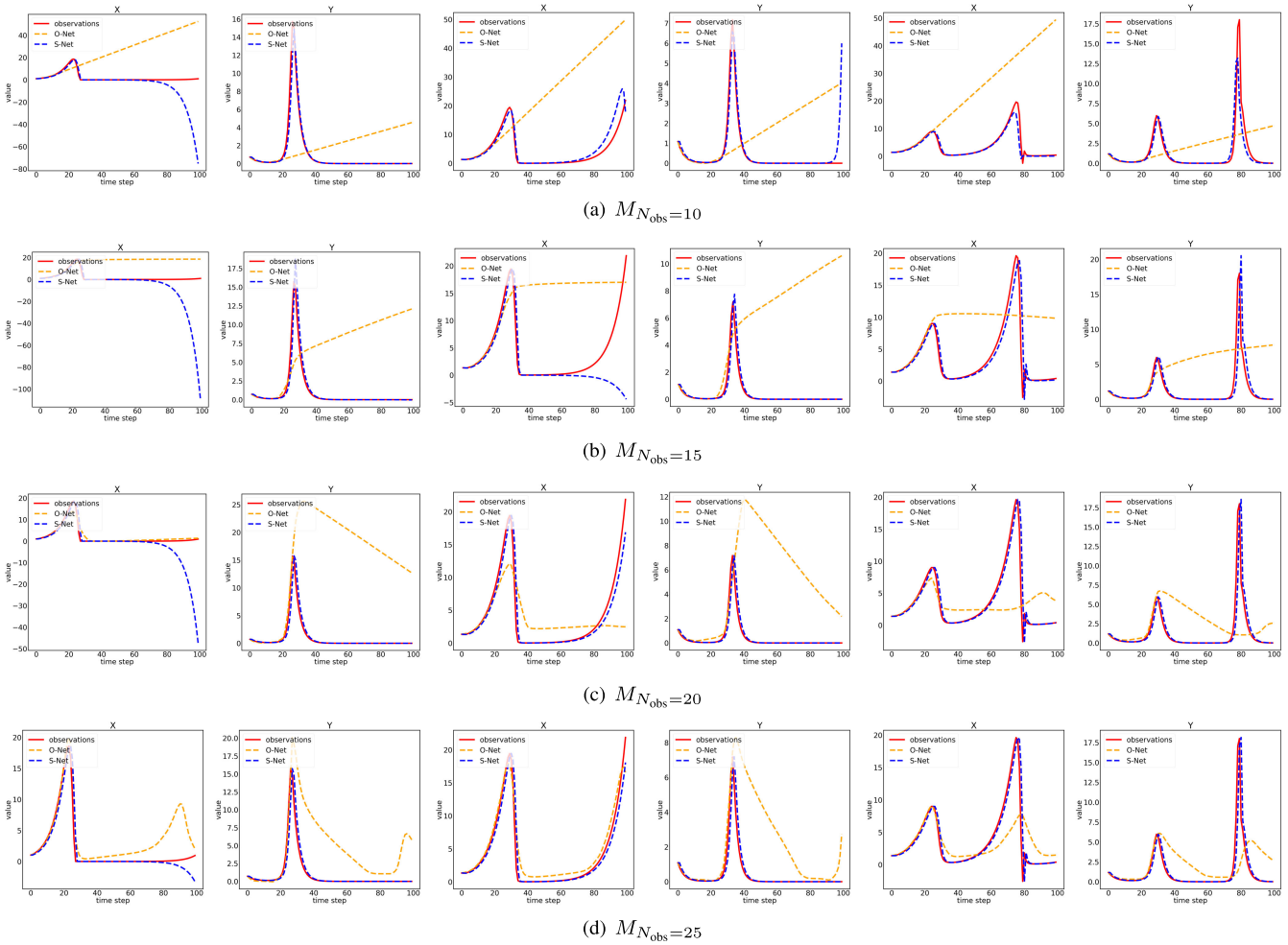
(e) y: $M_{N_{\text{obs}}=6}$ (f) z: $M_{N_{\text{obs}}=6}$ (g) x: $M_{N_{\text{obs}}=10}$ (h) y: $M_{N_{\text{obs}}=10}$

(i) z: $M_{N_{\text{obs}}=10}$ (j) x: $M_{N_{\text{obs}}=20}$ (k) y: $M_{N_{\text{obs}}=20}$ (l) z: $M_{N_{\text{obs}}=20}$

