# Mixed-Order Relation-Aware Recurrent Neural Networks for Spatio-Temporal Forecasting

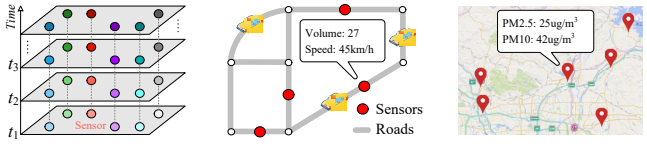
Yuxuan Liang, Kun Ouyang, Yiwei Wang, Zheyi Pan, Yifang Yin, Hongyang Chen, Junbo Zhang, Yu Zheng, David S. Rosenblum, Roger Zimmermann

**Abstract**—Spatio-temporal forecasting has a wide range of applications in smart city efforts, such as traffic forecasting and air quality prediction. Graph Convolutional Recurrent Neural Networks (GCRNN) are the state-of-the-art methods for this problem, which learn temporal dependencies by RNNs and exploit pairwise node proximity to model spatial dependencies. However, the spatial relations in real data are not simply pairwise but sometimes in a higher order among multiple nodes. Moreover, spatio-temporal sequences deriving from nature are often regulated by known or unknown physical laws. GCRNNs rarely take into account the underlying physics in real-world systems, which may result in degenerated performance. To address these issues, we devise a general model called Mixed-Order Relation-Aware RNN (MixRNN+) for spatio-temporal forecasting. Specifically, our MixRNN+ captures the complex mixed-order spatial relations of nodes through a newly proposed building block called Mixer, and simultaneously addressing the underlying physics by the integration of a new residual update strategy. Experimental results on three forecasting tasks in smart city applications (including traffic speed, taxi flow, and air quality prediction) demonstrate the superiority of our model against the state-of-the-art methods. We have also deployed a cloud-based system using our method as the bedrock model to show its practicality.

**Index Terms**—Spatio-Temporal Data Mining, Urban Computing, Reaction Kinetics, Physics-Informed Neural Networks.

## 1 Introduction

In recent years, large quantities of sensors have been deployed in different locations to collectively sense the environment, generating massive geospatially-correlated and time-varying data. Such sensors' readings have been termed *spatio-temporal (ST) sequences*, as shown in Figure 1(a). Notably, it is common that one sensor generates multivariate time series as it monitors different target conditions simultaneously. For instance, the loop detectors in Figure 1(b) report timely readings about the vehicles passing by as well as their travel speeds, providing insights for traffic management. Likewise, Figure 1(c) illustrates several air quality monitoring stations that constantly report the concentration of different pollutants, such as PM2.5 and PM10. In this paper, we study the problem of *spatio-temporal forecasting* which has facilitated a wide range of applications in smart cities, such as traffic forecasting [1], [2], [3], [4], air quality prediction [5], [6], [7], [8], and water quality prediction [9].



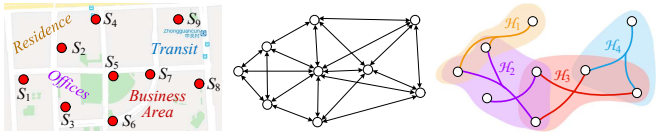

(a) ST Sequence (b) Loop detectors in roads (c) Air quality sensors

Fig. 1: Illustration/examples of spatio-temporal sequences.


- *Yuxuan Liang, Kun Ouyang, Yiwei Wang, Yifang Yin, David S. Rosenblum (Fellow, IEEE) and Roger Zimmermann (Senior Member, IEEE) are with School of Computing, National University of Singapore, Singapore. Email: {yuxliang, ouyangk, y-wang, david, rogerz}@comp.nus.edu.sg.*
- *Hongyang Chen (Senior Member, IEEE) is with the Research Center for Intelligent Network, Zhejiang Lab, China. Email: dr.h.chen@ieee.org.*
- *Zheyi Pan, Junbo Zhang (Member, IEEE) and Yu Zheng (Fellow, IEEE) are with JD iCity, JD Tech and JD Intelligent Cities Research, Beijing, China. Email: {zheyi.pan, msjunbozhang, msyuzheng}@outlook.com*


One key characteristic that must be considered in spatio-temporal forecasting is *spatio-temporal dependencies*. Primarily, a sensor's future is highly conditioned on its previous states, demonstrating strong temporal dependencies. Recurrent Neural Networks (RNN) have become one of the most popular architectures for learning such sequential information due to their flexibility in capturing nonlinear relationships [10]. Moreover, the future readings of a sensor are often impacted by its neighbors' histories. To model such structural relationships among sensor data, *graph* theory has long been adopted as a powerful tool, where the nodes denote sensors, and the edge weights represent the pairwise proximity measured by geographical distances. We thus use the term "node" and "sensor" interchangeably in the following parts. Based on this formulation, a new family of deep learning models called Graph Convolutional Recurrent Neural Networks (GCRNN) has achieved state-of-the-art performance in many spatio-temporal forecasting tasks [1], [4], [11], [12]. As its name suggests, they integrate Graph Convolutional Networks (GCN) [13] with RNNs, in which GCNs are employed as a basic building block to capture the spatial dependencies among sensors, and RNNs are applied to modeling the temporal dependencies.

Even though the above GCRNN methods have achieved promising results in ST forecasting, they only leverage the pairwise connections among data. In practice, the inter-sensor relationships are sometimes beyond pairwise (high-order relations among nodes, e.g., ternary, quaternary) and even much more complicated. Figure 2(a) shows an example of nine base stations at different locations with various land-use functions (e.g., $S_5$, $S_6$, $S_7$ and $S_8$ are in a business area), each of which reports the number of mobile users within a certain range. If a big sales promotion occurs in this business area, these four stations will simultaneously witness a rising




(a) Sensor distribution (b) Graph construction (c) Hypergraph structure

Fig. 2: (a)-(b) Layout and graph generation of nine sensors in a given area. (c) We can generate hyperedges ($\mathcal{H}_1$ to $\mathcal{H}_4$) to encode high-order relations among nodes based on their land-use functions, e.g., office areas, business districts, etc.

trend in occupancy. In this case, merely modeling pairwise connections may be insufficient to represent such complex and high-order structures among sensors. To better match reality, a powerful method that jointly captures the pairwise and high-order relations among sensors is acutely needed.

Meanwhile, GCRNNs have overlooked the prior knowledge of *underlying physics*. In reality, ST sequences deriving from nature are often governed by known or unknown physical laws [14]. For example, reaction kinetics describes the evolution of variables in a dynamic system continuously. To be more specific, reaction kinetics employs the first derivative $dx/dt$ to analyze the instantaneous rate of change of a variable $x$ with respect to time $t$, which has been widely explored in various real-world applications, such as describing chlorine decay in water supply systems [15], [16], evaluating air pollutant emissions [17] and forecasting traffic flows [18], [19]. Though GCRNN applied in the existing works can approximate the system dynamics (i.e., temporal dependencies) to some extent via the state transformations in RNNs, its discrete nature makes it an awkward fit to model the *continuous* changes driven by physical laws. Therefore, how to incorporate such physical rules (e.g., reaction kinetics) as prior knowledge to the neural network models for better modeling of ST sequences remains an open problem.

In this paper, we successively present two models (an *intermediate* model and its *physics-informed* version) to tackle the above two challenges, respectively. Firstly, we introduce **MixRNN**, a method allowing efficient mixed-order spatial relation modeling, by substituting the graph convolutions in GCRNNs with a new building block called **Mixer**. The mixed-order spatial relations are learned through two branches: one to capture the pairwise relations via GCNs, and the other to learn the high-order relations via hypergraph convolutional networks (HGCN) [20], [21]. Compared to conventional graphs in which each edge only connects two nodes, hypergraphs aim to handle more complicated high-order relations using degree-free hyperedges, as shown in Figure 2(c). Since the ground truth of high-order relations is absent in practice, our Mixer block avoids the time-consuming preprocessing in manual hypergraph construction [21], [22], and instead generates the hypergraph structure adaptively based on time-varying inputs.

Secondly, we recruit a new residual update strategy to endow MixRNN with physical knowledge (termed **MixRNN+**), by assuming that a dynamic system is governed by complex reaction kinetics. Inspired by reaction kinetics that utilizes a fixed and pre-defined ordinary differential equation (ODE) to describe the dynamics, a neural network is employed to approximate the derivatives of hidden states in MixRNN. As a result, MixRNN+ possesses continuous-time states which obey ODEs between successive time slots and can be updated upon new observations. In essence, this strategy can also be linked to recent advances that relate residual networks and ordinary/partial differential equations (ODEs/PDEs) [23], [24], [25], [26], [27]. Our residual update strategy not only helps the data-driven model (MixRNN) better generalize to different applications, but also enhances the interpretability by modeling the instantaneous rate of change of the hidden states.

In summary, our main contributions are four-fold:

- *MixRNN+*: We present a unified model that jointly considers the mixed-order spatial relations and the continuous-time hidden dynamics for spatio-temporal forecasting. A cloud-based system for urban flow management has been built to demonstrate its practicality.
- *Mixed-order spatial relation learning*: We introduce a new module for jointly capturing the pairwise and high-order spatial relations among sensors via different branches, which can be easily integrated into existing architectures including both RNNs and CNNs.
- *Physics-informed learning*: We couple physical laws (i.e., reaction kinetics) with the strong modeling capacity of a data-driven deep learning model to embrace both interpretability and accuracy.
- *Generalizability*: We evaluate our method on three typical tasks in smart cities using real-world datasets. Extensive experiments verify that our model displays very competitive performance compared to the state of the art.

## 2 PRELIMINARY

### 2.1 Notations

**Definition 1 (Spatio-temporal sequence)**: Let $\mathbf{X}_t \in \mathbb{R}^{N\times D}$ denote the signal observed from $N$ sensors at a certain time $t$, where $D$ is the number of measurements. Each entry $x_{ij}$ indicates the value of the $j$-th measurement of sensor $i$.

**Definition 2 (Graph)**: Generally, geo-sensors are interconnected with each other through an explicit network (e.g., loop detectors in road networks) or underlying structure measured by Euclidean distance (e.g., air quality stations in urban areas). We represent such prior knowledge as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, where $\mathcal{V}$ is a set of sensors and $\mathcal{E}$ is a set of edges. $\mathbf{A} \in \mathbb{R}^{N\times N}$ is a weighted adjacency matrix, describing the proximity between different nodes.

**Definition 3 (Hypergraph)**: Similar to the adjacency matrix in conventional graphs, we can use an incidence matrix $\mathbf{B} \in \mathbb{R}^{N\times M}$ to represent the underlying hypergraph structure among data, where each element $b_{ij}$ is the likelihood that node $i$ belongs to hyperedge $\mathcal{H}_j$ and $M$ is the number of hyperedges. As it is very difficult to pre-define the hypergraph structure among sensors, we generate it during the model training, which will be detailed in Sec. 4.2.

**Problem Statement**. Given a graph $\mathcal{G}$ and historical observations of all sensors from the past $T$ time steps, our target is to learn a function $f(\cdot)$ that predicts $D'$ measurements over the next $\tau$ steps:

$$[\mathbf{X}_{1:T}, \mathcal{G}] \xrightarrow{f(\cdot)} \mathbf{Y}_{1:\tau}.$$

where $\mathbf{X}_{1:T} \in \mathbb{R}^{N\times D\times T}$ and $\mathbf{Y}_{1:\tau} \in \mathbb{R}^{N\times D'\times \tau}$.

