Climate Modeling with Neural Diffusion Equations

Jeehyun Hwang[1], Jeongwhan Choi[1], Hwangyong Choi[1], Kookjin Lee[2], Dongeun Lee[3], Noseong Park[1]
Yonsei University[1], Seoul, South Korea
Arizona State University[2], Tempe, AZ, USA
Texas A&M University–Commerce[3], Commerce, TX, USA

***Abstract*—Owing to the remarkable development of deep learning technology, there have been a series of efforts to build deep learning-based climate models. Whereas most of them utilize recurrent neural networks and/or graph neural networks, we design a novel climate model based on the two concepts, the neural ordinary differential equation (NODE) and the diffusion equation. Many physical processes involving a Brownian motion of particles can be described by the diffusion equation and as a result, it is widely used for modeling climate. On the other hand, neural ordinary differential equations (NODEs) are to learn a latent governing equation of ODE from data. In our presented method, we combine them into a single framework and propose a concept, called *neural diffusion equation* (NDE). Our NDE, equipped with the diffusion equation and one more additional neural network to model inherent uncertainty, can learn an appropriate latent governing equation that best describes a given climate dataset. In our experiments with two real-world and one synthetic datasets and eleven baselines, our method consistently outperforms existing baselines by non-trivial margins.**

***Index Terms*—climate modeling, diffusion equation, neural ordinary differential equation**

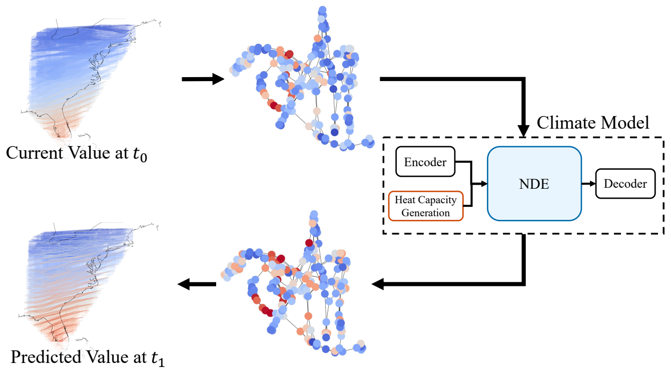
I. Introduction

Deep learning-based climate modeling (or weather forecasting) is an emerging topic and a brand-new application area [1]–[11]. Owing to the recent advancement of the differential equation-inspired deep learning [12]–[16], this specific topic is gathering much attention from the research community.

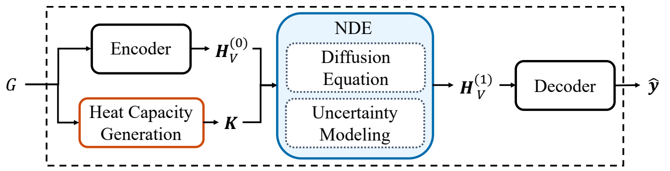
The seminal paper, titled neural ordinary differential equations (NODEs), discovered that residual networks are equivalent to the explicit Euler method to solve ODE problems [12]. Therefore, training residual networks is solving ODE problems specialized in image classification, according to them.

In NODEs, more specifically, we solve an integral problem of $\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t; \boldsymbol{\theta}_f)dt$, where $\boldsymbol{h}$ is a vector that contains a set of values that change over time $t \in [0, T]$ and $f(\boldsymbol{h}(t), t; \boldsymbol{\theta}_f) = \frac{d\boldsymbol{h}(t)}{dt}$. In other words, the multi-dimensional vector $\boldsymbol{h}(t_1)$ at time $t_1 > t_0$ is calculated by adding the sum of the changes in $[t_0, t_1]$ to $\boldsymbol{h}(t_0)$. The key of NODEs is the neural network f parameterized by $\boldsymbol{\theta}_f$, which is trained from data.

In the case of climate modeling, an area to model is frequently abstracted by a grid network (or a small-world network) and each grid cell corresponds to an element in $\boldsymbol{h}(t)$ [17]. We then learn $\boldsymbol{\theta}_f$ from data to minimize a climate modeling loss function. We typically use the mean squared error (MSE) as a loss function since climate modeling or weather forecasting is a regression problem.



(a)



(b)

Fig. 1. The overall workflow of our proposed *neural diffusion equation* (NDE). NDE stands for our proposed method as well as its key layer. (a) The weather stations (and their sensing values) are represented as a graph and our NDE predicts their future values. (b) Given a graph $G = (V, E)$ annotated with node and edge features, the core NDE layer evolves $\boldsymbol{H}_V^{(0)}$ to $\boldsymbol{H}_V^{(1)}$, followed by a decoding layer which produces predictions. The NDE layer consists of two parts: i) the diffusion equation and ii) the neural network-based uncertainty model f.

Whereas various recurrent neural network models and regression models can be used for this purpose, we propose to use the diffusion equation [18] under the regime of NODEs to model the physical dynamics governing the spread of temperature, air, etc. As a matter of fact, the diffusion equation is one of the most popular differential equations for climate modeling [19], [20].

The diffusion equation over a grid network uses the discrete Laplacian operator which is written as $\frac{d\boldsymbol{h}(t)}{dt} = -\Delta\boldsymbol{h}(t) = -k\boldsymbol{L}\boldsymbol{h}(t)$, where k is a heat capacity coefficient, Δ means the discrete Laplacian operator, and $\boldsymbol{L}$ is the symmetrically normalized graph Laplacian matrix. The physical process in this equation is governed by the current values $\boldsymbol{h}(t)$ at time t and the grid connectivity.

One thing worth mentioning in the above diffusion equation is that it does not have any parameters to learn since $\boldsymbol{L}$ and

$\boldsymbol{h}(0)$, i.e., the initial climate conditions, are given by data. In this regard, it is a pure ODE rather than NODE. To further increase the accuracy, in our case we extend the definition of the time-derivative of $\boldsymbol{h}(t)$ to $\frac{d\boldsymbol{h}(t)}{dt} = -k\boldsymbol{L}\boldsymbol{h}(t) + f(\boldsymbol{h}(t), t; \boldsymbol{\theta}_f)$ — we note that our model can be considered as a NODE since the time-derivative of $\boldsymbol{h}(t)$ is partially approximated by the neural network f. The neural network f is to learn the uncertainty of real-world diffusion processes. Therefore, we solve $\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} -k\boldsymbol{L}\boldsymbol{h}(t) + f(\boldsymbol{h}(t), t; \boldsymbol{\theta}_f)dt$, which we call *neural diffusion equation* (NDE). We also consider the case that k is not a scalar value but a matrix (see Section IV-C). For simplicity but without loss of generality, we assume k is a scalar in this section.

The overall workflow of our method is shown in Fig. 1. In most cases, weather stations and their sensing values are converted into a graph annotate with node and/or edge features, and we use our neural diffusion equation (NDE) method to predict future climate-related factors. In our proposed method, the heat capacity coefficients are also trained from data and so is the neural network-based uncertainty model. Since real-world environments accompany uncertainties and noises incurred by human activities and so forth, our uncertainty modeling is one of the most important part in the method. The heat capacity defines how quickly (easily) two neighboring nodes interact with each other, which is not explicitly given to us in many cases. Therefore, we propose to learn from data. Another key component in our method is the heat equation. We also implement the exact heat equation since non-trivial parts in climate modeling can still be modeled by it.

In our experiments, we show that our method, featured by i) the heat equation, ii) the uncertainty modeling, and iii) the heat capacity generation, shows the best accuracy in all datasets. We use five datasets: two of them are synthetic and the other three are real-world datasets. The synthetic datasets include a grid and small-world network (with and without noises injected into the data) — we note that climate models typically assume either a grid or a small-world network[1]. The real-world datasets includes the data collected in Los Angeles, San Diego, and the east side of the USA. We compare our method with eleven baseline methods and our proposed method NDE shows the best forecasting performance only except the multi-step forecasting in San Diego. Our contributions can be summarized as below:

1) We design a climate forecasting model with the diffusion equation, the neural network-based uncertainty modeling, and the heat capacity generation methods.
2) We conduct comprehensive experiments with synthetic and real-world datasets and compare with eleven baselines. Our proposed method, called *neural diffusion equation* (NDE), shows the best forecasting performance only except one experimental case. Our method's forecasting errors are up to 51% smaller than that of the best baseline method.

