APPENDIX

In this technical appendix, we provide additional details on experimental setups, model implementations, and related works. In addition, supplementary evaluation results and case studies are provided for additional discussion.

A DETAILED EXPERIMENTAL SETUPS

This section provides additional information about the adopted datasets, baseline models, and detailed experimental settings for the results presented in the main paper.

A.1 Dataset Descriptions

We adopt several public urban computing datasets to benchmark our model. They include traffic flow, environmental measurements, and energy records, which are widely used in relevant studies.

Traffic Flow Benchmarks. (1) To evaluate the short-term forecasting performance, we adopt six high-resolution highway traffic flow datasets, including two traffic speed datasets: `METR-LA` and `PEMS-BAY` [28], and four traffic volume datasets: `PEMS03`, `PEMS04`, `PEMS07`, and `PEMS08` [20]. The six datasets contain traffic measurements aggregated every 5 minutes from loop sensors installed on highway networks. `METR-LA` contains spot speed data from 207 loop sensors over a period of 4 months from Mar 2012 to Jun 2012, located at the Los Angeles County highway network. `PEMS-BAY` records 6 months of speed data from 325 static detectors in the San Francisco South Bay Area. `PEMS0X` contains the real-time highway traffic volume information in California, collected by the Caltrans Performance Measurement System (PeMS) [6] in every 30 seconds. The raw traffic flow is aggregated into a 5-minute interval for the experiments. Similarly to `METR-LA` and `PEMS-BAY`, `PEMS0X` also includes an adjacency graph calculated by the physical distance between the sensors. (2) To evaluate NexuSQN's long-term forecasting performance, we use the `TrafficL` benchmark [25]. This dataset provides hourly records of road occupancy rates (between 0 and 1) measured by 862 sensors on San Francisco Bay Area freeways over a period of 48 months. (3) To verify the scalability of models, we adopt the large-scale `LargeST` benchmarks. `LargeST` dataset contains 5 years of traffic readings from 01/01/2017 to 12/31/2021 collected every 5 minutes by 8600 traffic sensors in California. We adopt the largest subset `GLA` and use the readings from 2019 that are considered, aggregated into 15-minute intervals for experiments.

Air Quality Benchmarks. `AQI` data from the Urban Air project [65] record PM2.5 pollutant measurements collected by 437 air quality monitoring stations across 43 Chinese cities from May 2014 to April 2015 with an aggregation interval of 1 hour. Note that `AQI` data contains nearly 26% missing data.

Energy Production and Consumption Benchmarks. Large-scale `PU-VS` production data [23] consists of a simulated energy production by 5016 PV farms in the United States during 2006. The original observations are aggregated into a 30-minute window. `Electricity` benchmark is widely adopted to evaluate long-term forecast performance. It records load profiles (in kWh) measured hourly by 321 sensors from 2012 to 2014. `CER-En`: smart meters measuring energy consumption from the Irish Commission for Energy Regulation Smart Metering Project [1]. Following [9], we consider the full sensor network containing 6435 smart meters with a 30-minute aggregation interval.

Global Meteorological Benchmarks. `Global Wind` and `Global Temp` [52] contain the hourly averaged wind speed and hourly temperature of meteorological 3850 stations around the world from 1 January 2019 to 31 December 2020 from the National Centers for Environmental Information (NCEI) system.

A.2 Experimental Setups

Table 13: Input and output settings.