### 2.2 Graph Convolutional Recurrent Neural Networks

Recently, GCRNN has been one of the state-of-the-art methods for ST forecasting thanks to its power in capturing ST dependencies. GCRNN factorizes the modeling of spatial and temporal domains by using GCNs for spatial learning and RNNs for temporal learning, which is formulated as:

$$\begin{aligned}\mathbf{r}_t &= \sigma\left(\mathrm{GCN}_r\left(\mathbf{X}_t, \mathbf{H}_{t-1}\right) + \mathbf{b}_r\right),\\ \mathbf{u}_t &= \sigma\left(\mathrm{GCN}_u\left(\mathbf{X}_t, \mathbf{H}_{t-1}\right) + \mathbf{b}_u\right),\\ \mathbf{C}_t &= \tanh\left(\mathrm{GCN}_c\left(\mathbf{X}_t, (\mathbf{r}_t \odot \mathbf{H}_{t-1})\right) + \mathbf{b}_c\right),\\ \mathbf{H}_t &= \mathbf{u}_t \odot \mathbf{C}_t + (\mathbf{1} - \mathbf{u}_t) \odot \mathbf{H}_{t-1},\end{aligned}$$

where $\mathbf{X}(t)$ and $\mathbf{H}(t)$ indicate the input and output at time $t$; $\mathrm{GCN}$ is a function for learning spatial dependencies, such as a standard GCN [13] and diffusion convolution [1]; $\mathbf{r}_t$ and $\mathbf{u}_t$ are reset gates and update gates of GRUs [28]. $\mathbf{b}_r$, $\mathbf{b}_u$, $\mathbf{b}_c$ are the biases and $\mathbf{1}$ is an all-one matrix; $\sigma$ is the sigmoid function and $\odot$ denotes the Hadamard product. In this formulation, the hidden states $\mathbf{H}$ are recurrently updated based on the previous states along the time axis, considering the contextual information from spatial domains as well.

## 3 Related Works

### 3.1 Deep Learning for Graph & Hypergraph

A *graph* is a structure amounting to a set of objects in which some pairs of objects are in some sense "related". GCNs are popular building blocks for learning such graph-structured data [29], which can be categorized into spectral-based and spatial-based approaches. Spectral-based methods [13], [30] focus on the design of a fast approximation of spectral convolutions on graphs. Within the latter class [1], [31], [32], the convolution operation is defined in the groups of spatially close nodes. Though effective, graphs encounter challenges in describing more complex data, e.g., high-order relations among multiple nodes. A *hypergraph* is a generalization of a graph in which an edge can join any number of vertices, allowing us to model high-order correlations among different nodes. This new concept was first studied in [20] and revealed greater potential than a conventional graph in modeling real-world data. Then, [21], [33] generalized the convolution operation from graphs to hypergraphs, i.e., hypergraph convolutional networks (HGCN). These models cannot be directly applied for modeling ST sequences, since they mainly focus on feature aggregation in the spatial domain while overlooking the temporal dependencies.

### 3.2 Spatio-Temporal Forecasting

#### 3.2.1 Traditional physical models

Learning spatio-temporal dynamics of a system to forecast the future is crucial to fields as diverse as physics and environics. To this end, traditional physical models assume that a dynamic system is driven by certain physical laws, represented as ordinary or partial differential equations (ODEs/PDEs) that dominate the whole process irrespective of time or locations. Among these models, reaction kinetics is the most representative to analyze a dynamic system [15], [16], [17], [18], [19]. For example, [15] described both bulk and wall chlorine decay via simple first-order decay kinetics with linear dynamics. [19] modeled the instantaneous change of the traffic data using linear ODEs. [34] presented a numerical method to model kinematic wave (KW) traffic streams containing slow vehicles. Despite their success, most of these studies were built on strong domain knowledge, including extensive computation overheads when solving ODEs/PDEs by numerical methods.

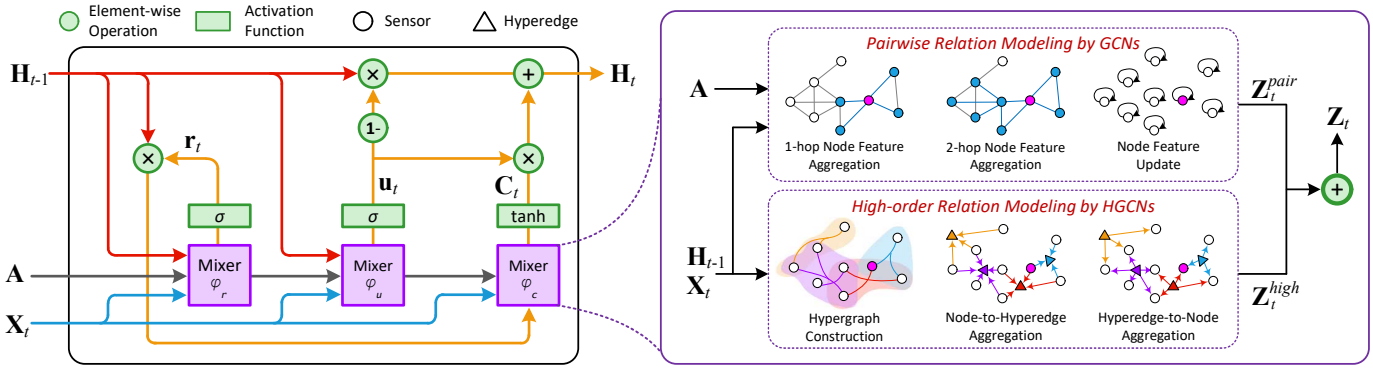
#### 3.2.2 Data-driven approaches

The second category is data-driven prediction models which learn from historical data to enable a system to give a desired output. Compared to classic physical models, data-driven methods have demonstrated their advantages in both effectiveness and flexibility in many applications [8], [9], [35]. Typically, autoregression models (e.g., ARIMA [36] and VAR [37]) have been widely applied for time series prediction. They however rely on the stationary assumption and show inferiority in learning non-linearity [7]. With recent advances in deep learning techniques, spatio-temporal graph convolutional networks have been the dominant class for ST sequence prediction, following two paradigms. They either integrate GCNs with RNNs [1], [4], [12], [38], [39] or CNNs [3], [40], [41], [42], [43], [44], [45], in which GCNs are used as basic building blocks to capture the spatial dependencies while RNNs or CNNs are used for modeling the temporal dependencies. Meanwhile, capturing the high-order relations among multiple nodes is crucial to ST forecasting as well. To this end, [22] proposed HGCRNN by integrating hypergraph convolutions with RNNs to jointly learn the high-order spatial relations and temporal dynamics in ST sequence data. However, HGCRNN fails to capture the lower-order (i.e., pairwise) relations without adding very restrictive constraints, and also ignores the underlying physics. To tackle these issues, we devise the intermediate model MixRNN, which will be introduced in Sec. 4.

### 3.3 Physics-Informed Deep Learning

Physics-Informed Neural Networks (PINN) are a new family of models that attempts to couple physical knowledge with the strong capacity of data-driven models. For instance, some pioneering studies introduced a physics-based regularization to guide the training of neural networks in lake temperature modeling [46], [47]. [48] presented a warping scheme with physical constraints for sea surface temperature prediction, but only suitable for the dynamic system that obeys general advection-diffusion principles. [49] further adopted a hybrid learning paradigm for turbulence modeling by marrying turbulent flow simulation with neural networks. However, using such certain physical constraints makes it hard to generalize to various applications.

In addition, there is another line of works linking differential equations with neural networks [24], [25], leading to the integration of ODEs, that is seen as continuous residual networks [50]. Following them, researchers started to apply this paradigm for the simulation of multivariate time series, such as irregular time series [27], hydropower generation [51] and reservoir inflows [52]. However, none of these methods can be directly applied to large-scale ST sequences as they are limited to low-dimensional time series and cannot well handle the spatial dependencies among a




(a) Details of MixRNN cell (b) Illustration of Mixer block

Fig. 3: Illustration of the MixRNN cell (left hand side) and Mixer block (right hand side), where the legend is at the top left.

large number of sensors. In this paper, by assuming that the dynamic system is driven by reaction kinetics, we further enhance the proposed MixRNN with physical knowledge by a novel residual update strategy. This new model termed MixRNN+ will be delineated in Sec. 5

## 4 Mixed-Order Relation-Aware RNNs

Figure 4 illustrates the framework of MixRNN, which follows the paradigm of the encoder-decoder architecture [53]. Firstly, we leverage a MixRNN as the encoder to transform the historical observations $\mathbf{X}_{1:T}$ into hidden states. In each MixRNN cell (i.e., at each time step), we replace the matrix multiplications in standard RNNs with the proposed Mixer block to learn the spatio-temporal dependencies. Compared with the GCRNN-based models that only consider the pairwise relations, our cell can jointly capture different orders of relations among data. Subsequently, we feed the last encoder state $\mathbf{H}_T$ as the initial state of the MixRNN decoder to generate the predictions $\hat{\mathbf{Y}}_{1:\tau}$ step by step. We will describe each model component in the following sections.

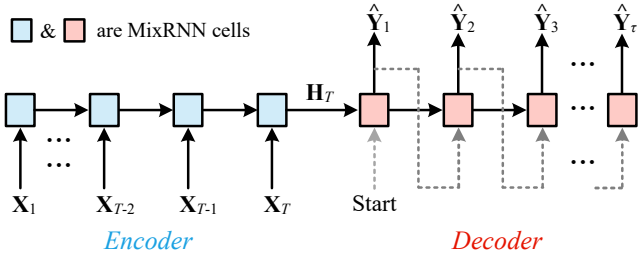



Fig. 4: Framework of MixRNN.

### 4.1 MixRNN Cell

As shown in Figure 3(a), the target of a MixRNN cell is to holistically capture both temporal and spatial dependencies by combining the previous hidden states and the current input. We follow the idea of a variant of RNN called Gate Recurrent Units (GRU) [54] using gating mechanisms to control the contribution of the previous state $\mathbf{H}_{t-1} \in \mathbb{R}^{N\times F}$ and the current observation $\mathbf{X}_t \in \mathbb{R}^{N\times D}$. Specifically, a MixRNN cell is formulated as:

$$\mathbf{r}_t = \sigma\left(\phi_r\left(\mathbf{X}_t, \mathbf{H}_{t-1}\right) + \mathbf{b}_r\right), \tag{1}$$

$$\mathbf{u}_t = \sigma\left(\phi_u\left(\mathbf{X}_t, \mathbf{H}_{t-1}\right) + \mathbf{b}_u\right), \tag{2}$$

$$\mathbf{C}_t = \tanh\left(\phi_c\left(\mathbf{X}_t, \left(\mathbf{r}_t \odot \mathbf{H}_{t-1}\right)\right) + \mathbf{b}_c\right), \tag{3}$$

$$\mathbf{H}_t = \mathbf{u}_t \odot \mathbf{C}_t + (\mathbf{1} - \mathbf{u}_t) \odot \mathbf{H}_{t-1}, \tag{4}$$

where $\mathbf{r}_t \in \mathbb{R}^{N\times F}$ and $\mathbf{u}_t \in \mathbb{R}^{N\times F}$ are reset gates and update gates to control which information should be passed to the output. $\mathbf{b}_r$, $\mathbf{b}_u$, $\mathbf{b}_c$ are the biases and $\mathbf{1}$ is an all-one matrix with equal size to $\mathbf{H}_t$; $\phi_r$, $\phi_u$ and $\phi_c$ are non-shared Mixer operators for modeling mixed-order relations among nodes in the spatial domain.

### 4.2 Mixer: Mixed-Order Relation Learning

The key element of a MixRNN cell is the Mixer block applied at each time step. As shown in Figure 3(b), it aims to learn the pairwise and high-order relations by GCNs and HGCNs separately in different branches, given the observed signals $\mathbf{X}_t$ at time $t$ and the last hidden state $\mathbf{H}_{t-1}$. Once we obtain the updated node features from these branches, the last step is to fuse them by element-wise addition. Compared with the previous studies [1], [22], our Mixer presents the first attempt to explicitly capture different orders of relations within one building block.

