Scalable Spatiotemporal Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural forecasting of spatiotemporal time series drives both research and industrial innovation in several relevant application domains. Graph neural networks (GNNs) are often the core component of the forecasting architecture. However, in most spatiotemporal GNNs, the computational complexity scales up to a quadratic factor with the length of the sequence times the number of links in the graph, hence hindering the application of these models to large graphs and long temporal sequences. While methods to improve scalability have been proposed in the context of static graphs, few research efforts have been devoted to the spatiotemporal case. To fill this gap, we propose a scalable architecture that exploits an efficient encoding of both temporal and spatial dynamics. In particular, we use a randomized recurrent neural network to embed the history of the input time series into high-dimensional state representations encompassing multi-scale temporal dynamics. Such representations are then propagated along the spatial dimension using different powers of the graph adjacency matrix to generate node embeddings characterized by a rich pool of spatiotemporal features. The resulting node embeddings can be efficiently pre-computed in an unsupervised manner, before being fed to a feed-forward decoder that learns to map the multi-scale spatiotemporal representations to predictions. The training procedure can then be parallelized node-wise by sampling the node embeddings without breaking any dependency, thus enabling scalability to large networks. Empirical results on relevant datasets show that our approach achieves results competitive with the state of the art, while dramatically reducing the computational burden.

1 Introduction

As graph neural networks (GNNs; [1, 2]) are gaining more traction in many application fields, the need for architectures scalable to large graphs – such as those associated with large sensor networks – is becoming a pressing issue. While research to improve the scalability of models for static graph signals has been very prolific [3–6], little attention has been paid to the additional challenges encountered when dealing with discrete-time dynamical graphs, i.e., spatiotemporal time series. Several of the existing scalable training techniques rely on subsampling graphs to reduce the computational requirements of the training procedure, e.g., [3, 5]. However, sampling the node-level observations as if they were i.i.d. can break relational (spatial) dependencies in static graphs and it is even more problematic in the dynamic case, as dependencies occur also across the temporal dimension. Indeed, complex temporal and spatial dynamics that emerge from the interactions across the whole graph over a long time horizon, can be easily disrupted by perturbing such spatiotemporal structure with subsampling. As an alternative, precomputing aggregated features over the graph allows for factoring out spatial propagation from the training phase in certain architetures [6]. However, similarly to the subsampling approach, extending this method to the spatiotemporal case is not trivial as the preprocessing step must account also for the temporal dependencies besides the graph topology.

Submitted to NeurIPS 2022 Temporal Graph Learning Workshop. Do not distribute.

In this paper, we propose a novel scalable encoder-decoder architecture for processing spatiotemporal data. Fig. 1, shows a high-level overview of the architecture. The spatiotemporal *encoding* scheme is training-free: first, it exploits a deep randomized recurrent neural network [7, 8] to encode the history of each sequence in a high-dimensional vector embedding; then, it uses powers of the graph adjacency matrix to build informative node representations of the spatiotemporal dynamics at different scales. According to the downstream task at hand, the *decoder* maps the node representations into the desired output, e.g., the future values of the time series associated with each node. To improve efficiency, we exploit the structure of the extracted embedding to design the decoder to act as a collection of filters localized at different spatiotemporal scales.

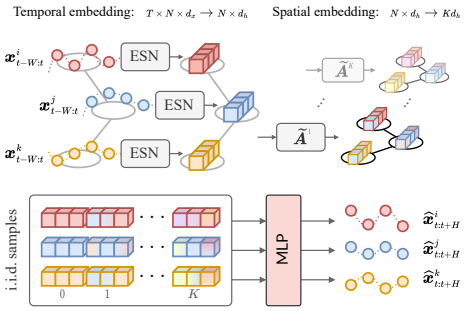

Figure 1: Overview of the framework. An Echo-State Network (ESN) – with shared parameters across nodes – encodes temporal dynamics. Then, K graph shift operators are used to propagate spatial information. The resulting $K+1$ representations are concatenated and fed to an MLP to obtain predictions.

Since the spatiotemporal encoder requires neither training nor supervision, the representation of each node and time step can be computed in a preprocessing stage, without the constraints that come from online training on GPUs with limited memory. The decoder is the only component of the architecture with trainable parameters. However, since spatiotemporal relationships are already embedded in the representations, the embeddings can be processed independently from their spatiotemporal context with two consequent advantages. First, training can be done node-wise, allowing for sampling node representations in mini-batches of a size proportional to the hardware capacity. Second, the decoder can be implemented similarly to a standard multilayer perceptron (MLP) readout, which is fast and easy to train. Let T and E be the number of steps and the number of edges in the input graph, respectively. The cost of training a standard spatiotemporal GNN on a mini-batch of data has a computational and memory cost that scales as $\mathcal{O}(TE)$, or $\mathcal{O}(T^2E)$ in attention-based architectures [9]. Conversely, in our approach mini-batches can be sampled disregarding the length of the sequence and size of the graph, thus making scalability in training constant, i.e., $\mathcal{O}(1)$, w.r.t. the spatiotemporal dimension of the problem.

Our contributions can be summarized as follows.

- We propose a general scalable deep learning framework for spatiotemporal time series, which exploits a novel encoding method based on randomized recurrent components and scalable GNNs architectures.
- We apply the proposed model to forecast multivariate time series, whose channels are subject to spatial relationships described by a graph.
- We carry out a rigorous and extensive empirical evaluation of the proposed architecture and variations thereof. Notably, we introduce *two* benchmarks for scalable spatiotemporal forecasting architectures.

Empirical results show that our approach performs on par with the state of the art while being easy to implement, computationally efficient, and extremely scalable. Given these considerations, we refer to our architecture as ***Scalable Graph Predictor*** **(SGP)**.