[1]Each research domain has its own preference on graph models. For instance, social networks typically assume scale-free networks.

II. Related Work

We introduce several base concepts in this section: i) neural ordinary differential equations, ii) diffusion equations, and iii) climate modeling.

A. Neural Ordinary Differential Equations (NODEs)

NODEs solve the following initial value problem (IVP), which involves an integral problem, to calculate $\boldsymbol{z}(t_1)$ from $\boldsymbol{z}(t_0)$ [12]:

$$\boldsymbol{z}(t_1) = \boldsymbol{z}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{z}(t); \boldsymbol{\theta}_f)dt, \tag{1}$$

where $f(\boldsymbol{z}(t); \boldsymbol{\theta}_f)$, which we call *ODE function*, is a neural network to approximate $\dot{\boldsymbol{z}} \stackrel{\text{def}}{=} \frac{d\boldsymbol{z}(t)}{dt}$. To solve the integral problem, NODEs rely on ODE solvers, such as the explicit Euler method, the Dormand–Prince (DOPRI) method, and so forth [21].

In general, ODE solvers discretize time variable t and convert an integral into many steps of additions. For instance, the explicit Euler method can be written as follows in a step:

$$\boldsymbol{z}(t+h) = \boldsymbol{z}(t) + h \cdot f(\boldsymbol{z}(t); \boldsymbol{\theta}_f), \tag{2}$$

where h, which is usually smaller than 1, is a pre-determined step size of the Euler method.The DOPRI method uses a much more sophisticated method to update $\boldsymbol{z}(t+h)$ from $\boldsymbol{z}(t)$ and dynamically controls the step size h. However, those ODE solvers sometimes incur unexpected numerical instability [22]. For instance, the DOPRI method sometimes keeps reducing the step size h and eventually, an underflow error is produced. To prevent such unexpected problems, several countermeasures were also proposed.

Instead of the backpropagation method, the adjoint sensitivity method is used to train NODEs for its efficiency and theoretical correctness [12]. After letting $\boldsymbol{a}_{\boldsymbol{z}}(t) = \frac{d\mathcal{L}}{d\boldsymbol{z}(t)}$ for a task-specific loss $\mathcal{L}$, it calculates the gradient of loss w.r.t model parameters with another reverse-mode integral as follows:

$$\nabla_{\boldsymbol{\theta}_f}\mathcal{L} = \frac{d\mathcal{L}}{d\boldsymbol{\theta}_f} = -\int_{t_m}^{t_0} \boldsymbol{a}_{\boldsymbol{z}}(t)^{\mathtt{T}} \frac{\partial f(\boldsymbol{z}(t); \boldsymbol{\theta}_f)}{\partial \boldsymbol{\theta}_f} dt.$$

$\nabla_{\boldsymbol{z}(0)}\mathcal{L}$ can also be calculate in a similar way and we can propagate the gradient backward to layers earlier than the ODE if any. It is worth of mentioning that the space complexity of the adjoint sensitivity method is $\mathcal{O}(1)$ whereas using the backpropagation to train NODEs has a space complexity proportional to the number of DOPRI steps. Their time complexities are similar or the adjoint sensitivity method is slightly more efficient than that of the backpropagation. Therefore, we can train NODEs efficiently.

B. Diffusion Equations

The diffusion equation is to describe the macroscopic behavior of many micro-particles in Brownian motion, resulting from the random movements and collisions of the particles [23]. In mathematics, it is related to Markov processes, such as random walks, and applied in many other fields, such as materials science, information theory, and biophysics [24]–[27]. The essence of the diffusion equation with a graph-based representation of data can be written as follows:

$$\frac{d\boldsymbol{h}(t)}{dt} = -\Delta \boldsymbol{h}(t) = -k\boldsymbol{L}\boldsymbol{h}(t), \tag{3}$$

where $\boldsymbol{h} \in \mathbb{R}^{|V|\times 1}$ is a vector that contains a value for each node, k is a heat capacity coefficient, Δ means the discrete Laplacian operator, and $\boldsymbol{L}$ is the symmetrically normalized augmented graph Laplacian matrix. Therefore, the change of the values in $\boldsymbol{h}$ over time t can be describe by the diffusion equation.

In our case, each node has multi-dimensional values, where the heat equation can be written as follows:

$$\frac{d\boldsymbol{H}(t)}{dt} = -\Delta \boldsymbol{H}(t) = -k\boldsymbol{L}\boldsymbol{H}(t), \tag{4}$$

where $\boldsymbol{H}$ is a matrix, each row of which contains multi-dimensional values for each node.

Analogy to Simple Graph Convolution (SGC): The simple graph convolution [28] method is one of the most efficient graph convolutional networks. Its main graph convolutional layer definition is as follows:

$$\boldsymbol{H}(m) = \boldsymbol{S}^m \boldsymbol{H}(0), \tag{5}$$

where $\boldsymbol{H}$ is a node hidden matrix, and $\boldsymbol{S}^m$ is the m-th power of the symmetrically normalized adjacency matrix $\boldsymbol{S}$ with added self-loops.

Now we derive Eq. (5) from Eq. (4) to show their analogy [29], [30]. When applying the Euler discretization to Eq. (4) with an interval step size dt, we have

$$\begin{aligned}\boldsymbol{H}(t+dt) &= \boldsymbol{H}(t) - dt\boldsymbol{L}\boldsymbol{H}(t)\\ &= \boldsymbol{H_t} - dt(\boldsymbol{I}-\boldsymbol{S})\boldsymbol{H}(t)\\ &= [(1-dt)\boldsymbol{I} + dt\boldsymbol{S}]\boldsymbol{H}(t)\\ &\overset{\text{def}}{=} \boldsymbol{S}^{dt}\boldsymbol{H}(t)\end{aligned} \tag{6}$$

We will get the following final $\boldsymbol{H}(T)$, if we keep evolving the ODE until the terminal time T.

$$\boldsymbol{H}(T) = [\boldsymbol{S}^{dt}]^m \boldsymbol{H}. \tag{7}$$

We regard that SGC corresponds to the Euler discretization with a unit step size $dt = 1$. This step size reduces the diffusion matrix to the Linear GCN diffusion matrix $\boldsymbol{S}$:

$$\boldsymbol{S}^{(dt)}|_{dt=1} = (1-1)\boldsymbol{I} + \boldsymbol{S} = \boldsymbol{S} \tag{8}$$

and the final $\boldsymbol{H}$ becomes equivalent:

$$\boldsymbol{H}(T)|_{dt=1} = \boldsymbol{H}(m) = \boldsymbol{H}^{(m)} = \boldsymbol{S}^m\boldsymbol{H}. \tag{9}$$

Therefore, Eq. (9) is equivalent to Eq. (4) when the Euler method with a unit step size is used. As mentioned earlier, however, we also have many other ODE solvers, such as DOPRI and so forth. Thus, SGC is a special case of the diffusion equation. In this regard, our method, whose one part is the diffusion equation, is able to learn from and infer about graphs. In other words, our proposed model is a spatio-temporal model.