	Datasets	window	Horizon	Graphs
Traffic	METR-LA	12	12	True
	PEMS-BAY	12	12	True
	PEMS03	12	12	True
	PEMS04	12	12	True
	PEMS07	12	12	True
	PEMS08	12	12	True
	TrafficL	96	384	False
	LargeST-GLA	12	12	True
Energy	Electricity	336	96	False
	CER-EN	36	22	True
	PV-US	36	22	True
Environ.	AQI	24	3	True
	Global Temp	48	24	False
	Global Wind	48	24	False

Basic Setups. We adopt the widely used input-output settings in related literature to evaluate the models. These settings are given in Tab. 13. For all the datasets, we adopt the same training (70%), validating (10%), and testing (20%) set splits and preprocessing steps as previous work. It is worth commenting that NexuSQN does not rely on redefined graphs, and results in Tab. 10 indicate that the benefits of incorporating predefined graphs are marginal. Time-of-day information is provided as exogenous variables for our model, and day-of-week feature is input to baselines that require this information. In addition, labels with nonzero values are used to compute the metrics. And the performance of all methods is recorded in the same evaluation environment. We evaluate the model's performance using metrics such as mean absolute error (MAE), mean squared error (MSE), and mean absolute percentage error (MAPE). For long-term benchmarks including `TrafficL` and `Electricity`, we do not use the standardization on labels and use the normalized mean absolute error (NMAE), mean relative error (MRE), and mean absolute percentage error (MAPE) as metrics.

Ablation Study. We consider the following three variations:

- w $\mathbf{A}_{\text{pre}}$: We add additional diffusion graph convolutions [28] with three forward and backward diffusion steps using distance-based graphs in `SpaceMixer`.
- w/o SpaceMixer: We remove the "SpaceMixer" design by keeping each channel independent [36]. In this scenario, our model shares a structure similar to that of STID [43].

[1] https://www.ucd.ie/issda/data/commissionforenergyregulationcer

- w/o $\mathbf{E}_T$: We remove the spatiotemporal node embedding, and our model degrades into a spatiotemporal MLP-MIXER [7].

For each of them, we keep the same experimental settings in Section 4 and report the forecasting results.

Robustness Study. To evaluate the robustness under model attack, we select several representative STGNNs and randomly corrupt a proportion of linear weights in the input embedding or the final readout layer. These corrupted values are filled with zeros.

Transfer Study. As suggested in [11], node embedding can facilitate transfer between different datasets. Therefore, we first pretrain our model on small PEMS08 data, then freeze all parameters except node embedding. During the transfer stage, we randomly initialize the node embedding according to the spatial dimension of the target larger datasets (i.e., PEMS03, PEMS04, and PEMS07) and fine tune the embedding parameter.

Table 14: Hyperparameters of NexuSQN.

Model configurations		PEMS03	PEMS04	PEMS07	PEMS08
Hyperparameter	input_emb_size	128	128	128	128
	n_spacemixer	1	1	1	1
	activation	GeLU			
	st_embed	False	True	True	False
	node_emb_size	64	64	64	96
Training configs	window_size	12			
	horizon	12			
	batch_size	32			
	learning_rate	0.005			
	lr_gamma	0.1			
	lr_milestones	[40,60]			

Model configurations		METR-LA	PEMS-BAY	TrafficL	LargeST
Hyperparameter	input_emb_size	48	64	128	128
	n_spacemixer	1	1	2	1
	activation	GeLU			
	st_embed	False	False	False	False
	node_emb_size	80	96	64	96
Training configs	window_size	12		96	12
	horizon	12		384	12
	batch_size	32	32	16	16
	learning_rate	0.005			
	lr_gamma	0.1			
	lr_milestones	[20,30,40]			

B REPRODUCIBILITY

In order to ensure the reproducibility of this paper, this sections provides the detailed implementations of NexuSQN, as a technical complement to the descriptions presented in the paper. **Our code to reproduce the reported results is available at https://github.com/tongnie/NexuSQN.**

Platform. All these experiments are conducted on a Windows platform with one single NVIDIA RTX A6000 GPU with 48GB memory. Our implementations and suggested hyperparameters are mainly based on PyTorch [40], Torch Spatiotemporal [8], and BasicTS [42] benchmarking tools.

Hyperparameters. To ensure a fair and unbiased comparison, we adopt the original implementations presented in each paper. We also use the recommended hyperparameters for the baselines wherever possible. During evaluation, all baselines are trained, validated, and tested in identical environments. Our NexuSQN contains only a few model hyperparameters and is easy to tune. The detailed configurations of traffic datasets are shown in Tab. 14. And the hyperparameters of other datasets can be found in our repository. The configurations of other baseline models and implementations follow the official resources as much as possible.