2 Preliminaries and Problem Definition

We consider discrete-time spatiotemporal graphs. In particular, given N interlinked sensors, we indicate with $\boldsymbol{x}_t^i \in \mathbb{R}^{d_x}$ the d_x-dimensional multivariate observation associated with the i-th sensor at time-step t, with $\boldsymbol{X}_t \in \mathbb{R}^{N \times d_x}$ the node attribute matrix encompassing measurements graph-wise, and with $\boldsymbol{X}_{t:t+T}$ the sequence of T measurements collected in the time interval $[t, t+T)$ at each sensor. Similarly, we indicate with $\boldsymbol{U}_t \in \mathbb{R}^{N \times d_u}$ the matrix containing exogenous variables (e.g., weather information related to a monitored area) associated with each sensor at the t-th time-step. Then, we indicate additional, optional, static node attributes as $\boldsymbol{V} \in \mathbb{R}^{N \times d_v}$. The relational information

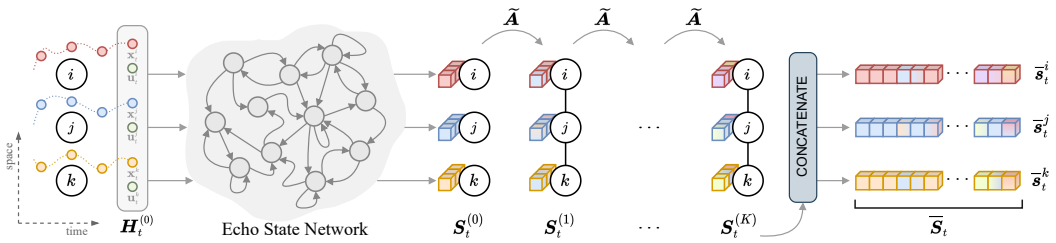

Figure 2: Overview of the SGP encoder. Input time series are fed into a randomized network with recurrent connections and embedded into a hierarchical vector representation. A graph shift operator is used to propagate and aggregate spatial information of different order which is then concatenated to obtain a final embedding.

is encoded in a, potentially dynamic, weighted adjacency matrix $\boldsymbol{A}_t \in \mathbb{R}^{N \times N}$. We indicate with the tuple $\mathcal{G}_t = \langle \boldsymbol{X}_t, \boldsymbol{U}_t, \boldsymbol{V}, \boldsymbol{A}_t \rangle$ the graph signal at the t-th time-step. Note that the number of sensors in a network is here considered fixed only to ease the presentation; we only request nodes to be distinguishable across time steps. The objective of spatiotemporal forecasting is to predict the next H observations given a window of W past measurements. In particular, we consider the family of forecasting models $F_\theta(\cdot)$ such that $\widehat{\boldsymbol{X}}_{t:t+H} = F_\theta\left(\mathcal{G}_{t-W:t}\right)$, where θ indicates the learnable parameters of the model and $\widehat{\boldsymbol{X}}_{t:t+H}$ the H-step ahead point forecast.

Echo-State Networks Echo state networks [7, 10] are a class of randomized architectures that consist of recurrent neural networks with random connections that encode the history of input signals into a high-dimensional state representation to be used as input to a (trainable) readout layer. The main idea is to feed an input signal into a high-dimensional, randomized, and non-linear reservoir, whose internal state can be used as an embedding of the input dynamics. An echo state network is governed by the state update equation $\boldsymbol{h}_t = \sigma\left(\boldsymbol{W}_x \boldsymbol{x}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}\right)$, where $\boldsymbol{x}_t$ indicates a generic input to the system, $\boldsymbol{W}_x \in \mathbb{R}^{d_h \times d_x}$ and $\boldsymbol{W}_h \in \mathbb{R}^{d_h \times d_h}$ are the random matrices defining the connectivity pattern in the reservoir, $\boldsymbol{b} \in \mathbb{R}^{d_h}$ is a randomly initialized bias, $\boldsymbol{h}_t$ indicates the reservoir state, and σ is a nonlinear activation function (usually tanh). If the random matrices are defined properly, the reservoir will extract a rich pool of dynamics characterizing the system underlying the input time series $\boldsymbol{x}_t$ and, thus, the reservoir states become informative embeddings of $\boldsymbol{x}_{t-T:t}$ [10]. Thanks to the non-linearity of the reservoir, the embeddings are commonly processed with a linear readout that is optimized with a least squares procedure to perform classification, clustering, or time series forecasting [11].

3 Scalable Spatiotemporal GNNs

This section presents our approach to building scalable GNN architectures for time series forecasting. Our method is based on a hybrid encoder-decoder architecture. The encoder first constructs representations of the time series observed at each node by using a reservoir that accounts for dynamics at different time scales. Representations are further processed to account for spatial dynamics described by the graph structure. In particular, as shown on the right-hand side of Fig. 2, we use incremental powers of the graph adjacency matrix to propagate and aggregate information along the spatial dimension. Each power of the propagation matrix accounts for different scales of spatial dynamics. The final embedding is then built by concatenating representations obtained w.r.t. each propagation step, thus resulting in a rich encoding of both spatial and temporal features.

The encoder does not need any training and, once computed, the embeddings can be uniformly sampled over time and space when training a nonlinear readout to perform H-step-ahead predictions. The straightforward choice for the decoder (i.e., readout) is to map the encodings to the outputs (i.e., predictions) by using a linear transformation or a standard MLP. However, to further enhance scalability, our decoder exploits the structure of the embedding to reduce the number of parameters and learn filters that are localized in time and space. As we will discuss, this is done by learning separate weight matrices for each spatiotemporal scale. In the following, we describe each component of the architecture in detail.

Spatiotemporal Encoder We consider as temporal encoders deep echo state networks (DeepESN; Gallicchio et al. 8) with leaky integrator neurons [12]. In particular, we consider networks where the signal associated with each node is encoded first as $\boldsymbol{h}_t^{i,(0)} = \left[\boldsymbol{x}_t^i \| \boldsymbol{u}_t^i\right]$ and then by a stack of L randomized recurrent layers s.t.

$$\begin{aligned} \hat{\boldsymbol{h}}_t^{i,(l)} &= \tanh\left(\boldsymbol{W}_u^{(l)} \boldsymbol{h}_t^{i,(l-1)} + \boldsymbol{W}_h^{(l)} \boldsymbol{h}_{t-1}^{i,(l)} + \boldsymbol{b}^{(l)}\right), \\ \boldsymbol{h}_t^{i,(l)} &= (1-\gamma_l)\boldsymbol{h}_{t-1}^{i,(l)} + \gamma_l \hat{\boldsymbol{h}}_t^{i,(l)}, \qquad l = 1, \dots, L \end{aligned} \tag{1}$$

where $\gamma_l \in (0, 1]$ is a discount factor associated with l-th layer, $\boldsymbol{W}_u^{(l)} \in \mathbb{R}^{d_{h^l} \times d_{h^{l-1}}}$, $\boldsymbol{W}_h \in \mathbb{R}^{d_{h^l} \times d_{h^l}}$, $\boldsymbol{b} \in \mathbb{R}^{d_{h^l}}$ are random weight matrices, $\boldsymbol{h}_t^{i,(l)}$ indicates the hidden state of the system w.r.t. the i-th node at the l-th layer, and $\|$ indicates node-wise concatenation. As Eq. 1 shows, DeepESNs are a hierarchical stack of reservoir layers that, e.g., by changing the discount factor at each layer, extract a rich pool of multi-scale temporal dynamics [8][1]. Given a DeepESN encoder, the input is represented by the concatenation of the states from each layer, i.e., we obtain node-level temporal encodings $\overline{\boldsymbol{h}}_t^i$ for each node i and time-step t as $\overline{\boldsymbol{h}}_t^i = \left(\boldsymbol{h}_t^{i,(0)} \| \boldsymbol{h}_t^{i,(1)} \| \dots \| \boldsymbol{h}_t^{i,(L)}\right)$. We indicate as $\overline{\boldsymbol{H}}_t$ the encoding for the whole graph at time t. The extraction of the node-level temporal embeddings is depicted on the left-end side of Fig. 2, where, to simplify the drawing, we depict an ESN with a single layer.