C. Climate Modeling and Weather Forecasting

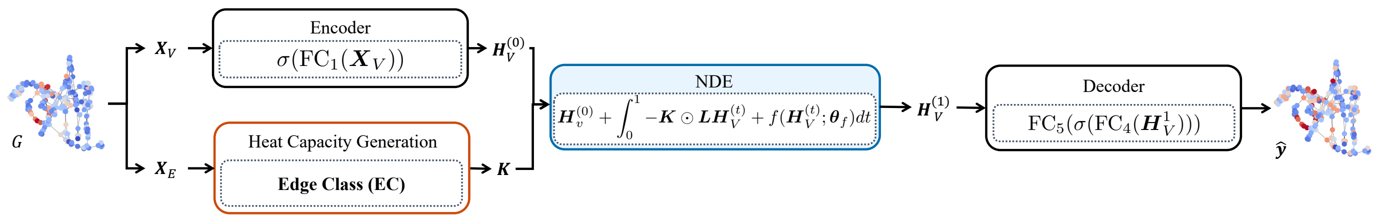
Climatologists study the natural factors that cause climate change, using past information to help predict future climate change. Climatological phenomena includes (radiative, convective, and latent) heat transfer, interactions among atmosphere, oceans and surface, and chemical and physical composition of the atmosphere. Climate model elements that can constitute such phenomena are typically differential equations based on physics, fluid motion, and chemistry [31]. In weather forecasting tasks, temperature change is related to a transport problem which occurs by diffusion principles. This equation describes a large family of physical processes(heat conduction, wind dynamics, fluid dynamics, etc.) [32]. For this reason, we focus on designing our proposed method with the diffusion equation.

Various deep learning-based models are designed for climate and weather forecasting [1], [6], [9], [10], [33], [34], including near-surface air temperature predictions [17], [35], air quality inference [8], [36], precipitation predictions [2], [3], [37], wind speed predictions [7], [38], [39] and extreme weather predictions [4], [5]. Climate and weather consists of spatio and temporal data, leading to the development of spatio-temporal models [11], [40]. However, models accounting spatial and temporal dependencies do not consider the differential equations related to the climate model. Recently, researches are trying to leverage physical knowledge for climate predictions. De Bezenac et al. [41] used transport physics (diffusion and advection) to predict the sea surface temperature, but this is limited to a regular grid. DPGN [17] designed a physics-informed learning architecture to predict the air temperature. DPGN incorporated differentiable physics equations with a graph neural network framework and used as a physics-informed regularizer. Unlike the above models, we learn the heat capacity of the diffusion equation, which is one important point that have been overlooked for a while, and consider the uncertain nature from the real-word climate data.

III. Problem Definition

We solve various climate modeling-related problems in this work, using our proposed method. One typical problem is predicting a climate-related factor a's next value from recent values of a set B factors, where i) $a \in B$, e.g., a is next temperature and B includes recent temperature, or ii) $a \notin B$, e.g., a is next temperature and B does not include temperature but humidity, wind velocity, and so on.

It is well known that these processes can be described by diffusion equations under ideal conditions. Due to the uncertain nature of real-world climate data, however, our problem definition is to model the relationships among various noisy climate factors. This problem definition falls into the category of regression.



G
X_V
X_E
Encoder
$\sigma(\mathrm{FC}_1(X_V))$
Heat Capacity Generation
Edge Class (EC)
$H_V^{(0)}$
K
NDE
$H_v^{(0)} + \int_0^1 -K \odot LH_V^{(t)} + f(H_V^{(t)};\theta_f)dt$
$H_V^{(1)}$
Decoder
$\mathrm{FC}_5(\sigma(\mathrm{FC}_4(H_V^1)))$
$\hat{y}$

Fig. 2. The detailed workflow of NDE

IV. Proposed Method

We describe our proposed neural diffusion equation method to model the spatio-temporal relationships among climate factors.

A. Overall Architecture

Our method consists of four modules as shown in Fig. 2: i) encoding layer, ii) heat capacity generation layer, iii) neural diffusion equation layer, and iv) decoding layer. Given a graph $G = (V, E)$, a node feature matrix $\boldsymbol{X}_V \in \mathbb{R}^{|V|\times M}$, and an edge feature matrix $\boldsymbol{X}_E \in \mathbb{R}^{|E|\times N}$, the role of each module is as follows:

1) The encoding layer converts $\boldsymbol{X}_V$ into a hidden matrix $\boldsymbol{H}_V^{(0)} \in \mathbb{R}^{|V|\times D}$, where D is the size of hidden vector;
2) The heat capacity generation layer creates the heat capacity coefficient(s). Our method supports four different heat capacity concepts and the details will be described shortly;
3) The neural diffusion equation layer processes $\boldsymbol{H}_V^{(0)}$ to generate $\boldsymbol{H}_V^{(1)}$, following the learned neural diffusion equation;
4) The decoding layer predicts a target climate factor for each node from $\boldsymbol{H}_V^{(1)}$.

The node feature matrix $\boldsymbol{X}_V$ can have values for each node for the previous P time steps to learn from historical patterns.

B. Initial Encoding Layer

The node feature matrix $\boldsymbol{X}_V \in \mathbb{R}^{|V|\times M}$ contains the input features of nodes. We use the following encoding layer to produce their initial hidden vectors:

$$\sigma(\mathtt{FC}_1(\boldsymbol{X}_V)), \tag{10}$$

where σ is a rectified linear unit (ReLU) and $\mathtt{FC}$ is a fully-connected layer.

Instead of directly feeding the raw climate factors contained by $\boldsymbol{X}_V$ into the neural diffusion layer, we first create initial hidden vectors.

C. Heat Capacity Generation Layer

The heat capacity generation layer produces one of the four different types of the heat capacity from the edge feature matrix $\boldsymbol{X}_E \in \mathbb{R}^{|E|\times N}$. In diffusion equations, a heat capacity coefficient of an edge represents how quickly (or easily) climate factors are diffused via the edge. Therefore, it is one of the most crucial points in our model to learn a reliable heat capacity. We support the following four types:

1) **Edge Class (EC):** We learn a heat capacity coefficient for each class of edges when edge features are one-hot vectors. Those edges which share the same one-hot vector also share the same heat capacity coefficient, denoted $k_c \in \mathbb{R}$, where c means a class of edges;
2) **Heat Matrix (HM):** We learn a heat capacity matrix $\boldsymbol{K} \in \mathbb{R}^{|V|\times|V|}$. We learn the entire matrix in this case;
3) **Single Coefficient (SC):** We learn a single heat capacity coefficient $k \in \mathbb{R}$ which will be shared by all edges;
4) **Fixed Coefficient (FC):** We also consider the case of $k = 1$. In fact, this is one of the ablation study models.

The first method, denoted **EC**, can be efficiently implemented using the embedding layer API of PyTorch and TensorFlow, which outputs an embedding vector given a one-hot vector. In our case, it outputs a scalar heat capacity coefficient.

D. Neural Diffusion Equation Layer

This neural diffusion equation layer is the core of our proposed method. It evolves $\boldsymbol{H}_V^{(0)}$ to $\boldsymbol{H}_V^{(1)}$ using the learned neural diffusion equation. The key computation method is as follows:

$$\boldsymbol{H}_V^{(1)} = \boldsymbol{H}_V^{(0)} + \int_0^1 -\boldsymbol{K} \odot \boldsymbol{L}\boldsymbol{H}_V^{(t)} + f(\boldsymbol{H}_V^{(t)}, t; \boldsymbol{\theta}_f)dt, \tag{11}$$

where $\odot$ is an element-wise product, and $\boldsymbol{L}$ is the symmetrically normalized augmented graph Laplacian matrix of the graph G. In the above equation, we assume the heat capacity of **EC** and **HM** since in both cases the heat capacity is represented by a matrix $\boldsymbol{K}$. For **SC** and **FC**, we use a scalar coefficient k.