C SUPPLEMENTARY RESULTS

Full Results of Traffic Data. Full results of short-term traffic benchmarks including the MAE at {3,6,12} steps are given in Tab. 15.

Full Results of Efficiency Analysis. Full results of computational performance on METR-LA and PEMS-BAY data are shown in Tab. 16. And additional illustrations are given in Figs. 6 and 7.

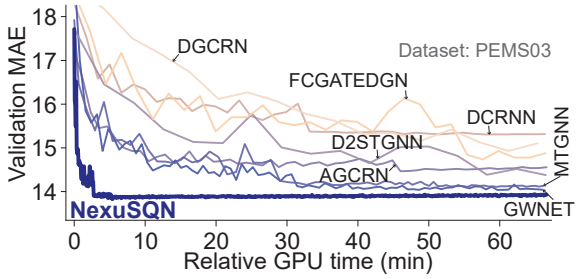

Figure 6: Validation MAE curves of different models.

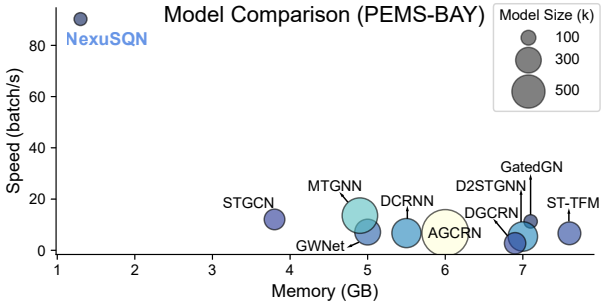

Figure 7: Computational performance.

Case Study on Temporal Contextualization. An illustration of the temporal contextualization issue and the behavior of different models is provided in Fig. 8.

Additional Results on Private data. Full evaluation results in five private data are shown in Tab. 17. The models are trained or fine-tuned to forecast the future 192 steps using the historical 192 steps. Online parallel testing results are given in Fig. 9.

D ADDITIONAL INTERPRETATIONS

To give a clear exposition of the design concept of our model, we provide more discussions and interpretations on the model architecture and the connections with related works in this section.

Table 15: Full results on METR-LA, PEMS-BAY, PEMS03, PEMS04, PEMS07, and PEMS08 datasets. MAE for {15, 30, 60} minutes forecasting horizons, as well as MAE, MSE, and MAPE averaged over one hour (12 time steps) are reported.

