Product Graph U-networks using Temporal pooling for Spatiotemporal imputation

Aman Atman
Santosh Nannuru
aman.atman@research.iiit.ac.in
santosh.nannuru@iiit.ac.in
IIIT Hyderabad
Hyderabad, Telangana, India

ABSTRACT

Missing values in time-series is a frequently occurring problem in multi-sensor data analysis. The current state-of-the-art imputation networks don't generalize well across the benchmarks. Further, some directly operate on space-time product graphs which may minimize the information loss but are prohibitively expensive. We propose Product Graph U-networks using Temporal pooling for Spatiotemporal imputation (P-GUTS). It uses pooled representations of the space-time sequence in the bottleneck part of the U-network and is memory efficient. Since datasets can have differing levels of redundancy, relying on a single view (pooled representation) of the data can be limiting. Therefore, P-GUTS uses multiple U-Nets with different pooling factors and aggregates the representations. We demonstrate the effectiveness of P-GUTS on multiple real-world imputation datasets and robustness across increasing missing rates. P-GUTS also gives encouraging results in spatiotemporal forecasting.

CCS CONCEPTS

• **Information systems** → **Sensor networks**; • **Computing methodologies** → **Temporal reasoning**.

KEYWORDS

Time series, U-networks, Attention

ACM Reference Format:
Aman Atman and Santosh Nannuru. 2018. Product Graph U-networks using Temporal pooling for Spatiotemporal imputation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Multi-variate time streams consist of multiple time series correlated to each other. For example, traffic speed across the roads in a city can be linked. Accident on one road may affect the speeds on nearby roads in the future. Similarly, the particulate matter in the air can show complex spatio-temporal dynamics. It may be possible to predict, with high accuracy, the values at any space-time coordinate if neural networks can learn the patterns in the data. In addition to unknown future values, intermediate missing values are common in time-series datasets. The sensors may malfunction or the network may fail momentarily. Time series imputation is a vital pre-processing step for forecasting, anomaly detection and other downstream tasks.

Attention [27] has been successfully used to design effective networks for various time series tasks – imputation [18, 24], forecasting [12, 36] and anomaly detection [10, 25]. Being non-recurrent it allows the missing points to attend over the entire series, preventing compounding errors, and can be parallelized. But, only using positional encodings can cause loss of temporal information as they only maintain relative ordering information [35]. Therefore, time-series attention networks explicitly include temporal inductive biases (like day, week embeddings) to encode seasonality information. Similarly, spatial node embeddings can be used. These may be either simple learnable vectors [18] or further use additional graph inductive biases [17].

SPIN [18] is one of the state-of-the-art imputation networks using graph attention applied on the spatiotemporal product graph. SPIN-H is the smaller variant of SPIN where the temporal axis is pooled before message passing. SPIN-H does pooling by doing message passing on the bipartite graph between original and the reduced time dimensions. Thus, SPIN and SPIN-H operate at different temporal resolutions and are observed to be effective on separate datasets[18]. This motivates using an architecture that can view data at multiple resolutions making it generalize across benchmarks. Previous works in computer vision [6, 16, 33] also have demonstrated the advantage of multi-scale features. We use the terms down-sampling and pooling interchangeably.

Inspired by image in-painting or pixel-wise prediction, we formulate imputation as a node regression problem on space-time product graph. U-Networks [23] have been successfully been used in pixel-wise prediction and graph U-Net [11] for node classification.

We present a novel imputation network – Product Graph U-networks using Temporal pooling for Spatiotemporal imputation (**P-GUTS**). Graph U-Net can't directly operate on space-time vertices, where explicit edges are not always available. Therefore, instead of using Graph Convolutional Networks (GCN) [13], we use space-time graph attention based U-Networks. By using multiple U-Networks each with a different pooling factor we leverage multiple views of the data. We use the initial space-time positional

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

embeddings procedure from SPIN to enable a direct comparison of the imputation networks. P-GUTS achieves state-of-the-art results in spatio-temporal imputation across various benchmarks. It is also robust against high missing rates.

2 RELATED WORK

Missing values plague multi-sensor data analysis [22, 30]. Statistical methods for multi-variate time series imputation using interpolation [1, 20, 31], or matrix factorization [19, 34] have significantly higher error compared to the modern methods.

Auto-regressive methods like ARIMA [4], and even RNN-based networks including BRITS [5], and graph based recurrent network GRIN [7], suffer from compounding error problem [24]. As predictions rely on the previous outputs, errors can propagate and accumulate. Sequential nature makes them slow and impractical for long term imputation and forecasting.