Without the neural network f, Eq. (11) reduces to the diffusion equation with the discrete Laplace operator. In real-world environments, however, it is hard to say that diffusion processes are completely governed by the diffusion equation — in particular, we observe in our experiments that diffusion processes around large cities have non-trivial uncertainties that cannot be solely described by the diffusion equation. To this end, we let the neural network f learn the uncertainty. Therefore, $\frac{d\boldsymbol{H}_V^{(t)}}{dt}$ is governed by the heat equation and the learned neural network. The definition of f is as follows:

$$f(\boldsymbol{H}_V^{(t)}, t; \boldsymbol{\theta}_f) \overset{\text{def}}{=} \sigma(\mathtt{FC}_3(\sigma(\mathtt{FC}_2(\boldsymbol{H}_V^{(t)})))). \tag{12}$$

Algorithm 1: How to train our proposed NDE

```
Input: Training data $D_{train}$, Validating data $D_{val}$,
       Maximum iteration number $max\_iter$
1 Initialize $\boldsymbol{\theta}_f$ and other parameters $\boldsymbol{\theta}_{others}$ if any, e.g., the
    parameters of the encoding, decoding, and other layers;
2 $k \leftarrow 0$;
3 while $k < max\_iter$ do
4     Train $\boldsymbol{\theta}_f$ and $\boldsymbol{\theta}_{others}$;
5     Validate and update the best parameters, $\boldsymbol{\theta}^*_f$ and $\boldsymbol{\theta}^*_{others}$,
        with $D_{val}$;
6     $k \leftarrow k + 1$;
7 return $\boldsymbol{\theta}^*_f$ and $\boldsymbol{\theta}^*_{others}$;
```

E. Decoding Layer

The decoding layer is to produce predictions from $\boldsymbol{H}_V^{(1)}$. The decoding layer definition is as follows:

$$\hat{\boldsymbol{y}} = \mathtt{FC}_5(\sigma(\mathtt{FC}_4(\boldsymbol{H}_V^{(1)}))), \tag{13}$$

where $\hat{\boldsymbol{y}} \in \mathbb{R}^{|V|\times 1}$ contains a climate factor value for each node.

F. Multi-step Forecasting

While the default forecasting is to predict very next value for a target climate factor from $\boldsymbol{X}_V$, our method also supports predicting for next multiple S time steps. We keep evolving $\boldsymbol{H}_V^{(i-1)}$ to $\boldsymbol{H}_V^{(i)}$ until $i = S$ as follows, using the neural diffusion equation layer:

$$\begin{aligned}
\boldsymbol{H}_V^{(1)} &= \boldsymbol{H}_V^{(0)} + \int_0^1 -\boldsymbol{K} \odot \boldsymbol{L}\boldsymbol{H}_V^{(t)} + f(\boldsymbol{H}_V^{(t)}, t; \boldsymbol{\theta}_f)dt,\\
\boldsymbol{H}_V^{(2)} &= \boldsymbol{H}_V^{(1)} + \int_1^2 -\boldsymbol{K} \odot \boldsymbol{L}\boldsymbol{H}_V^{(t)} + f(\boldsymbol{H}_V^{(t)}, t; \boldsymbol{\theta}_f)dt,\\
&\vdots\\
\boldsymbol{H}_V^{(S)} &= \boldsymbol{H}_V^{(S-1)} + \int_{S-1}^S -\boldsymbol{K} \odot \boldsymbol{L}\boldsymbol{H}_V^{(t)} + f(\boldsymbol{H}_V^{(t)}, t; \boldsymbol{\theta}_f)dt.
\end{aligned} \tag{14}$$

One can consider that this mechanism corresponds to a continuous recurrent network [42], [43]. The decoding layer then predicts for each time step $s \in \{1, 2, \cdots, S\}$ using $\boldsymbol{H}_V^{(s)}$.

G. Training Algorithm

We use Alg. 1 to train our proposed NDE. For the gradient calculation, we use the following mean squared error loss function:

$$\mathcal{L} \overset{\text{def}}{=} \frac{\sum_{i=1}^{|V|}(\boldsymbol{y}_{[i]} - \hat{\boldsymbol{y}}_{[i]})^2}{|V|}, \tag{15}$$

where $\boldsymbol{y}_{[i]}$ means the i-th element of $\boldsymbol{y}$.

The training algorithm follows a standard method to update the parameters and validate with a validation set. One more thing worth mentioning is that the training process can be theoretically well-posed.

TABLE I
DATASET DESCRIPTION

Dataset	LA & SD	NOAA
Train	2012/6/28 9pm-2012/07/8 7am	2015/1/1 0am-2015/9/13 12pm
Valid	2012/7/8 7am-2012/07/10 9am	2015/9/13 12pm-2015/10/20 0am
Test	2012/7/10 9am-2012/07/14 10pm	2015/10/20 0am-2016/1/1 0am

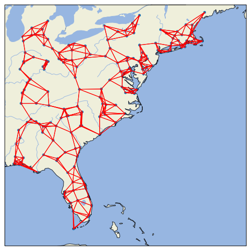


Fig. 3. Weather stations in the eastern states of the US and its 4-NN graph.

Well-posedness of the Solution of NDE: The well-posedness[2] of ODEs was already proved in [44, Theorem 1.3] under the mild condition of the Lipschitz continuity of $\frac{d\boldsymbol{h}(t)}{dt}$, i.e., $\frac{d\boldsymbol{H}(t)}{dt}$ in our case. The matrix multiplication of the heat equation is a representative Lipschitz continuous operator with a specific Lipschitz constant. In addition, almost all activations, such as ReLU, Leaky ReLU, SoftPlus, Tanh, Sigmoid, ArcTan, and Softsign, have a Lipschitz constant of 1. Other common neural network layers, such as dropout, batch normalization and other pooling methods, have explicit Lipschitz constant values. Therefore, the Lipschitz continuity of $\frac{d\boldsymbol{H}(t)}{dt}$ can be fulfilled in our case, and our training algorithm solves a well-posed problem so its training process is stable in practice.

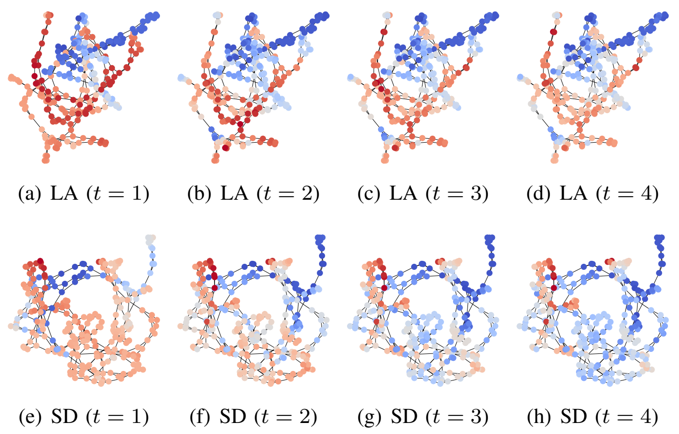
V. Experiments

We conduct experiments with synthetic and real-world datasets for climate modeling. All experiments were conducted in the following software and hardware environments: UBUNTU 18.04 LTS, PYTHON 3.8.0, NUMPY 1.18.5, SCIPY 1.5, MATPLOTLIB 3.3.1, PYTORCH 1.7.0, CUDA 10.0, and NVIDIA Driver 417.22, i9 CPU, and NVIDIA RTX TITAN. We repeat the training and testing procedures with ten different random seeds and report their mean and standard deviation accuracy — resources are accessible at https://github.com/jeehyunHwang/Neural-Diffusion-Equation.

A. Experimental Environments

1) Synthetic Data: We test with synthetic data from physics. Network dynamics are generated based on the analytical solutions of the diffusion equation referred in Eq. (4). We consider a grid network and a small-world network. Each node of the grid network has eight connected neighbors. In particular, we add the small-world network due to the fact that climate networks can be regarded as small-world networks [45], [46]. In fact, the real-world datasets we use later are classified as small-world networks because all the small-coefficients, which are coefficients for quantifying small-worldness [47], [48], are greater than 1. We use the Watts-Strogats model to generate the small-world network [49]. We first generate

[2]A well-posed problem means i) its solution uniquely exists, and ii) its solution continuously changes as input data changes.