FIGURE 7. Lorenz: Prediction errors of the O-Net(orange) and S-Net(green) with different training time points 4, 6, 10, and 20. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 1]$, and the vertical axis shows the errors. The banded curves indicate the 25% – 100% percentile of the relative errors among 60 test samples and the dark regions indicate the 25% – 75% percentile of the relative error.

of time points increased, the performance of S-Net improve significantly, but the performance of O-Net remain almost unchanged, as shown in Figures 6(b), 6(c), and 6(d). Even in the initial stages of each subfigure, O-Net was unable to perform well.

The error plots for four different time points (4, 6, 10, and 20) are presented in Figure 7. In each case, O-Net exhibits significantly larger errors than S-Net. For instance, consider Figure 7(l). The error of O-Net increases sharply with time and soon reaches 1400, while the error of S-Net increases relatively slowly with time and reaches a maximum of about 100. Table 2 shows the errors of S-Net and O-Net of various time points. The error of O-Net can be up to a thousand times larger than that of S-Net. Clearly, S-Net outperforms



(a) $M_{N_{\text{obs}}=10}$

(b) $M_{N_{\text{obs}}=15}$

(c) $M_{N_{\text{obs}}=20}$

(d) $M_{N_{\text{obs}}=25}$

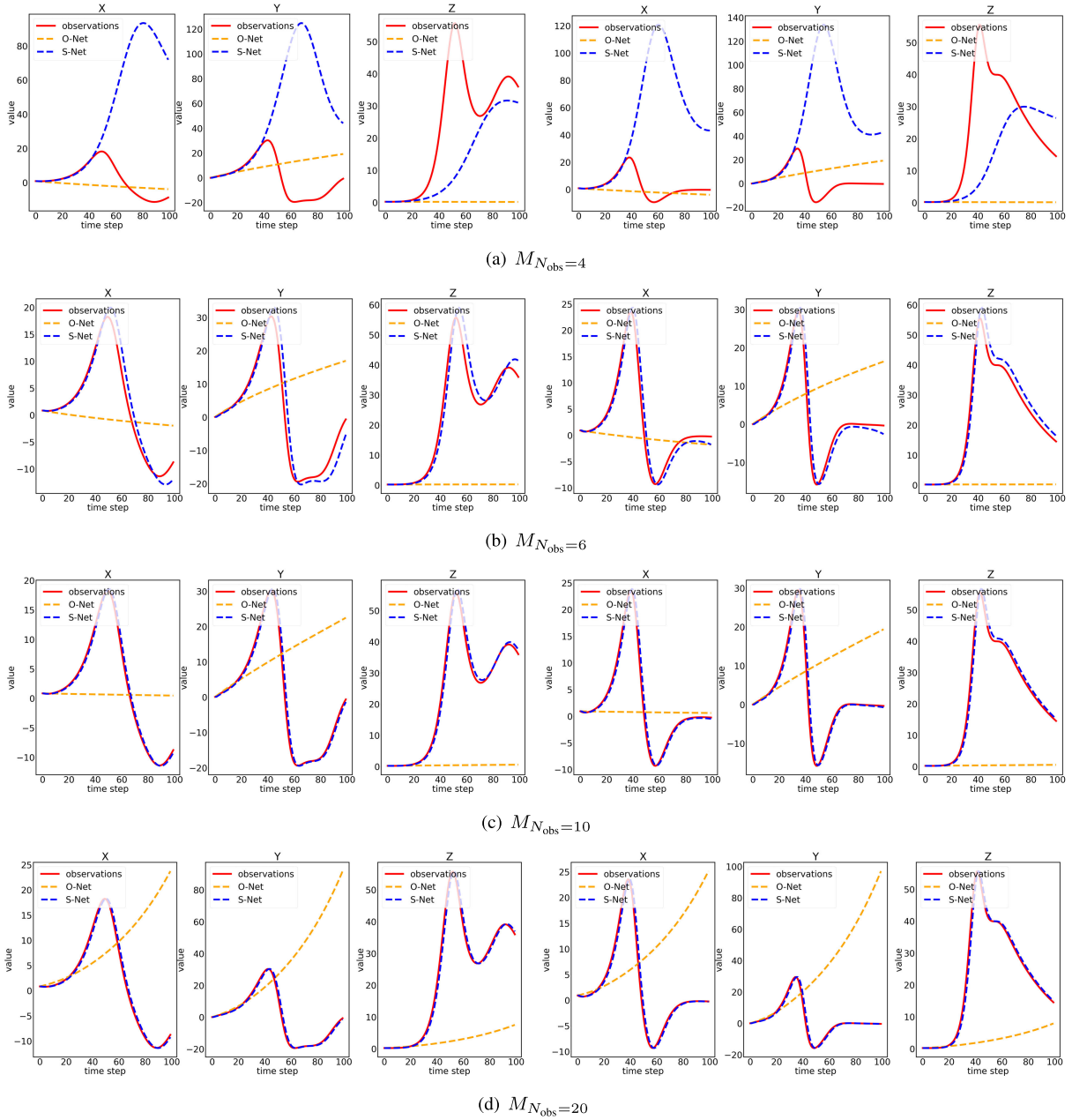
FIGURE 8. **Testing results on $(x_0, y_0, \alpha, \beta, \delta, \gamma) \in \{(1.03591, 0.71055, 1.64399, 0.73789, 2.63207, 0.58110), (1.34048, 1.08388, 1.29828, 1.44437, 3.24866, 0.58960), (1.39070, 1.15958, 1.20000, 1.01820, 3.32212, 0.93040)\}$ by S-Net and O-Net with time points 10, 15, 20 and 25 for the x-component (left) and y-component (right).**