Several state-of-the-art networks have leveraged attention [27] for learning better representations. SPIN [18] uses additive attention [3] on spatiotemporal graph with excellent results, at the expense of high memory requirements. SPIN-H, its hierarchical variant, is more efficient but does not perform as well generally. Trafformer [12] is also memory intensive, using multiplicative attention [27] directly on the space-time product graph for traffic forecasting. NRTSI [24] uses transformer encoder for stochastic time series performing hierarchical imputation.

MCNN [8] extracts features at different scales and frequencies using convolutional layers, leading to superior feature representation for time series classification. MICN [28] also leverages multi-branch network combining local features and global correlations for forecasting. MTST [36] achieves state-of-the-art results in multivariate forecasting using multiple resolution features. ST-UNet [32] combines spatial graph convolutions with dilated recurrent skip-connections, having separate operations across space and time. It is most related to our proposed P-GUTS, where we use pure attention backbone instead of GRU and operate on the space-time product graph for joint reasoning. Instead of a single U-net, P-GUTS uses multiple U-net branches to capture comprehensive patterns across fine to more coarsely pooled data.

Diffusion models have also been used for time series imputation – CSDI [26], and the recent SSSD [2] and MIDM [29] have iterative noising and denoising phases to fill the missing values.

3 PROBLEM SETUP

We now describe the multi-variate imputation problem. Let $Y \in \mathbb{R}^{N \times T}$ denote the spatiotemporal series, without any missing values. Here N denotes the number of spatial nodes and T the number of temporal nodes. $M \in \{0,1\}^{N \times T}$ is a binary mask of the same shape, denoting element wise missing (0) or present (1) values. Let the input multivariate time-series be $X \in \mathbb{R}^{N \times T}$ where $X \odot M = Y \odot M$. We want to learn a network f_θ which minimizes the following objective $\text{MAE}(Y, \hat{Y} = f_\theta(X, M))$ where MAE is the mean absolute error,

$$\text{MAE} = \frac{1}{NT} \sum_{i=1}^{NT} |y_i - \hat{y}_i| \tag{1}$$

and y is the flattened Y.

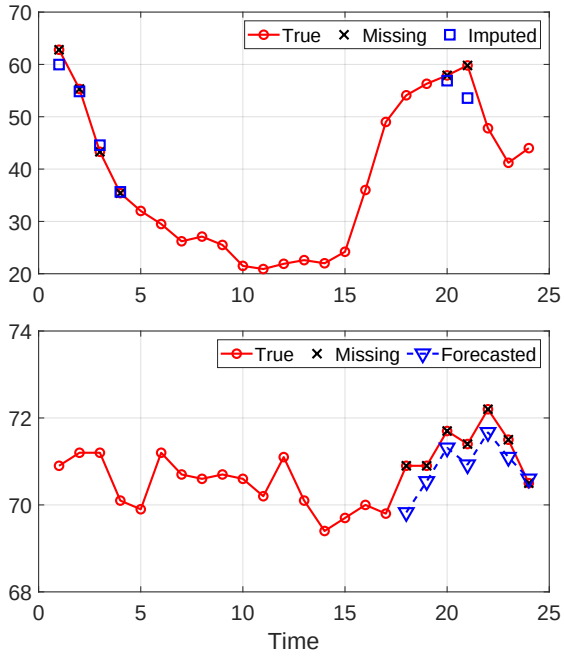

Figure 1: Imputation and forecasting outputs from a typical imputation model. Forecasting can be considered as imputation on future values.

Forecasting is the task of predicting future F values in a time series based on patterns and trends in historical data. As seen in Figure 1 we can also frame it as special-case of imputation where the the last F columns out of M are missing.

Directly processing the entire time-series ($T \sim 10,000$) can become intractable for many of the real word datasets. Therefore, we divide it into small batches (~ 10) as is conventionally done by other imputation and forecasting models.

4 ARCHITECTURE

In this section we provide an overview of the P-GUTS architecture. A realization with L layers is shown in Figure 2 with S_l indicating the lth P-GUTS processing block. Output of each layer acts as input to the next layer.

