DeepGraphONet: A Deep Graph Operator Network to Learn and Zero-Shot Transfer the Dynamic Response of Networked Systems

Yixuan Sun, Christian Moya, Guang Lin, and Meng Yue, *Member, IEEE*

***Abstract*—This article develops a deep graph operator network (DeepGraphONet) framework that learns to approximate the dynamics of a complex system (e.g., the power grid or traffic) with an underlying subgraph structure. We build our DeepGraphONet by fusing the ability of graph neural networks to exploit spatially correlated graph information and deep operator networks to approximate the solution operator of dynamical systems. The resulting DeepGraphONet can then predict the dynamics within a given short/medium-term time horizon by observing a finite history of the graph state information. Furthermore, we design our DeepGraphONet to be resolution independent. That is, we do not require the finite history to be collected at the exact/same resolution. In addition, to disseminate the results from a trained DeepGraphONet, we design a zero-shot learning strategy that enables using it on a different subgraph. Finally, empirical results on the transient stability prediction problem of power grids and traffic flow forecasting problem of a vehicular system illustrate the effectiveness of the proposed DeepGraphONet.**

***Index Terms*—Deep learning, graph neural networks (GNNs), networked dynamical systems, operator regression.**

I. Introduction

NETWORKED dynamical systems are ubiquitous in science and engineering, e.g., the power grid or traffic networks. To simulate and predict the dynamics of such systems, researchers have developed sophisticated, high-fidelity numerical schemes that can accurately solve the corresponding governing equations. However, for tasks requiring multiple forward simulations, e.g., optimization, uncertainty quantification, and control, these high-fidelity schemes may become prohibitively expensive [1].

Deep learning techniques have been proposed to simulate and predict complex dynamical systems and to address the computational cost problem associated with traditional approaches. These techniques act as fast surrogate models trained using data, and they can learn either the underlying governing equations [2], [3], [4] or the future dynamic response [5], [6]. Traditional neural networks, such as fully connected neural networks or recurrent neural networks (RNN), have been used for time-dependent prediction tasks [7], [8], [9]. However, traditional neural networks often require massive data to learn the system's dynamic response, and retraining is often needed for different operating conditions. This limitation has hindered the wider use of traditional neural networks for solving complex dynamical systems in science and engineering, where data are often scarce and expensive to collect.

To address such a limitation, several works have proposed to learn the solution operator (i.e., a mapping between infinite-dimensional spaces) of complex dynamical systems using, for example, deep operator networks (DeepONet) [10], graph neural operators [11], or Fourier neural operators [12]. In particular, the DeepONet, introduced in the seminal paper [10] and developed based on the universal approximation theorem for nonlinear operators [13], has demonstrated remarkable accuracy and generalization capability for learning the solution operator of nonautonomous systems.

We used the DeepONet framework in a recent study [14] to learn the dynamic response of the power grid after a disturbance without using recurrence and fixed resolution, requirements typically adopted by RNNs. However, our work only considered the dynamic response at the bus level. Thus, it neglects the possibly rich information from the interaction between a bus and its neighbors. Such a spatial correlation is critical for learning the solution operator of networked dynamical systems, e.g., the power grid or traffic networks.

To enable learning the solution operator of complex networked dynamical systems, we propose, in this article, the deep graph operator network (DeepGraphONet) framework. DeepGraphONet is a multiinput multioutput DeepONet that learns the dynamic response of complex networked systems by exploiting spatially correlated information from the given underlying graph or subgraph structure.

Manuscript received 18 July 2022; revised 27 March 2023; accepted 18 July 2023. Date of publication 7 August 2023; date of current version 30 August 2023. This work was supported by the Advanced Grid Modeling Program, Office of Electricity Delivery and Energy Reliability of the U.S. Department of Energy. The work of G. Lin, C. Moya, and Y. Sun was supported in part by the National Science Foundation under Grant OAC-2311848, Grant DMS-1555072, Grant DMS-2053746, and Grant DMS-2134209, in part by Brookhaven National Laboratory Subcontract382247, and in part by the U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research Program (DE-SC0021142). *(Corresponding author: Guang Lin.)*

Yixuan Sun was with the School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907 USA. He is now with Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL 60439 USA (e-mail: yixuan.sun@anl.gov).

Christian Moya is with the Department of Mathematics, Purdue University, West Lafayette, IN 47907 USA (e-mail: cmoyacal@purdue.edu).

Guang Lin is with the School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907 USA, and also with the School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: guanglin@purdue.edu).

Meng Yue is with the Interdisciplinary Science Department, Brookhaven National Laboratory, Upton, NY 11973-5000 USA (e-mail: yuemeng@bnl.gov).

Digital Object Identifier 10.1109/JSYST.2023.3298884

A. Related Work

1) Learning Dynamical Systems: Many works have proposed using machine and deep learning to learn unknown dynamical systems from time-series data. In particular, we classify such works into learning the system's governing equations [2], [4], [15] and future response [5], [6]. For example, Brunton et al. [2], use a dictionary of functions to learn a sparse representation of the system's governing equations. Zhang et al. [15] adopted Bayesian sparse regression to identify differential equation terms from a large pool of candidates with error bars.

On the other hand, Qin et al. [5] train a neural network to learn the next state response of the system given the current state. The framework then can predict the system's future response by recursively using the trained network. Raissi et al. [6] proposed a multistep method with a feed-forward neural network to approximate the dynamical system response. Most of the aforementioned works can effectively learn the dynamical system for a single operating condition. However, the aforementioned works will require a prohibitive amount of data and training resources to learn the system's response to many operating conditions. To alleviate such limitations, we will use the novel framework of deep operator learning in this work.

2) Deep Operator Learning: Traditional deep learning techniques [16] focus on approximating the mapping between Euclidean spaces. However, these traditional techniques may not be adequate for learning the solution operator of a complex dynamical system. To learn such an operator, several works have proposed designing operator learning frameworks based on deep neural networks [10], [11], [12], [17].

We build our work on the DeepONet framework introduced in [10]. DeepONet is a neural network architecture that can learn nonlinear operators by using a Branch network to process input information and a Trunk network to process query locations within the output function domain. Then, one computes the output function at a query location by merging the features from both the Branch and Trunk nets using a dot product. DeepONet has demonstrated its exceptional approximation and generalization capabilities using a low amount of data for problems in power engineering [14], electroconvection, and multiphysics tasks [18], or material science tasks [19]. DeepONet, however, is only a single-input single-output operator framework. Thus, one cannot directly use DeepONet for networked dynamical systems.

Jin et al. [20] proposed MIONet, a multiinput function single-output function DeepONet to alleviate the multiple-input problem. MIONet is composed of multiple branch networks to encode the input functions, and a trunk network that is tasked with encoding the output function's domain. Nevertheless, the MIONet framework fails to consider the spatial correlation among input functions of a networked system and only produces a single output function. Our work effectively alleviates both limitations by incorporating a graph neural network (GNN) within the DeepONet framework.

3) Graph Neural Networks (GNNs): In traditional deep learning, the training data, e.g., time series, tabular-like data, and images, are well-structured, and one assumes it belongs to Euclidean space. However, we cannot naturally assume that other forms of data, such as power grid and traffic data, belong to Euclidean space. As a result, researchers developed GNNs [21] to capture the spatial correlation of data with an underlying graph representation.

At the early stage, GNNs were used within recursive schemes [22], [23] to reach steady node states for subsequent tasks. These schemes, however, suffered from high-computational costs and were limited to some specific graph structures. To alleviate such issues, other works [24], [25], [26] build on graph signal processing to develop graph spectral convolution networks and their localized variants. In graph spectral convolution, one performs the graph convolution operation in the Fourier domain. Such spectral convolution requires the eigen-decomposition of the graph Laplacian. Spectral convolution, however, is expensive and does not allow transferring to another graph with a different topology [21], [27]. This limitation prevents the trained GNNs from achieving zero-shot learning.