(a) LA ($t = 1$) (b) LA ($t = 2$) (c) LA ($t = 3$) (d) LA ($t = 4$)

(e) SD ($t = 1$) (f) SD ($t = 2$) (g) SD ($t = 3$) (h) SD ($t = 4$)

Fig. 4. The graph representation of the LA and SD areas. There are 274 and 282 nodes, respectively. These graphs show the change of the air temperature over time.

ground-truth values of each network dynamics including 400 nodes using the Dormand-Prince method [50]. We sample 100 snapshots of the created continuous-time dynamics with the same time-interval, and use first 80 snapshots for training and the remaining 20 snapshots for testing.

2) LA and SD Data: We also use a dataset that consists of hourly climate observations over 16 days (June/28/2012 21:00 to July/14/2012 22:00) in Los Angeles and San Diego areas, including 274 and 282 nodes, respectively (as shown in Fig. 4). Each node includes 10 climate observations: air temperature, albedo, precipitation, soil moisture, relative humidity, specific humidity, surface pressure, planetary boundary layer height, and wind vector (2 directions). Listed observations are selected due to its characteristic of high relation to diffusion equation [19]. The dataset also includes edge attributes representing the land-usage types of source and destination nodes, resulting in 43 different edge connection classes. The discrete time unit is one hour. The statistics of all real-world datasets we use for our experiments are summarized in Table I.

3) NOAA Data: We conduct experiments with a dataset including hourly average temperature derived from the Online Climate Data Directory of the National Oceanic and Atmospheric Administration(NOAA)[3]. We select 188 stations located in a region of the latitude in [23.886, 44.371] and the longitude in [-87.188, -67.605] as shown in Fig. 3, from 2015 of hourly climate normals dataset. The imputation of missing values is done using the average value of 2 time-wise nearest neighbors for experimental purposes. We adopt 4-NN (Nearest Neighbor) algorithm to construct graph structure, and the created adjacency matrix $\boldsymbol{A}$ is converted by $\boldsymbol{A} = (\boldsymbol{A} + \boldsymbol{A}^T)/2$ to make it symmetric. The discrete time unit is one hour.

4) Evaluation Method: For the synthetic dataset, we perform a time-series forecasting of reading the first 80% and predicting the last 20% of observations, i.e., extrapolation. We exclude the interpolation from this evaluation since almost all models show reasonable accuracy for it. However, they show significantly different accuracy for the extrapolation. Since each node has one scalar value strictly following the diffusion equation, this dataset is relatively more straightforward than other real-world datasets.

With the LA and SD datasets, we feed P recent observations, where $P = \{1, 3, 5\}$, and predict S next values, where $S = \{1, 10\}$, i.e., sequence-to-sequence predictions. This specific experimental setting had been used in [17] as well. In particular, the input node feature does not include the air temperature but the ground-truth output is the air temperature, i.e., the case of $a \notin B$ in Section. III. This forecasting can be used to reconstruct the sensing values of malfunctioning/dead sensors from others.

In the NOAA dataset, we predict next S temperature values, where $S = \{1, 6\}$, from recent P temperature values, where $P = \{1, 6\}$, i.e., the case of $a \in B$ in Section. III.

In all datasets, we use the evaluation metrics of mean absolute error (MAE) and/or mean squared error (MSE). In all cases, we run with 10 different random seeds and report their mean and standard deviation accuracy.

5) Baselines: We consider the following baseline to compare with our method:

1) In the first group of baselines, they ignore graph connectivity and perform individual forecasting for each node as follows:
   a) Multi-Layer Perceptron (MLP) is a basic neural network model which uses a series of fully-connected layers. We set the number of layers to 2, and the hidden size to 64.
   b) Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and Gated Recurrent Units (GRU) are popular models to deal with time-series data. We set the number of layers to 2, and the hidden size to 64.
2) The next group includes the temporal-GNN models, which are combinations of GNN and RNN models. The temporal-GNN model adopt the GNN to capture the graph structure and RNN/LSTM/GRU to learn the temporal information.
3) DPGN [17] and NDCN [51] are two differential equation-based models to predict values that follow the diffusion equation. They do not consider learning the heat capacity coefficients and modeling uncertainty.
   a) The key of DPGN is injecting the knowledge of the diffusion equation into neural networks. To this end, it uses a technique known as *physics-informed neural network* (PINN [52]).
4) We also consider two variants of DPGN: GN-skip and GN-only. The GN-only model includes three modules, which are a graph encoder, a graph network (GN) block [53], and a graph decoder. The GN-skip model connects the input and output of the GN block with a skip-connection [54]. For these two variants of DPGN,

[3]https://www.ncdc.noaa.gov/cdo-web/

however, we do not inject the knowledge of the heat equation.

6) Hyperparameters: We consider the following ranges of the hyperparameters: the hidden vector size D is in $\{8, 20, 16, 32\}$. The number of layers of f in Eq. (12) is in $\{2,3\}$. The hidden vector size D is in $\{128, 256\}$. The learning rate in all methods is in $\{1 \times 10^{-2}, 1 \times 10^{-3}, 1 \times 10^{-4}\}$.

For reproducibility, we report the best hyperparameters for our method as follows:

1) Synthetic Data: the learning rate is 1×10^{-2}, the weight decay is 1×10^{-3}, the hidden vector size D is 20;
2) LA: the learning rate is 1×10^{-3}, the weight decay is 5×10^{-4}, the hidden vector size D is 32;
3) SD: the learning rate is 1×10^{-3}, the weight decay is 5×10^{-4}, the hidden vector size D is 32;
4) NOAA: the learning rate is 1×10^{-3}, the weight decay is 0, the hidden vector size D is 32.

B. Experimental Results

1) Synthetic Data: Table II summarizes the mean and standard deviation accuracy of the extrapolation experiments with the synthetic datasets for various models. To inject some uncertainty into the data, we add temporal noises and consider both with and without the noises. RNN-based models, such as RNN and RNN-GNN, show relatively poor accuracy. This is because RNN models do not have enough capacity to learn from the first 80% of observations, which is rather long to be processed by RNNs. LSTM and GRU-based models show reasonable accuracy. GRU-based models are better than LSTM-based models in our experiments without noises in the grid network. When there exist noises, LSTM outperforms GRU. Both LSTM and GRU are capable of learning from the long sequence, i.e., the first 80% of observations, and show better accuracy than RNN. One more result worth mentioning is that GRU-GNN outperforms GRU in the grid network when there are no noises, which shows the efficacy of processing both temporal and spatial information.

However, all the best outcomes are made by differential equation-based models, NDCN and NDE. Among them, our NDE shows much smaller MAE values. The MAE of NDE is 55% of that of NDCN, which is about 45% smaller for the grid network. In the case of the small-world network, NDE shows a 15% smaller MAE than that of NDCN. Since our NDE explicitly models noise (uncertainty), it shows better accuracy than others.

Fig. 5 visualizes some ground-truth and predicted values in the grid network. The true diffusion examples in Fig. 5 (a-c) show that there exist three thermal points in (a) but as time goes by, they are diffused over the 2-dimensional space. We note that the time points we use in these figures, i.e., $t \in \{21.5, 81, 90\}$, do not belong to the training set. Therefore, $t = 21.5$ means an interpolation and the other two cases mean extrapolations. Our prediction by NDE successfully interpolate and extrapolate the diffusion process as shown in the figures. Fig. 6 visualizes for the small-world network and we can observe similar patterns.