The next step is to propagate information along the spatial dimension. As discussed at the beginning of the section, we use powers of a graph shift operator $\widetilde{\boldsymbol{A}}$ to propagate and aggregate node representations at different scales. We then obtain spatiotemporal encodings as

$$\begin{aligned} \boldsymbol{S}_t^{(0)} &= \overline{\boldsymbol{H}}_t = \left(\boldsymbol{H}_t^{(0)} \| \boldsymbol{H}_t^{(1)} \| \dots \| \boldsymbol{H}_t^{(L)}\right), \\ \boldsymbol{S}_t^{(k)} &= \widetilde{\boldsymbol{A}} \boldsymbol{S}_t^{(k-1)} = \left(\widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(0)} \| \widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(1)} \| \dots \| \widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(L)}\right), \end{aligned} \tag{2}$$

with $\overline{\boldsymbol{S}}_t = \left(\boldsymbol{S}_t^{(0)} \| \boldsymbol{S}_t^{(1)} \| \dots \| \boldsymbol{S}_t^{(K)}\right)$ and where $\widetilde{\boldsymbol{A}}$ indicates a generic graph shift operator matching the sparsity pattern of the graph adjacency matrix. In practice, by indicating with $\boldsymbol{D}$ the graph degree matrix, we use $\widetilde{\boldsymbol{A}} = \boldsymbol{D}^{-1}\boldsymbol{A}$ in the case of a directed graph and the symmetrically normalized adjacency $\widetilde{\boldsymbol{A}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$ in the undirected case. Furthermore, for directed graphs we optionally increase the number of representations to $2K + 1$ to account for bidirectional dynamics, i.e., we repeat the encoding process w.r.t. the transpose adjacency matrix similarly to [14]. Intuitively, each propagation step $\widetilde{\boldsymbol{A}} \boldsymbol{S}_t^{(k-1)}$ propagates and aggregates – properly weighted – features between nodes connected by paths of length k in the graph. As shown in Eq. 2, features corresponding to each order k can be computed recursively with K sparse matrix-matrix multiplications (Fig. 2). Alternatively, each matrix $\widetilde{\boldsymbol{A}}^k$ can be precomputed and the computation of the different blocks of matrix $\overline{\boldsymbol{S}}_t$ can be distributed in a parallel fashion. Even in the case of extremely large graphs, features $\overline{\boldsymbol{S}}_t$ can be computed offline by exploiting distributed computing as they do not need to be loaded on the GPU memory.

Multi-Scale Decoder The role of the decoder is that of selecting and weighing from the pool of the (possibly redundant) features extracted by the spatiotemporal encoder and mapping them to the desired output. Representations $\overline{\boldsymbol{S}}_t$ can be fed into an MLP that performs node-wise predictions. Since the representations are large vectors, a naïve implementation of the MLP results in many parameters that hinder scalability. Therefore, we replace the first MLP layer with a more efficient implementation that exploits the structure of the embeddings. As we described in the previous paragraph, $\overline{\boldsymbol{S}}_t$ is the concatenation of the representations corresponding to different spatial propagation steps which, in turn, are obtained from the concatenation of multi-scale temporal features. To exploit this structure, we design the first layer of the decoder with a sparse connectivity pattern to learn representations $\overline{\boldsymbol{Z}}_t$ such that

$$\boldsymbol{Z}_t^{(k)} = \sigma\left(\widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(0)} \boldsymbol{\Theta}_k^{(0)} \| \dots \| \widetilde{\boldsymbol{A}}^k \boldsymbol{H}_t^{(L)} \boldsymbol{\Theta}_k^{(L)}\right) = \sigma\left(\boldsymbol{S}_t^{(k)} \begin{bmatrix} \boldsymbol{\Theta}_k^{(0)} & & 0 \\ & \ddots & \\ 0 & & \boldsymbol{\Theta}_k^{(L)} \end{bmatrix}\right), \tag{3}$$

[1]We refer to [13] for more details on the properties and stability of DeepESNs.

where $\boldsymbol{\Theta}_k^{(l)} \in \mathbb{R}^{d_{h^l} \times d_z}$ are the learnable parameters and σ is an activation function. We indicate with $\overline{\boldsymbol{Z}}_t$ the concatenated representations $\overline{\boldsymbol{Z}}_t = \left(\boldsymbol{Z}_t^{(0)} \| \boldsymbol{Z}_t^{(1)} \| \dots \| \boldsymbol{Z}_t^{(K)}\right)$. In practice, $\overline{\boldsymbol{Z}}_t$ can be efficiently computed by exploiting grouped 1-d convolutions (e.g., see Krizhevsky et al. 15) to parallelize computation on GPUs. In particular, if we indicate the 1-d grouped convolution operator with g groups and kernel size r as $\star_{r,g}$, and the collection of the decoder parameters $\boldsymbol{\Theta}_k^{(l)}$ as $\boldsymbol{\Theta}$, we obtain $\overline{\boldsymbol{Z}}_t = \sigma\left(\boldsymbol{\Theta} \star_{1,g} \overline{\boldsymbol{S}}_t\right)$, with $g = L(K+1)$ in the case of undirected graphs and $g = L(2K+1)$ for the directed case. Besides reducing the number of parameters by a factor of $L(K+1)$, this architecture localizes filters $\boldsymbol{\Theta}_k^{(L)}$ w.r.t. the dynamics of spatial order k and temporal scale l. In fact, as highlighted in Eq. 3, representation $\overline{\boldsymbol{Z}}_t$ can be seen as a concatenation of the results of $L(K+1)$ graph convolutions of different order. Finally, the obtained representations are fed into an MLP that predicts the H-step-ahead observations as $\widehat{\boldsymbol{x}}_{t:t+H}^i = \texttt{MLP}\left(\overline{\boldsymbol{z}}_t^i, \boldsymbol{v}^i\right)$ where the static node-level attributes $\boldsymbol{v}^i$ can also be augmented by concatenating a set of learnable parameters (i.e., a learnable positional encoding).