4.1 Network Overview

We perform an initial embedding (of dimension c) of the multivariate time-series $X \in \mathbb{R}^{N \times T}$ and missing mask $M \in \mathbb{R}^{N \times T}$. The initial representations $H^{(0)} \in \mathbb{R}^{N \times T \times c}$ are obtained using temporal positional encodings, learnable node embeddings and embedded week and day information as described in the introduction [18]. We use multiple hidden layers to increase model complexity, obtaining $H^{(l)} \in \mathbb{R}^{N \times T \times c}$ for each layer l. A deeper network further necessitates residual connections to prevent vanishing gradients and enable faster training. Thus the inputs X and M are made available at each layer. The representation from the final layer, $H^{(L)}$, is transformed using a multi-layer perceptron (MLP) $\text{MLP}_1 : \mathbb{R}^c \to \mathbb{R}^1$ to obtain the required space-time point wise imputations $\hat{Y} \in \mathbb{R}^{N \times T}$.

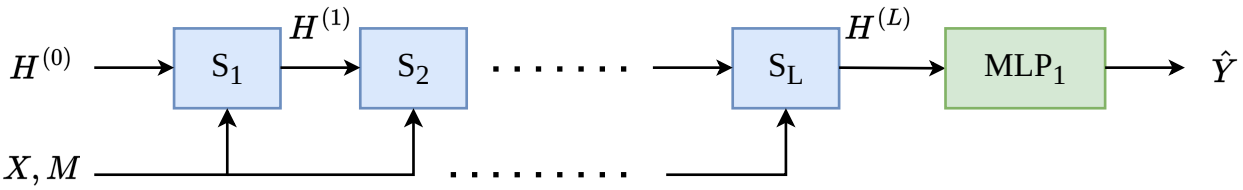

Figure 2: Taking time series X and mask M as input, the network obtains imputations $\hat{Y}$ by passing through several P-GUTS blocks with residual connections.

4.2 P-GUTS block

The detailed architecture of an individual P-GUTS block is given in Figure 3. It consists of three parts – data embedding, U-Net branches and, representation aggregation. Each P-GUTS block learns the embedded representation $E^{(l)}$ from the inputs $\{H^{(l)}, X, M\}$ which is transmitted through multiple U-Network branches to obtain distinct pooled representations $\{R_i^{(l)}\}_{i=1}^{B}$ which are aggregated and transformed using an MLP to give the output $H^{(l+1)}$.

While embedding inputs residual connections are used to obtain intermediate representation $E^{(l)} \in \mathbb{R}^{N\times T\times c}$,

$$E^{(l)} = \text{MLP}_2(X) \times M + H^{(l-1)} \tag{2}$$

where $\text{MLP}_2 : \mathbb{R}^1 \to \mathbb{R}^c$ is used to transform X into a tensor. We only consider the input X for the known space-time coordinates by masking out the missing positions using mask M.

The intermediate representation $E^{(l)}$ is processed by a bank of U-Networks. Each U-Net branch observes $E^{(l)}$ and produces pooled representation after reducing the space and/or time dimension by a given pooling factor. Corresponding to each factor and chosen axis space or time, the network learns specific patterns $R_i^{(l)} \in \mathbb{R}^{N\times T\times c}$. Each $R_i^{(l)}$ corresponds to a different view of the input features. By having multiple views, we expect to achieve accurate imputations as the network can leverage complementary information sources.

We aggregate the various representations $R_i^{(l)}$ by directly concatenating along the hidden dimension to obtain a tensor of size $\mathbb{R}^{N\times T\times Bc}$. This tensor is then transformed using $\text{MLP}_3 : \mathbb{R}^{Bc} \to \mathbb{R}^c$ to obtain $H^{(l+1)}$.

Further, an MLP (similar to $\text{MLP}_1 : \mathbb{R}^c \to \mathbb{R}^1$) is applied to each layer (not shown in the figure) resulting in layer-wise space-time imputations. During training we aggregate all the losses – each layer is learning to de-noise and refine the outputs from the previous layer. Layer wise losses can be useful for regularization, as real-world time-series are notoriously noisy and over-fitting is common. While testing we only consider the last layer's outputs.

4.3 U-Network branch

The effectiveness of U-Networks for pixel-wise predictions [23] motivates us to use them for node-wise prediction in space-time graph. We emphasize three main advantages of U-Networks in our implementation: **(i)** They alleviate the memory requirements by giving pooled latent representations. For instance, the direct space-time product graph becomes huge for most of the real-world datasets. By reducing its size, we can use attention on the resulting space-time graph without significant memory overhead. **(ii)** Further, as the output from the U-Network is of the same shape as input, the outputs across all the B branches have the same shape despite the different pooling factors. **(iii)** Pooling in the U-Net can be performed along time as well as spatial axis.