On the other hand, similar to standard convolutional neural networks (CNNs), several works [28], [29] proposed nonspectral and spatial convolution directly defined on the graph. In particular, the spatial convolution locally operates on nodes, and thus, one can use it on different graphs. Furthermore, by stacking spatial convolutional layers, we can effectively process information from nodes located more than one hop away.

To learn with graph data efficiently and effectively, the authors in [30], [31], and [32] employed the spatial graph convolution within a message-passing framework. The main idea of such a framework is to update the node state information by aggregating information from neighboring nodes, followed by a neural-network-based transformation. In this work, we adopt such a message-passing framework, and in particular, the framework from GraphSage [32], to process the input information to the DeepONet.

This work aims at learning the dynamic evolution of graph signals using a fixed graph structure. Many studies [33], [34], [35], [36], [37] have combined deep learning techniques for time series (e.g., RNNs [34], [35] and CNNs [33], [36], [37]) with GNNs to produce time-dependent predictions that can take into account spatial correlations. However, such methods may lack flexibility and require high computational cost to learn the solution operator of a networked dynamical system. Thus, in our work, we will employ the deep operator learning framework to learn the parametric mapping between graph functions directly.

B. Our Work

The objectives of this work are twofold.

1) *Approximating the Solution Operator:* We aim to derive a deep-learning-based method to approximate the solution operator of a system in which the dynamics evolve within an underlying subgraph structure. For instance, we aim to use our method to learn the dynamics of a control area within a large-scale power grid or the traffic dynamics of a city's neighborhood.
2) *Zero-Shot Learning:* We aim to apply the trained proposed model directly to approximate the solution operator on

unseen graphs or subgraphs with different structures. In other words, we aim to achieve zero-shot learning. For instance, we aim to train the model on a small subgraph, and then, use it to make highly accurate predictions on a larger graph.

The contributions of this work are as follows.

1) We first build (in Section III) a DeepGraphONet framework that learns to approximate the solution operator of a dynamical system with an underlying connected subgraph structure. DeepGraphONet fuses the ability of GNNs to exploit the graph information and DeepONets [10] to approximate the solution operator of nonlinear systems. The proposed DeepGraphONet takes as inputs a finite history of the graph state information and the desired query location with an arbitrary resolution for short/medium-term prediction. Compared to the vanilla DeepONet [10], the proposed DeepGraphONet is a multiinput multioutput framework and resolution-independent in the input function; that is, we do not require the Branch sensors to be fixed.
2) We then propose (in Section IV) a *zero-shot learning* strategy that exploits the property of message-passing GNNs to enable directly using a trained DeepGraphONet on a different graph or subgraph for the same task.
3) Finally, we verify the efficacy of the DeepGraphONet on the transient stability prediction problem of the IEEE 16-machine 68-bus system and the traffic dynamics forecasting problem using the METR-LA dataset, which contains traffic information measured using loop detectors in the highway of Los Angeles County [38].

The rest of this article is organized as follows. Section II introduces the problem of approximating the solution operator of a dynamical system with an underlying subgraph structure. The DeepGraphONet framework to approximate such a solution operator is detailed in Section III-A. We present the resolution-independent DeepGraphONet and our zero-shot learning strategy in Sections III-B and III-C, respectively. Numerical experiments illustrate the efficacy of DeepGraphONet in Section IV. Section V discusses our future work, and finally, Section VI concludes this article.

II. Problem Settings

We consider a complex networked dynamical system. We model its *networked structure* using the undirected graph $G = (V, E)$, where V is the set of $|V|$ nodes and E is the set of $|E|$ edges, and *dynamics* using the initial value problem

$$\begin{aligned} \frac{d}{dt}\mathbf{x}(t) &= f(\mathbf{x}(t); G) \\ \mathbf{x}(t_0) &= \mathbf{x}_0. \end{aligned} \tag{1}$$

Here, $\mathbf{x}(t) \in \mathcal{X} \subset \mathbb{R}^{d\times|V|}$ is the graph-valued state function, $\mathbf{x}_0 \in \mathcal{X}$ the initial condition, and $f : \mathcal{X} \to \mathcal{X}$ the *unknown* vector field. For simplicity, we assume each node's state has the same dimension $d = 1$, and with some abuse of notation, we make explicit the dependence of the dynamics f on the underlying graph structure G. We also let for future work the case when we know an approximate model of the dynamics $f_{\text{approx}} \approx f$.

A. Solution Operator

Our goal is to approximate the dynamic response of (1) described via the solution operator (also known as flow map)

$$\mathcal{F}(\mathbf{x}(t_0); G)(t) \equiv \mathbf{x}(t) = \mathbf{x}(t_0) + \int_{t_0}^{t} f(\mathbf{x}(\beta); G)d\beta \tag{2}$$

for all $t \in [t_0, t_f]$, where $[t_0, t_f] \subset [t_0, \infty)$ denotes a finite time interval where the solution operator exists and is unique.

For large-scale networked dynamical systems, i.e., when $|V| \gg 1$, learning the aforementioned solution operator may be prohibitively expensive. Moreover, one may not have the access to all graph states and may be interested in learning the dynamics of a subregion within the network, i.e., a *subgraph*. For example, for power grids, we may want to learn the dynamic response of a control area, or for traffic networks, we may want to approximate the dynamic response of a city or a district.

To this end, let us split the graph-valued state function as $\mathbf{x} = (\mathbf{x}_S, \mathbf{x}_{S^c})$. Here, $S = (V_S, E_S)$ denotes the subgraph region of interest within the network, where $V_S \subset V$ is a *connected set* of nodes within V and $E_s \subset E$, and S^c denotes the complement of S. To describe the dynamics of $\mathbf{x}_S(t)$, we employ the Mori–Zwanzig formalism [39], [40], which yields

$$\frac{d}{dt}\mathbf{x}_S(t) = R\left(\mathbf{x}_S(t); S\right) + \int_{t_0}^{t} \mathcal{K}\left(\mathbf{x}_S(t-\beta), \beta\right) d\beta + \mathcal{N}\left(\mathbf{x}_0\right) \tag{3}$$

where the first term R is the Markovian term, which depends on the current value of $\mathbf{x}_S$; the second term is the *memory* integral, which depends on the values of the state and input variables from the initial time $\beta = t_0$ to the current time $\beta = t$. This memory integral involves $\mathcal{K}$, which is commonly known as the memory kernel. Finally, the third term is the "noise" term, which depends on the initial condition $\mathbf{x}_0$.