Training and sampling The main improvement introduced by the proposed approach in terms of scalability concerns the training procedure. Representations $\overline{\boldsymbol{S}}_t$ embed both the temporal and spatial relationships among observations over the sensor network. Consequently, each sample $\overline{\boldsymbol{s}}_t^i$ can be processed independently since no further spatiotemporal information needs to be collected. This allows for training the decoder with SGD by uniformly and independently sampling mini-batches of data points $\overline{\boldsymbol{s}}_t^i$. This is the key property that makes the training procedure extremely scalable and drastically reduces the lower bound on the computational complexity required for the training w.r.t. standard spatiotemporal GNN architectures.

4 Related works

Spatiotemporal GNNs are essentially based on the idea of integrating message-passing modules in architectures to process sequential data. Notably, Seo et al. [16] and Li et al. [14] use message-passing to implement gates of recurrent neural networks. Yu et al. [17] and Wu et al. [18, 19] proposed architectures alternating temporal and spatial convolutions. Wu et al. [9] and Marisca et al. [20], instead, exploit the attention mechanism to propagate information along both time and space. Modern architectures often combine some type of relational inductive bias, with full Transformer-like attention [21] along the spatial dimension [22–24], which, however, makes the computation scale quadratically with the number of nodes. SGP falls within the category of *time-then-graph models*, i.e., models where the temporal information is encoded before being propagated along the spatial dimension. Gao and Ribeiro [25] showed that such models can be more expressive than architectures that alternate temporal and spatial processing steps.

Research on scalable models for discrete-time dynamic graphs has been relatively limited. Practitioners have mostly relied on methods developed in the context of static graphs which include node-centric, GraphSAGE-like, approaches [3] or subgraph sampling methods, such as ClusterGCN [4] or GraphSAINT [5]. Wu et al. [19], Gandhi et al. [26], Wu et al. [27] are examples of such approaches. Among scalable GNNs for static graphs, SIGN [6] is the approach most related to our method. Like in our approach, SIGN performs spatial propagation as a preprocessing step by using different shift operators to aggregate across different graph neighborhoods, which are then fed to an MLP. However, SIGN is limited to static graphs and propagates raw node-level attributes. Finally, similar to our work, DynGESN [28] processes dynamical graphs with a recurrent randomized architecture. However, the architecture in DynGESN is completely randomized, while ours is hybrid as it combines randomized components in the encoder with trainable parameters in the decoder.

5 Empirical evaluation

We empirically evaluate our approach in 2 different scenarios. In the first, we compare the performance of our forecasting architecture against state-of-the-art methods on popular, medium-scale, traffic forecasting benchmarks. In the second, we evaluate the scalability of the proposed method on large-scale spatiotemporal time series datasets by considering two novel benchmarks for load forecasting and PV production prediction.

Datasets In the first experiment we consider the **METR-LA** and **PEMS-BAY** datasets [14], which are popular medium-sized benchmarks used in the spatiotemporal forecasting literature. In particular, METR-LA consists of traffic speed measurements taken every 5 minutes by 207 detectors in the Los Angeles County Highway, while PEMS-BAY includes analogous observations recorded by 325 sensors in the San Francisco Bay Area. We use the same preprocessing steps of previous works to extract a graph and obtain train, validation and test data splits [18]. For the second experiment, we introduce two larger-scale datasets derived from energy analytics data. The first dataset contains data coming from the Irish Commission for Energy Regulation Smart Metering Project (**CER-E**; Commission for Energy Regulation 29), which has been previously used for benchmarking spatiotemporal imputation methods [30]; however, differently from previous works, we consider the full sensor network consisting of 6435 smart meters measuring energy consumption every 30 minutes at both residential and commercial/industrial premises. The second large-scale dataset is obtained from the synthetic **PV-US**[2] dataset [31], consisting of simulated energy production by 5016 PV farms scattered over the United States given historic weather data for the year 2006, aggregated in half an hour intervals. Since the model does not have access to weather information, PV production at neighboring farms is instrumental for obtaining good predictions. Notably, CER-E and PV-US datasets are at least an order of magnitude larger than the datasets typically used for benchmarking spatiotemporal time series forecasting models. Note that for both PV-US and CER-En the (weighted) adjacency is obtained by applying a thresholded Gaussian kernel to the similarity matrix obtained by considering the geographic distance among the sensors and the correntropy [32] among the time series, respectively. We provide further details on the datasets in the supplemental material.

Baselines We consider the following baselines:

1. **LSTM**: a single standard gated recurrent neural network [33] trained by sampling window of observations from each node-level time series by disregarding the spatial information;
2. **FC-LSTM**: an LSTM processing input sequences as if they were a single high-dimensional multivariate time series;
3. **DCRNN**: a recurrent graph network presented in [14] – differently from the original model we use a recurrent encoder followed by a linear readout (more details in the appendix);
4. **Graph WaveNet**: a residual network that alternates temporal and graph convolutions over the graph that is given as input and an adjacency matrix that is learned by the model [18];
5. **Gated-GN**: a state-of-the-art time-than-graph [25] model introduced in [24] for which we consider two different configurations. The first one – indicated as **FC** – uses attention over the full node set to perform spatial propagation, while the second one – indicated as **UG** – constrains the attention to edges of the underlying graph.
6. **DynGESN**: the echo state network for dynamical graphs proposed in [28].

For all the baselines, we use, whenever possible, the configuration found in the original papers or in their open-source implementation; in all the other cases we tune hyperparameters on the holdout validation set.

Experimental setup For the traffic datasets, we replicate the setup used in previous works. In particular, each model is trained to predict the 12-step-ahead observations. In SGP, the input time series are first encoded by the spatiotemporal encoder, and then the decoder is trained by sampling mini-batches along the temporal dimension, i.e., by sampling B sequences $\mathcal{G}_{t-W:t}$ of observations.

For the large-scale datasets, we focus on assessing the scalability of the different architectures rather than maximizing forecasting accuracy. In particular, for both datasets, we consider the first 6 months of data (4 for months for training, 1 month for validation, and 1 month for testing). The models are trained to predict the next $\{00{:}30, 07{:}30, 11{:}00\}$ hours. We repeat the experiment in two different settings to test the scalability of the different architectures w.r.t. the number of edges. In the first setting, we extract the graph by sparsifying the graph adjacency matrix imposing a maximum of 100 neighbors for each node, while in the second case we do not constrain the density of the adjacency matrix.