The common architecture for the individual branches U_i is presented in Figure 4. We modify graph U-Networks [11] leveraging attention for message passing. We enrich the inputs by adding learnable positional mask embeddings having shape $\mathbb{R}^c$. These are conditioned on whether the space-time point is absent or present, resulting in $M_i^{(l)} \in \mathbb{R}^{N\times T\times c}$ which is added to $E^{(l)}$ from the data embedding stage.

TE denotes the transformer encoder using typical multi-head attention from [27] – TE_0, TE_1, TE_2 are transformer encoders applied along the temporal sequence, and Space-Time TE acts along the flattened space-time sequence. TE architecture consists of single layer of the transformer encoder with specific configuration described in the Simulation Setup section. We initially apply attention along the time axis using TE_0 (moving the space to the batch axis). Let the output of this step be denoted $F^{(l)}$. Though $F^{(l)}$ has a branch dependence, we are dropping the suffix i for brevity.

4.3.1 Down-sampling. The following down-sampling process is used to select the top $k = \frac{T}{f_i}$ indices from temporal (or $\frac{N}{f_i}$ spatial) dimension, where f_i is the pooling factor. Each branch i, has learnable embedding $p_i^{(l)} \in \mathbb{R}^c$ for selecting these indices. It converts each spatiotemporal point in $F^{(l)}$ to a scalar,

$$F^{(l)} \cdot p_i^{(l)} = F_p^{(l)} \in \mathbb{R}^{N\times T}. \tag{3}$$

We now sort $F_p^{(l)}$ along the temporal axis for each spatial point giving the sorted indices,

$$I = \arg\text{sort}(F_p^{(l)}) \in \mathbb{Z}^{N\times T}. \tag{4}$$

The indices I are flattened along the spatial axis – first N values are the first ranked temporal nodes for each spatial node, then the second, and so on. The first k unique indices from the flattened I give us the temporally pooled indices t_{idx}^k.

After applying sigmoid activation, $F_p^{(l)}$ is used as a scaling attention matrix across the hidden dimension. This explicitly includes $p_i^{(l)}$ in the output computation making it differentiable and learnable. LayerNorm [14] is applied to stabilize the training. Finally, we feature-encoded these intermediate representations O_i by using a

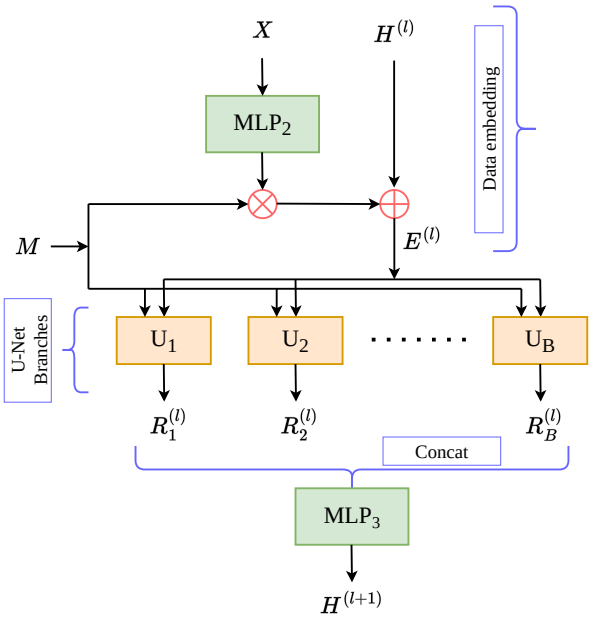

Figure 3: P-GUTS block. Space-time data embeddings from input time series X and mask M are combined with the previous block output $H^{(l)}$. These are then passed through multiple U-network branches (U_i) each with distinct pooling factors followed by aggregation to obtain the output $H^{(l+1)}$.

transformer encoder (TE_1) along the time axis to obtain $P_i^{(l)}$ as,

$$O_i^{(l)} = \text{LayerNorm}\Big(E^{(l)}[:, t_{\text{idx}}^k] \odot \Big(\sigma(F_p^{(l)}[:, t_{\text{idx}}^k]) \otimes \mathbf{1}_c\Big)\Big)$$
$$P_i^{(l)} = \text{TE}_1(O_i, \text{axis} = \text{time}). \quad (5)$$

Here $\otimes$ indicates Kronecker product. The final output of the down-sampling process is the pooled representation $P_i^{(l)} \in \mathbb{R}^{N\times k\times c}$ (or $P_i^{(l)} \in \mathbb{R}^{k\times T\times c}$ if spatial pooling is done instead of temporal pooling).