B. Decaying Memory Assumption

Learning the memory integral from the initial time $\beta = t_0$ is challenging as the computational cost increases with the integral bounds. Also, from the neural network approximation point of view, integrating from the initial time leads to increasing input length for the network, which is impractical for network training. To alleviate such a challenge, we assume the memory decays over time, that is, there exists $t_M \in (t_0, t)$ (we treat t_M as a *hyperparameter* in this article) such that we can neglect the effect of the memory for all $\beta \in [t_0, t - t_M)$. Formally, this corresponds to the following approximation of the memory integral. For any given $\epsilon > 0$, there exists $t_0 < t_M < t$ such that

$$\left| \int_{t_0}^{t} \mathcal{K}\left(\mathbf{x}_S(t-\beta), \beta\right) d\beta - \int_{t_0}^{t_M} \mathcal{K}\left(\mathbf{x}_S(t-\beta), \beta\right) d\beta \right| < \epsilon. \tag{4}$$

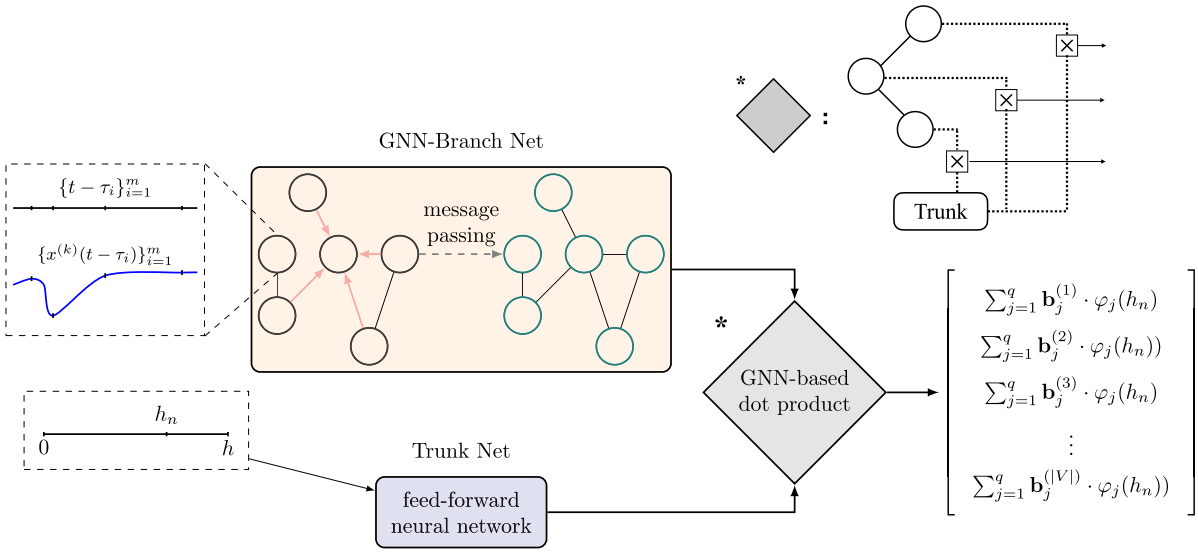

Fig. 1. DeepGraphONet architecture. The *Branch Net* is a GNN that uses message passing to learn the nodes' representation on the networked system's graph S and takes as input the graph state signals $\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m, \tau \in [0, t_M]$. The *Trunk Net* is a feed-forward neural network that takes as input the query location h_n within the prediction horizon $[0, h]$. We obtain the DeepGraphONet's output $\mathcal{F}_\theta(\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m; S)(h_n)$ by applying the GNN-based dot product between the GNN-Branch Net and the Trunk Net. The GNN-based dot product performs the standard dot product between each node in the Branch Net-produced graph and the Trunk Net.

C. Discrete Representation of the Integral

In practice, one computes the truncated integral using some discretized scheme (e.g., quadrature) such that for any given $\kappa > 0$, we have

$$\left| \int_{t_0}^{t_M} \mathcal{K}\left(\mathbf{x}_S(t-\beta), \beta\right) d\beta - Q\left(\mathbf{x}_S\left(t-\tau_1\right), \ldots, \mathbf{x}_S\left(t-\tau_m\right)\right) \right| < \kappa \tag{5}$$

where the memory partition of size m (we treat this memory resolution m as a *hyperparameter* in this article) is *arbitrary* and satisfies $0 \leq \tau_1, \ldots, \tau_m \leq t_M$ and $\{t-\tau_i\}_{i=1}^m \subset [t-t_M, t]$.

These decaying memory and discrete representation assumptions lead to the following approximate dynamics of $\mathbf{x}_S$ as

$$\frac{d}{dt}\mathbf{x}_S(t) = R(\mathbf{x}_S(t)) + Q\left(\mathbf{x}_S\left(t-\tau_1\right), \ldots, \mathbf{x}_S\left(t-\tau_m\right)\right). \tag{6}$$

In this article, we employ the *local solution operator* within the arbitrary interval $[t, t+h_n]$, $h_n \in [0, h]$, where h is the predicting horizon, and we treat it as a hyperparameter in this article. For all $t > t_0$, the local memory sensor locations satisfy $\{t-\tau_i\}_{i=1}^m \subset [t-t_M, t]$. With this local memory, the local solution operator is

$$\begin{aligned}\tilde{\mathcal{F}}\left(\mathbf{x}_S(t), \{\mathbf{x}_S\left(t-\tau_i\right)\}_{i=1}^m; S\right)(h_n) &= \mathbf{x}_S(t) \\ &+ \int_t^{t+h_n} R\left(\mathbf{x}_S(\beta); S\right) d\beta \\ &+ Q\left(\mathbf{x}_S\left(t-\tau_1\right), \ldots, \mathbf{x}_S\left(t-\tau_m\right)\right).\end{aligned} \tag{7}$$

In the aforementioned, we emphasized that the solution operator depends on the finite history of the subgraph state information $\mathbf{x}_S(t)$ within the time interval $[t-t_M, t]$.

Our goal in this article is to design a DeepGraphONet $\mathcal{F}_\theta$, with trainable parameters $\theta \in \mathbb{R}^p$, to approximate the solution operator $\tilde{\mathcal{F}}$ for all $h_n \in [0, h]$. To control the approximation power of the proposed DeepGraphONet, we will use *the following two* hyperparameters: the memory's size t_M and predicting horizon h.

III. Methods

This section describes the proposed DeepGraphONet $\mathcal{F}_\theta$ to approximate the solution operator $\tilde{\mathcal{F}}$.

A. Deep Graph Operator Network (DeepGraphONet)

Building on the DeepONet, introduced in [10], we propose DeepGraphONet $\mathcal{F}_\theta$, a multiinput multioutput DeepGraphONet, which consists of two neural networks (see Fig. 1): the *Branch* network and the *Trunk* network.

1) Branch Network: The *Branch* net is a GNN that processes the finite history of the graph-state information. The *Branch* maps the graph-state information $\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m$, where $\mathbf{x}_S(t-\tau_i) \in \mathbb{R}^{|S|}$, to the graph-based coefficients $\mathbf{b} \in \mathbb{R}^{q \times |S|}$.

To build the proposed GNN-based Branch network, we use a collection of *message passing convolutional graph layers*, which we adopted from [32]. For all $i \in S$, the forward pass of the corresponding message passing layer reads

$$x_i'(t) = \sigma\left(W_1 x_i(t) + W_2 \cdot \text{mean}_{j \in \mathcal{N}_i}(x_j(t))\right) \tag{8}$$

where $x_i(t)$ is the ith node's state at time t, $\mathcal{N}_i$ is the set of nodes adjacent to the ith node, W_1 and W_2 are trainable weights, and σ

is a nonlinear activation function. In our experiments, we chose to have 20 message-passing layers and rectified linear unit as the activation function. Within each graph dataset, the nodes are entities of interest (e.g., power grid buses and traffic network sensors), and the edges describe the connectivity between nodes. At this point, we do not consider edge features/weights according to (8).

By stacking these message passing layers, we can transport information from nodes located *two or more* hops away to the ith node. Such a process allows the network to learn automatically *spatial dependencies* among nodes. Furthermore, since our goal is to obtain the dynamic response of all nodes, we do not use a readout (pooling) in the proposed GNN-based Branch net.

Remark. Fixed versus resolution-independent sensors: If the sensor locations used to discretize the input representing the finite history of graph-state information $\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m$ are fixed (as in [10]), then we denote the proposed framework as the *standard* DeepGraphONet. If, on the other hand, the sensor locations can change over time, then we denote the proposed framework as the *resolution-independent* DeepGraphONet (see Section III-B for more details).