#### 4.2.1 High-order Relation Learning

In real-world systems, the relationships between objects can be high-order, beyond pairwise formulation [33]. A hypergraph is a natural solution to model such characteristics, in which each hyperedge can jointly connect multiple vertices. Recall that we employ an incidence matrix $\mathbf{B}$ to represent the underlying hypergraph structure among data, where each element $b_{ij}$ is the likelihood (not binary indicators) that node $i$ belongs to hyperedge $j$. Motivated by HGCRNN [22] for ST forecasting, we modify HGCNs for high-order relation modeling as the first branch of MixNet. For simplicity, we concatenate $\mathbf{X}_t$ and $\mathbf{H}_{t-1}$ to be $\mathbf{P}_t \in \mathbb{R}^{N\times(F+D)}$ as the input of both branches. Figure 3(b) shows the pipeline of high-order relation modeling, which contains three steps:

**(a) Hypergraph Construction**.

One major issue of HGCRNN is that its performance relies heavily on the quality of the hypergraph structure. However, in practice, we have no ground truth of the hypergraph describing the high-order relations among data. HGCRNN

requires extensive expert knowledge to engineer a hypergraph construction for each application or dataset, which largely reduces its practicality. Moreover, the spatial correlations in ST sequence data are highly dynamic, changing over time [7], [55]. Using a fixed hypergraph structure across all time steps as HGCRNN does will degrade the performance.

In this paper, we solve this problem by adaptively generating the hypergraph structure (i.e., the incidence matrix $\mathbf{B}_t$) at an arbitrary time $t$ using a 2-layer GCN $\psi$:

$$\mathbf{B}_t = \mathrm{softmax}\left(\psi(\mathbf{P}_t)\right), \tag{5}$$

where $\mathrm{softmax}$ is similar to applying normalization in each hyperedge, which guarantees that the sum of each column equals one. When creating hyperedges, such transformation not only considers the similarity between the time series readings at different locations, but also captures the spatial proximity between nodes via GCNs. In this way, we can easily achieve a dynamic hypergraph at different time. Compared with HGCRNN, our data-driven hypergraph is more flexible and can be easily adapted to a variety of real-world tasks. It is worth noting that we do not need to introduce an additional regularizer such as a clustering loss [56] to guide the learning of $\phi$, because the regularizer might restrict modeling capacity [13] and cause extra training issues (see more details in Sec. 6.3.4).

**(b) Node-to-Hyperedge Aggregation**.

After the hypergraph construction, the next step is node-to-hyperedge aggregation. Instead of using complex or time-consuming operations like attention mechanisms [7], [22], [33], we produce the features of each hyperedge by directly aggregating features of the corresponding nodes that belong to this hyperedge (see Figure 3), which can be implemented through the multiplication of the transpose of the incidence matrix as follows:

$$\mathbf{E}_t = \mathrm{Aggregate}_{n2h}(\mathbf{B}, \mathbf{P}_t) = \mathbf{B}_t^\top \mathbf{P}_t \mathbf{W}_\mathbf{e}, \tag{6}$$

where $\mathbf{E}_t \in \mathbb{R}^{M\times F}$ denotes the the features of $M$ hyperedges; Meanwhile, we employ learnable parameters $\mathbf{W}_e \in \mathbb{R}^{(F+D)\times F}$ for an additional feature transformation. In other words, we formulate the node-to-hyperedge mapping function as a linear combination (a.k.a weighted global pooling) of node features such that the new features can aggregate information from multiple locations.

**(c) Hyperedge-to-Node Aggregation**.

Once we obtain the features of $M$ hyperedges, the final step is to update the features of each node by aggregating their related hyperedge features, Figure 3(b) illustrates how we achieve this. Analogy to the former step, we implement the node-to-hyperedge by matrix multiplication as

$$\mathbf{Z}_t^{high} = \mathrm{Aggregate}_{h2n}(\mathbf{B}, \mathbf{E}_t) = \mathbf{B}_t \mathbf{E}_t, \tag{7}$$

where $\mathbf{Z}_t^{high} \in \mathbb{R}^{N\times F}$ are the new node representations. In summary, we perform a node-hyperedge-node transformation to learn the high-order relations in this branch.

#### 4.2.2 Pairwise Relation Learning

On the other hand, pairwise relations also play an important role in the prediction task. The modern tool for learning such structural relations is GCNs. For example, [1] presented the first attempt to introduce diffusion convolutions for traffic forecasting by relating traffic flow to a diffusion process. Following this study, we use diffusion convolutions to model the pairwise relationships among the directed graph. Let $\mathbf{D}_O$ and $\mathbf{D}_I$ denote the out- and in-degree matrix based on $\mathbf{A}$. We characterize the diffusion process by a random walk on $\mathcal{G}$ and a state transition matrix $\mathbf{D}_O^{-1}\mathbf{A}$. Noticing that it is also necessary to model the information from the upstream nodes, we formulate the pairwise relations by considering the bidirectional diffusion process as

$$\mathbf{Z}_t^{pair} = \sum_{k=1}^{K} \left( \theta_{k,1} \left(\mathbf{D}_O^{-1}\mathbf{A}\right)^k + \theta_{k,2} \left(\mathbf{D}_I^{-1}\mathbf{A}^\top\right)^k \right) \mathbf{P}_t \mathbf{W}_p,$$

where $k$ is the diffusion step; $\boldsymbol{\theta} \in \mathbb{R}^{K\times 2}$ assigns trainable weights to each diffusion step and $\mathbf{W}_p \in \mathbb{R}^{(F+D)\times F}$ is for feature conversion. By this, we simulate the pairwise diffusion process from both directions to update the node features in this branch. We set $K = 2$ in our experiments to consider the 2-hop neighbors of each node. Besides, a self-loop is added to $\mathbf{A}$ to allow our model learning self relations. Finally, we fuse the updated features from both branches to obtain the output of a Mixer block:

$$\mathbf{Z}_t = \mathrm{LayerNorm}\left(\mathbf{Z}_t^{high} + \mathbf{Z}_t^{pair}\right), \tag{8}$$

where $\mathrm{LayerNorm}$ is a layer normalization layer [57] performed for faster training.

#### 4.2.3 Generalization to Other Models

Mixer can be easily generalized to existing models for structural relation modeling. By turning off one of the branches, it will be degraded to a GCN or an HGCN, respectively. In the high-order branch, our method does not need heavy human effort to determine the hypergraph structure compared to HGCRNN. Although MixRNN has one more branch than a GCRNN or an HGCRNN, we find that they require very close computation time in practice (see Sec. 6.4). Apart from RNNs, our Mixer can also be easily integrated with existing CNN network architectures [2], [3] and achieves promising improvements (see Sec. 6.3.3). Thus, it has great potential in a wide range of applications, especially in multi-modal data with complex relationships between objects.

### 4.3 Encoder-Decoder & Optimization

For multi-step ahead forecasting, we employ the *sequence-to-sequence* architecture [53] where both encoder and decoder are the proposed MixRNNs. *Scheduled sampling* [58] is employed to alleviate the discrepancy between the input distributions of training phase and testing phase. As MixRNN provides an end-to-end mapping from the historical data to the predictions and is differentiable everywhere, we can train the whole network with the back-propagation rule. During the training phase, the Adam optimizer [59] is used to minimize the Mean Absolute Error (MAE) between the prediction results $\hat{\mathbf{Y}}_{1:\tau}$ and the ground truth $\mathbf{Y}_{1:\tau}$:

$$L\left(\boldsymbol{\Theta}\right) = \frac{1}{\tau N D'} \sum_{t=1}^{\tau} \sum_{i=1}^{N} \sum_{j=1}^{D'} \left|\hat{y}_{tij} - y_{tij}\right|, \tag{9}$$

where $\boldsymbol{\Theta}$ are all trainable parameters in MixRNN.

## 5 PHYSICS-INFORMED MIXRNNS

Recall that MixRNN allows us to jointly capture the mixed-order spatial relations and the temporal dependencies in ST sequences. To enable physics-informed learning, we further couple the intermediate model MixRNN with continuous-time dynamics that are driven by *reaction kinetics*. Figure 5 depicts the framework of the advanced model, where two MixRNN+ are used as the encoder and decoder respectively. As can be seen in Figure 5, MixRNN+ possesses continuous states which obey an ordinary differential equation (ODE) between successive steps, and can be updated upon observations. This new feature is the major difference against MixRNN as shown in Figure 4. To achieve this, we present a simple yet effective residual update strategy, in which a neural network is used to infer the instantaneous rate of change of the hidden states at any time.

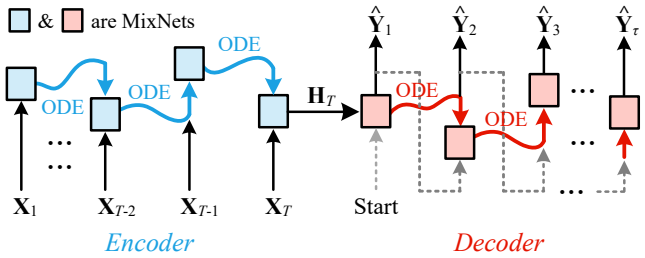



Fig. 5: Framework of MixRNN+.

### 5.1 Recap of Reaction Kinetics

As mentioned in Sec. 3.2.1, traditional physical models for spatio-temporal forecasting usually rely on the assumption that the dynamic system is driven by certain physical laws. To avoid the heavy computation overheads in solving high-order ODEs/PDEs, reaction kinetics employs the first-order ODE to analyze the dynamic system [16], [19]. In contrast to deep learning models such as GCRNNs, traditional physical models are always easier to interpret since they explicitly model the instantaneous rate of change of the variable everywhere. A general form of reaction kinetics is to represent the differential equation of the reactant $x$ as

$$\frac{dx}{dt} = f(x, t), \tag{10}$$

where $f$ is a *pre-defined* function based on expert knowledge according to the specific application, which takes the time and the current value of the variable as inputs to compute the derivative. For example, first-order kinetics describe the rates of reactions as $\frac{dx}{dt} = -kx$, where $k$ is the exponential decay constant. A combination of higher power of $x$, e.g., $\sum_0^n k_n x^n$, can indicate more complex reaction equations.

### 5.2 Residual Update Strategy

Although the above physical models are easy to interpret by virtue of modeling the derivative, most of them were built based on domain knowledge and not flexible to generalize to different real-world systems. For instance, we cannot directly transfer the exponential decay rule from modeling water pressure to urban traffic. Hence, we begin with thinking from another perspective: *can we couple reaction kinetics with the strong modeling capacity of a data-driven model to embrace both interpretability and accuracy?*

To answer this question, let us first briefly revisit the state transformation in standard RNNs. Given the previous state $\mathbf{H}_{t-1} \in \mathbb{R}^{N\times F}$ and its current input $\mathbf{X}_t \in \mathbb{R}^{N\times D}$, an RNN (in particular, GRU) generates its current hidden state $\mathbf{H}_t$ by

$$\mathbf{H}_t = f_{rnn}(\mathbf{H}_{t-1}, \mathbf{X}_t) \tag{11}$$

where $f_{rnn}$ is a GRU function for state transformation. From Eq. 11, we can observe that the hidden state is updated discretely and fully depends on the time-invariant weights of RNNs. However, ST sequences usually come from nature and evolve continuously over time, which means that discrete RNNs may not be optimal for modeling them.