4.3.2 Space-time Bottleneck. To avoid missing complex spatiotemporal patterns, a product graph (PG) with Nk nodes (or kT) is considered. The space and time dimensions are flattened into a single dimension. A transformer encoder – TE (Space-Time) is applied on this. We reshape the tensor back into separate space and time axis resulting in $B_i^{(l)} \in \mathbb{R}^{N\times k\times c}$ as the output of this step.

4.3.3 Up-sampling. We initialize the up-sampling step tensor $Q_i^{(l)}$ with $E^{(l)}$. We use t_{idx}^k to scatter the pooled values back to their initial positions,

$$Q_i^{(l)}[:, t_{\text{idx}}^k] = Q_i^{(l)}[:, t_{\text{idx}}^k] + B_i^{(l)}. \quad (6)$$

We finally apply another transformer encoder TE_2 to attend over the temporal axis to obtain $R_i^{(l)}$. We have described the pooling across temporal axis but a similar process can be applied along the spatial axis.

5 EXPERIMENTS

5.1 Simulation Setup

We use RTX 2080 Ti with 11 GB memory for training the models. We implement in PyTorch [21] using open-source code from the baselines wherever available. We report performance statistics on 3 random initializations. While P-GUTS can also perform spatial pooling, we focus on temporal pooling across the experiments. We denote the pooling factors in brackets, for e.g. **P-GUTS**(3, 6, 12) means we are using *temporal* pooling factors of 3, 6 and 12.

5.2 Benchmarks

We evaluate on the benchmarks presented in Table 1 as frequently considered in the literature [7, 18, 29]. The evaluation mask is created by artificially adding missing values. We use real-world datasets for evaluation – AQI, AQI36 [37] which consist of air quality measurements recorded over cities in China; PEMS BAY, METR LA [15] which record traffic speed using road sensors in the Bay Area and Los Angeles respectively. The spatial graphs are included with the dataset. These are constructed from the inter-node distance matrix applying a thresolded gaussain kernel.

The missing values for evaluation are not uniformly distributed, but dependent on the datasets similar to [18]. For air quality datasets having high true missing percentage, evaluation mask is generated following the distribution of naturally missing values. For traffic datasets, we use BLOCK mask which introduces contiguous time steps of missing values, simulating sensor failure for consecutive timestamps. Duration of consecutive missing timestamp failures are uniformly sampled, $S \sim \mathcal{U}(12, 48)$, with failure probability $p_f = 0.15$. Additionally, 5% of the values are dropped uniformly.

We use recent state-of-the-art (SOTA) imputation models SPIN, SPIN-H [18] and GRIN [7], CSDI [26], SSSD [2], and MIDM [29] as baselines, and report their publicly available results. Mean Absolute Error (MAE) is the evaluation metric for imputation.

5.3 Configuration

Across all benchmarks, we use the training configuration and common hyperparameters from the transformer baseline in [18] to skip the exhaustive hyperparameter search. We use 8 as batch size, 8×10^{-4} as learning rate, 5-layers of P-GUTS blocks, cosine scheduler with restarts, Adam optimizer, 5.0 as gradient clipping value, 300 as number of epochs, 64 as the hidden dimension c, 2 layers in each MLP, with each transformer encoder having 4 heads and 40 epochs as patience. The imputation is done in sliding windows of size 24 usually, but for AIR36 it is 36. Window stride is 1.

We have several choices for pooling factors, which can go from 1 to the size of time window. For e.g in traffic datasets, resulting in dimension $\frac{24}{1}$ to $\frac{24}{24}$. In Figure 5, we select the best initial pooling factor f^* from a wide range of magnitudes. We start with $f \geq 3$ due to memory limitations. For progressively adding the remaining factors, we select $2f^*$ and $4f^*$ to cover a broad range.

5.4 Results

In this section, we provide empirical evidence for the usefulness of P-GUTS on benchmarks described previously.