O-Net significantly. We provide the findings of two other cases using different models obtained by $N_{\text{obs}} = 4$, 6, 10, and 20, respectively, in Appendix D.

VII. CONCLUSION

Exploring parameterized ODEs is a broad area in computational science and engineering. Parameterized ODEs problems are particularly challenging to solve because the solutions to these problems can vary with the parameters used. Black-box methods offer some insights, but their generalization performance is often subpar due to a limited understanding of underlying mechanisms. To address this issue, we propose a novel framework that combines the S-Net with an ODE Solver to learn parameterized ODEs. Unlike the black-box O-Net method that simply fits the observed data, S-Net learns the expressions of the equations based on the observed data. We use the Euler method as the ODE Solver due to its simplicity. Experiments show the S-Net framework surpasses the O-Net method in learning parameterized ODEs, evidenced by superior efficiency and accuracy in tests involving the Lotka-Volterra and Lorenz Equations. The proposed framework marks a significant advancement in studying parameterized ODEs in computational science and engineering.

However, there are several limitations to address in future research. Firstly, the current version of S-Net only supports basic operators such as addition, subtraction, and multiplication, which limits its application to complex systems. To broaden its applicability, it is necessary to incorporate more advanced operators such as division, trigonometric functions, powers, and fractional calculation operators. Second, enhancing simulation accuracy requires implementing other ODE Solvers. Thirdly, the present approach is not applicable to arbitrary order systems, which present a more complex challenge. The combinatorial relationship between variables and parameters must be known, and the observations are supplied based on the information on variables. An additional step may be required to predict the number of variables or parameters based on the current method. Finally, while S-Net has proven resilient against noisy data in [9], testing with real data is necessary to establish its applicability to diverse domains.



(a) $M_{N_{\text{obs}}=4}$

(b) $M_{N_{\text{obs}}=6}$

(c) $M_{N_{\text{obs}}=10}$

(d) $M_{N_{\text{obs}}=20}$

FIGURE 9. Testing results on $(x_0, y_0, z_0, \sigma, \rho, \beta) \in \{(0.84731, 0.11332, 0.24186, 5.24679, 29.69917, 2.26212), (0.92617, 0.04462, 0.21900, 14.47371, 27.97391, 2.57754)\}$ by S-Net and O-Net with time points 4, 6, 10 and 20 for the x-component (left), y-component (middle) and z-component (right).

APPENDIX A
CONVERGENCE OF THE EULER METHOD

Our subsequent analysis establishes the convergence of the Euler method when solving the initial-value problem of a first-order differential equation, as stated in (18)

$$\begin{cases} \dfrac{dy}{dx} = f(x, y), x > x_0, \\ y(x_0) = y_0. \end{cases} \tag{18}$$

Here, the unknown function is denoted by $y(x)$, the known function by $f(x, y)$, and the initial data by y_0.