To solve this issue, we draw inspiration from the above reaction kinetics and devise a new model called MixRNN+, which casts the physical mechanism into MixRNNs for a better modeling of ST sequences. We first assume that the dynamic system is driven by reaction kinetics, and then approximate the instantaneous rate of change of the hidden state everywhere via a residual update strategy. Specifically, we elaborate to change the rule of state transformation to be continuous in MixRNNs:

$$\hat{\mathbf{H}}_t = \mathbf{H}_{t-1} + \int_{t-1}^{t} \frac{d\mathbf{H}_s}{ds} ds, \text{ where } \frac{d\mathbf{H}_s}{ds} = f_\theta(\mathbf{H}_s, s), \tag{12}$$

$$\mathbf{H}_t = f_{rnn}(\hat{\mathbf{H}}_t, \mathbf{X}_t), \tag{13}$$

where $\theta$ is the parameters of a neural network $f$. From a mathematical viewpoint, the intermediate state $\hat{\mathbf{H}}_t$ can be interpreted as the left limit of the current state $\mathbf{H}_t$. In contrast to reaction kinetics in Eq. 10 that compute the derivative based on a pre-defined function $f$, we parameterize the derivative of hidden states by a trainable neural network $f_\theta$. Note that computing $\hat{\mathbf{H}}_t$ is equivalent to solving an ODE, we thereby employ a numerical black-box ODE solver (here it is the Euler method, a first-order numerical procedure for solving ODEs with a given initial value) to iteratively obtain the target state in Eq. 12 as:

$$\mathbf{H}_{\tau+\Delta t} = \mathbf{H}_\tau + \Delta t \cdot f_\theta\left(\mathbf{H}_\tau, \tau\right), \tag{14}$$

where $\tau \in [t-1, t)$ and $\Delta t$ is a hyperparameter to control the temporal resolution. This operation allows our model to generate hidden states at an arbitrary frame rate, leading to continuous-time dynamics. Since Eq. 14 updates its hidden state $\mathbf{H}_t$ based on the residual $\Delta t \cdot f_\theta\left(\mathbf{H}_t, t\right)$, we call this strategy *residual update* (ResUpdate), and for simplicity rewrite it as

$$\hat{\mathbf{H}}_t^i = \text{ResUpdate}(f_\theta, \mathbf{H}_{t-1}, (t-1, t)). \tag{15}$$

**Comparison with ResNet.** Our residual update strategy can also be interpreted as a continuous version of ResNet for deep residual learning [50]. Formally, a residual layer updates the hidden state at the $t$-th layer by using a transformation $f_t$ over the previous state, computed as $\mathbf{H}_t = \mathbf{H}_{t-1} + f_t(\mathbf{H}_{t-1})$. In contrast to ResNet, our ResUpdate (Eq. 14) has the following differences:

- The primary distinction is that, *the rationale of ResUpdate is coupling reaction kinetics into RNN to derive continuous hidden dynamics between observations*, rather than forming a very deep neural network by mitigating the gradient issue as ResNet does.

- The second difference is that *ResUpdate has shared parameters across all layers*. Without weights and biases depending on time, the transformation in ResUpdate is defined for all $t$, giving us a continuous expression for the derivative of the function we are approximating.
- The third difference is that, because of shared weights, *there are much fewer parameters in a ResUpdate than in an ordinary ResNet*. For example, assuming the number of evaluation steps between observations is $N_e$, there would be $N_e$ times the number of parameters in a ResNet compared to a ResUpdate.

In addition, we conduct some experiments to study the effects of ResUpdate vs. ResNet on three datasets. See Sec. 6.3.5 for experimental results and related discussion.

### 5.3 Procedure of MixRNN+

We present a physics-informed approach called MixRNN+ by integrating MixRNN with the residual update strategy, which not only enables the data-driven model MixRNN to have continuous-time dynamics, but also enhances the interpretability by modeling the instantaneous rate of change. Algorithm 1 illustrates the procedure of MixRNN+. Recall that $\mathbf{H}_t \in \mathbb{R}^{N \times F}$ denotes the hidden states of all nodes at a given time $t$. First, the initial hidden states are set all zeros in line 1. For each observation time $t$, we compute the intermediate state $\hat{\mathbf{H}}_t$ by our residual update rule based on the integration of $\frac{d\mathbf{H}_t}{dt}$ (see line 3). After that, we compute the current hidden states using a MixRNN cell in line 4. In other words, we update the hidden states upon new observations $\mathbf{X}_t$ by jointly considering the mixed-order spatial dependencies and the continuous historical states. During the training phase, we employ the same setting (e.g., schedule sampling, optimizer, and loss) as MixRNN, detailed in Sec. 4.3.

As $f_\theta$ determines the evolution of the hidden states, we need to pay more attention to the design of $f_\theta$. The simplest way is a multi-layer perceptron (MLP) that updates a node's current state fully based on its previous state, rather than considering the impacts of all nodes at each evaluate step. We can also use a 2-layer GCN as $f_\theta$ to allow message passing between a node and its neighbors at each residual update, or employ a Mixer block to capture mixed-order spatial relations at once. We test different settings of $f_\theta$ and they achieve very similar performances on three real-world datasets. Therefore, we represent $f_\theta$ as an MLP since we have already modeled the spatial dependencies by the Mixer block in the MixRNN cell. The other reason is to reduce the computational costs and memory usage caused by the message passing scheme like GCNs.

**Algorithm 1:** The MixRNN+ algorithm

**Input:** Historical observations $\{\mathbf{X}_t\}_{t=1\dots T}$, Adjacency matrix $\mathbf{A}$
**Output:** The hidden states $\{\mathbf{H}_t\}_{t=1\dots T}$
1: $\mathbf{H}_0 = \mathbf{0}$ ▷ Initial hidden state
2: **for** $t = 1 \dots T$ **do**
3: $\quad \hat{\mathbf{H}}_t = \mathrm{ResUpdate}\left(f_\theta, \mathbf{H}_{t-1}, (t-1, t)\right)$
4: $\quad \mathbf{H}_t = \mathrm{MixRNN}(\mathbf{X}_t, \hat{\mathbf{H}}_t, \mathbf{A})$
5: **end for**

### 5.4 Encoder-Decoder & Optimization

MixRNN+ shares the same manner to perform predictions as MixRNN, i.e., using an encoder-decoder architecture. The training strategy, including optimizer and schedule sampling, is also identical. Given the prediction results $\hat{\mathbf{Y}}_{1:\tau}$ and the ground truth $\mathbf{Y}_{1:\tau}$, we utilize MAE as the loss function to train MixRNN+, computed as:

$$L\left(\mathbf{\Theta}\right) = \frac{1}{\tau N D'} \sum_{t=1}^{\tau} \sum_{i=1}^{N} \sum_{j=1}^{D'} \left|\hat{y}_{tij} - y_{tij}\right|, \tag{16}$$

where $\mathbf{\Theta}$ are all trainable parameters in our model.

### 5.5 Discussion

#### 5.5.1 Comparison to Existing Methods

We appreciate the contributions of GCRNN and HGCRNN in modeling spatio-temporal (ST) sequences, and build our model based on them. Next, we emphasize how our MixRNN+ differs. Additionally, we compare our method with CNN-based approaches as well.

**Comparison to GCRNNs.** In Sec. 1, we have introduced two limitations of GCRNNs in spatio-temporal forecasting: 1) GCRNNs have difficulty with addressing high-order spatial relations; 2) GCRNNs have overlooked underlying physical knowledge. To overcome these issues, we empower the RNN-based methods by integrating the two proposed components, including the *Mixer block* for capturing mixed-order relations, and the *residual update strategy* for learning continuous hidden dynamics. In other words, our approach is built on GCRNNs and our major contributions include the two novel components.

**Comparison to HGCRNNs.** We draw inspiration from HGCRNNs to capture high-order relations using hypergraph convolutions, however, the branch of high-order relation modeling in the Mixer block is not identical to HGCRNN. The major difference is that our Mixer block constructs hypergraph structures adaptively depending on the input data, whereas HGCRNN requires extensive human engineering to obtain such structures (see Sec. 4.2.1). Besides, the aggregation method in HGCRNN is based on an averaging function, while ours uses matrix multiplication.

**Comparison to CNN-based Methods.** Besides RNN-based methods, MixRNN+ differs from CNN-based methods (e.g., STGCN [2], Graph WaveNet [3], and MTGNN [60]) mainly in the way it captures temporal dependencies. In terms of spatial dependencies, the mixed-order relations consisting of both pairwise and high-order relations are still less explored in the literature of CNN-based methods.

#### 5.5.2 Can Residual Update Improve CNN-based Models?

It is worth noting that our residual update strategy can be only integrated into RNN-based approaches, as it seeks to infer continuous hidden dynamics between two observations (see the comparison between Figure 4 and 5). In the CNN-based methods such as GWNet and MTGNN, there is no state transformation between two consecutive time steps (i.e., no internal state across different time steps), which indicates such a strategy cannot be integrated into these approaches for further improvements.

# 6 EXPERIMENTS

## 6.1 Experimental Settings

### 6.1.1 Tasks & Datasets

To show the generalizability of the proposed models, we evaluate them on three popular tasks in smart city efforts:

- *Traffic speed forecasting*: We first use a traffic benchmark called **METR-LA** [1] to evaluate our approach. It reports the 4-month readings of traffic speed from 207 loop detectors on the highways of Los Angeles County. The traffic speed readings are ranging from Mar 1st 2012 to Jun 30th 2012 and aggregated into 5-minute windows.
- *Air quality prediction*: We also adopt the Beijing Air Quality Dataset (**AirBJ**) [7] for evaluation. Specifically, the concentration of several air pollutants (e.g., $PM_{2.5}$, $SO_2$, and CO) together with some meteorological readings (e.g., temperature) are collected by 35 sensors every hour in Beijing. We make predictions on the concentration of $PM_{2.5}$, which is the primary air pollutant in most cases [5], [6].
- *Crowd flow forecasting*: The **TaxiNYC** dataset derives from taxi trajectories in New York City (NYC) [61]. The authors partitioned NYC into 100 irregular regions based on the technique of morphological image processing, and then compute the crowd flow by projecting the GPS trajectories into these regions. Here, we follow the authors to collectively forecast the inflow and outflow of all regions.

Table 1 provides the details of the datasets. Following [1], [3], we forecast the target series over the next 12 time steps based on the previous 12 steps in all tasks. Each dataset is partitioned in chronological order with 70% for training, 10% for validation and 20% for testing. We also combine the inputs with the time of day to make predictions. Z-score normalization is applied to the model inputs for fast convergence. To construct the graphs, we compute the network distances between sensors and build the adjacency matrix using the thresholded Gaussian kernel method [1].

TABLE 1: Dataset statistics.

| Dataset | METR-LA | AirBJ | TaxiNYC |
|---|---|---|---|
| Type | Traffic speed | Air quality | Crowd flow |
| #Instances | 34,272 | 28,752 | 48,121 |
| # Interval | 5 minutes | 1 hour | 1 hour |
| Start time | 03/01/2012 | 08/20/2014 | 01/01/2011 |
| End time | 06/30/2012 | 11/30/2017 | 06/30/2016 |
| #Nodes | 207 | 35 | 100 |
| #Edges | 1,515 | 302 | 484 |
| In/Output Dim | 2/1 | 19/1 | 3/2 |

TABLE 2: Model characteristics. Cont. denotes continuous.

| Model | Year | Spatial | | Temporal | |
|---|---|---|---|---|---|
| | | pairwise | high-order | discrete | cont. |
| ARIMA | 1970 | | | ✓ | |
| VAR | 2006 | ✓ | | ✓ | |
| STGCN | 2018 | ✓ | | ✓ | |
| GWNet | 2019 | ✓ | | ✓ | |
| MTGNN | 2020 | ✓ | | ✓ | |
| FC-LSTM | 1997 | | | ✓ | |
| DCRNN | 2017 | ✓ | | ✓ | |
| HGCRNN | 2020 | | ✓ | ✓ | |
| MixRNN | 2021 | ✓ | ✓ | ✓ | |
| MixRNN+ | 2021 | ✓ | ✓ | | ✓ |

### 6.1.2 Baselines

We compare our model with eight baselines that belong to the following three classes:

- **Traditional data-driven models**: ARIMA [36] and VAR [37] are two well-known shallow autoregression models.
- **CNN-based approaches**: STGCN [2], Graph WaveNet (GWNet for short) [3] and MTGNN [60] integrate GCNs with 1D temporal convolutions to capture the spatio-temporal dependencies. We also set them as baselines.
- **RNN-based methods**: Since our methods are based on RNNs, we compare them with the existing RNN-based models introduced in Sec. 3.2.2, including FC-LSTM [62], DCRNN [1] and HGCRNN [22].