2) Trunk network: The *Trunk* net is a multilayer perceptron (MLP) that maps a given query location h_n, within the prediction horizon $[0, h]$, to a collection of *trainable* Trunk basis functions

$$\varphi := (\varphi_1(h_n), \varphi_2(h_n), \ldots, \varphi_q(h_n))^\top \in \mathbb{R}^q. \tag{9}$$

Finally, we compute the DeepGraphONet's output for each node $i \in S$ by merging the corresponding Branch coefficients $\mathbf{b}^{(i)}$ with the Trunk basis functions φ using the dot product:

$$\mathcal{F}_\theta^{(i)}\left(\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m; S\right)(h_n) = \sum_{j=1}^{q} \mathbf{b}_j^{(i)} \cdot \varphi_j(h_n). \tag{10}$$

3) Training the DeepGraphONet $\mathcal{F}_\theta$: To train the parameters θ of the proposed DeepGraphONet, we minimize, using gradient-based optimization schemes (e.g., Adam [41]), the loss function

$$\mathcal{L}(\theta; \mathcal{D}) = \frac{1}{N}\sum_{k=1}^{N} \left| \mathbf{x}_S^k(t+h_n^k) - \mathcal{F}_\theta\left(\{\mathbf{x}_S^k(t-\tau_i)\}_{i=1}^m\right)(h_n^k) \right|_1 \tag{11}$$

using the dataset $\mathcal{D}$ of $N := |\mathcal{D}|$ triplets

$$\mathcal{D} := \left\{\{\mathbf{x}_S^k(t-\tau_i)\}_{i=1}^m, h_n^k, \mathbf{x}_S^k(t+h_n^k)\right\}_{k=1}^N \tag{12}$$

generated using the true solution operator $\mathcal{F}$.

B. Resolution-Independent DeepGraphONet

The vanilla DeepONet, introduced in [10], effectively and accurately approximates single-input single-output solution operators $\mathcal{F}$. Thus, DeepONet can predict arbitrary locations within the output function domain $[0, h]$. However, the main limitation of the vanilla DeepONet is that it requires a fixed set of sensors within the input function domain $[t - t_M, t]$. This requirement forces the network to take new data points with the same resolution, which, in turn, prevents the flexible application of the trained network, e.g., in scenarios where sampling may be inexact or recursive prediction is needed.

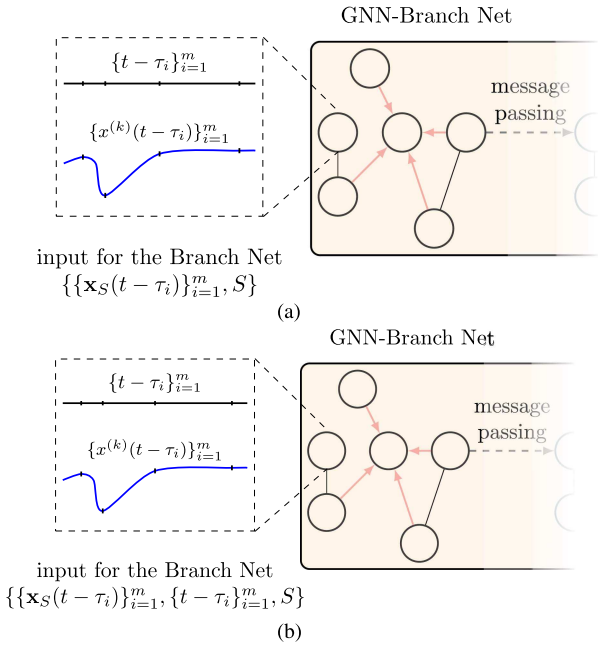

Fig. 2. Input for the Branch Net in standard and resolution-independent DeepGraphONet. (a) Input for the Branch Net in the *standard* DeepGraphONet, where the network takes the input function values at a fixed set of sensors. (b) In a *resolution-independent* DeepGraphONet, the Branch Net takes both input function values and their locations. The fixed set of sensors requirement is eliminated.

To address such a limitation, we propose a simple, intuitive solution that allows using resolution-independent sensors, as shown in Fig. 2. More specifically, we enable the GNN-based Branch network to take as inputs not only the graph-state memory input $\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m$, but also the corresponding, possibly time-variant, sensor locations. To this end, for each node $i \in S$, we construct the input function by concatenating the node-state function values $\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m$ with its corresponding sensor location $\{t-\tau_i\}_{i=1}^m$. These function-location value pairs provide complete information about the input, enabling us to relax the fixed sensors' requirement. In Section IV, we will demonstrate that the *resolution-independent* DeepGraphONet does not cause performance degradation when compared to the *standard* DeepGraphONet.

C. Zero-Shot Learning Scheme

In this article, we build the proposed DeepGraphONet framework to be versatile. That is, once we have trained the DeepGraphONet $\mathcal{F}_{\theta^*}$ using the graph S, we expect to use it with high accuracy on a different graph $S' \neq S$ (with possibly different number of nodes) for the same task.

Such a *zero-shot learning* capability of the proposed DeepGraphONet is achieved immediately due to the message-passing nature of the GNN-based Branch network. Zero-shot learning aims to apply the trained network to unseen instances with different underlying structures, e.g., domain discretization or graph. With message-passing GNN layers, the DeepGraphONet

learns the weights associated with the node feature dimension after concatenating the messages from the neighboring nodes. The trained DeepGraphONet can then directly apply the learned weights to a different graph after the message-passing operation for the same task.

In particular, we can directly apply DeepGraphONet to S' because the trainable weights of the GNN-based Branch depend on the node feature dimension and not on the adjacency matrix. Thus, the proposed DeepGraphONet is inductive, and we may use it beyond the graph S used during training. In Section IV, we will show that DeepGraphONet can effectively approximate the solution operator of networked dynamical systems within subgraph structures S' not used during training, thus, achieving zero-shot learning.

IV. Experiments

To evaluate the accuracy and effectiveness of the proposed DeepGraphONet, we test it on the following *two* different tasks:

1) predicting the transient response of power grids (see Section IV-A);
2) forecasting traffic on highway networks (see Section IV-B).

In the experiments, we use t_M, m, h, and h_n to denote the memory length, number of sensors within the memory, prediction horizon, and query location within the prediction horizon, respectively. We treat t_M and h as hyperparameters in this article and explore the model performance. We evaluated the proposed DeepGraphONet against several baseline models. The baseline models include the following:

1) autoregressive integrated moving average (ARIMA), which is a commonly used linear model for time-series forecasting;
2) MLP, which is a fully connected feedforward neural network that takes the input history at different timesteps as features and outputs the horizon-length predictions;
3) vanilla DeepONet [10] that does not consider the graph structure.

Neural networks and implementation protocols: We use the same neural network architecture to build the following two DeepGraphONets: 1) the *standard* DeepGraphONet, which has fixed sensors as in [10], and 2) *resolution-independent DeepGraphONet*, which we proposed in this article (see Fig. 2) to alleviate the fixed-sensors constraint. To build the DeepGraphONets' architecture, we proceed as follows. The *Branch Net* uses 20 *message passing* graph convolution layers, and the *Trunk Net* uses five feed-forward layers. The output dimension is 100 neurons for both the Branch and Trunk Nets.

We prepared the data to align with the DeepGraphONet's formulation, where we set a memory partition and an output function domain bound h. With regard to the resolution-independent DeepGraphONet, with the memory partition size m, we randomly sampled $\lfloor \frac{m}{2} \rfloor$ sensors and stacked them with their locations as additional feature channels for the *Branch Net*.

Algorithm 1 shows the procedure of using trained DeepGraphONet to predict the complete future trajectories. With the trained network, we use the initial memory and partition to produce predictions covering the entire prediction horizon. We then repeat this process with the newly observed *ground truth* as the memory for the next prediction horizon until the trajectory's end.

Algorithm 1: Complete Trajectory Prediction Scheme.