[2] `https://www.nrel.gov/grid/solar-power-data.html`

Table 1: Results on benchmark traffic datasets (averaged over 3 independent runs). We report metrics averaged over a one-hour (12 steps) forecasting horizon and MAE for $H \in \{15, 30, 60\}$ minutes time steps. Bold numbers are within a standard deviation from the best reported average result.

	METR-LA					**PEMS-BAY**				
	15 min	30 min	60 min	Average		15 min	30 min	60 min	Average	
	MAE	MAE	MAE	MAE	MAPE (%)	MAE	MAE	MAE	MAE	MAPE (%)
LSTM	2.99 ± 0.00	3.58 ± 0.00	4.43 ± 0.01	3.58 ± 0.00	10.19 ± 0.05	1.39 ± 0.00	1.83 ± 0.01	2.35 ± 0.01	1.79 ± 0.00	4.16 ± 0.05
FC-LSTM	3.33 ± 0.01	3.43 ± 0.01	3.67 ± 0.01	3.46 ± 0.01	10.15 ± 0.09	2.22 ± 0.01	2.25 ± 0.01	2.34 ± 0.02	2.26 ± 0.01	5.33 ± 0.04
DynGESN	3.27 ± 0.00	3.99 ± 0.00	5.00 ± 0.00	3.98 ± 0.00	11.11 ± 0.01	1.57 ± 0.00	2.13 ± 0.01	2.81 ± 0.02	2.09 ± 0.01	4.74 ± 0.01
DCRNN	2.82 ± 0.00	3.23 ± 0.01	3.74 ± 0.01	3.20 ± 0.00	8.88 ± 0.05	1.36 ± 0.00	1.71 ± 0.00	2.08 ± 0.01	1.66 ± 0.00	3.76 ± 0.01
Graph WaveNet	2.72 ± 0.01	3.10 ± 0.02	3.54 ± 0.03	3.06 ± 0.02	8.40 ± 0.03	1.31 ± 0.00	1.64 ± 0.01	1.94 ± 0.01	1.58 ± 0.00	3.58 ± 0.02
FC-Gated-GN	2.72 ± 0.01	$\mathbf{3.05 \pm 0.01}$	$\mathbf{3.44 \pm 0.01}$	3.01 ± 0.00	$\mathbf{8.27 \pm 0.00}$	1.32 ± 0.00	1.63 ± 0.01	$\mathbf{1.89 \pm 0.01}$	1.56 ± 0.01	3.51 ± 0.03
UG-Gated-GN	2.72 ± 0.00	3.10 ± 0.00	3.54 ± 0.01	3.06 ± 0.00	8.40 ± 0.04	1.33 ± 0.00	1.67 ± 0.01	1.99 ± 0.01	1.61 ± 0.01	3.59 ± 0.03
SGP	$\mathbf{2.69 \pm 0.00}$	$\mathbf{3.05 \pm 0.00}$	$\mathbf{3.45 \pm 0.00}$	$\mathbf{3.00 \pm 0.00}$	$\mathbf{8.27 \pm 0.02}$	$\mathbf{1.30 \pm 0.00}$	$\mathbf{1.60 \pm 0.00}$	$\mathbf{1.88 \pm 0.00}$	$\mathbf{1.54 \pm 0.00}$	$\mathbf{3.44 \pm 0.01}$
Ablations										
–No-Space-Enc.	2.84 ± 0.00	3.26 ± 0.00	3.74 ± 0.00	3.22 ± 0.00	9.20 ± 0.01	1.34 ± 0.00	1.68 ± 0.00	2.02 ± 0.00	1.62 ± 0.00	3.67 ± 0.01
–FC-Dec.	2.76 ± 0.01	3.13 ± 0.01	3.52 ± 0.02	3.08 ± 0.01	8.63 ± 0.11	1.35 ± 0.01	1.67 ± 0.01	1.96 ± 0.01	1.61 ± 0.01	3.61 ± 0.04
–GC-Dec.	2.77 ± 0.00	3.17 ± 0.00	3.63 ± 0.00	3.12 ± 0.00	8.74 ± 0.01	1.32 ± 0.00	1.65 ± 0.00	1.97 ± 0.00	1.59 ± 0.00	3.60 ± 0.01

Table 2: Results on large-scale datasets (averaged over at least 3 independent runs). We report MAE over H-step-ahead predictions, $H = \{30\text{m}, 7\text{h}30\text{m}, 11\text{h}\}$, together with timings and memory consumption. $*$ indicates that subsampling was needed to comply with the memory constraints. Bold numbers are within a standard deviation from the best reported average result.

		PV-US						**CER-En**					
		Prediction error (MAE)			Resource utilization			Prediction error (MAE)			Resource utilization		
		30 mins	7 hours 30 mins	11 hours	Batch/s	Memory	Batch size	30 mins	7 hours 30 mins	11 hours	Batch/s	Memory	Batch size
100-NN	DCRNN	1.39 ± 0.09	3.34 ± 0.22	$\mathbf{3.54 \pm 0.48}$	2.04 ± 0.01	9.63 GB	2	0.22 ± 0.00	$\mathbf{0.28 \pm 0.00}$	0.29 ± 0.00	1.43 ± 0.02	11.10 GB	2
	Graph WaveNet	1.45 ± 0.13	5.09 ± 0.63	5.26 ± 1.34	2.01 ± 0.02	11.64 GB	2	0.23 ± 0.00	0.36 ± 0.01	0.36 ± 0.01	2.41 ± 0.03	8.39 GB	1
	UG-Gated-GN	1.33 ± 0.08	$\mathbf{2.94 \pm 0.05}$	$\mathbf{3.12 \pm 0.14}$	8.41 ± 0.09	11.46 GB	5	0.22 ± 0.00	$\mathbf{0.28 \pm 0.00}$	$\mathbf{0.28 \pm 0.00}$	8.21 ± 0.08	11.70 GB	4
	SGP	$\mathbf{1.09 \pm 0.01}$	$\mathbf{3.14 \pm 0.21}$	$\mathbf{3.16 \pm 0.19}$	$\mathbf{116.58 \pm 8.74}$	**2.21 GB**	4096	$\mathbf{0.21 \pm 0.00}$	0.30 ± 0.00	0.31 ± 0.01	$\mathbf{117.32 \pm 8.36}$	**2.21 GB**	4096
Full	DCRNN	1.59 ± 0.17	4.10 ± 0.27	4.93 ± 0.60	1.37 ± 0.00	11.59 GB	1^*	0.23 ± 0.00	0.29 ± 0.00	$\mathbf{0.29 \pm 0.00}$	1.13 ± 0.01	11.10 GB	1^*
	Graph WaveNet	1.65 ± 0.23	6.93 ± 0.58	7.93 ± 0.17	0.77 ± 0.00	11.35 GB	2	0.25 ± 0.01	0.38 ± 0.03	0.37 ± 0.01	1.26 ± 0.01	8.58 GB	1
	UG-Gated-GN	1.61 ± 0.06	3.25 ± 0.04	$\mathbf{3.04 \pm 0.05}$	8.83 ± 0.10	11.14 GB	1^*	0.22 ± 0.00	$\mathbf{0.28 \pm 0.00}$	$\mathbf{0.29 \pm 0.00}$	8.77 ± 0.10	11.14 GB	1^*
	SGP	$\mathbf{1.09 \pm 0.00}$	$\mathbf{3.06 \pm 0.11}$	$\mathbf{3.13 \pm 0.13}$	$\mathbf{118.64 \pm 8.35}$	**2.21 GB**	4096	$\mathbf{0.21 \pm 0.00}$	0.30 ± 0.00	0.31 ± 0.01	$\mathbf{115.85 \pm 10.60}$	**2.21 GB**	4096