Dataset	METR-LA						PEMS-BAY						PEMS03				
Metric	15 min	30 min	60 min	Average			15 min	30 min	60 min	Average			15 min	30 min	60 min	Average	
	MAE	MAE	MAE	MAE	MSE	MAPE (%)	MAE	MAE	MAE	MAE	MSE	MAPE (%)	MAE	MAE	MAE	MAE	MAPE (%)
AGCRN	2.85	3.19	3.56	3.14	40.94	8.69	1.38	1.69	1.94	1.63	13.59	3.71	14.65	15.72	16.82	15.58	15.08
DCRNN	2.81	3.23	3.75	3.20	41.12	8.90	1.37	1.72	2.09	1.67	14.52	3.78	14.59	15.76	18.18	15.90	15.74
GWNet	2.74	3.14	3.58	3.09	38.86	8.52	1.31	1.65	1.96	1.59	13.31	3.57	13.67	14.60	16.23	14.66	14.88
GatedGN	2.72	3.05	3.43	3.01	37.19	8.20	1.34	1.65	1.92	1.58	13.17	3.55	13.72	15.49	19.08	17.08	14.44
GRUGCN	2.93	3.47	4.24	3.46	48.47	9.85	1.39	1.81	2.30	1.77	16.70	4.03	14.63	16.44	19.87	16.62	15.96
EvolveGCN	3.27	3.82	4.59	3.81	52.64	10.56	1.53	2.00	2.53	1.95	18.98	4.43	17.07	19.03	22.22	19.11	18.61
ST-Transformer	2.97	3.53	4.34	3.52	51.49	10.06	1.39	1.81	2.29	1.77	17.01	4.09	14.03	15.72	18.74	15.85	15.37
STGCN	2.84	3.25	3.80	3.23	40.34	9.02	1.37	1.71	2.08	1.66	14.11	3.75	14.27	15.49	18.02	15.61	16.07
STID	2.82	3.18	3.56	3.13	41.83	9.07	1.32	1.64	1.93	1.58	13.05	3.58	13.88	15.26	17.41	15.27	16.39
MTGNN	2.79	3.12	3.46	3.07	39.91	8.57	1.34	1.65	1.91	1.58	13.52	3.51	13.74	14.76	16.13	14.70	14.90
D2STGNN	2.74	3.08	3.47	3.05	38.30	8.44	**1.30**	**1.60**	1.89	1.55	12.94	3.50	13.36	14.45	15.96	14.45	14.55
DGCRN	2.73	3.10	3.54	3.07	39.02	8.39	1.33	1.65	1.95	1.59	13.12	3.61	13.83	14.77	16.14	14.73	14.86
SCINet	3.06	3.47	4.09	3.47	44.61	9.93	1.52	1.85	2.24	1.82	14.92	4.14	14.39	15.25	17.37	15.43	15.37
FreTS	3.03	3.67	4.62	3.67	54.68	10.49	1.41	1.87	2.45	1.84	18.28	4.23	14.53	16.78	21.07	17.04	16.18
TSMixer	2.93	3.31	3.79	3.28	41.63	8.97	1.44	1.80	2.12	1.73	14.93	3.91	14.47	15.34	17.11	15.43	15.41
NexuSQN	**2.66**	**2.99**	**3.36**	**2.95**	**35.35**	**8.04**	**1.30**	**1.60**	**1.86**	**1.54**	**12.42**	**3.45**	**13.15**	**14.20**	**15.79**	**14.18**	**13.77**

Dataset	PEMS04					PEMS07					PEMS08				
Metric	15 min	30 min	60 min	Average		15 min	30 min	60 min	Average		15 min	30 min	60 min	Average	
	MAE	MAE	MAE	MAE	MAPE (%)	MAE	MAE	MAE	MAE	MAPE (%)	MAE	MAE	MAE	MAE	MAPE (%)
AGCRN	18.27	18.95	19.83	18.90	12.69	19.55	20.65	22.40	20.64	9.42	14.50	15.16	16.41	15.23	10.46
DCRNN	19.19	20.54	23.55	20.75	14.32	20.47	22.11	25.77	22.30	9.51	14.84	15.93	18.22	16.06	10.40
GWNet	18.17	18.96	20.23	18.95	13.59	19.87	20.97	23.53	21.13	9.96	14.21	14.99	16.45	15.02	9.70
GatedGN	18.10	18.84	20.03	18.81	13.11	21.01	22.74	25.81	22.68	10.18	14.07	14.96	16.27	14.91	9.64
GRUGCN	19.89	22.28	27.37	22.68	15.81	21.15	24.00	29.91	24.37	10.29	15.46	17.32	21.16	17.55	11.43
EvolveGCN	23.21	25.82	30.97	26.21	17.79	24.72	28.09	34.41	28.40	12.11	18.21	20.46	24.45	20.64	13.11
ST-Transformer	19.13	21.33	25.69	21.63	15.12	20.17	22.80	27.74	23.05	9.77	14.39	15.87	18.69	16.00	10.71
STGCN	19.26	20.75	24.03	20.95	14.79	20.26	22.33	26.58	22.53	9.66	14.81	16.02	18.76	16.20	10.57
STID	17.77	18.60	20.01	18.60	12.87	18.66	19.93	21.91	19.88	8.82	13.53	14.29	15.75	14.32	9.69
MTGNN	17.88	18.49	19.62	18.48	12.76	18.94	20.23	22.23	20.19	8.59	13.80	14.57	15.82	14.57	9.46
D2STGNN	17.60	18.39	19.63	18.39	12.65	18.49	20.74	23.10	20.84	9.99	13.65	14.53	16.00	14.52	9.40
DGCRN	18.05	18.76	20.07	18.77	13.13	18.58	21.10	22.55	21.24	10.45	13.96	14.68	15.96	14.70	9.62
SCINet	18.30	19.03	20.85	19.19	13.31	21.63	22.89	25.96	23.11	10.00	14.57	15.60	17.75	15.77	10.13
FreTS	19.91	22.64	28.23	23.06	15.74	21.21	24.66	31.45	25.05	10.82	15.37	17.62	22.11	17.92	11.75
TSMixer	19.12	19.78	21.03	19.81	13.40	20.15	21.73	24.67	21.81	8.98	15.84	16.82	18.80	17.00	12.37
NexuSQN	**17.37**	**18.05**	**19.14**	**18.03**	**12.34**	**18.15**	**19.34**	**21.05**	**19.28**	**8.05**	**13.17**	**13.96**	**15.28**	**13.98**	**9.02**