Require: system graph S; trained DeepGraphONet $\mathcal{F}_{\theta^*}$; hyperparameters (t_M, m, h, h_n); initial local memory $\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m$; complete trajectory length L;
initialize an empty list for saving the predictions C;
let $N = \lceil \frac{(L-t_M)}{h} \rceil$;
for $n = 0, \ldots, N-1$ **do**
  for *each* $h_n \in [0, h]$ **do**
    use the DeepGraphONet's forward pass to compute the prediction
    $$\bar{\mathbf{x}}(t+h_n) = \mathcal{F}_{\theta^*}(\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m, \{\tau\}_{i=1}^m; S)(h_n).$$
    append $\bar{\mathbf{x}}(t+h_n)$ to C;
  end for
  update the current time
  $$t \leftarrow t + h;$$
  new observations come in
  $$\{\mathbf{x}_S(t-\tau_i)\}_{i=1}^m;$$
end for
concatenate C;
Return the concatenated predicted trajectories;

For example, in the power grid problem, we have a trained DeepGraphONet with memory length $t_M = 200$ ms and prediction horizon $h = 20$ ms. In testing time, we want to produce predictions of the entire trajectory of size 700 ms after the initial memory. We use the initial memory and the trained network to make predictions for the next 20 ms and save the predictions to a list. We then use the *new observations* to produce predictions for another 20 ms until the trajectory ends. Finally, we concatenate the saved predictions as the predicted complete trajectory.

We implemented the DeepGraphONet in `PyTorch` and trained and tested it with an Nvidia GPU. The source code of the model can be accessed on https://github.com/cmoyacal/graph-deepOnetGitHub.[1] We measured and compared the performance of the proposed model and baseline models in terms of L_1 relative error (L_1%), root mean squared error (RMSE), and coefficient of determination (R^2), and investigated the DeepGraphONet's sensitivity to history length, t_M, and prediction horizon, h, using L_1 relative error. The formulations of the metrics are in Appendix A.

[1]The repository will be public upon the acceptance of the manuscript.

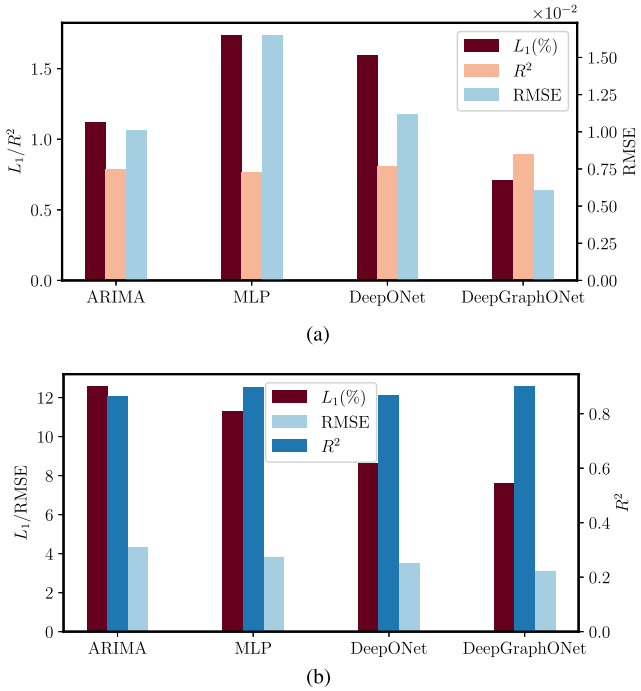

Fig. 3. Comparisons of ARIMA, MLP, vanilla DeepONet, and the proposed DeepGraphONet in terms of L_1 relative error (%), RMSE, and coefficient of determination (R^2). (a) Model performances on the power grid problem testing set with a horizon, $h = 20$ ms. (b) Model performances on the testing set of the traffic prediction problem with a horizon, $h = 30$ min.

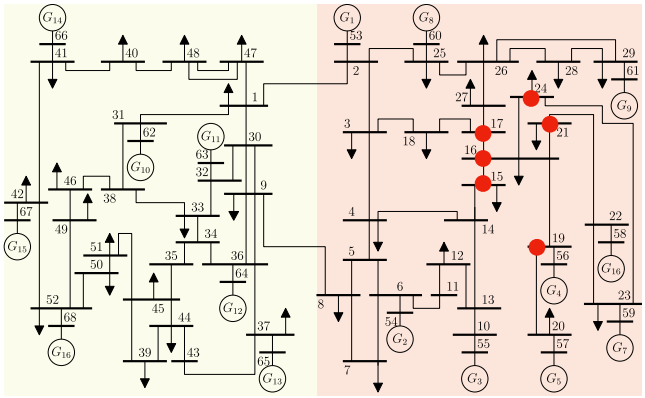

Fig. 4. One-line diagram of the New York–New England 16-generator 68-bus power grid. We train our DeepGraphONet using the *arbitrary* subgraph of buses $S := \{15, 16, 17, 19, 21, 24\} \subset G$, highlighted with red dots.

A. Experiment 1: Power Transient Stability

1) Dataset: We simulated the datasets for the New York–New England 16-generator 68-bus power grid model, shown in Fig. 4, using time-domain simulations with EPTOOL [42]. The simulations considered the full dynamical model of the generators, including turbine governors, excitation systems, and power system stabilizers. The faults were simulated by randomly disconnecting one transmission line. The postfault trajectory occurred within the interval $(t_{cl}, T]$, with t_{cl} fixed at 2 s and simulation time T at 7 s. More details about the data generation are in [14]. The power grid dataset consisted of 1830 independent

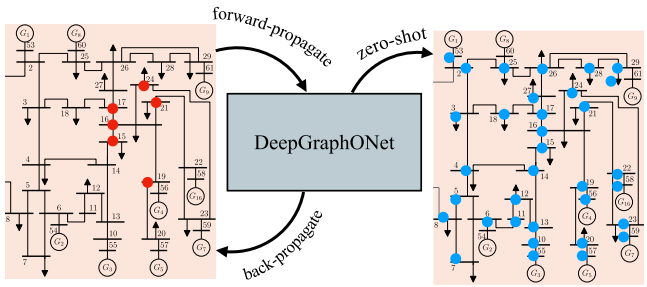

Fig. 5. Zero-shot learning on a larger subgraph. We train the DeepGraphONet on an arbitrary small subgraph (marked in red) and directly apply the trained network to a much larger subgraph (marked in blue) to achieve high predicting performance (0.686% L_1 relative error).

TABLE I
MEAN AND STANDARD DEVIATION (ST.DEV.) OF THE L_1-RELATIVE ERROR (%) BETWEEN THE PREDICTED AND ACTUAL TRANSIENT RESPONSE TRAJECTORIES FOR MULTIPLE TIME HORIZONS h AND USING THE (I) *STANDARD* DEEPGRAPHONET AND (II) RESOLUTION-INDEPENDENT DEEPGRAPHONET

h (ms)		2	10	20	200
standard	mean	0.21	0.64	0.71	2.57
	st.dev.	0.74	2.71	2.62	4.89
res.ind.	mean	0.37	0.62	0.72	1.71
	st.dev.	1.62	2.57	2.61	3.58

TABLE II
MEAN AND STANDARD DEVIATION (ST.DEV.) OF THE L_1-RELATIVE ERROR (%) BETWEEN THE PREDICTED AND ACTUAL TRANSIENT RESPONSE TRAJECTORIES FOR MULTIPLE MEMORY RESOLUTIONS m AND USING THE (I) *STANDARD* DEEPGRAPHONET AND (II) RESOLUTION-INDEPENDENT DEEPGRAPHONET

t_M (ms)		50	100	200
standard	mean	0.71	0.70	0.66
	st.dev.	2.62	2.82	3.14
res.ind.	mean	0.72	0.71	0.68
	st.dev.	2.60	3.46	2.98

trajectories on a subgraph with six nodes (red buses in Fig. 5). Each trajectory had 701 time steps with a resolution of 1 ms. Among the available trajectories, 60%, 20%, and 20% were used for training, validation, and testing, respectively. We then applied the trained model to the entire region (marked in blue in Fig. 5), achieving zero-shot learning.