To assess the performance in terms of scalability, we fix a maximum GPU memory budget of 12 GB and select the batch size accordingly; if a batch size of 1 does not fit in 12 GB, we uniformly subsample edges of the graph to reduce the memory consumption. Differently from the other baselines, in SGP we first preprocess the data to obtain spatiotemporal embeddings and then train the decoder by uniformly sampling the node representations. We train each model for 1 hour, then restore the weights corresponding to the minimum training error and evaluate the forecasts on the test set. The choice of not running validation at each epoch was dictated by the fact that for some of the baselines running a validation epoch would take a large portion of the 1 hour budget. The time required to encode the datasets with SGP's encoder ranges from tens of seconds to ≈ 4 minutes on an AMD EPYC 7513 processor with 32 parallel processes. To ensure reproducibility, the time constraint is not imposed as a hard time out; conversely, we measure the time required for the update step of each model on an NVIDIA RTX A5000 GPU and fix the maximum number of updates accordingly. For SGP, the time required to compute node embeddings was considered as part of the training time and the number of updates was appropriately reduced to make the comparison fair. For all the baselines, we keep the same architecture used in the traffic experiment. For SGP we use the same hyperparameters for the decoder, but we reduce the dimension of the embedding (the value of K) so that a preprocessed dataset can fit in a maximum of ≈ 80 GB of storage. To account for the different temporal scales, we increase the window size for all baselines and increase the number of layers in the ESN (while keeping the final size of $\overline{\boldsymbol{H}}_t$ similar). Additional details and the exact values of the hyperparameters are provided in the supplementary material.

5.1 Results

Results for the traffic benchmarks are reported in Tab. 1; while the outcomes of the scalability experiments are shown in Tab. 2. We consider *mean absolute error* (MAE), and *mean absolute percentage error* (MAPE) as evaluation metrics.

Traffic experiment The purpose of the first experiment is to demonstrate that the proposed method achieves performance comparable to that of the state of the art. In this regard, results in Tab. 1 show that in all the considered scenarios SGP is always among the best performing forecasting architectures. The full-attention baseline is the strongest competitor which, however, has time and memory complexities that scale quadratically with the number of nodes. Regarding the other baselines, DCRNN underperforms compared to the other spatiotemporal GNN architectures. DynGESN, the fully randomized architecture, despite being very fast to train, obtains reasonable performance in short-range predictions but falls short over longer forecasting horizons in the considered scenarios. In light of these results, it is worth commenting on the efficiency of SGP compared to the baselines. Approaches like DCRNN and Graph Wavenet, perform graph convolutions whose time and space of complexity is $\mathcal{O}(LTE)$, being E the number of edges, L the number of layers (8 in Graph Wavenet), and T the time steps. Such complexity is completely amortized by the preprocessing step in our architecture. Similarly, Gated-GN, while being architecturally much simpler, propagates spatial information by relying on the attention mechanism that is known to scale poorly with the dimensionality of the problem. The bottom of Tab. 1 reports results for the ablation of key elements of the proposed architecture: **No-Space-Enc.** indicates that the embeddings are built without the spatial propagation step; **FC-Dec.** consider the case where the structure of the embedding is ignored in the readout and the sparse weight matrix in Eq. 3 is replaced by a fully-connected one; **GC-Dec.** indicates that the spatial propagation is limited to the neighbors of order $K = 1$ and, thus, the decoder behaves similarly to a single-layer graph convolutional network. Results clearly show the optimality of the proposed architectural design.

Large-scale experiment Tab. 2 reports the results of the scalability experiment where we considered only the spatiotemporal GNNs trained by gradient descent. We excluded the full-attention baseline (FC-Gated-GN) as its $\mathcal{O}(N^2)$ complexity prevented scaling to the larger datasets; however, we considered the UG version where attention is restrained to each node's neighborhood. There are several comments that need to be made here. First of all, batch size has a different meaning for our model and the other baselines. In our case, each sample corresponds to a single spatiotemporal (preprocessed) observation; for the other methods, a sample corresponds to a window of observations $\mathcal{G}_{t-W:t}$ where edges of the graph are eventually subsampled if the memory constraints could not be met otherwise. In both cases, the loss is computed w.r.t. all the observations in the batch. The results clearly show that SGP can be trained efficiently also in resource-constrained settings, with contained GPU memory usage. In particular, the update frequency (batch/s) is up to 2 order of magnitude higher. Notably, resource utilization at training time remains constant (by construction) in the two considered scenarios, while almost all the baselines require edge subsampling in order to meet the resource constraints. Fig. 3 shows learning curves for the PV-US dataset, further highlighting the vastly superior efficiency, scalability, and learning stability of SGP. Finally, results concerning the forecasting accuracy show that performance is competitive with the state of the art in all the considered scenarios.

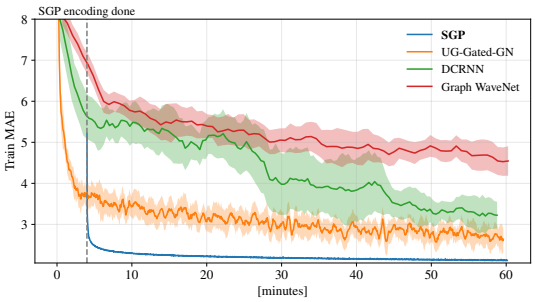

Figure 3: Training curves on PV-US. The plot shows the average $\pm$ the standard deviation of 3 independent runs. The plotted curves are smoothed with a running average of 8 steps.