Theorem 1: Assume the following one-step method corresponding to the initial value problem (18) be the p-order accuracy

$$y_{n+1} = y_n + h\phi(x_n, y_n, h)$$

and the function ϕ satisfies the Lipschitz condition for y, i.e., $\exists L > 0$,

$$|\phi(x, y_1, h) - \phi(x, y_2, h)| \leq L|y_1 - y_2|, \quad \forall y_1, y_2$$

and $y_0 = y(x_0)$, then the one-step method is convergent and $y(x_n) - y_n = O(h^p)$.

Proof: Let $e_n = y(x_n) - y_n$, then

$$y(x_{n+1}) = y(x_n) + h\phi(x_n, y(x_n), h) + T_{n+1},$$

i.e.,

$$e_{n+1} = e_n + h[\phi(x_n, y(x_n), h) - \phi(x_n, y_n, h)] + T_{n+1}.$$

Since the one-step method is the p-order accuracy, then $\exists h_0, 0 < h \leq h_0$, satisfy $|T_{n+1}| \leq Ch^{p+1}$ where C is a constant. That is

$$|e_{n+1}| \leq |e_n| + hL|e_n| + Ch^{p+1} = \alpha|e_n| + \beta,$$

where $\alpha = 1 + hL$, $\beta = Ch^{p+1}$. So we can get

$$\begin{aligned} |e_n| &\leq \alpha|e_{n-1}| + \beta \leq \alpha^2|e_{n-2}| + \alpha\beta + \beta \\ &\leq \alpha^3|e_{n-3}| + \beta(1 + \alpha + \alpha^2) \leq \ldots \leq . \end{aligned} \tag{19}$$

From the known conditions, we get

$$\begin{aligned} |e_n| &\leq \exp(L(x_n - x_0))|e_0| + Ch^pL^{-1}(\exp(L(x_n - x_0)) - 1) \\ &= Ch^pL^{-1}(\exp(L(x_n - x_0)) - 1). \end{aligned} \tag{20}$$

From (20), we know when $h \to 0$, then $|e_n| \to 0$. ■

APPENDIX B
THE SEMI-IMPLICIT SOLVERS

Our approach can be adapted to learn unknown functions utilizing semi-implicit solvers with suitable modifications. For instance, consider the semi-implicit Euler method, which can be employed for a pair of differential equations with the form

$$\begin{aligned} \frac{dx}{dt} &= f(t, y), \\ \frac{dy}{dt} &= g(t, x), \end{aligned} \tag{21}$$

where f and g are unknown functions that we want to learn. The semi-implicit Euler method produces an approximateÂ discreteÂ solution by iterating

$$\begin{aligned} y_{n+1} &= y_n + g(t_n, x_n)\Delta t, \\ x_{n+1} &= x_n + f(t_n, y_{n+1})\Delta t, \end{aligned} \tag{22}$$

where Δt is the time step. The difference with the standard Euler method is that the semi-implicit Euler method uses y_{n+1} in the equation for x_{n+1}, while the standard Euler method uses y_n. Given that the expressions for f and g are represented by S-Nets, S-Net$_f$ and S-Net$_g$, it is necessary to provide them with known information. In the semi-implicit Euler method, we use a positive time step to compute x_{n+1} from y_{n+1} generated by S-Net$_g$ based on starting points x_0, y_0, that is

$$x_{n+1} = x_n + \text{S-Net}_f(t_n, \text{S-Net}_g(t_n, x_n)\Delta t). \tag{23}$$

In this context, y is not utilized as an observable variable; instead, it serves as a component to generate the observable variable x. With this approach, as the expression for g must be learned through the x variable, we require additional observations on the x variable and no longer need observations on the y variable.

APPENDIX C
LOTKA-VOLTERRA

In Figure 8, we present the outcomes of three distinct models obtained by utilizing $N_{\text{obs}} = 10, 15, 20$, and 25, respectively. The rows indicate the prediction results of the different models for three test samples. The first two columns in each row demonstrate the evolution of the two variables x and y for the first test sample $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (1.03591, 0.71055, 1.64399, 0.73789, 2.63207, 0.58110)$. The center two columns illustrate the changes in the variables x and y over time for the second test sample $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (1.34048, 1.08388, 1.29828, 1.44437, 3.24866, 0.58960)$. Finally, the last two columns show how the variables x and y varied over time for the third test sample $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (1.39070, 1.15958, 1.20000, 1.01820, 3.32212, 0.93040)$. Each subplot displays the time of prediction on the horizontal axis within the interval $(0, 100\Delta t] = (0, 10]$, while the vertical axis indicates the corresponding values. The solid red line represents the ground truth, while the orange and blue dashed lines show the O-Net and S-Net prediction results, respectively. Notably, within a specific range, the prediction accuracy of the model increases with a higher number of observation points used to train the model.