2) Testing Accuracy: We show the model testing results in Figs. 3(a) and 6 and Tables I and II. Overall, our trained both standard and resolution-independent DeepGraphONets accurately approximated the solution operator for the power grid problem. Compared to other baseline models, the DeepGraphONet resulted in the lowest L_1 relative error and RMSE and the highest R^2 score. The L_1 relative errors of all short- and medium-term forecasting settings were within 1%. Moreover, the resolution-independent DeepGraphONet, taking the input function representation with an arbitrary resolution, did not show performance degradation compared to the standard [see Fig. 7(b)].

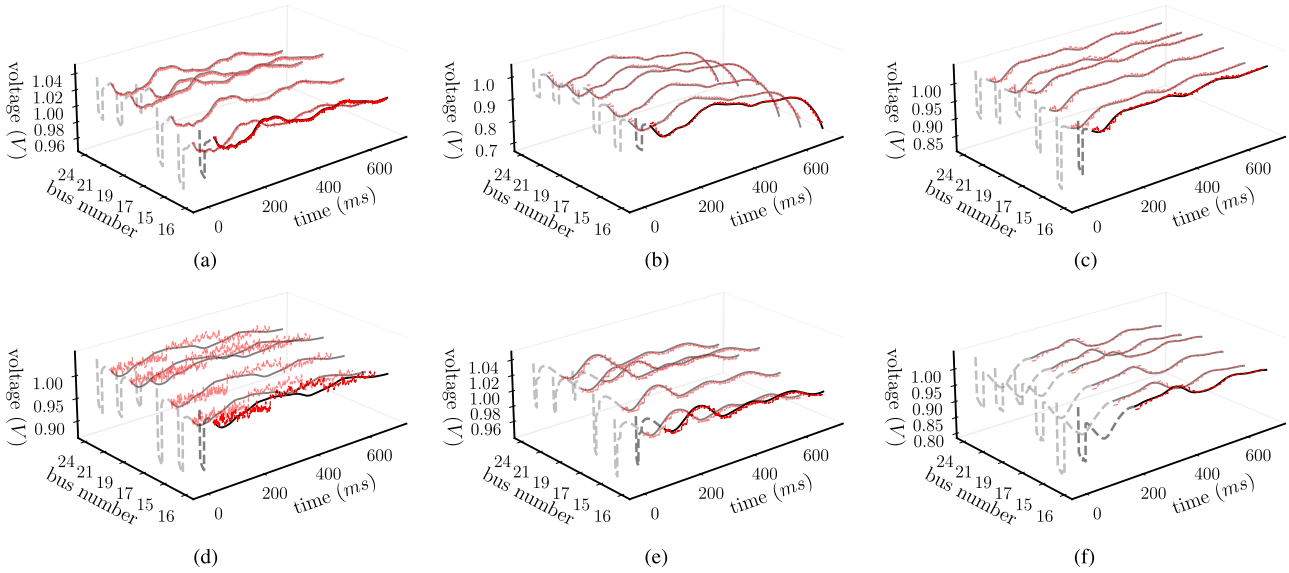

Fig. 6. Comparison of the DeepGraphONet predicted transient trajectories (dashed red lines) with the actual trajectories (solid black lines) on the subgraph of buses $S := \{15, 16, 17, 19, 21, 24\} \subset G$. DeepGraphONet generated all predictions at the same time given the memory trajectory (dashed gray lines). We test multiple prediction horizons h and memory resolutions t_M. In particular, (a) $h = 2$ ms with $t_M = 50$ ms; (b) $h = 10$ ms with $t_M = 50$ ms; (c) $h = 20$ ms with $t_M = 50$ ms; (d) $h = 200$ ms with $t_M = 50$ ms; (e) $h = 20$ ms with $t_M = 100$ ms; and (f) $h = 20$ ms with $t_M = 200$ ms.

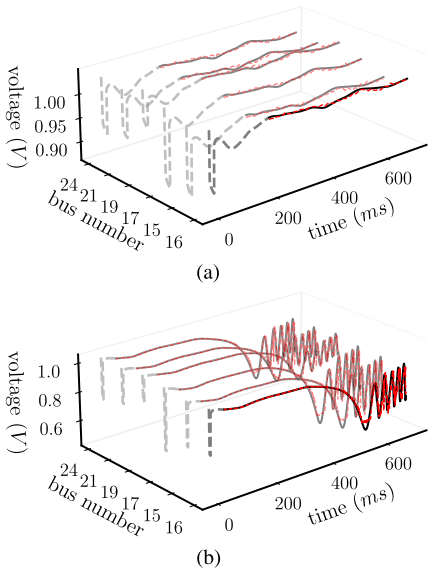

Fig. 7. Comparison of the DeepGraphONet predicted transient trajectories (dashed red lines) with the actual trajectories (solid black lines) on the subgraph of buses $S := \{15, 16, 17, 19, 21, 24\} \subset G$ using the proposed resolution-independent DeepGraphONet. We test two time horizons h and memory resolutions t_M. (a) $h = 500$ ms with $t_M = 200$ ms. (b) $h = 2$ ms with $t_M = 50$ ms.

3) Varying Memory Lengths: We explored the impact and sufficiency of local memory lengths. We trained the DeepGraphONet with different memory lengths, $t_M = 50$ ms, $t_M = 100$ ms, $t_M = 200$ ms, and evaluated the networks on the testing dataset. The prediction results suggest that with larger t_M, the average model performance improves [see Fig. 6(d), (e), and (f)]. We observed the same trend in both original and resolution-independent DeepGraphONets, as shown in Table II.

4) Varying Horizons: To investigate the performance change of DeepGraphONet with respect to the change of the output function domain, we varied the prediction horizons h. In particular, we trained the networks with $h = 2$ ms, $h = 10$ ms, $h = 20$ ms, and $h = 200$ ms with the same memory length, $t_M = 50$ ms. The results (see Fig. 6(a), (b), and (c), and Table I) show for all prediction horizons, both the standard and resolution-independent DeepGraphONet made accurate approximations to the solution operators. Furthermore, the network performance increases as h decreases. For short- and medium-term forecasting, the networks led to L_1 relative errors of less than 1%, whereas when $h = 200$ ms, the relative error was over 2%. However, with longer local memory ($t_M = 200$ ms), the trained DeepGraphONet accurately predicted the longer horizon ($h = 500$ ms), shown in Fig. 7(a).

5) Zero-Shot Learning: Finally, we applied the trained model to a larger subgraph from the same power grid model. The new subgraph has 34 nodes and contains the graph used during training. By directly applying the trained model to the larger subgraph without further training, we achieved zero-shot learning with high accuracy. Fig. 8 depicts the prediction results on nine randomly selected nodes of the new subgraph. For these nodes, the predictions aligned with the ground truth well with a L_1 relative error of 0.686%.

With the aforementioned, we conclude that the proposed DeepGraphONet can successfully learn the solution operator with local memories for predicting the transient stability of power grids. Particularly, the resolution-independent DeepGraphONet, although having the flexibility of taking the discrete input function representation with an arbitrary resolution, does not show performance degradation. In addition, one can directly apply the trained DeepGraphONet to solve the power grid transient stability problem with a different underlying graph structure, achieving zero-shot learning.