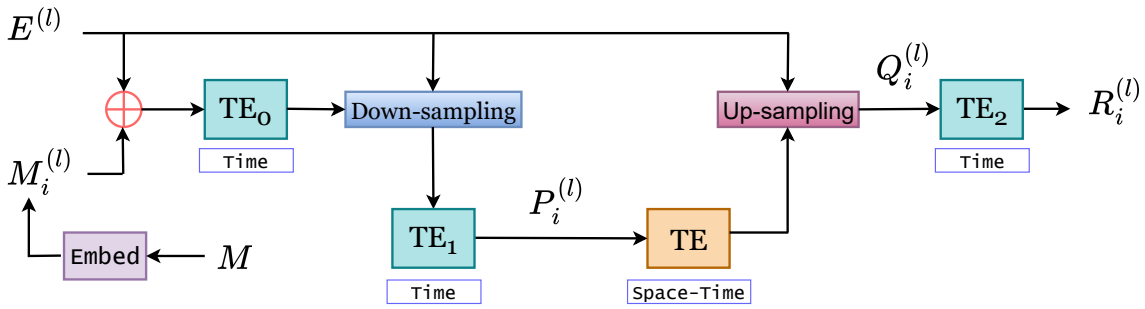

Figure 4: U-network branch within a P-GUTS block. Across every layer l, each branch i consists of attention U-network which processes embedded space-time series E and mask M using transformer encoder (TE) to obtain point wise space-time outputs. Multiple transformer encoders operating along time as well as space-time are used.

Table 1: Statistics of Spatio-Temporal Imputation Benchmarks

	Nodes	Edges	Freq.	Timesteps	Window	True Miss. (%)	Proc. Miss. (%)
LA BLOCK	207	1515	5 min	34272	24	8.11	16.52
BAY BLOCK	325	2369	5 min	52128	24	0.02	9.10
AQI	437	5398	1 hour	8760	24	25.67	36.34
AQI36	36	654	1 hour	8760	36	13.25	24.58

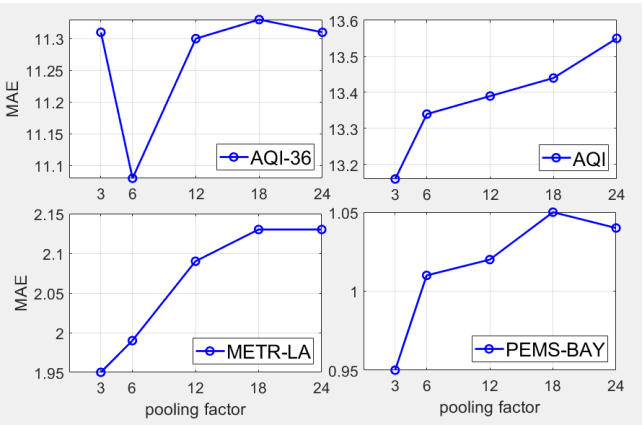

Figure 5: MAE versus the pooling factor f used in P-GUTS.

Table 2: Performance (in terms of MAE)

	Block missing		Simulated failures	
	PEMS-BAY	METR-LA	AQI-36	AQI
GRIN	1.14 ± 0.01	2.03 ± 0.00	12.08 ± 0.47	14.73 ± 0.15
SPIN	1.06 ± 0.02	1.98 ± 0.01	11.77 ± 0.54	13.92 ± 0.15
SPIN-H	1.05 ± 0.01	2.05 ± 0.02	10.89 ± 0.27	14.41 ± 0.13
P-GUTS (f^*)	0.95 ± 0.00	1.95 ± 0.01	11.08 ± 0.20	13.16 ± 0.18
P-GUTS ($f^*, 2f^*$)	**0.93 ± 0.01**	**1.92 ± 0.01**	**10.84 ± 0.09**	**13.12 ± 0.07**
P-GUTS ($f^*, 2f^*, 4f^*$)	**0.93 ± 0.01**	1.95 ± 0.04	10.94 ± 0.03	13.25 ± 0.07

5.4.1 Imputation. From Figure 5, we find $f^* = 6$ for AQI36, and $f^* = 3$ for the remaining datasets. In Table 2 we compare P-GUTS against the baselines across the common imputation benchmarks. P-GUTS is performing significantly better than the baselines. Interestingly, we observe the number of pooling factors has a nonlinear relation to network performance. While using multiple pooling factors can indeed be better than both individual factors (f^* and $2f^*$), the network performance degrades after further adding $4f^*$. Each additional branch we add leads to more parameters which can cause overfitting especially for time-series[35]. While we report the publicly available results of baselines, we also obtained comparable results running them on our machines. P-GUTS requires lesser memory, making it possible to train on BAY BLOCK for which SPIN gives out-of-memory (OOM) error even for batch size of 1 on our device.