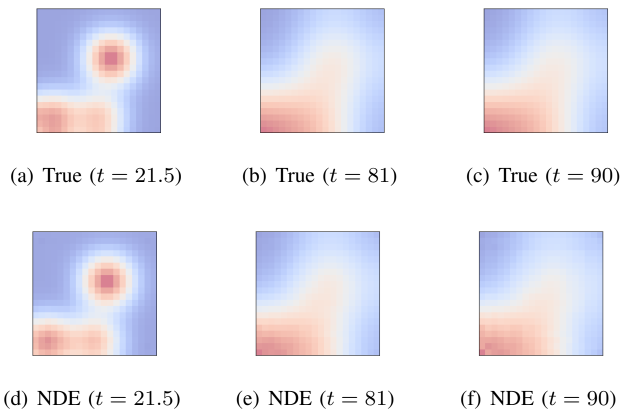

(a) True ($t = 21.5$) (b) True ($t = 81$) (c) True ($t = 90$)
(d) NDE ($t = 21.5$) (e) NDE ($t = 81$) (f) NDE ($t = 90$)

Fig. 5. The visualization of the interpolation ($t = 21.5$) and the extrapolation ($t = 81$ and $t = 90$) on the grid network

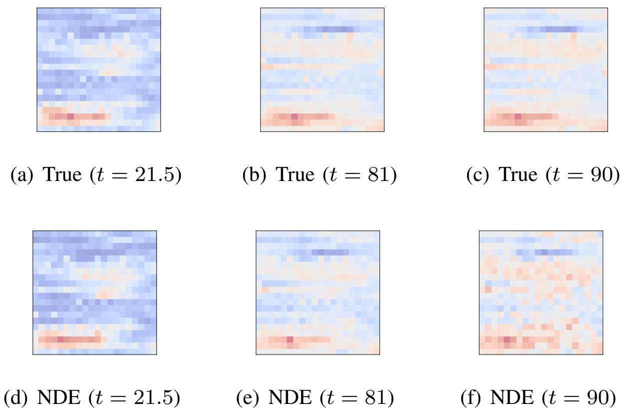

(a) True ($t = 21.5$) (b) True ($t = 81$) (c) True ($t = 90$)
(d) NDE ($t = 21.5$) (e) NDE ($t = 81$) (f) NDE ($t = 90$)

Fig. 6. The visualization of the interpolation ($t = 21.5$) and the extrapolation ($t = 81$ and $t = 90$) on the small-world network

2) LA and SD Data: Table III summarizes the forecasting results in the LA and SD datasets. For the one next step forecasting of LA, MLP and RNN-based models show the biggest errors (MSEs) whereas other differential equation-based models show reasonable accuracy. One interesting point is that the combination of RNN and GNN does not significantly improve the accuracy over the only RNN-based models. In comparison with GN-skip, both DPGN and NDCN significantly decrease the errors, i.e., a MSE of 0.5654 for GN-skip vs. 0.4435 of DPGN. However, our NDE further decreases the error to 0.2628, which is about 41% smaller. For the one next step forecasting in SD, we observe similar patterns with a little smaller margin of 31% between DPGN and NDE. However, the gap between them is still non-trivial.

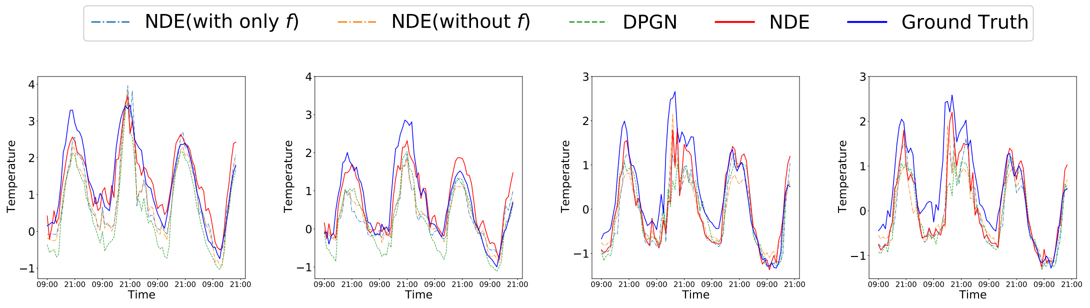
For the multi-step forecasting in LA, many baseline methods show unreliable outcomes in comparison with our method, e.g., an MSE of 1.2588 by NDCN vs. 0.7594 by NDE. Except our method, DPGN shows a small error.

For the multi-step forecasting in SD, however, our method does not show the best accuracy. Our NDE is comparable to NDCN and RNN-GNN. The smallest error is achieved by

TABLE II

PREDICTION ERRORS (MAES) IN SYNTHETIC DATA FROM HEAT DIFFUSION. EACH RESULT IS THE MEAN AND THE STANDARD DEVIATION WITH 10 RUNS.

Model	Grid without noise	Small-world without noise	Grid with noise	Small-world with noise	#Params
RNN	0.7188 ± 0.2464	0.1452 ± 0.0697	1.0087 ± 0.0014	0.6423 ± 0.0590	24,530
LSTM	0.5374 ± 0.1352	0.1561 ± 0.0162	0.8230 ± 0.0531	0.5984 ± 0.0017	84,890
GRU	0.4887 ± 0.0145	0.1050 ± 0.0179	0.8815 ± 0.1021	0.6121 ± 0.0149	64,770
RNN-GNN	0.7511 ± 0.2253	0.1783 ± 0.0641	1.0088 ± 0.0017	0.6426 ± 0.0585	24,530
LSTM-GNN	0.5303 ± 0.0984	0.1739 ± 0.0524	0.8182 ± 0.1016	0.6287 ± 0.0214	84,890
GRU-GNN	0.4530 ± 0.0774	0.1054 ± 0.0178	1.0042 ± 0.0692	0.6249 ± 0.0247	64,770
NDCN	0.2007 ± 0.3963	0.0799 ± 0.0194	0.6882 ± 0.0641	0.5921 ± 0.2421	901
NDE	**0.1121 ± 0.0229**	**0.0687 ± 0.0260**	**0.4569 ± 0.0347**	**0.4822 ± 0.2124**	1,761



(a) Node 1 in LA (b) Node 2 in LA (c) Node 1 in SD (d) Node 2 in SD

Fig. 7. We visualize the ground-truth and predicted values for two nodes of the LA and SD graphs. (a-b) One-step near-surface air temperature forecasting in LA from 2012/07/10 9:00 to 2012/07/14 22:00 (c-d) One-step near-surface air temperature forecasting in SD from 2012/07/10 9:00 to 2012/07/14 22:00

DPGN (although both DPGN and NDE rely on the diffusion equation in their model designs). We think that the training mechanism of PINN used by DPGN is more effective than our training method for SD. Recall that our method is more effective than the PINN mechanism of DPGN for LA, which shows the difficulty of learning diffusion processes to model climate. One more difficulty of SD is that it contains many unpaved regions, as noted in [17].

Fig. 7 shows the ground-truth and predicted values by time. In these figures, we compare only the highly performing differential equation-based methods. With the human visual evaluation in the figures, our NDE shows the best matches with the ground-truth values. DPGN's predictions are clearly worse than those of our method.

3) NOAA Data: For this dataset which does not have edge features, DPGN, GN-only and GN-skip cannot be tested because they require edge features. In NOAA, differential equation-based methods show a good match. MLP and RNN-based methods show much larger errors than those of NDCN and NDE. Among RNN-based models, LSTM shows relatively smaller errors. However, NDCN shows much smaller errors than them, and our NDE shows the smallest errors among all methods.

C. Ablation and Sensitivity Studies

With the LA and SD datasets, we conduct ablation studies. First, we compare the heat capacity generation methods in Table V. In all cases, as shown, the heat capacity generation by edge class shows the smallest errors, followed by the single coefficient and the fixed coefficient methods.

We also carried out sensitivity studies by varying P, i.e., varying how much past information we feed into the model. In Table VI, it shows that climate modeling does not require long history. The diffusion equation also requires only $\boldsymbol{H}(t)$ to derive $\frac{d\boldsymbol{H}(t)}{dt}$, which means past history is not needed in the diffusion equation. Our experimental results also show similar patterns in LA and SD.