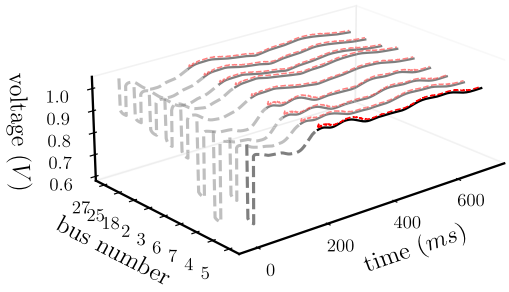

Fig. 8. *Zero-shot learning results.* Comparison of the DeepGraphONet predicted transient trajectories (dashed red lines) with the actual trajectories (solid black lines) on a subgraph of buses $S' := \{2, 3, 4, 5, 6, 7, 18, 25, 27\} \subset G$ different from the *training* subgraph S.

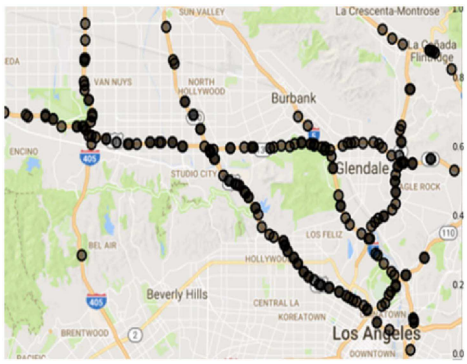

Fig. 9. METR-LA traffic network (adopted from [35]). The black dots represent the loop detectors, which we regard as the nodes of the graph G.

B. Experiment 2: Traffic Forecasting

1) Dataset: We used the METR-LA dataset [38], containing 179 loop detectors, that collect the speed data on the freeway network in Los Angeles County, shown in Fig. 9. The dataset contained speed measurements of the year 2018, with a resolution of 5 min. Due to the lack of independent trajectories, we split the data based on the time for training, validation, and testing purposes. We used the data from January to June 2018 for training, July to September 2018 for validation, and October to December 2018 for testing. We trained and evaluated the network using the first 20 nodes in the graph and directly applied the trained network to the entire graph for zero-shot learning.

We followed the same training protocol as in the power grid experiment and reported model performances on the testing set. Fig. 3(b) compares the DeepGraphONet and other baseline models. The results suggest that the DeepGraphONet has superiority over other baseline models in terms of all metrics.

2) Varying Horizons: We investigated the DeepGraphONet's performance in traffic forecasting under the commonly used horizons [43], $h = 15$ min, $h = 30$ min, and $h = 60$ min with the local memory length fixed as $t_M = 60$ min. Fig. 10 shows the prediction results from the randomly picked four nodes in the graph where the predicting horizon was $h = 60$ min. DeepGraphONet captured the future speed of all nodes at the same time accurately using the history observation, while at some sharp turning points, the model did not match the oscillation exactly but captured the average behavior. The results suggest that with an increasing horizon, the model performance decreased overall and the error range became larger. Table III lists the mean and standard deviation of the L_1 relative error of the models, both for original and resolution-independent DeepGraphONet, with different predicting horizons on the testing set. The results imply the great ability of the proposed DeepGraphONet in predicting the traffic dynamics for all nodes present in the network simultaneously, with ∼6% error for short-term (5 min) prediction and ∼11% for long-term (1 h).

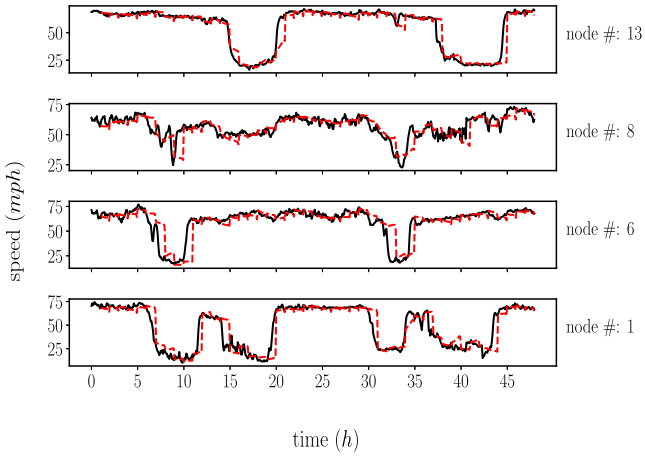

Fig. 10. Comparison of the DeepGraphONet predicted trajectories (dashed red lines) with the actual trajectories (solid black lines) on the subgraph $S := \{1, 6, 8, 13\}$ of the highway network G using $t_M = 60$ (minutes) memory length.

TABLE III
MEAN AND STANDARD DEVIATION (ST.DEV.) OF THE L_1-RELATIVE ERROR (%) BETWEEN THE PREDICTED AND ACTUAL TRAFFIC TRAJECTORIES FOR MULTIPLE TIME HORIZONS T AND USING THE (I) *STANDARD* DEEPGRAPHONET AND (II) *RESOLUTION-INDEPENDENT* DEEPGRAPHONET

h (min)		15	30	60
standard	mean	5.52	7.60	10.79
	st.dev.	9.49	14.74	22.05
res.ind.	mean	6.84	8.62	11.87
	st.dev.	12.56	16.79	23.57

3) Zero-Shot Learning: To test the DeepGraphONet's potential of zero-shot learning for traffic forecasting, we directly applied the trained network to the entire METR-LA network with 179 nodes. Fig. 11 shows the visualized model prediction results on four randomly picked nodes among the 179. With 1-h predicting horizon, the trained model closely predicted the future on the unseen graph structure for all nodes at the same time, resulting in the L_1 relative error of 10.79%.

The experiment results for the traffic forecasting problem strongly suggest that our proposed DeepGraphONet was powerful in learning the solution operator for traffic dynamics. Also, the DeepGraphONet can be better implemented for practical applications for it is easy to train, can produce predictions at arbitrary time step within the horizon, and does not require forward dependence to obtain the value at this step.

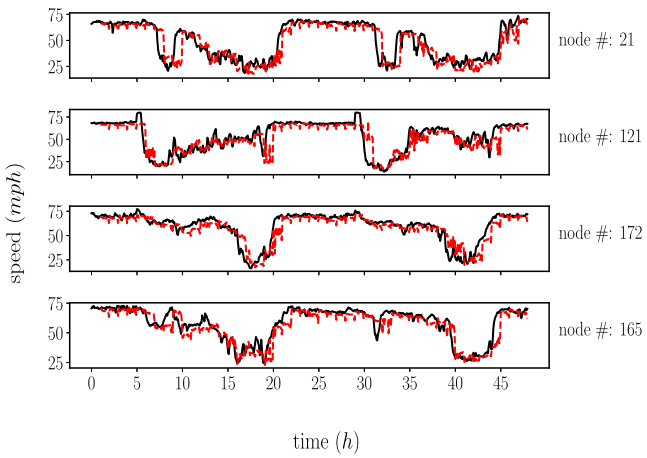

Fig. 11. *Zero-shot learning results.* Comparison of the DeepGraphONet predicted trajectories (dashed red lines) with the actual trajectories (solid black lines) on all the 179 nodes of the METR-LA highway network G. We illustrate the predicted trajectories for the following *four* nodes $\{21, 121, 165, 172\}$, which we selected uniformly at random.

V. Discussion and Future Work

We used the proposed DeepGraphONet to learn the solution operator of networked dynamical systems and demonstrated its effectiveness in power grid transient stability and traffic forecasting tasks. In particular, the resolution-independent DeepGraphONet demonstrated excellent accuracy in approximating the solution operator without performance degradation while having the ability of flexible discrete input function representation.

We demonstrated that the trained DeepGraphONet can produce predictions at arbitrary query locations within the output function domain and does not require forward dependence in the output sequence. In addition, we showed that one could directly apply the DeepGraphONet, trained on a small subgraph region, to a larger graph region for the same task, achieving zero-shot learning with high accuracy. With the promising wide-ranged applications of the proposed DeepGraphONet, we provide next a preamble for our future work.