In Table 2, we further present the characteristics of these models. For example, ARIMA and FC-LSTM only consider the recent time slots within each node while ignoring the spatial correlations. CNN-based models including DCRNN, GWNet and MTGNN can capture the pairwise relationships but overlook higher-order relationships among multiple nodes. Only MixRNN+ can jointly capture the mixed-order spatial relations and the continuous temporal dynamics.

### 6.1.3 Implementation Details & Hyperparameters

**Our Models.** We implement our model by PyTorch 1.1 with a Quadro RTX 6000 GPU, where the batch size is 64. The learning rate starts from 0.01 and reduces to $\frac{1}{10}$ at epoch 10, 40 and 70. For the hidden dimensionality $F$, we conduct a grid search over $\{16, 32, 64, 96, 128\}$. As introduced in Sec. 5.3, the function $f_\theta$ for approximating the derivative is a 3-layer MLP with $F$ hidden units in each fully-connected layer. Note that setting $\Delta t$ is equivalent to specifying the number of evaluations $N_e$ between observations. We set $N_e$ to 10 in our experiments for a trade-off between performance and speed. Since our model generates the hypergraph structure through model training, we also test different numbers of hyperedges ($M$) in our experiments.

**Baselines.** Next, we introduce the implementation details of the deep-learning-based methods. On METR-LA, DCRNN, STGCN, GWNet and MTGNN are the previous and recent state-of-the-art methods. We therefore employ the default settings by their authors on this dataset. For FC-LSTM, we stack two LSTM layers and set the hidden dimensionality to 64. For HGCRNN without public code, we implemented it by ourselves. To be more specific, we use stacked HGCRNN with two layers, where the number of hidden units is 64.

As the baselines were not previously examined on AirBJ and TaxiNYC, we search the best hyperparameters for them, respectively. For the RNN-based approaches (FC-LSTM, DCRNN and HGCRNN), we conduct a grid search over the number of RNN layers (ranging from 1 to 3) and its hidden dimensionality (over $\{16, 32, 64, 128\}$). For the CNN-based methods, we conduct a grid search for the number of channels in their blocks over $\{16, 32, 64, 128\}$. Besides, we need to tune whether to utilize adaptive adjacency matrix for GWNet and MTGNN. After tuning hyperparameters, we observe that *most of the methods have similar parameter sizes on all these datasets*, e.g., ranging from 200K to 400K on AirBJ, with the only exception of STGCN which is a simple yet effective model. Meanwhile, merely increasing the parameter size (i.e., capacity) cannot always bring gains

on the prediction tasks, because of the overfitting problem. For example, using a STGCN with 528K parameters cannot even outperform its small version with 179K parameters on METR-LA.

#### 6.1.4 Evaluation Metrics

We adopt two popular metrics in regression tasks [1], [3] to evaluate the model performance, including mean absolute error (MAE) and root mean squared error (RMSE). Smaller metric scores indicate better model performance. We do not use Mean Percentage Errors since there are a large number of entries that equal to zero in the second and third dataset. Missing values are excluded when calculating these metrics.

### 6.2 Model Comparison

For each dataset, we run each method 5 times and report the mean results of 3, 6 and 12 step-ahead forecasting in Table 3. To be fair, we present the best performance of each method under different hyperparameter settings. For example, we report MixRNN+ with $F = 64$ and $M = 60$ on METR-LA.

It can be seen that MixRNN+ consistently outperforms the baselines over both metrics in most cases, with the only exception of for one entry on AirBJ. These improvements are significant according to the Student's t-test at level 0.01. Such improvements are more obvious on long-term forecasting since predicting a longer horizon (e.g., 12-step ahead) is far more challenging than a shorter horizon (e.g., 3-step ahead). From the table, we also have the following observations. First, deep learning-based methods exhibit much fewer errors than ARIMA/VAR due to their capability of learning non-linear relationships. Second, FC-LSTM, ODERNN and LatentODE perform much worse than other DL methods, revealing the importance of capturing the spatial dependencies in such applications. Third, MixRNN bests DCRNN and HGCRNN over all future horizons, which shows the superiority of our Mixer block that jointly models the pairwise and high-order spatial relations. Meanwhile, we find that learning pairwise relations is more useful than high-order relations on METR-LA and TaxiNYC, but obtain the opposite results on AirBJ by comparing DCRNN and HGCRNN. This fact corroborates the motivation of using our Mixer block for learning both of them. Despite the fact that the CNN-based approaches like STGCN and GWNet achieve higher accuracy than the RNN-based baselines on the whole, MixRNN+ still outperform the CNN-based models by virtue of coupling MixRNNs with the physical laws. The results of MixRNN+ vs. MixRNN will be further discussed later.

Additionally, we find that the baselines and the proposed model perform similarly on short-term prediction, while being distinctive on long-term prediction. Prior investigations have found similar observations (small gains on short-term prediction) [2], [3], [60], [63]. The major reason, we argue, is that *the task of short-term prediction is much simpler than long-term forecasting*, which makes all models produce extremely close results on short-term prediction. For an easy task (predicting the value at the next time step), even a very simple model such as an LSTM can achieve adequate performance, i.e., the predictive accuracy between an easy model and ours is not distinguishable. Conversely, when predicting on a long-term horizon, the simple model finds it incredibly difficult to compete with our models due to its lower model capacity and worse generalization ability.

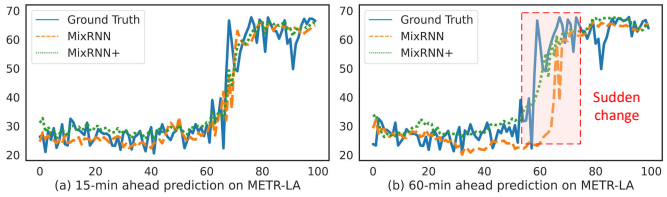



Fig. 6: The prediction curves of our models on the 158-th sensor on METR-LA. The y axis is the traffic speed (m/s).

TABLE 3: Model comparison of 12-step ahead prediction over the three datasets, where we use **bold** and **underline** fonts to highlight the best and the second best performance, respectively. We train and test each method five times, and present results using the format: "mean ± standard deviation".

| Methods | | | ARIMA | VAR | STGCN | GWNet | MTGNN | FC-LSTM | DCRNN | HGCRNN | MixRNN | MixRNN+ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| METR-LA | 3 | MAE | 3.19 | 3.05 | 2.73 ± 0.01 | 2.68 ± 0.01 | 2.69 ± 0.01 | 2.98 ± 0.02 | 2.73 ± 0.01 | 2.79 ± 0.01 | 2.65 ± 0.01 | **2.63 ± 0.01** |
| | | RMSE | 5.96 | 5.84 | 5.29 ± 0.01 | 5.15 ± 0.02 | 5.18 ± 0.02 | 5.92 ± 0.02 | 5.25 ± 0.02 | 5.41 ± 0.03 | 5.10 ± 0.02 | **5.06 ± 0.02** |
| | 6 | MAE | 3.85 | 3.71 | 3.13 ± 0.01 | 3.05 ± 0.01 | 3.05 ± 0.01 | 3.58 ± 0.03 | 3.15 ± 0.01 | 3.23 ± 0.02 | 3.04 ± 0.01 | **3.00 ± 0.01** |
| | | RMSE | 7.94 | 7.71 | 6.40 ± 0.03 | 6.16 ± 0.02 | 6.17 ± 0.02 | 7.30 ± 0.06 | 6.38 ± 0.03 | 6.54 ± 0.03 | 6.16 ± 0.02 | **6.08 ± 0.03** |
| | 12 | MAE | 5.24 | 5.01 | 3.56 ± 0.01 | 3.53 ± 0.01 | 3.49 ± 0.01 | 4.44 ± 0.03 | 3.69 ± 0.02 | 3.76 ± 0.01 | 3.48 ± 0.01 | **3.42 ± 0.01** |
| | | RMSE | 9.97 | 9.55 | 7.50 ± 0.03 | 7.36 ± 0.03 | 7.23 ± 0.04 | 9.01 ± 0.07 | 7.67 ± 0.04 | 7.81 ± 0.04 | 7.28 ± 0.03 | **7.16 ± 0.02** |
| AirBJ | 3 | MAE | 18.57 | 16.35 | 13.28 ± 0.27 | 12.93 ± 0.31 | 12.90 ± 0.35 | 14.57 ± 0.59 | 12.81 ± 0.31 | 12.70 ± 0.36 | 12.85 ± 0.27 | **12.46 ± 0.29** |
| | | RMSE | 30.28 | 30.10 | 24.36 ± 0.24 | 23.98 ± 0.36 | 24.01 ± 0.33 | 27.10 ± 0.41 | 23.91 ± 0.37 | 23.87 ± 0.32 | 24.01 ± 0.32 | **23.16 ± 0.25** |
| | 6 | MAE | 26.84 | 24.85 | 18.41 ± 0.25 | 18.20 ± 0.37 | 18.31 ± 0.32 | 21.50 ± 0.59 | 19.08 ± 0.72 | 18.60 ± 0.37 | 18.20 ± 0.34 | **17.94 ± 0.30** |
| | | RMSE | 39.82 | 37.09 | 32.97 ± 0.40 | 32.91 ± 0.46 | 33.01 ± 0.41 | 37.43 ± 0.70 | 34.60 ± 0.43 | 33.38 ± 0.51 | 33.08 ± 0.42 | **32.74 ± 0.45** |
| | 12 | MAE | 35.08 | 33.73 | 25.23 ± 0.41 | 25.01 ± 0.38 | 24.96 ± 0.46 | 30.58 ± 0.71 | 27.24 ± 0.52 | 25.29 ± 0.50 | 25.24 ± 0.47 | **24.86 ± 0.45** |
| | | RMSE | 49.94 | 48.80 | 42.48 ± 0.71 | 42.31 ± 0.64 | **42.05 ± 0.82** | 47.28 ± 0.94 | 45.51 ± 0.80 | 44.84 ± 0.77 | 42.47 ± 0.67 | 42.11 ± 0.75 |
| TaxiNYC | 3 | MAE | 32.60 | 26.79 | 19.42 ± 0.35 | 19.46 ± 0.41 | 19.85 ± 0.36 | 21.25 ± 0.65 | 20.59 ± 0.45 | 20.85 ± 0.48 | 19.73 ± 0.40 | **19.04 ± 0.36** |
| | | RMSE | 85.48 | 72.25 | 45.95 ± 1.14 | 48.22 ± 1.32 | 50.28 ± 1.26 | 53.65 ± 1.89 | 50.70 ± 1.40 | 52.19 ± 1.28 | 49.82 ± 1.33 | **43.69 ± 1.39** |
| | 6 | MAE | 36.72 | 29.04 | 21.75 ± 0.34 | 22.22 ± 0.47 | 23.84 ± 0.42 | 26.10 ± 0.68 | 24.03 ± 0.44 | 25.73 ± 0.57 | 21.57 ± 0.44 | **20.58 ± 0.40** |
| | | RMSE | 98.40 | 78.74 | 52.70 ± 1.04 | 54.58 ± 1.26 | 57.09 ± 1.19 | 65.84 ± 1.94 | 59.04 ± 1.58 | 64.89 ± 1.45 | 52.54 ± 1.39 | **50.47 ± 1.31** |
| | 12 | MAE | 39.31 | 32.19 | 23.30 ± 0.44 | 23.72 ± 0.40 | 24.09 ± 0.55 | 29.30 ± 0.81 | 26.80 ± 0.59 | 28.04 ± 0.54 | 24.30 ± 0.40 | **22.10 ± 0.49** |
| | | RMSE | 110.4 | 85.8 | 57.62 ± 1.10 | 59.84 ± 1.32 | 58.82 ± 1.28 | 71.55 ± 1.74 | 68.90 ± 1.47 | 70.52 ± 1.52 | 59.05 ± 1.32 | **55.79 ± 1.30** |

### 6.3 Ablation Study

Next, we evaluate the effectiveness of each model component. Unless otherwise specified, we report the *average errors* of all future horizons in the following figures and tables.