Table VII shows the performance by varying the hidden dimension size D. We test up to $D = 32$ and $D = 32$ shows the smallest errors. In particular, the performance gap between $D = 16$ and $D = 32$ is large in SD and NOAA.

In Fig. 7 and other main result tables, the two ablation study models, i.e., NDE(without f) and NDE(with only f), do not produce as good predictions as those of the full model, NDE. One more point worth mentioning is that NDE(with only f) sometimes requires a large model for f since it relies only on f, e.g., 85,089 parameters in NOAA by NDE(with only f) vs. 53,585 by NDE.

D. Model Efficiency Analyses

Fig. 8 shows the number of parameters and the error. Models at the bottom left corner in this figure are preferred. Our NDE and its variants are located around the bottom left corner.

In general, RNN-based models show low efficiencies in the figure, followed by RNN-GNN, GRU-RNN, and NDCN. One more interesting point is that our ablation study models also show good efficiencies in general.

TABLE III
PREDICTION ERRORS (MSEs) ON THE LA AND SD

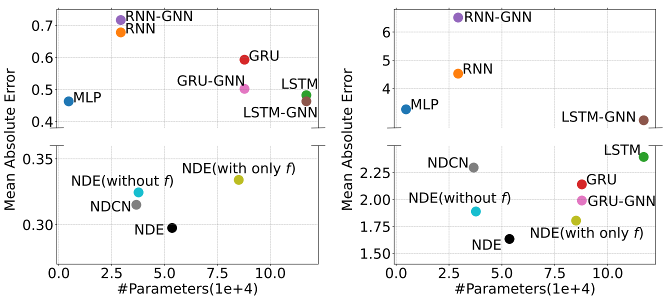
Model	LA One-step	SD One-step	LA Multi-step	SD Multi-step	LA #Params	SD #Params
MLP	0.6902 ± 0.0171	0.5863 ± 0.0123	1.2766 ± 0.0143	1.0030 ± 0.0154	4,865	4,865
RNN	0.6295 ± 0.0178	0.5411 ± 0.0231	1.2869 ± 0.0172	0.9065 ± 0.0172	29,825	29,825
LSTM	0.6723 ± 0.0074	0.5651 ± 0.0269	1.3043 ± 0.0106	0.8811 ± 0.0225	119,105	119,105
GRU	0.6486 ± 0.0049	0.5749 ± 0.0082	1.2872 ± 0.0141	0.8852 ± 0.0084	89,345	89,345
RNN-GNN	0.6607 ± 0.0064	0.5291 ± 0.0578	1.1844 ± 0.0727	0.7862 ± 0.0475	29,825	29,825
LSTM-GNN	0.7007 ± 0.0008	0.5754 ± 0.0180	1.1392 ± 0.0252	0.7954 ± 0.0110	119,105	119,105
GRU-GNN	0.6914 ± 0.0098	0.5705 ± 0.0057	1.1722 ± 0.0726	0.7414 ± 0.0294	89,345	89,345
GN-only	0.6035 ± 0.0832	0.7007 ± 0.0848	1.3415 ± 0.1195	1.0422 ± 0.0673	45,696	45,696
GN-skip	0.5654 ± 0.1015	0.6543 ± 0.1195	1.0257 ± 0.1912	0.9872 ± 0.2425	45,696	45,696
DPGN	0.4435 ± 0.0378	0.5149 ± 0.0831	0.8677 ± 0.1033	**0.6714 ± 0.1106**	45,696	45,696
NDCN	0.5380 ± 0.0469	0.5296 ± 0.0274	1.2588 ± 0.0654	0.7542 ± 0.0730	75,525	75,525
NDE(with only f)	0.3439 ± 0.0922	0.6009 ± 0.2001	0.8635 ± 0.0267	0.7960 ± 0.0854	4,851	9,889
NDE(without f)	0.3493 ± 0.0488	0.5522 ± 0.0582	0.9730 ± 0.1266	0.8731 ± 0.0236	492	1,454
NDE	**0.2621 ± 0.0026**	**0.3561 ± 0.0055**	**0.7594 ± 0.0225**	0.7301 ± 0.0048	4,894	9,934

TABLE IV
PREDICTION ERRORS (MAEs) ON NOAA

Model	One-step	Multi-step	#Params
MLP	0.4629 ± 0.0187	3.2582 ± 0.5518	4,673
RNN	0.6783 ± 0.0675	4.5232 ± 0.2807	29,313
LSTM	0.4822 ± 0.0356	2.4247 ± 0.0906	117,057
GRU	0.5930 ± 0.0838	2.1406 ± 0.0409	87,809
RNN-GNN	0.7169 ± 0.1131	6.5125 ± 1.4519	29,313
LSTM-GNN	0.4631 ± 0.0292	2.3955 ± 0.7086	117,057
GRU-GNN	0.5020 ± 0.1098	1.9906 ± 0.1261	87,809
NDCN	0.3151 ± 0.0122	2.2967 ± 0.0415	36,657
NDE(with only f)	0.3340 ± 0.0265	1.8039 ± 0.1017	85,089
NDE(withouf f)	0.3245 ± 0.0161	1.8890 ± 0.0565	37,713
NDE	**0.2975 ± 0.0062**	**1.6337 ± 0.0467**	53,585

TABLE V
SENSITIVITY W.R.T. THE HEAT CAPACITY GENERATION METHOD FOR ONE-STEP PREDICTION

Heat Capacity Generation Method	LA	SD
FC	0.3657 ± 0.0117	0.7701 ± 0.1164
SC	0.2901 ± 0.0096	0.5863 ± 0.0550
HM	0.5063 ± 0.1031	0.6033 ± 0.0382
EC	**0.2621 ± 0.0026**	**0.3561 ± 0.0055**



(a) One-step prediction (b) Multi-step prediction

Fig. 8. Model efficiency in NOAA. The bottom left corner is preferred.

VI. CONCLUSION

In this paper, we tackled the problem of one-step and multi-step climate factor forecasting. The diffusion equation with

TABLE VI
SENSITIVITY W.R.T. P

P	LA	SD
1	0.2621 ± 0.0026	0.3561 ± 0.0055
3	0.3332 ± 0.0459	0.4495 ± 0.0899
5	0.3229 ± 0.0868	0.4289 ± 0.0576

TABLE VII
SENSITIVITY W.R.T. THE HIDDEN DIMENSION SIZE D

Size of D	LA	SD	NOAA
8	0.2787 ± 0.0049	0.5223 ± 0.0492	0.4355 ± 0.0092
16	0.2628 ± 0.0132	0.5491 ± 0.0671	0.3789 ± 0.0135
32	0.2621 ± 0.0026	0.3561 ± 0.0055	0.2975 ± 0.0062

uncertainty modeling is used in our research. To this end, our presented method learns i) the heat capacity generation method and ii) the neural network-based uncertainty model. In the end, these two modules are combined with the diffusion equation. Our comprehensive experiments with synthetic and real-world datasets show the best efficiency of our method, NDE. In general, NDE achieves the best (or nest-best) accuracy with a relatively smaller number of parameters and GPU memory space complexity during inference.

In the future, we will study a more principled method to model uncertainty. One possible approach is to use stochastic differential equations (SDEs) [55], [56].

ACKNOWLEDGEMENT

Noseong Park (noseong@yonsei.ac.kr) is the corresponding author. This work was supported by the Yonsei University Research Fund of 2021, and the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University), and No. 2021-0-00155, Context and Activity Analysis-based Solution for Safe Childcare).

REFERENCES

[1] M. A. Zaytar and C. El Amrani, "Sequence to sequence weather forecasting with long short-term memory recurrent neural networks," *International Journal of Computer Applications*, 2016.

[2] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *NeurIPS*, vol. 28, 2015.

[3] X. Shi, Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Deep learning for precipitation nowcasting: A benchmark and a new model," *arXiv preprint*, 2017.