APPENDIX D
LORENZ

In Figure 9, we present the outcomes of two examples utilizing distinct models derived from $N_{\text{obs}} = 4, 6, 10$, and 20, respectively. Each row represents the prediction results of different models on two test samples. The first three columns of each row demonstrate the progression of the three variables x, y, and z for the first test sample $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.84731, 0.11332, 0.24186, 5.24679, 29.69917, 2.26212)$, while the last three columns depict how the three variables x, y, and z evolve over time for the third test sample $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.92617, 0.04462, 0.21900, 14.47371, 27.97391, 2.57754)$. Each plot shows the prediction time in the interval $(0, 100\Delta t] = (0, 1]$ on the horizontal axis and the corresponding values on the vertical axis. The solid red line in each subplot represents the actual ground truth, while the orange and blue dashed lines show the O-Net and S-Net prediction results, respectively. Notably, the prediction accuracy of the model improves with an increase in the number of observation points used to train the model.

REFERENCES

[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, Dec. 2018, pp. 6572–6583.

[2] Y. Rubanova, R. T. Chen, and D. K. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, Dec. 2019, pp. 5321–5331.

[3] C. Herrera, F. Krach, and J. Teichmann, "Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Austria, May 2021, pp. 1–44. [Online]. Available: https://openreview.net/forum?id=JFKR3WqwyXR

[4] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou, "Symplectic recurrent neural networks," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–23. [Online]. Available: https://openreview.net/forum?id=BkgYPREtPr