- Best results are bold marked. Note that - indicates the model runs out of memory with the minimum batch size.

Table 16: Model computational performances.

	METR-LA				PEMS-BAY			
	Speed (Batch/s)	Memory (GB)	Batch size	Model size (k)	Speed (Batch/s)	Memory (GB)	Batch size	Model size (k)
AGCRN	10.43	4.1	64	989	6.78	6.0	64	991
DCRNN	9.90	3.8	64	387	6.72	5.5	64	387
GWNet	10.50	3.4	64	301	7.11	5.0	64	303
GatedGN	13.53	6.0	32	74.6	11.27	7.1	16	82.2
ST-Transformer	5.77	8.2	64	236	6.63	7.6	32	236
STGCN	17.31	2.9	64	194	12.05	3.8	64	194
MTGNN	21.39	2.9	64	405	13.54	4.9	64	573
D2STGNN	5.02	6.9	32	392	5.38	7.0	16	394
DGCRN	2.82	8.0	64	199	2.74	6.9	32	208
NexuSQN	**137.35**	**1.2**	**64**	**60.5**	**90.26**	**1.3**	**64**	**75.6**

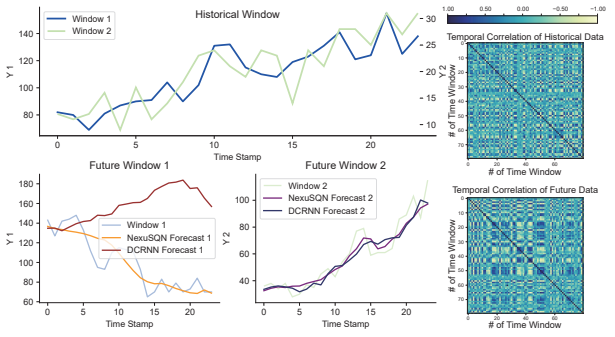

Figure 8: Temporal contextualization issue.

D.1 Additional Architecture Details

Time Stamp Encoding. We adopt sinusoidal positional encoding to inject the time-of-day information along time dimension:

$$\begin{aligned} PE_{\text{sine}} &= \texttt{sin}(p_i * 2\pi/\delta_D), \\ PE_{\text{cosine}} &= \texttt{cos}(p_i * 2\pi/\delta_D), \\ \mathbf{U}_t &= [PE_{\text{sine}} \| PE_{\text{cosine}}], \end{aligned} \tag{15}$$

Table 17: Results on Baoding, Yizhuang, Beijing, Shenzhen and Shanghai urban traffic data. (192 to 192)