#### 6.3.1 Effect of Residual Update

Since MixRNN+ enhances MixRNN to possess continuous-time dynamics, we analyze the results of them to demonstrate the advantage of integrating our residual update strategy. As illustrated in Table 3, MixRNN+ reduces the prediction errors by a considerable gap in AirBJ and TaxiNYC, while it slightly outperforms MixRNN on METR-LA. The major reason is the length of time interval: the time interval on METR-LA is 5 minutes, which is much shorter than the 1-hour interval on the other two datasets. With the increase of the time interval, e.g., from 5 minutes to 1 hour, it becomes more challenging for RNN models to conduct state transformation between consecutive observations. In other words, obtaining the continuous temporal changes between observations becomes more necessary to help the modeling of ST sequences when the time interval is larger.

Besides, we notice that MixRNN+ yields better performance than MixRNN more clearly in long-term forecasting, e.g., at the 12-step ahead horizon. To further investigate it, we randomly select a sensor from METR-LA and conduct a case study on it. Figure 6(a) and (b) depict 15 and 60 minute-ahead (i.e., 3 and 12 step-ahead) predicted values vs. real values on a snapshot of the test data of this sensor, respectively. By comparing these two figures, we find that MixRNN+ outperforms MixRNN by a larger margin on the 60 minute-ahead prediction, especially at the sudden change (see the red rectangle in b). This is because the integration of physical laws enables our model to capture the long-term temporal dependencies more effectively.

As MixRNN+ is a variant of RNNs, we can learn from RNNs to increase the model capacity by stacking multiple MixRNN+ layers. According to the results in Table 4, the performance of MixRNN+ on METR-LA is not very sensitive to the number of stacked MixRNN+ layers. On the other datasets, even though stacking 2 or 3 layers can bring slight improvement, it requires far more GPU storage and induces much higher computational costs. Therefore, we choose the single-layer model as our default setting on these datasets.

TABLE 4: Effects of different number of layers. The metrics are computed by averaging the errors across all future steps.

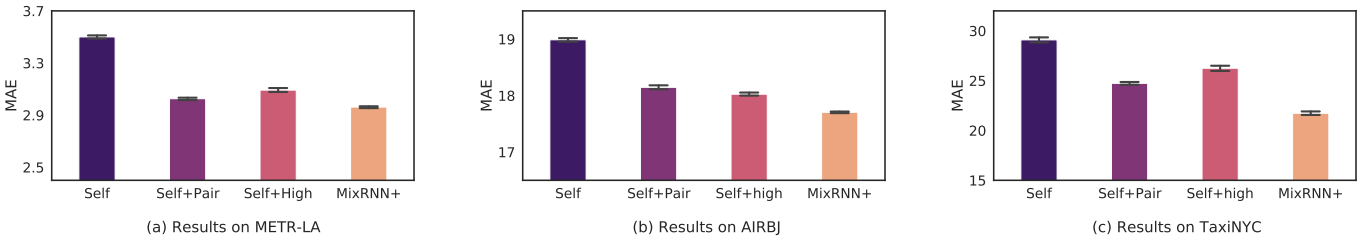
| MixRNN+ | METR-LA | | AirBJ | | TaxiNYC | |
|---|---|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE | MAE | RMSE |
| 1 layer | **2.96** | **5.95** | 17.71 | **31.62** | 20.35 | **50.14** |
| 2 layers | 2.97 | 6.02 | 17.72 | 31.68 | **20.21** | 50.31 |
| 3 layers | 3.02 | 6.04 | **17.69** | 31.88 | 20.47 | 50.56 |

#### 6.3.2 Generalizability of Residual Update

Furthermore, we integrate our residual update strategy into the RNN-based baselines for a more thorough comparison, denoted as FC-LSTM+, DCRNN+ and HGCRNN+. From Table 5, it can be seen that our residual update strategy brings consistent gains across all the baselines, demonstrating its broad generalizability. More importantly, our MixRNN+ still surpasses the variants of baselines (e.g., HGCRNN+) by a considerable margin on all datasets. These facts reveal the necessity of addressing mixed-order relations in spatio-temporal forecasting tasks.

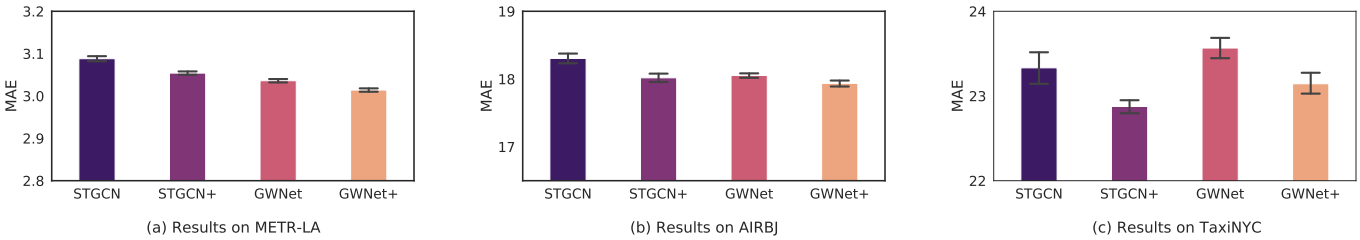
TABLE 5: Comparison with RNN-based methods that incorporates the residual update. The error metrics are computed by averaging the errors across all future time steps.

| Method | METR-LA | | AirBJ | | TaxiNYC | |
|---|---|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE | MAE | RMSE |
| FC-LSTM | 3.57 | 7.01 | 21.20 | 36.11 | 26.83 | 66.12 |
| DCRNN | 3.15 | 6.27 | 18.85 | 33.02 | 24.26 | 58.10 |
| HGCRNN | 3.22 | 6.45 | 18.27 | 32.45 | 25.42 | 63.39 |
| MixRNN | **3.00** | **6.06** | **18.03** | **31.98** | **21.42** | **53.77** |
| FC-LSTM+ | 3.54 | 6.95 | 20.76 | 34.92 | 24.82 | 61.50 |
| DCRNN+ | 3.09 | 6.14 | 18.62 | 32.81 | 23.47 | 54.96 |
| HGCRNN+ | 3.17 | 6.39 | 18.27 | 32.06 | 23.85 | 60.49 |
| MixRNN+ | **2.96** | **5.95** | **17.71** | **31.62** | **20.35** | **50.14** |




(a) Results on METR-LA (b) Results on AIRBJ (c) Results on TaxiNYC

Fig. 7: Effect of different branches in the proposed Mixer block.




(a) Results on METR-LA (b) Results on AIRBJ (c) Results on TaxiNYC

Fig. 8: Performance comparison for CNN-based methods.

#### 6.3.3 Effect of Mixer Block

Learning mixed-order spatial relations is one of the key challenges in ST prediction. We compare MixRNN+ with its variants to investigate the effectiveness of each branch in the Mixer block. Due to the page limit, we mainly discuss the results of MAE in this part. In particular, we use *Pair* and *High* to represent the two branches, respectively. By turning off both of the branches, our model will degrade to a standard GRU that ignores spatial relations, denoted as *Self*. For example, MixRNN+ without HGCN for high-order relation modeling can be denoted as *self+pair*. As depicted in Figure 7, the variant that only captures the self relations performs much worse than other variants in both tasks. We also observe that it can achieve a large improvement with the consideration of pairwise relations or high-order relations. By jointly learning the pairwise and high-order relations, our Mixer block can learn the complex relations in ST data more effectively and achieve better performance.

To further verify its generalizability in other popular architectures, we replace graph convolution layers in two CNN-based models (STGCN and GWNet) with our Mixer block for mixed-order relation modeling. Figure 8 presents the comparison results, where the models with a postfix "+" mean integrating our module. The consistent improvements over the base models reveal that Mixer block can also be a useful off-the-shelf plugin for the CNN-based methods.

#### 6.3.4 Effect of Hypergraph Generation

One possible concern of the Mixer block is hypergraph generation. Recall that the generation method introduced in Sec. 4.2.1 is a sort of soft clustering, but we do not introduce any objective function to guide the learning of the function $\psi$ (a 2-layer GCN). Would this be a problem? To answer this question, we add an auxiliary loss $L_a$ to Eq. 16. A natural idea is to employ the spectral clustering loss [56] as $L_a$ to enforce strongly-connected nodes to be grouped into the same hyperedge. Meanwhile, it encourages the assignment to be orthogonal to avoid sub-optimal solutions. We tune the trade-off between the MAE loss $L$ and $L_a$ and report the best results in Table 6. From this table, it can be easily seen that adding such auxiliary loss will not improve the predictive performance. That is because strongly-connected nodes sometimes perform disparately due to external factors. Adding this kind of regularizer might restrict the model capacity [13]. By using GCNs for generating the incidence matrix, our model already considers the inherent proximity between nodes [13] as well as the real-time observations. Besides, tuning the trade-off parameter between two loss functions is very hard and time-consuming. Thus, we prefer not to use the auxiliary loss function for additional guidance. Moreover, we compare MixRNN+ with its variant that uses an MLP as $\psi$ (denoted as w. MLP in Table 6) for hypergraph generation. The degraded performance (base vs. w. MLP) reveals the importance of encoding spatial proximity to the generation process.

TABLE 6: Evaluation on the hypergraph generation; Base: MixRNN+; w. $L_a$: MixRNN+ with an auxiliary loss for spectral clustering; w. MLP: MixRNN+ with an MLP as $\psi$.

| Loss | METR-LA | | AirBJ | | TaxiNYC | |
|---|---|---|---|---|---|---|
| | MAE | RMSE | MAE | RMSE | MAE | RMSE |
| Base | **2.96** | **5.95** | **17.71** | **31.62** | **20.35** | **50.14** |
| w. $L_a$ | 2.99 | 6.14 | 18.01 | 33.85 | 20.76 | 51.69 |
| w. MLP | 2.98 | 6.12 | 18.05 | 33.81 | 20.63 | 51.95 |

#### 6.3.5 Comparison with ResNet

To further compare ResUpdate with ResNet, we consider the following models and variants for comparison:

- **ResUpdate**: it is equivalent to the proposed MixRNN+.
- **ResNet-10**: As mentioned in Sec. 6.1.3, we set $N_e = 10$ for a trade-off between speed and accuracy. Thus, we compare MixRNN+ with a variant that replaces the residual update between observations with an equivalent number of residual layers (i.e., 10), denoted as ResNet-10. Note that we use the identical neural designs in one layer of ResNet and ResUpdate for fairness.
- **ResNet-5**: Noting that using 10 residual layers leads to much more learnable parameters, we also explore a baseline with 5 residual layers for comparison.
- **ResNet-1**: This variant replaces ResUpdate with only one residual layer, which guarantees its parameter number (model capacity) to be equal to MixRNN+.
- **Base**: The base model means MixRNN+ without the residual update strategy, namely MixRNN.

The prediction errors (MAE) of each method and their parameter numbers are shown in Table 7. The errors achieved by ResNet-10/5/1 and Base are not distinctive, revealing that simply stacking residual layers (for solving gradient problems) cannot enhance model performance. However, by using our ResUpdate, MixRNN+ considerably outperforms ResNet-10/5/1 even though these ResNet-based approaches have more parameters, e.g., ResNet-10 has around twice the number of parameters than ResUpdate on METR-LA. This result demonstrates that the rationale between ResUpdate and ResNet is totally different. ResUpdate computes the instantaneous rate of change between observations via a deep neural network, thus leading to a more natural and powerful modeling of real-world ST sequences.