[5] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, Feb. 2019.
[6] Z. Long, Y. Lu, X. Ma, and B. Dong, "PDE-Net: Learning PDEs from data," in *Proc. Mach. Learn. Res.*, Jul. 2018, pp. 3208–3216.
[7] Z. Long, Y. Lu, and B. Dong, "PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network," *J. Comput. Phys.*, vol. 399, Dec. 2019, Art. no. 108925.
[8] S. Sahoo, C. Lampert, and G. Martius, "Learning equations for extrapolation and control," in *Int. Conf. Mach. Learn.*, Jul. 2018, pp. 4442–4450.
[9] Q. Li and S. Evje, "Learning the nonlinear flux function of a hidden scalar conservation law from data," *Netw. Heterogeneous Media*, vol. 18, no. 1, pp. 48–79, 2022.
[10] R. Maulik, K. Fukami, N. Ramachandra, K. Fukagata, and K. Taira, "Probabilistic neural networks for fluid flow surrogate modeling and data recovery," *Phys. Rev. Fluids*, vol. 5, no. 10, Oct. 2020, Art. no. 104401.
[11] K. Lee and E. J. Parish, "Parameterized neural ordinary differential equations: Applications to computational physics problems," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 477, no. 2253, Sep. 2021, Art. no. 20210162.
[12] Y. S. Shimizu and E. J. Parish, "Windowed space–time least-squares Petrov–Galerkin model order reduction for nonlinear dynamical systems," *Comput. Methods Appl. Mech. Eng.*, vol. 386, Dec. 2021, Art. no. 114050.
[13] K. Lee and N. Trask, "Parameter-varying neural ordinary differential equations with partition-of-unity networks," 2022, *arXiv:2210.00368*.
[14] H. Owhadi, "Bayesian numerical homogenization," *Multiscale Model. Simul.*, vol. 13, no. 3, pp. 812–828, Jan. 2015.
[15] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Inferring solutions of differential equations using noisy multi-fidelity data," *J. Comput. Phys.*, vol. 335, pp. 736–746, Apr. 2017.
[16] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Machine learning of linear differential equations using Gaussian processes," *J. Comput. Phys.*, vol. 348, pp. 683–693, Nov. 2017.
[17] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 15, pp. 3932–3937, Mar. 2016.
[18] H. Schaeffer, "Learning partial differential equations via data discovery and sparse optimization," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 473, no. 2197, Jan. 2017, Art. no. 20160446.
[19] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Sci. Adv.*, vol. 3, no. 4, Apr. 2017, Art. no. e1602614.
[20] Z. Wu and R. Zhang, "Learning physics by data for the motion of a sphere falling in a non-Newtonian fluid," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 67, pp. 577–593, Feb. 2019.
[21] V. Iakovlev, M. Heinonen, and H. Lähdesmäki, "Learning continuous-time PDEs from sparse data with graph neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Austria, May 2021, pp. 1–15.
[22] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Austria, May 2021, pp. 1–18.
[23] B. Chang, M. Chen, E. Haber, and E. H. Chi, "AntisymmetricRNN: A dynamical system view on recurrent neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–15.
[24] E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2019, pp. 7377–7388.
[25] A. Yazdani, L. Lu, M. Raissi, and G. E. Karniadakis, "Systems biology informed deep learning for inferring parameters and hidden dynamics," *PLOS Comput. Biol.*, vol. 16, no. 11, Nov. 2020, Art. no. e1007575.
[26] C. K. Buhler, R. S. Terry, K. G. Link, and F. R. Adler, "Do mechanisms matter? Comparing cancer treatment strategies across mathematical models and outcome objectives," *Math. Biosci. Eng.*, vol. 18, no. 5, pp. 6305–6327, 2021.
[27] K. Ohta and H. Ishida, "Comparison among several numerical integration methods for Kramers–Kronig transformation," *Appl. Spectrosc.*, vol. 42, no. 6, pp. 952–957, Aug. 1988.
[28] G. R. W. Quispel and D. I. McLaren, "A new class of energy-preserving numerical integration methods," *J. Phys. A, Math. Theor.*, vol. 41, no. 4, Feb. 2008, Art. no. 045206.
[29] K. Atkinson, *An Introduction to Numerical Analysis*. Hoboken, NJ, USA: Wiley, 2008.
[30] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems*, vol. 146. New York, NY, USA: Wiley, 1991.
[31] H. Deng, F. Chen, Z. Zhu, and Z. Li, "Dynamic behaviors of Lotka–Volterra predator–prey model incorporating predator cannibalism," *Adv. Difference Equ.*, vol. 2019, no. 1, pp. 1–17, Dec. 2019.
[32] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, nos. 1–3, pp. 503–528, Aug. 1989.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

QING LI received the B.S. degree in applied mathematics from Qingdao University, China, in 2015, with a focus on solid foundation in mathematical theory and its practical applications, and the M.S. degree in mathematics from the South China University of Technology, China, with a focus on advanced mathematical concepts and numerical methods. She is currently pursuing the Ph.D. degree with the Department of Energy and Petroleum Technology, University of Stavanger, Norway, with a focus on developing novel machine learning algorithms that leverage her expertise in differential equations, both ordinary and partial, and numerical calculation. Her research interests include the intersection of these fields, with a particular emphasis on developing innovative solutions to complex problems in energy and petroleum technology.

STEINAR EVJE received the M.S. and Ph.D. degrees in applied mathematics from the University of Bergen, in 1992 and 1998, respectively. He is currently a Professor in applied and computational mathematics with the Department of Energy and Petroleum Engineering, University of Stavanger. With more than two decades of experience in the field, he has developed a keen interest in the development of mathematical models that can be used to gain insight into fundamental mechanisms for various multiphase transport and reaction processes within fluid mechanics and medical engineering. In addition to his work on mathematical modeling, he has also focused on leveraging the power of data-driven modeling, including machine learning methods to solve problems related to partial differential equations. He has published numerous papers and articles that have helped advance the state of the art in both machine learning and mathematical modeling.

JIAHUI GENG (Graduate Student Member, IEEE) received the B.S. degree from the School of Automation, Southeast University, China, in 2015, and the M.S. degree from the Department of Computer Science, RWTH Aachen University, Germany, in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Computer Engineering, University of Stavanger, Norway. His research interests include robustness, privacy, and security, as well as the development of blockchain systems and the application of dynamic systems in machine learning. With a passion for exploring the cutting edge of his field, he is committed to making meaningful contributions to the development of robust, secure, and privacy-preserving machine learning systems that can be used to drive innovation and progress in a wide range of industries and applications.