	Baoding		Yizhuang		Beijing		Shenzhen		Shanghai	
	MAE	WAPE	MAE	WAPE	MAE	WAPE	MAE	WAPE	MAE	WAPE
MTGNN	13.66	59.55	16.41	77.20	OOM	OOM	22.38	55.91	OOM	OOM
AGCRN	OOM	OOM	12.24	59.61	OOM	OOM	OOM	OOM	OOM	OOM
StemGNN	12.34	55.59	12.97	61.16	OOM	OOM	OOM	OOM	OOM	OOM
TimesNet	10.12	48.77	11.78	56.60	15.52	45.69	18.22	45.37	16.39	40.88
Pyraformer	8.42	37.92	10.86	51.05	OOM	OOM	OOM	OOM	OOM	OOM
Autoformer	9.09	40.94	11.55	54.31	17.47	47.11	21.22	52.67	17.75	45.07
FEDformer	8.44	37.87	10.06	47.33	17.69	47.72	22.25	55.21	18.10	45.96
Informer	8.36	37.68	10.91	49.59	17.37	46.83	21.23	52.71	17.09	43.39
DLinear	8.65	38.76	10.71	50.36	13.69	36.92	13.61	37.79	13.48	34.24
PatchTST	9.99	44.98	10.53	49.52	13.62	36.74	14.55	36.11	12.95	32.89
NexuSQN	**7.89**	**35.54**	**9.55**	**44.91**	**11.21**	**30.23**	**10.89**	**32.01**	**11.03**	**27.98**

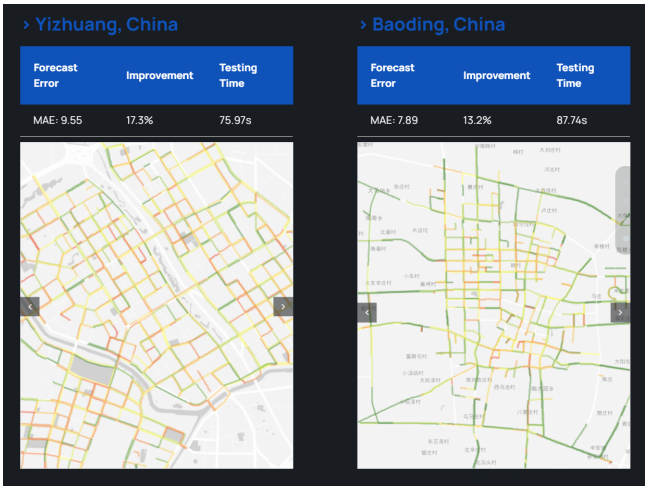

Figure 9: Online testing results of NexuSQN.

where p_i is the index of i-th time point in the series, and δ_D is the day-unit time mapping. We concatenate PE_{sine} and PE_{cosine} as the final temporal encoding. In fact, day-of-week embedding can also be applied using the one-hot encoding.

Node Embedding. For the spatial dimension, we adopt a unique identifier for each sensor. While an optional structural embedding is the random-walk diffusion matrix [14], for simplicity, we use the learnable node embedding [43] as a simple index positional encoding without any structural priors. Learnable node embedding can be easily implemented by initializing a parameter with its gradient trackable, i.e.,

```
self.emb = nn.Parameter(
    Tensor(self.num_nodes, self.emb_size),
    requires_grad=True)
```

The initialization method (e.g., Gaussian or uniform distribution) can be used to specify its initial distribution.

Dense Readout. For a multi-step STDF task, we adopt a MLP and a reshaping layer to directly output the predictions:

$$\begin{aligned}\widehat{\mathbf{X}}_{T+1:T+H} &= \text{MLP}(\mathbf{H}^{(L+1)}),\\ \widehat{\mathcal{X}}_{T+1:T+H} &= \texttt{UNFOLD}(\widehat{\mathbf{X}}_{T+1:T+H}),\end{aligned} \tag{16}$$

where $\texttt{UNFOLD}(\cdot)$ is the inverse linear operator of $\texttt{FOLD}(\cdot)$. For simplicity, we avoid a complex sequential decoder and directly make multi-step predictions through regression.