6 Conclusion

We proposed SGP, a scalable architecture for graph-based spatiotemporal time series forecasting. Our approach can compete with the state of the art in popular medium-sized benchmark datasets, while greatly improving the scalability in large sensor networks. We believe that SGP constitutes an important stepping stone for future research on scalable spatiotemporal forecasting and has the potential of being widely adopted by practitioners in both academia and industry.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks and Learning Systems*, 20(1):61–80, 2008.

[2] Davide Bacciu, Federico Errica, Alessio Micheli, and Marco Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 129:203–221, 2020.

[3] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

[4] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, 2019.

[5] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2019.

[6] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Benjamin Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

[7] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[8] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.

[9] Zonghan Wu, Da Zheng, Shirui Pan, Quan Gan, Guodong Long, and George Karypis. Traversenet: Unifying space and time in message passing for traffic forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[10] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[11] Filippo Maria Bianchi, Simone Scardapane, Sigurd Løkse, and Robert Jenssen. Reservoir computing approaches for representation and classification of multivariate time series. *IEEE Transactions on Neural Networks and Learning Systems*, 32(5):2169–2179, 2020.

[12] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3): 335–352, 2007.

[13] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Design of deep echo state networks. *Neural Networks*, 108:33–47, 2018.

[14] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[16] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[17] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3634–3640, 2018.

[18] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1907–1913, 2019.

[19] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 753–763, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Ivan Marisca, Andrea Cini, and Cesare Alippi. Learning to reconstruct missing data from spatiotemporal graphs with sparse observations. *arXiv preprint arXiv:2205.13479*, 2022.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[22] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. Gman: A graph multi-attention network for traffic prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 1234–1241, 2020.

[23] Boris N Oreshkin, Arezou Amini, Lucy Coyle, and Mark Coates. Fc-gaga: Fully connected gated graph architecture for spatio-temporal traffic forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9233–9241, 2021.

[24] Victor Garcia Satorras, Syama Sundar Rangapuram, and Tim Januschowski. Multivariate time series forecasting with latent graph inference. *arXiv preprint arXiv:2203.03423*, 2022.

[25] Jianfei Gao and Bruno Ribeiro. On the equivalence between temporal and static equivariant graph representations. In *International Conference on Machine Learning*, pages 7052–7076. PMLR, 2022.

[26] Ankit Gandhi, Sivaramakrishnan Kaveri, Vineet Chaoji, et al. Spatio-temporal multi-graph networks for demand forecasting in online marketplaces. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 187–203. Springer, 2021.

[27] Yuankai Wu, Dingyi Zhuang, Aurelie Labbe, and Lijun Sun. Inductive graph neural networks for spatiotemporal kriging. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4478–4485, 2021.

[28] Alessio Micheli and Domenico Tortorella. Discrete-time dynamic graph echo state networks. *Neurocomputing*, 496:85–95, 2022.

[29] Commission for Energy Regulation. CER Smart Metering Project - Electricity Customer Behaviour Trial, 2009-2010 [dataset]. *Irish Social Science Data Archive. SN: 0012-00*, 2016. URL `https://www.ucd.ie/issda/data/commissionforenergyregulationcer`.

[30] Andrea Cini, Ivan Marisca, and Cesare Alippi. Filling the g_ap_s: Multivariate time series imputation by graph neural networks. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=kOu3-S3wJ7`.

[31] Marissa Hummon, Eduardo Ibanez, Gregory Brinkman, and Debra Lew. Sub-hour solar data for power system modeling from static spatial variability analysis. Technical report, National Renewable Energy Lab.(NREL), Golden, CO (United States), 2012.

[32] Weifeng Liu, Puskal P Pokharel, and Jose C Principe. Correntropy: Properties and applications in non-gaussian signal processing. *IEEE Transactions on signal processing*, 55(11):5286–5298, 2007.

[33] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[34] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[36] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[37] Andrea Cini and Ivan Marisca. Torch Spatiotemporal, 3 2022. URL `https://github.com/TorchSpatiotemporal/tsl`.

[38] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL `https://github.com/Lightning-AI/lightning`.

[39] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[40] neptune.ai. Neptune: Metadata store for mlops, built for research and production teams that run a lot of experiments, 2021. URL `https://neptune.ai`.

[41] Hosagrahar V Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Communications of the ACM*, 57(7):86–94, 2014.

[42] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 95–104, 2018.

[43] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

[44] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[45] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[46] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units. *arXiv preprint arXiv:1606.08415*, 2016.

[47] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Detailed experimental settings

In this appendix, we provide additional details on the experimental settings for the results presented in the paper.

A.1 Software platform

The Python [34] code used to run all the computational experiments will be open-sourced in a future release. We relied on the following open-source libraries:

- PyTorch [35];
- PyTorch Geometric [36];
- Torch Spatiotemporal [37];
- PyTorch Lightning [38];
- `numpy` [39].

We relied on the Neptune[3] [40] DevOps infrastructure for the logging of the experiments. For all the baselines, we run all the experiments by relying on their open-source implementations.

A.2 Hardware platform

Experiments were run on a server equipped with two AMD EPYC 7513 processors and four NVIDIA RTX A5000. Reproducibility of the scalability experiments was ensured by taking timings for the update step of each model and setting the number of updates performed by each model accordingly (more details in Sec. A.5).

Table 3: Additional information on the considered datasets.

Dataset	# steps	# nodes	# edges	sparsity
METR-LA	34272	207	1515	3.54%
PEMS-BAY	52116	325	2369	2.24%
PV-US (100nn)	8868	5016	417,199	1.66%
CER-En (100nn)	8868	6435	639,369	1.54%
PV-US	8868	5016	3,710,008	14.75%
CER-En	8868	6435	3,186,369	7.69%

A.3 Datasets

All datasets used in our study are open-source or freely available for research purposes. The input graphs are extracted by at first computing a weighted, dense adjacency matrix $\boldsymbol{W}$ from (side) spatial information, e.g., the geographic position of the sensors, or by computing a (dis)similarity metric among the time series. The adjacency is then sparsified to obtain $\boldsymbol{A}$ by zeroing out values under a certain threshold and, optionally, capping the maximum number of neighbors for each node. For all datasets, the only exogenous variable we consider is the encoding of the time of the day with two sinusoidal functions. Tab/ 3 shows additional details of the considered datasets.