5.4.2 Robustness. In Table 3 the models are evaluated on their robustness to increasing missing rates. The Block missing mask is generated by simulating sensor failure where contiguous timesteps are absent. The failure probability p_f of 5%, 10% and 15% corresponds to an overall missing rate ranging between ≈ 70-75%, ≈ 90-92%, and ≈ 96-97% respectively. We chose (3, 6) as it performs well across the considered imputation datasets. P-GUTS gives better performance generally, and by a significant margin particularly for the AIR dataset. P-GUTS has higher error for the very high missing rate scenario. We suspect that this may be due to redundant pooled representations obtained from extremely sparse input. We further investigate this by comparing against single factor variants in Table 4. We observe that a single factor performs better for these datasets than multiple factors under extremely high missing rates.

Table 3: Performance (MAE) with an increasing number of simulated failures in the *Block missing* setting.

	METR-LA			PEMS-BAY			AQI		
	Failure probability			Failure probability			Failure probability		
	5 %	10 %	15 %	5 %	10 %	15 %	5 %	10 %	15 %
GRIN	3.05 ± 0.02	4.52 ± 0.05	5.82 ± 0.06	2.26 ± 0.03	3.45 ± 0.06	4.35 ± 0.04	15.62 ± 0.24	22.08 ± 0.39	29.03 ± 0.42
SPIN	2.71 ± 0.02	3.32 ± 0.02	3.87 ± 0.05	1.78 ± 0.03	**2.15 ± 0.03**	**2.41 ± 0.02**	14.29 ± 0.24	18.71 ± 0.34	24.34 ± 0.46
SPIN-H	2.64 ± 0.02	3.17 ± 0.02	**3.61 ± 0.04**	1.75 ± 0.04	**2.16 ± 0.03**	2.48 ± 0.02	14.55 ± 0.26	19.37 ± 0.36	25.38 ± 0.37
P-GUTS (3, 6)	**2.53 ± 0.02**	**3.07 ± 0.03**	3.80 ± 0.06	**1.69 ± 0.03**	2.20 ± 0.03	2.86 ± 0.03	**13.15 ± 0.13**	**15.85 ± 0.17**	**19.92 ± 0.31**

Table 4: Performance (MAE) under extreme missing *Block missing* setting (15%. failure probability)

	METR-LA	PEMS-BAY
[3, 6]	3.80 ± 0.06	2.86 ± 0.03
[3]	3.68 ± 0.05	2.63 ± 0.03
[6]	3.71 ± 0.06	2.76 ± 0.03

Table 5: Performance on PEMS-BAY for 1 hour Forecasting.

	GMAN	MIDM	P-GUTS(3, 4, 6)
MAE	1.86 ± 0.02	1.83 ± 0.04	**1.57 ± 0.01**
MRE (%)	4.31 ± 0.02	4.21 ± 0.06	**2.51 ± 0.01**

5.4.3 Forecasting. We use the spatiotemporal forecasting benchmark given in [29] to evaluate the forecasting ability of P-GUTS in Table 5. It considers the PEMS-BAY dataset with 1 hour forecasting horizon and 1 hour look-back window. We observe a significant improvement both in MAE and MRE. Therefore, we can directly use P-GUTS for forecasting in addition to imputation.

5.5 Ablations

Table 6: Ablation study: Performance (MAE) of P-GUTS(f^*) and modified versions of itself.

Model	AQI 36	METR LA
P-GUTS	11.08 ± 0.2	1.95 ± 0.01
P-GUTS w/o Space-Time TE	14.92 ± 0.19	2.68 ± 0.01
P-GUTS w/o TE_0	11.26 ± 0.27	2.00 ± 0.02
P-GUTS w/o TE_1	11.37 ± 0.15	1.97 ± 0.01
P-GUTS w/o TE_2	10.87 ± 0.10	1.98 ± 0.01

In Table 6 we evaluate the effectiveness of components of P-GUTS. We remove components one at a time and evaluate the performance of the resulting model. We use pooling factor $f^* = 6$ for AQI36 and $f^* = 3$ for METR LA with 3 independent initializations.

(1) **Space-Time TE** is an integral part of P-GUTS using attention on the space-time product graph. Removing it deteriorates the performance significantly.
(2) **Time TE** additionally improves the network performance, particularly TE_0 and TE_1. We notice improvement for the AQI36 dataset after removing TE_2. AQI36 is a smaller dataset which may make it easier for complex networks to overfit (smaller SPIN-H outperforms SPIN for this case). We further evaluate the need for TE_2 when we have multiple pooling factors.
(3) In Table 7, we evaluate the performance of best performing P-GUTS, i.e using factors 3, 6 for METR LA and 6, 12 for AQI36. We now notice that TE_2 becomes more relevant with multiple factors, the performance gap reduces for AQI36, and improves for METR LA as we add this component.