TABLE 7: ResUpdate vs ResNet. MAEs are computed by averaging the errors across all future time steps. #Param: the number of trainable parameters (in thousands).

| Method | METR-LA | | AirBJ | | TaxiNYC | |
|---|---|---|---|---|---|---|
| | MAE | #Param | MAE | #Param | MAE | #Param |
| ResUpdate | **2.96** | 642 | **17.71** | 422 | **20.35** | 508 |
| ResNet-10 | 3.04 | 1266 | 18.05 | 694 | 21.52 | 912 |
| ResNet-5 | 3.04 | 876 | 18.08 | 524 | 21.45 | 660 |
| ResNet-1 | 3.02 | 642 | 18.04 | 422 | 21.49 | 508 |
| Base | 3.01 | 487 | 18.10 | 354 | 21.49 | 407 |

#### 6.3.6 Hyperparameter Study

Finally, we test the effect of the two major hyperparameters in our model on the first two datasets, i.e., the number of hyperedges $M$ in the Mixer block and the hidden dimensionality $F$. Figure 9(a) depicts how prediction errors changes over $M$ on the first two datasets. We observe that our model achieves the lowest errors when $M = 60$ and $40$ on them, respectively. When $M$ is very small (such as 20), it is hard to aggregate the nodes into such few hyperedges. On the other hand, using a large number of hyperedges

such as $M = 100$ also leads to higher errors since it makes the node features over-smooth [21]. In terms of the hidden dimensionality, $F = 64$ shows superiority in both datasets. As $F$ increases, it will induce more model parameters and may result in the overfitting problem. That is the reason why $F = 96$ and 128 perform worse than $F = 64$. On the contrary, using $F = 16$ or $F = 32$ causes the highest MAEs in contrast to other variants as it restricts the model capacity.

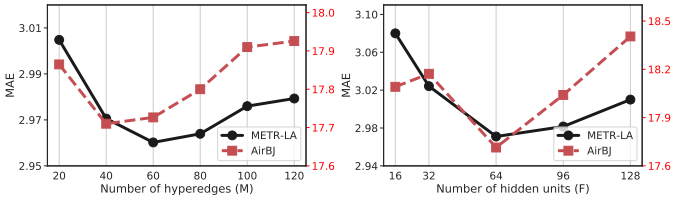



Fig. 9: Effect of the two hyperparameters in the proposed MixRNN+. (a) Study on the number of hyperedges $M$; (b) Study on the number of hidden units $F$.

### 6.4 Computation Time

Despite the good performance, we notice that adding the residual update to MixRNN induces extra computational overheads and lengthens training time. Hence, we conduct a study on the training speed of the proposed models as well as the competitive baselines from two perspectives: a) *training time per epoch* and b) *the epoch number of convergence*.

Figure 10 presents the training time per epoch of each approach. From this figure, we can observe that FC-LSTM requires the least training time since it overlooks the spatial information from neighbors. MixRNN achieves very close speed to DCRNN and HGCRNN, while largely improving the predictive performance in most cases (see Table 3). Although MixRNN+ extends MixRNN to be physics-guided by integrating the residual update strategy, it requires more training time per epoch to compute the continuous states. Thus, MixRNN+ is the priority when the application requires higher prediction accuracy. In a time-sensitive application, we prefer using MixRNN rather than MixRNN+.

In addition, we compare the epoch number of convergence of these RNN-based approaches, where we employ the same learning rate and scheduler for fairness (see Sec. 6.1.3). As shown in Table 8, MixRNN and MixRNN+ consistently achieve the fastest or the second fastest convergence speed among these methods. We argue that our models can obtain more useful gradients for training by considering both high-order and pairwise relations. Besides, we also observe that the convergence epoch number of the three baselines is not quite distinctive.

TABLE 8: Convergence epoch numbers of RNN methods.

| Models | METR-LA | AirBJ | TaxiNYC |
|---|---|---|---|
| FC-LSTM | 76 | 21 | 72 |
| DCRNN | 68 | 25 | 65 |
| HGCRNN | 74 | 24 | 69 |
| MixRNN | **52** | 20 | 46 |
| MixRNN+ | 55 | **16** | **43** |

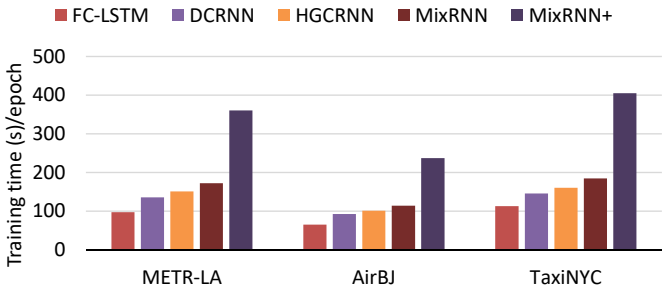
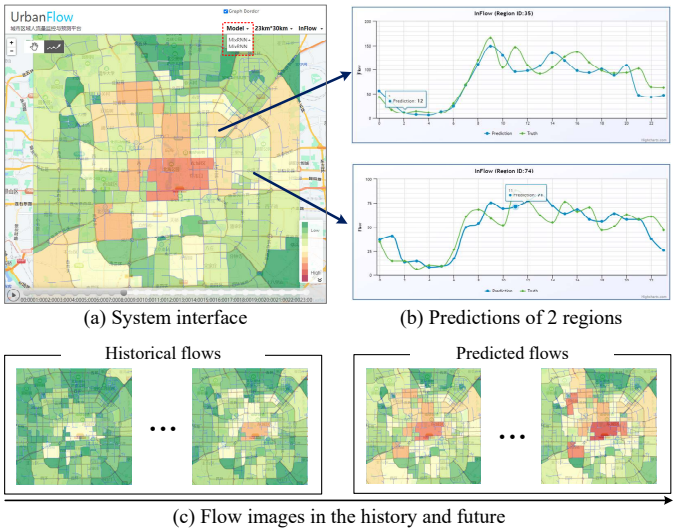



Fig. 10: Computation time of the methods.

## 7 System Deployment

Being able to predict the crowd flows in each and every part of a city, especially in irregular regions, is strategically important for traffic control, risk assessment, and public safety [64]. To solve this problem, we have upgraded the previously deployed cloud-based system (**UrbanFlow** [61]) by using our MixRNN or MixRNN+ as the bedrock model for the prediction. This system enables governors to monitor the real-time crowd flows and forecast crowd flows in the near future. Here, we briefly give an overview of the functions of UrbanFlow. Figure 11 illustrates the main interface of UrbanFlow, where each region on the map represents an irregular urban area. The color of each cell indicates its flow density, where "red" means dense and "green" means sparse. A user can click any region on the interface to see the flow details like Figure 11(b), including the ground truth as well as the prediction results. In different scenarios, users can specify to use MixRNN or MixRNN+ as the prediction model in the red rectangle in Figure 11. By clicking the "play button" at the bottom left of the main interface, we can watch a movie of flow heatmaps. More details about the system deployment can be found in [64].




(a) System interface (b) Predictions of 2 regions

(c) Flow images in the history and future

Fig. 11: The UrbanFlow system for crowd flow monitoring and forecasting in irregular regions. We upgrade it using our models as the bedrock model. (a) The main interface of UrbanFlow. (b) An example of the prediction interface of two regions. (c) A movie-style heatmap.

## 8 Conclusion and Future Work

We have presented the MixRNN+ for improving spatio-temporal forecasting, a small step that may benefit a variety of smart city applications. Specifically, we first propose an intermediate model called MixRNN to capture the mixed-order spatial relations as well as the temporal dependencies. Then, by assuming that the dynamic system is driven by reaction kinetics, we generalize MixRNNs to possess continuous-time dynamics via a residual update strategy. Our model has combined the flexibility of neural networks with interpretability provided by physical laws. The experiments on three real-world smart city applications have demonstrated our state-of-the-art performance. We will release our source code for public use soon. In the future, we plan to reduce the number of function evaluations in the ODE solver while preserving the accuracy, as we have noticed that the major efficiency bottleneck is the evaluation between two observations. Another direction is to explore how to integrate neural networks with more complex physical laws, such as second-order ODEs.

## Acknowledgments

This research is supported by Singapore Ministry of Education Academic Research Fund Tier 2 under MOE's official grant number T2EP20221-0023, also supported by the Beijing Nova Program (Z201100006820053), and the Beijing Natural Science Foundation (4212021). H. Chen is supported in part by the funding project of Zhejiang Lab under Grant 2020LC0PI01. We thank all reviewers for their constructive suggestions in improving this paper.

## References

[1] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[2] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *IJCAI*, 2018.

[3] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph wavenet for deep spatial-temporal graph modeling," in *IJCAI*, 7 2019, pp. 1907–1913.

[4] Z. Pan, W. Zhang, Y. Liang, W. Zhang, Y. Yu, J. Zhang, and Y. Zheng, "Spatio-temporal meta learning for urban traffic prediction," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[5] Y. Zheng, X. Yi, M. Li, R. Li, Z. Shan, E. Chang, and T. Li, "Forecasting fine-grained air quality based on big data," in *KDD*. ACM, 2015, pp. 2267–2276.

[6] Z. Qi, T. Wang, G. Song, W. Hu, X. Li, and Z. Zhang, "Deep air learning: Interpolation, prediction, and feature analysis of fine-grained air quality," *IEEE TKDE*, vol. 30, no. 12, pp. 2285–2297, 2018.

[7] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction," in *IJCAI*, 2018.

[8] X. Yi, J. Zhang, Z. Wang, T. Li, and Y. Zheng, "Deep distributed fusion network for air quality prediction," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 965–973.

[9] Y. Liu, Y. Zheng, Y. Liang, S. Liu, and D. S. Rosenblum, "Urban water quality prediction based on multi-task multi-view learning," in *IJCAI*, ser. IJCAI'16, 2016, p. 2576–2582.

[10] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 240–254, 1994.

[11] Z. Cui, K. Henrickson, R. Ke, and Y. Wang, "Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 11, pp. 4883–4894, 2019.

[12] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-gcn: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[14] 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, pp. 686–707, 2019.

[15] L. A. Rossman and P. F. Boulos, "Numerical methods for modeling water quality in distribution systems: A comparison," *Journal of Water Resources planning and management*, vol. 122, no. 2, pp. 137–146, 1996.

[16] L. Monteiro, D. Figueiredo, S. Dias, R. Freitas, D. Covas, J. Menaia, and S. Coelho, "Modeling of chlorine decay in drinking water supply systems using epanet msx," *Procedia Engineering*, vol. 70, pp. 1192–1200, 2014.

[17] V. Alexandrov, A. Sameh, Y. Siddique, and Z. Zlatev, "Numerical integration of chemical ode problems arising in air pollution models," *Environmental Modeling & Assessment*, vol. 2, no. 4, pp. 365–377, 1997.

[18] C. Lattanzio, A. Maurizi, and B. Piccoli, "Moving bottlenecks in car traffic flow: a pde-ode coupled model," *SIAM Journal on Mathematical Analysis*, vol. 43, no. 1, pp. 50–67, 2011.

[19] Y. Chen, B. Yang, Q. Meng, Y. Zhao, and A. Abraham, "Time-series forecasting using a system of ordinary differential equations," *Information Sciences*, vol. 181, no. 1, pp. 106–114, 2011.

[20] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in neural information processing systems*, 2007, pp. 1601–1608.

[21] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *AAAI*, vol. 33, 2019, pp. 3558–3565.

[22] J. Yi and J. Park, "Hypergraph convolutional recurrent neural network," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, p. 3366–3376.

[23] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," *Inverse problems*, vol. 34, no. 1, p. 014004, 2017.

[24] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations," in *International Conference on Machine Learning*. PMLR, 2018, pp. 3276–3285.

[25] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Advances in neural information processing systems*, 2018, pp. 6571–6583.