[4] Y. Liu, E. Racah, J. Correa, A. Khosrowshahi, D. Lavers, K. Kunkel, M. Wehner, W. Collins *et al.*, "Application of deep convolutional neural networks for detecting extreme weather in climate datasets," *arXiv preprint*, 2016.

[5] E. Racah, C. Beckham, T. Maharaj, S. E. Kahou, C. Pal *et al.*, "Extremeweather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events," *arXiv preprint*, 2016.

[6] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica *et al.*, "Exascale deep learning for climate analytics," in *International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018.

[7] L. Cheng, H. Zang, T. Ding, R. Sun, M. Wang, Z. Wei, and G. Sun, "Ensemble recurrent neural network based probabilistic wind speed forecasting approach," *Energies*, vol. 11, no. 8, 2018.

[8] W. Cheng, Y. Shen, Y. Zhu, and L. Huang, "A neural attention model for urban air quality inference: Learning the weights of monitoring stations," in *AAAI*, vol. 32, no. 1, 2018.

[9] M. Hossain, B. Rekabdar, S. J. Louis, and S. Dascalu, "Forecasting the weather of nevada: A deep learning approach," in *IJCNN*. IEEE, 2015.

[10] X. Ren, X. Li, K. Ren, J. Song, Z. Xu, K. Deng, and X. Wang, "Deep learning-based weather prediction: A survey," *Big Data Research*, vol. 23, 2021.

[11] S. F. Tekin, O. Karaahmetoglu, F. Ilhan, I. Balaban, and S. S. Kozat, "Spatio-temporal weather forecasting and attention mechanism on convolutional lstms," *arXiv preprint*, 2021.

[12] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *NeurIPS*, 2018.

[13] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," *arXiv preprint*, 2019.

[14] M. Finzi, K. A. Wang, and A. G. Wilson, "Simplifying hamiltonian and lagrangian neural networks via explicit constraints," *arXiv preprint*, 2020.

[15] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," *arXiv preprint*, 2020.

[16] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," *arXiv preprint*, 2019.

[17] S. Seo and Y. Liu, "Differentiable physics-informed graph networks," *arXiv preprint*, 2019.

[18] W. A. Strauss, *Partial differential equations: An introduction*. John Wiley & Sons, 2007.

[19] T. Stocker, *Introduction to climate modelling*. Springer Science & Business Media, 2011.

[20] B. Larwa, "Heat transfer model to predict temperature distribution in the ground," *Energies*, vol. 12, no. 1, 2019.

[21] J. Dormand and P. Prince, "A family of embedded runge-kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19 – 26, 1980.

[22] J. Zhuang, N. Dvornek, X. Li, S. Tatikonda, X. Papademetris, and J. Duncan, "Adaptive checkpoint adjoint method for gradient estimation in neural ode," in *ICML*, 2020.

[23] D. Freedman, *Brownian motion and diffusion*. Springer Science & Business Media, 2012.

[24] Y. Shikano, T. Wada, and J. Horikawa, "Discrete-time quantum walk with feed-forward quantum coin," *Scientific reports*, 2014.

[25] R. dos Santos Mendes, E. K. Lenzi, L. C. Malacarne, S. Picoli, and M. Jauregui, "Random walks associated with nonlinear fokker–planck equations," *Entropy*, vol. 19, no. 4, 2017.

[26] A. Plastino, E. Curado, F. Nobre, and C. Tsallis, "From the nonlinear fokker-planck equation to the vlasov description and back: Confined interacting particles with drag," *Physical Review E*, 2018.

[27] G. Mendes, M. Ribeiro, R. Mendes, E. Lenzi, and F. Nobre, "Nonlinear kramers equation associated with nonextensive statistical mechanics," *Physical Review E*, 2015.

[28] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *ICML*. PMLR, 2019.

[29] Y. Wang, Y. Wang, J. Yang, and Z. Lin, "Dissecting the diffusion process in linear graph convolutional networks," *arXiv preprint*, 2021.

[30] L.-P. Xhonneux, M. Qu, and J. Tang, "Continuous graph neural networks," in *ICML*. PMLR, 2020.

[31] A. DeWan, N. Dubois, K. Theoharides, and J. Boshoven, "Understanding the impacts of climate change on fish and wildlife in north carolina," *Defenders of Wildlife, Washington, DC*, 2010.

[32] S. R. Hanna, G. A. Briggs, and R. P. Hosker Jr, "Handbook on atmospheric diffusion," National Oceanic and Atmospheric Administration, Tech. Rep., 1982.

[33] S. Rasp and S. Lerch, "Neural networks for postprocessing ensemble weather forecasts," *Monthly Weather Review*, 2018.

[34] S. Scher, "Toward data-driven weather and climate forecasting: Approximating a simple general circulation model with deep learning," *Geophysical Research Letters*, vol. 45, no. 22, 2018.

[35] S. Seo, C. Meng, and Y. Liu, "Physics-aware difference graph networks for sparsely-observed dynamics," in *ICLR*, 2019.

[36] Y. Lin, N. Mago, Y. Gao, Y. Li, Y.-Y. Chiang, C. Shahabi, and J. L. Ambite, "Exploiting spatiotemporal patterns for accurate air quality forecasting using deep learning," in *ACM SIGSPATIAL*, 2018.

[37] P. Zhang, Y. Jia, J. Gao, W. Song, and H. Leung, "Short-term rainfall forecasting using multi-layer perceptron," *IEEE Transactions on Big Data*, 2018.

[38] H. Liu, X. Mi, and Y. Li, "Smart deep learning based wind speed prediction model using wavelet packet decomposition, convolutional neural network and convolutional long short term memory network," *Energy Conversion and Management*, vol. 166, 2018.

[39] Q. Zhu, J. Chen, L. Zhu, X. Duan, and Y. Liu, "Wind speed prediction with spatio–temporal correlation: A deep learning approach," *Energies*, 2018.

[40] A. Chattopadhyay, P. Hassanzadeh, and S. Pasha, "Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data," *Scientific reports*, 2020.

[41] E. De Bézenac, A. Pajot, and P. Gallinari, "Deep learning for physical processes: Incorporating prior scientific knowledge," *Journal of Statistical Mechanics: Theory and Experiment*, 2019.

[42] E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau, "Gru-ode-bayes: Continuous modeling of sporadically-observed time series," in *NeurIPS*, 2019.

[43] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural controlled differential equations for irregular time series," *arXiv preprint*, 2020.

[44] T. Lyons, M. Caruana, and T. Lévy, *Differential Equations Driven by Rough Paths*. Springer, 2004, École D'Eté de Probabilités de Saint-Flour XXXIV - 2004.

[45] A. Gozolchiani, S. Havlin, and K. Yamasaki, "Emergence of el niño as an autonomous component in the climate network," *Physical review letters*, vol. 107, no. 14, 2011.

[46] X.-S. Yang, "Small-world networks in geophysics," *Geophysical research letters*, vol. 28, no. 13, 2001.

[47] M. D. Humphries, K. Gurney, and T. J. Prescott, "The brainstem reticular formation is a small-world, not scale-free, network," *Proceedings of the Royal Society B: Biological Sciences*, 2006.

[48] M. D. Humphries and K. Gurney, "Network 'small-world-ness': a quantitative method for determining canonical network equivalence," *PloS one*, 2008.

[49] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, 1998.

[50] J. R. Dormand, *Numerical methods for differential equations: a computational approach*. CRC press, 1996, vol. 3.

[51] C. Zang and F. Wang, "Neural dynamics on complex networks," in *KDD*, 2020.

[52] 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," *Journal of Computational Physics*, vol. 378, 2019.

[53] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *ICML*. PMLR, 2018.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[55] R. T. Q. Chen and D. K. Duvenaud, "Neural networks with cheap differential operators," in *NeurIPS*, 2019.

[56] L. Kong, J. Sun, and C. Zhang, "SDE-net: Equipping deep neural networks with uncertainty estimates," in *ICML*, 2020.