Traffic datasets Both METR-LA and PEMS-BAY are widely popular benchmarks. We use the same setup of previous works [18] for all the preprocessing steps. As mentioned in Sec. 5, PEMS-BAY contains 6 months of data from 325 traffic sensors in the San Francisco Bay Area, while METR-LA contains 4 months of analogous readings acquired from 207 detectors in the Los Angeles County Highway [41]. In both datasets, observations are aggregated at a 5 minutes time scale.

CER-En The data from the Irish Commission for Energy Regulation Smart Metering Project [29] contains measurements of the energy consumption aggregated at a 30 minutes scale in households and small/medium enterprises. The full dataset consists of observations from 6435 smart meters measuring energy consumption every 30 minutes. As mentioned in the paper, we use the same preprocessing of [30], and, in particular, an analogous strategy to extract a graph from the correntropy [32] among time series. Note that, differently from [30], we consider the full sensor network. For all the spatiotemporal GNN baselines, we set the window size to 36 steps. Access to the dataset can be obtained free of charge by following the information provided at `https://www.ucd.ie/issda/data/commissionforenergyregulationcer`.

PV-US The PV-US[4] dataset [31] instead consists in a collection of simulated energy production by 5016 PV farms for the year 2006. In the raw datasets, samples are generated every 5 minute, we aggregate observations at 30 minutes intervals by taking their mean. A (small) subset of this dataset (often referred to as "Solar Energy"[5]) with only the 137 PV plants in Alabama state has been used as a multivariate time series forecasting benchmark [42]. To obtain an adjacency matrix, we consider the virtual position of the farms in terms of geographic coordinates, and we apply a Gaussian kernel over the pairwise Haversine distances, as described at the beginning of this section. Similarly to the CER-En dataset, we set the window size of the baselines to 36 steps. In the supplementary material, we provide the code to download and preprocess the data.

A.4 Additional details on SGP architecture

We implemented the DeepESN encoder following the design principles assessed in previous works [13, 43]. In particular, we decrease the discount factor λ progressively at each layer by subtracting 0.1 from its initial value. We also randomly set 30% of the weights of the networks to 0 to obtain a sparse reservoir. We use $tanh$ as nonlinear activation function. The recurrent weights are normalized so that the spectral radius of the corresponding matrix is lower than one [7].

[3] `https://neptune.ai/`

[4] `https://www.nrel.gov/grid/solar-power-data.html`

[5] `https://github.com/laiguokun/multivariate-time-series-data`

For the spatial encoding, we compute the embeddings at the different spatial scales iteratively. Additionally, we also concatenate to the spatiotemporal embedding $\overline{\boldsymbol{S}}_t$ the graph-wise average of the temporal embedding $\overline{\boldsymbol{H}}_t$ to act as a sort of global attribute [44].

The MLP decoder is implemented as standard feed-forward network with parametrized residual connections between layers [45], SiLU activation function [46] and optional Dropout [47] regularization.

A.5 Training and evaluation procedure

A.5.1 Traffic

As previously mentioned, for the traffic datasets we used the same training settings of previous works. For all the baselines we kept the same parameters of previous works whenever possible. For SGP we selected the hyperparameters by performing an initial random search and then manually adjusting the hyperparameters of the reservoir and selecting the best performing configuration on the validation set. In particular, for METR-LA we used a DeepESN with 3 layers of 32 units each, an initial decay factor of 0.9, and a spectral radius of 0.9. For PEMS-BAY, instead, we used an encoder with a single layer of 128 units, a decay rate of 0.8, and a spectral radius of 0.9. For both datasets, we set $K = 4$ and used the bidirectional encoding scheme. In the decoder, for the first layer we used 32 units for each group in METR-LA and 96 PEMS-BAY, followed by 2 fully connected layers of 256 units each with a dropout rate of 0.3. The model is trained with early stopping for a maximum of 200 epochs of 300 batch each with the Adam optimizer and a multi-step learning rate scheduler.

A.5.2 Large-scale

In Tab. 2 of the paper, we report the time required for a single model update (in terms of batches per second) and GPU memory usage for every considered method. To ensure a fair assessment, we record the time interval between the beginning of the inference step and the end weights' update for 150 batches and exclude the first 5 and last 5 measurements (that may have overheads). We exclude from the computation the overhead introduced – for every batch – by the edge subsampling strategy adopted for the scalability of the baselines.

To measure the GPU memory required, we exploit NVIDIA System Management Interface[6], which provides near real-time GPU usage monitoring.

All the experiments designed to measure time and memory requirements have been run on the same machine on a dedicated reserved GPU. We kept the models mostly unchanged w.r.t. the traffic experiment. However, we increased the window size to 36 for the baselines and updated the configuration of the reservoir for SGP to account for the different time scales. In particular, we increased the number of reservoir layers to 8 and 6 in PV-US and to in CER-En, respectively, and reduced the number of units accordingly. The difference in the number of layers between the two datasets is motivated by the choice of keeping the size of the preprocessed sequences similar. For this reason, we also set $K = 2$ and use the unidirectional encoding to limit the amount of required storage to a maximum ≈ 80 GB for each dataset.

A.5.3 Baselines

For **LSTM** and **FC-LSTM** we consider a single-layer LSTM with 128 units for the temporal embedding and an MLP with one hidden layer with 256 units and dropout rate of 0.1. For **DCRNN**, as reported in [14], we set the number of units in the hidden state to 64 and the order of the diffusion convolution to $K = 2$; compared to the original mode, we use a feed-forward readout instead of a recurrent one to enable scalability on the larger benchmarks. For **Graph WaveNet** and **Gated-GN** we use the same hyperparameters and learning rate schedulers reported in the relative papers. We implemented all the baselines in PyTorch and PyTorch Geometric (for graph-based methods) following the open-source implementations provided by the authors. To improve memory and computation efficiency in message-passing layers, we use sparse matrix-matrix multiplications instead of scatter-gather operations whenever possible. We fix the maximum number of training epochs to 300 to allow all the models to reach convergence, and stop the training if the MAE computed

[6] `https://developer.nvidia.com/nvidia-system-management-interface`

on the validation set does not decrease for 50 epochs. We evaluate the models using the weights corresponding to the minimum validation MAE.

For **DynGESN** we set the hyperparameters of the reservoir to the same ones used for SGP and increase the number of units to approximately match the dimensions of the final embeddings extracted by our method. We trained the readout with Ridge regression by selecting the weight of the L2-regularization term on the validation set.