1) *On physics-informed DeepGraphONet.* This article focuses on developing a physics-informed version of DeepGraphONet. We plan to use first-principles models to train the DeepGraphONet without data or use them as prior information during data-driven training.
2) *On anomaly detection with DeepGraphONet.* We will focus part of our future work on using DeepGraphoNet for detecting anomalies. To this end, we must endow the proposed DeepGraphONet with the ability to estimate the network's edges status using graph trajectory data.
3) *On learning networked control systems.* We plan to extend the proposed DeepGraphONet for learning networked control dynamical systems. This problem is more challenging because we need to learn the system's response using graph trajectory and its response to external inputs. If successful, our DeepGraphONet for control may become an essential tool for implementing model-based multiagent continuous reinforcement learning.
4) *On predicting long-term trajectories.* We plan to develop effective training strategies to alleviate error accumulation such that we can recursively use the trained DeepGraphONet to forecast trajectories for long-term horizons with high accuracy.

VI. Conclusion

In this article, we introduced the DeepGraphONet framework to predict the dynamics of complex systems with underlying subgraph structures. By fusing the ability of GNNs to correlate graph trajectory information and DeepONet to approximate nonlinear operators, we achieved significant results on complex problems such as predicting the transient stability of a control area of a power grid and the traffic flow within a city or district. Moreover, we built our DeepGraphONet to be resolution independent, i.e., we do not require a set of fixed sensors to encode the graph trajectory history. Finally, we designed a zero-shot learning strategy that enables using the proposed DeepGraphONet on a different subgraph with high predicting performance.

Appendix

A. Evaluation Metrics

In the following formulations of evaluation metrics, we use y to represent the true values, $\hat{y}$ the model approximated values, and N the data size.

1) *L_1 relative error*

$$L_1 \text{ relative error} = \frac{||y - \hat{y}||_1}{||y||_1} \times 100\%. \tag{13}$$

2) *Root mean square error (RMSE)*

$$\text{RMSE} = \sqrt{\frac{\sum (y - \hat{y})^2}{N}}. \tag{14}$$

3) *Coefficient of determination*

$$R^2 = 1 - \frac{\sum (y - \hat{y})^2}{\sum (y - \bar{y})}, \tag{15}$$

where $\bar{y}$ is the average of the true values.

References

[1] A. Iserles, *A First Course in the Numerical Analysis of Differential Equations*. Cambridge, U.K.: Cambridge Univ. Press, 2009, no. 44.
[2] 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.*, vol. 113, no. 15, pp. 3932–3937, 2016.
[3] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proc. Roy. Soc. A*, vol. 474, no. 2219, 2018, Art. no. 20180335.
[4] H. Schaeffer, "Learning partial differential equations via data discovery and sparse optimization," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 473, no. 2197, 2017, Art. no. 20160446.
[5] T. Qin, K. Wu, and D. Xiu, "Data driven governing equations approximation using deep neural networks," *J. Comput. Phys.*, vol. 395, pp. 620–635, 2019.
[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep neural networks for data-driven discovery of nonlinear dynamical systems," 2018, *arXiv:1801.01236*.
[7] J. Li, M. Yue, Y. Zhao, and G. Lin, "Machine-learning-based online transient analysis via iterative computation of generator dynamics," in *Proc. IEEE Int. Conf. Commun., Control, Comput. Technol. Smart Grids*, 2020, pp. 1–6.

[8] H. Wilms, M. Cupelli, A. Monti, and T. Gross, "Exploiting spatio-temporal dependencies for RNN-based wind power forecasts," in *Proc. IEEE PES GTD Grand Int. Conf. Expo. Asia*, 2019, pp. 921–926.
[9] Y. Sun, T. Mallick, P. Balaprakash, and J. Macfarlane, "A data-centric weak supervised learning for highway traffic incident detection," *Accident Anal. Prevention*, vol. 176, 2022, Art. no. 106779.
[10] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Mach. Intell.*, vol. 3, no. 3, pp. 218–229, 2021.
[11] A. Anandkumar et al., "Neural operator: Graph kernel network for partial differential equations," in *Proc. ICLR Workshop Integration Deep Neural Models Differ. Equ.*, 2020.
[12] Z. Li et al., "Fourier neural operator for parametric partial differential equations," 2020, in *Proc. Int. Conf. Learn. Representations.*, 2020.
[13] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Trans. Neural Netw.*, vol. 6, no. 4, pp. 911–917, Jul. 1995. [Online]. Available: https://ieeexplore.ieee.org/document/392253/
[14] C. Moya, S. Zhang, M. Yue, and G. Lin, "DeepoNet-grid-UQ: A trustworthy deep operator framework for predicting the power grid's post-fault trajectories," *Neurocomputing*, vol. 535, pp. 166–182, 2023.
[15] S. Zhang and G. Lin, "Robust data-driven discovery of governing physical laws with error bars," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 474, no. 2217, 2018, Art. no. 20180305.
[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[17] N. Winovich, K. Ramani, and G. Lin, "ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains," *J. Comput. Phys.*, vol. 394, pp. 263–279, 2019.
[18] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis, "DeepM&MNet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks," *J. Comput. Phys.*, vol. 436, 2021, Art. no. 110296.
[19] S. Goswami, M. Yin, Y. Yu, and G. Karniadakis, "A physics-informed variational DeepONet for predicting the crack path in brittle materials," *Comput. Methods Appl. Mech. Eng.*, vol. 391, 2022, Art. no. 114587.
[20] P. Jin, S. Meng, and L. Lu, "Mionet: Learning multiple-input operators via tensor product," *SIAM J. Sci. Comput.*, vol. 44, no. 6, pp. A3490–A3514, 2022.
[21] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
[22] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 714–735, May 1997.
[23] M. Gori, G. Monfardini, and F. Scarselli, "A new model for earning in raph domains," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2005, vol. 2, pp. 729–734.
[24] J. B. Estrach, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. 2nd Int. Conf. Learn. Representations*, 2014.
[25] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory," *Appl. Comput. Harmon. Anal.*, vol. 30, no. 2, pp. 129–150, 2011.
[26] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc Int. Conf. Learn. Representations*, 2016.
[27] J. Zhou et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://doi.org/10.1016/j.aiopen.2021.01.001
[28] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, vol. 29.
[29] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.
[30] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.
[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learn. Representations*, 2018.
[32] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30.
[33] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 922–929.
[34] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proc. 4th Int. Conf. Learn. Representations*, 2016, pp. 1–20.
[35] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. Int. Conf. Learn. Representations*, 2018.
[36] L. Zhao et al., "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 9, pp. 3848–3858, Sep. 2020.
[37] J. Zhu, Q. Wang, C. Tao, H. Deng, L. Zhao, and H. Li, "AST-GCN: Attribute-augmented spatiotemporal graph convolutional network for traffic forecasting," *IEEE Access*, vol. 9, pp. 35973–35983, 2021.
[38] H. V. Jagadish et al., "Big data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, 2014.
[39] A. J. Chorin and O. H. Hald, *Stochastic Tools in Mathematics and Science*, vol. 1, Berlin, Germany: Springer, 2009.
[40] A. J. Chorin, O. H. Hald, and R. Kupferman, "Optimal prediction with memory," *Physica D, Nonlinear Phenomena*, vol. 166, no. 3-4, pp. 239–257, 2002.
[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, vol. 2015, 2015.
[42] J. H. Chow and K. W. Cheung, "A toolbox for power system dynamics and control engineering education and research," *IEEE Trans. Power Syst.*, vol. 7, no. 4, pp. 1559–1564, Nov. 1992.
[43] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 753–763.