Interpretations of the Temporal Contextualization Issue. Different from the spatial contextualization issue, the temporal contextualization issue can be shown in a typical linear predictive model with weight $\mathbf{W}$ and bias $\mathbf{b}$, expressed as follows:

$$\begin{aligned}\mathbf{x}_{t+1:t+H} &= \mathbf{W}\mathbf{x}_{t-W:t} + \mathbf{b},\\ \text{or: } x_{t+h} &= \sum_{k=0}^{W} w_{k,h} x_{t-k} + b_{k,h},\ h \in \{1,\dots,H\}.\end{aligned} \tag{17}$$

Eq. (17) is an autoregressive model (AR) for each forecast horizon. Its weight $w_{k,h}$ depends solely on the relative time order and is agnostic to the absolute position in the sequence, rendering it incapable of contextualizing the series in temporal dimension. In this case, the temporal context is needed.

In the discussion by [7], predictive models like Eq. (17) are referred to as time-dependent. An upgrade to this type is termed data-dependent, where the weight becomes pattern-aware and conditions on temporal variations:

$$x_{t+h} = \sum_{k=0}^{W} \mathcal{F}_k(\mathbf{x}_{t-W:t}) x_{t-k} + b_{k,h}, \tag{18}$$

where $\mathcal{F}_k(\cdot)$ represents a data-driven function, such as self-attention. Eq. (18) creates a fully time-varying AR process, which is parameterized by time-varying coefficients [3]. However, its overparameterization can lead to overfitting of the data, rather than capturing the temporal relationships, such as the position on the time axis.

D.2 Remarks on Parameter-Efficient Designs

The proposed scalable mixing layers have several parameter-efficient designs. As NexuSQN conducts space mixing only after one time mixing rather than at every time point, it is more efficient than alternately stacking spatial-temporal blocks. We introduce these techniques and explain how they lighten the model structure.

Residual Connection. In each block, we incorporate a shortcut for the linear part. When all residual connections are activated (e.g., when spatial modeling is unnecessary), NexuSQN can degrade into a family of channel-independent linear models. This feature makes it promising for long-term series forecasting tasks [12, 60].

Parameter Sharing. In contrast to recent design trends that use separate node parameters or graphs for different layers [37, 61], we use a globally shared node embedding for all modules, including the TIMEMIXER and SPACEMIXER. This aims to simplify the end-to-end training of random embedding while reducing the model size. Furthermore, Yang et al. [57] and Han et al. [21] showed that MLP and GNN can share similar feature spaces. Without the contextualization function, SPACEMIXER can collapse into TIMEMIXER. Considering this, we instantiate consecutive SPACEMIXER layers

with a shared feedforward weight. This parameter-sharing design is inspired by the connection between MLP and GNN in spatiotemporal graphs.

Shallow-Layer Structure. Instead of using deep multilayer architectures, we use shallow layer structures with larger receptive fields. Specifically, we adopt a small number of structured space mixing layers (e.g., 1 layer in most cases) with all-to-all connections to capture long-range interactions. This approach avoids the use of multiple stacking of sparse graph aggregators or hierarchical operations such as diffusion convolutions [28]. This treatment is consistent with the previous finding that wider graphs can be more expressive than deeper ones under some conditions [53]. In other words, when the graph is dense, a single global mixing can gather adequate information adaptively from arbitrary nodes. Therefore, despite the simple structure, it has sufficient expressivity and a large receptive field to capture pairwise interactions. Another crucial aspect to consider when utilizing an all-to-all connection is that it prompts the model to gather comprehensive information from other windows to obtain a precise "temporal context."

D.3 Connections with Related Works

This detailed section on related work highlights recent advances in simplified neural forecasting models and advanced embedding techniques. We aim to provide a nexus between notable achievements by offering a meticulous exposition of two primary research avenues: (1) simplified models for time series forecasting; and (2) node embedding for spatiotemporal data, the unification of which served as the catalyst for driving the development of this work.