[26] L. Ruthotto and E. Haber, "Deep neural networks motivated by partial differential equations," *Journal of Mathematical Imaging and Vision*, pp. 1–13, 2019.

[27] Y. Rubanova, R. T. Chen, and D. K. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," *Advances in Neural Information Processing Systems*, vol. 32, pp. 5320–5330, 2019.

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[29] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, 2020.

[30] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

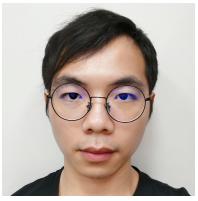
[31] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," *arXiv preprint arXiv:1511.02136*, 2015.

[32] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *arXiv preprint arXiv:1706.02216*, 2017.

[33] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *arXiv preprint arXiv:1901.08150*, 2019.

[34] C. F. Daganzo and J. A. Laval, "Moving bottlenecks: A numerical method that converges in flows," *Transportation Research Part B: Methodological*, vol. 39, no. 9, pp. 855–863, 2005.
[35] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and L. Zhenhui, "Deep multi-view spatial-temporal network for taxi demand prediction," in *AAAI*, 2018.
[36] G. E. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *Journal of the American statistical Association*, vol. 65, no. 332, pp. 1509–1526, 1970.
[37] E. Zivot and J. Wang, "Vector autoregressive models for multivariate time series," *Modeling Financial Time Series with S-Plus®*, pp. 385–429, 2006.
[38] W. Zhang, H. Liu, Y. Liu, J. Zhou, and H. Xiong, "Semi-supervised hierarchical recurrent graph neural network for city-wide parking availability prediction," in *AAAI*, vol. 34, no. 01, 2020, pp. 1186–1193.
[39] J. Ye, L. Sun, B. Du, Y. Fu, and H. Xiong, "Coupled layer-wise graph convolution for transportation demand prediction," *arXiv preprint arXiv:2012.08080*, 2020.
[40] S. Fang, Q. Zhang, G. Meng, S. Xiang, and C. Pan, "Gstnet: Global spatial-temporal network for traffic flow prediction." in *IJCAI*, 2019, pp. 2286–2293.
[41] M. Zhang, T. Li, H. Shi, Y. Li, P. Hui *et al.*, "A decomposition approach for urban anomaly detection across spatiotemporal data," in *IJCAI*, 2019.
[42] X. Geng, Y. Li, L. Wang, L. Zhang, Q. Yang, J. Ye, and Y. Liu, "Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting," in *AAAI*, vol. 33, no. 01, 2019, pp. 3656–3663.
[43] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," in *AAAI*, vol. 34, no. 01, 2020, pp. 1234–1241.
[44] R. Huang, C. Huang, Y. Liu, G. Dai, and W. Kong, "Lsgcn: Long short-term traffic prediction with graph convolutional networks," in *IJCAI*, 7 2020, pp. 2355–2361.
[45] F. Xu, Y. Li, and S. Xu, "Attentional multi-graph convolutional network for regional economy prediction with open migration data," in *KDD*, 2020, pp. 2225–2233.
[46] A. Karpatne, W. Watkins, J. Read, and V. Kumar, "Physics-guided neural networks (pgnn): An application in lake temperature modeling," *arXiv preprint arXiv:1710.11431*, 2017.
[47] X. Jia, A. Karpatne, J. Willard, M. Steinbach, J. Read, P. C. Hanson, H. A. Dugan, and V. Kumar, "Physics guided recurrent neural networks for modeling dynamical systems: Application to monitoring water temperature and quality in lakes," *arXiv preprint arXiv:1810.02880*, 2018.
[48] E. De Bézenac, A. Pajot, and P. Gallinari, "Deep learning for physical processes: Incorporating prior scientific knowledge," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, no. 12, p. 124009, 2019.
[49] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu, "Towards physics-informed deep learning for turbulent flow prediction," in *KDD*, 2020, pp. 1457–1466.
[50] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[51] F. Zhou, L. Li, K. Zhang, G. Trajcevski, F. Yao, Y. Huang, T. Zhong, J. Wang, and Q. Liu, "Forecasting the evolution of hydropower generation," in *KDD*, 2020, pp. 2861–2870.
[52] F. Zhou and L. Li, "Forecasting reservoir inflow via recurrent neural odes," in *AAAI*, 2021.
[53] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014, pp. 3104–3112.
[54] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *EMNLP*, 2014, pp. 1724–1734.
[55] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," in *AAAI*, vol. 33, 2019, pp. 5668–5675.
[56] F. M. Bianchi, D. Grattarola, and C. Alippi, "Spectral clustering with graph neural networks for graph pooling," in *International Conference on Machine Learning*. PMLR, 2020, pp. 874–883.
[57] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.
[58] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 1171–1179.
[59] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[60] 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 *KDD*, 2020, pp. 753–763.
[61] J. Sun, J. Zhang, Q. Li, X. Yi, Y. Liang, and Y. Zheng, "Predicting citywide crowd flows in irregular regions using multi-view graph convolutional networks," *IEEE Transactions on Knowledge and Data Engineering*, 2020.
[62] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[63] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *AAAI*, vol. 33, 2019, pp. 922–929.
[64] J. Zhang, Y. Zheng, D. Qi, R. Li, X. Yi, and T. Li, "Predicting citywide crowd flows using deep spatio-temporal residual networks," *Artificial Intelligence*, vol. 259, pp. 147–166, 2018.

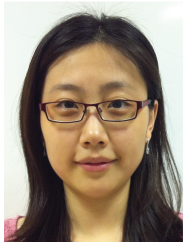
**Yuxuan Liang** is now a PhD student in School of Computing at National University of Singapore. He has published over twenty papers in the refereed conferences and journals, such as KDD, TKDE, NeurIPS, WWW, IJCAI, AAAI, MM and UbiComp. He previously worked as a Research Intern at Microsoft Research, Beijing, China and JD.COM, Beijing, China. His research interests lie in deep learning, machine learning, and their applications in urban areas.



**Kun Ouyang** is a PhD with major in Computer Science graduated from the National University of Singapore (NUS). He is now working in E-commerce advertising strategy and algorithms. He was also a research scholar in SAP Singapore. His research interests include spatial-temporal data mining and deep learning, with applications in human mobility analytics and online advertising respectively.

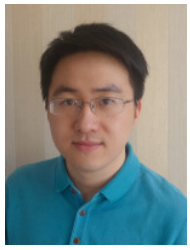
**Yiwei Wang** is currently a PhD student majored in Computer Science at National University of Singapore. He has published papers in the conferences and journals, including KDD, WWW, IJCAI, ICDM, ECML-PKDD, ECCV, ISSRE, and TSP. He previously worked as a Research Intern at Tencent, Shenzhen, China. His current research focuses on Graph Neural Network, and its applications in Natural Language Processing and Recommendation.

**Zheyi Pan** is a computer science Ph.D. candidate in Apex Data & Knowledge Management Lab, Department of Computer Science, Shanghai Jiaotong University, supervised by Prof. Yong Yu. He received his B.E. degree from Zhiyuan College, Shanghai Jiao Tong University in 2015. His research interests include deep learning and data mining with a special focus on urban computing and spatio-temporal data.



**Yifang Yin** received the B.E. degree from the Department of Computer Science and Technology, Northeastern University, Shenyang, China, in 2011, and received the Ph.D. degree from the National University of Singapore, Singapore, in 2016. She is currently a senior research fellow with the Grab-NUS AI Lab at the National University of Singapore. She worked as a Research Intern at the Incubation Center, Research and Technology Group, Fuji Xerox Co., Ltd., Japan, from October, 2014 to March, 2015. Her research interests include machine learning, spatiotemporal data mining, and multimodal analysis in multimedia.

**Hongyang Chen** (Senior Member, IEEE) received his B.S. and M.S. degrees from Southwest Jiaotong University, China, in 2003 and 2006, and Ph.D. degree from University of Tokyo, Japan, in 2011. He is currently a Senior Research Expert with Zhejiang Lab, China. He has authored 100+ refereed journal and conference papers in ACM Sensor Networks, IEEE TSP, IEEE TWC and IEEE ICC. His research interests include IoT, data-driven intelligent systems, machine learning, locationbased big data, and statistical signal processing.



**Junbo Zhang** (Member, IEEE) is a Senior Researcher of JD Intelligent Cities Research. He is leading the Urban AI Product Department of JD iCity at JD Technology, as well as AI Lab of JD Intelligent Cities Research. His research interests include Spatio-Temporal Data Mining and AI, Urban Computing, Deep Learning, Federated Learning. He has published over 50 research papers (e.g., AI Journal, IEEE TKDE, KDD, AAAI, IJCAI, WWW, ACL, UbiComp) in refereed journals and conferences. He serves as an Associate Editor of ACM Transactions on Intelligent Systems and Technology. He received the ACM Chengdu Doctoral Dissertation Award in 2016, the Chinese Association for Artificial Intelligence (CAAI) Excellent Doctoral Dissertation Nomination Award in 2016, the Si Shi Yang Hua Medal of SWJTU in 2012, and the Outstanding Ph.D. Graduate of Sichuan Province in 2013. He is a senior member of CCF (China Computer Federation), a member of IEEE and ACM.

**Yu Zheng** (Fellow, IEEE) is the Vice President of JD.COM and head JD Intelligent Cities Research. Before Joining JD.COM, he was a senior research manager at Microsoft Research. He currently serves as the Editor-in-Chief of ACM Transactions on Intelligent Systems and Technology and has served as the program co-chair of ICDE 2014 (Industrial Track), CIKM 2017 (Industrial Track) and IJCAI 2019 (industrial track). He is also a keynote speaker of AAAI 2019, KDD 2019 Plenary Keynote Panel and IJCAI 2019 Industrial Days. His monograph, entitled Urban Computing, has been used as the first text book in this field. In 2013, he was named one of the Top Innovators under 35 by MIT Technology Review (TR35) and featured by Time Magazine for his research on urban computing. In 2016, Zheng was named an ACM Distinguished Scientist and elevated to an IEEE Fellow in 2020 for his contributions to spatio-temporal data mining and urban computing.

**David S. Rosenblum** (Fellow, IEEE) is Provost's Chair Professor of Computer Science at the National University of Singapore (NUS). His research interests span many problems in software engineering, distributed systems and ubiquitous computing, and his current research focuses on probabilistic verification, uncertainty in software testing, and machine learning. He is a Fellow of the ACM and IEEE and was previously Editor-in-Chief of the ACM Transactions on Software Engineering and Methodology (ACM TOSEM) and Chair of the ACM Special Interest Group in Software Engineering (ACM SIGSOFT). He has received two "test-of-time" awards for his research papers, including the ICSE 2002 Most Influential Paper Award for his ICSE 1992 paper on assertion checking, and the inaugural ACM SIGSOFT Impact Paper Award in 2008 for his ESEC/FSE 1997 on Internet-scale event observation and notification (co-authored with Alexander L. Wolf). He also received the ACM SIGSOFT Distinguished Service Award in 2018.

**Roger Zimmermann** (Senior Member, IEEE) received the M.S. and Ph.D. degrees from the University of Southern California (USC), in 1994 and 1998, respectively. He is a Professor with the Department of Computer Science, National University of Singapore (NUS). He is also the Deputy Director with the Smart Systems Institute (SSI) and a Key Investigator with the Grab-NUS AI Lab. He has coauthored a book, seven patents, and more than 300 conference publications, journal articles, and book chapters. His research interests include streaming media architectures, multimedia networking, applications of machine/deep learning, and spatial data analytics. He is currently an Associate Editor for IEEE MultiMedia, Transactions on Multimedia Computing, Communications, and Applications (TOMM) (ACM), Multimedia Tools and Applications (MTAP) (Springer), and the IEEE Open Journal of the Communications Society (OJ-COMS). He is a distinguished member of the ACM and a senior member of the IEEE.