Simplified Models for Time Series Forecasting. Within the literature on time series studies, long-term series forecasting (LTSF) is considered one of the most challenging tasks. Encouragingly, recent studies have revisited the design of neural architectures and sought a simpler solution for this long-standing topic and empirically show the superiority of simple models over their complex counterparts, especially using the MLP-based architectures [7, 12, 60, 62]. In particular, Zeng et al. [60] analyzed the ineffectiveness of Transformer-based models on this task and outperformed an array of advanced Transformer-based baselines with a family of linear models. N-BEATS [38] and H-HITS [5] combine multiscale learning with residual MLP structures for univariate forecasting. Chen et al. [7] developed a fully MLP model that performs mixing operations along both the time and the feature dimensions, called TSMixer. The cross-sensor information is utilized by a weighted sum of all series. TiDE proposed by [12] enjoys a dense encoder-decoder structure with a residual MLP backbone. FreTS [58], PITS [26] and MSD-Mixer [66] further incorporate elaborate inductive biases into the MLP structures and have shown improved performance. Note that the channel-independence design in DLinear [60], TiDE [12] and PatchTST [36] makes the model a global univariate one. While the channel mixing in [7, 62] implicitly models the multivariate relationships between series.

However, investigations into the short-term spatiotemporal data forecasting are still lacking. The studied STDF differs from LTSF in two ways: (1) complex temporal patterns (e.g., nonstationary) with higher resolution; (2) subtle interaction between multivariate series. Therefore, we conjecture that the emerging Transformers-like or MLP-based models designed for LTSF may be less effective in the context of spatiotemporal data, as the above methods basically adopt a channel-independence assumption and ignore the explicit spatial correlations between series [12, 60]. To adapt to the spatiotemporal case, several pioneering works have attempted to simplify the neural forecasting architectures in different aspects. In particular, Cini et al. [9] proposed a scalable graph predictor based on the random-walk diffusion and the echo-state network to encode spatiotemporal representations prior to model training. Liu et al. [31] developed two alternative spatial techniques including a preprocessing-based ego graph method and a global sensor embedding to model spatial correlations. The processed spatial features are further fed to temporal models such as RNNs, TCNs, and WaveNets. A graph sampling strategy is required to improve training performance. However, both approaches rely on complex temporal encoders and precomputed graph features, which can cause high complexity in engineering applications. Zhang et al. [63] and Wang et al. [49] extend MLP-Mixers in spatiotemporal traffic forecasting problems and demonstrate their effectiveness and efficiency. These findings prompt us to further explore the MLP-Mixer paradigm.

Node Embedding for Spatiotemporal Data. Node embedding or positional encoding in the graph machine learning domain is a commonly used technique to enhance the expressive power of GNNs [15]. However, methodologies and applications in spatiotemporal graph modeling have been relatively limited. In this line of research, Deng et al. [13] identify the bottleneck of deep forecasting models as ineffective in distinguishing high-frequency components from input signals. A spatiotemporal normalization scheme is proposed to enhance the WaveNet models. Shao et al. [43] discuss the spatiotemporal identification problem and propose to assign a learnable embedding for each node and time step in a MLP structure. In contrast to the focus of these efforts, we focus on the spatiotemporal contextualization problems posed by the mixing operation in the canonical MLP-Mixer model rather than the data itself. Furthermore, GatedGN model proposed in [41] follows a time-then-graph template and considers the node index as a static identifier. Global attention is used to infer the latent graphs. However, all pairwise attention with time complexity $O(N^2)$ is computationally expensive. More recently, Cini et al. [11] interpreted the role of node embedding as local effects in global models and provided a systematic framework for incorporating it into different architectures. Several regularization and transfer learning strategies are also discussed.

Building upon existing work, our work lies at the crossroads of these two research trajectories. We leverage these advances and extend the MLP-Mixer architecture for large-scale urban data. We also showcase its practical applications and deployments.