Table 7: Ablation study: P-GUTS is compared with modified versions of itself when using multiple pooling factors.

Model	AQI 36	METR LA
P-GUTS	10.84 ± 0.09	1.92 ± 0.01
P-GUTS w/o TE_2	10.78 ± 0.05	1.97 ± 0.01

6 CONCLUSION

P-GUTS outperforms the state-of-the-art baselines for spatiotemporal imputation, showing improvements on several real-world benchmarks and robustness against increasing missing rates. We also observed that using multiple representations is useful – if we use more pooling factors, **P-GUTS** performs better. It is flexible in terms of memory requirements, as larger pooling factors can reduce the memory required. Moreover, the architecture allows for pooling along any axis – space, time, or both. We have evaluated the network on a limited range of pooling factors as exhaustive search can get intractable. Similarly to a deep mixture of experts [9], future work can explore directly identifying the pooling factors in a data-driven manner eliminating the expensive hyperparameter search.

REFERENCES

[1] Edgar Acuña and Caroline Rodriguez. 2004. The Treatment of Missing Values and its Effect on Classifier Accuracy. In *Classification, Clustering, and Data Mining Applications*, David Banks, Frederick R. McMorris, Phipps Arabie, and Wolfgang Gaul (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 639–647.
[2] Juan Miguel Lopez Alcaraz and Nils Strodthoff. 2023. Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models. arXiv:2208.09399 [cs.LG]

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473 [cs.CL] https://arxiv.org/abs/1409.0473

[4] Faraj Bashir and Hua-Liang Wei. 2018. Handling missing data in multivariate time series using a vector autoregressive model-imputation (VAR-IM) algorithm. *Neurocomputing* 276 (2018), 23–30. https://doi.org/10.1016/j.neucom.2017.03.097 Machine Learning and Data Mining Techniques for Medical Complex Data Analysis.

[5] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. 2018. Brits: Bidirectional recurrent imputation for time series. *Advances in neural information processing systems* 31 (2018).

[6] Liang-Chieh Chen. 2017. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587* (2017).

[7] Andrea Cini, Ivan Marisca, and Cesare Alippi. 2022. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=kOu3-S3wJ7

[8] Zhicheng Cui, Wenlin Chen, and Yixin Chen. 2016. Multi-Scale Convolutional Neural Networks for Time Series Classification. arXiv:1603.06995 [cs.CV] https://arxiv.org/abs/1603.06995

[9] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. arXiv:2101.03961 [cs.LG] https://arxiv.org/abs/2101.03961

[10] Yuye Feng, Wei Zhang, Haiming Sun, and Weihao Jiang. 2024. Spatial-Temporal Transformer with Error-Restricted Variance Estimation for Time Series Anomaly Detection. In *Advances in Knowledge Discovery and Data Mining*, De-Nian Yang, Xing Xie, Vincent S. Tseng, Jian Pei, Jen-Wei Huang, and Jerry Chun-Wei Lin (Eds.). Springer Nature Singapore, Singapore, 3–14.

[11] Hongyang Gao and Shuiwang Ji. 2019. Graph u-nets. In *International Conference on Machine Learning*. PMLR, 2083–2092.

[12] Di Jin, Jiayi Shi, Rui Wang, Yawen Li, Yuxiao Huang, and Yu-Bin Yang. 2023. Trafformer: Unify Time and Space in Traffic Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 7 (June 2023), 8114–8122. https://doi.org/10.1609/aaai.v37i7.25980

[13] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *ArXiv e-prints* (2016), arXiv–1607.

[15] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *International Conference on Learning Representations*.

[16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2117–2125.

[17] Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet K. Dokania, Mark Coates, Philip Torr, and Ser-Nam Lim. 2023. Graph Inductive Biases in Transformers without Message Passing. arXiv:2305.17589 [cs.LG] https://arxiv.org/abs/2305.17589

[18] Ivan Marisca, Andrea Cini, and Cesare Alippi. 2022. Learning to reconstruct missing data from spatiotemporal graphs with sparse observations. *Advances in Neural Information Processing Systems* 35 (2022), 32069–32082.

[19] Jiali Mei, Yohann De Castro, Yannig Goude, and Georges Hébrail. 2017. Nonnegative Matrix Factorization for Time Series Recovery From a Few Temporal Aggregates. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, 2382–2390. https://proceedings.mlr.press/v70/mei17a.html

[20] Stefan Oehmcke, Oliver Zielinski, and Oliver Kramer. 2016. kNN ensembles with penalized DTW for multivariate time series imputation. In *2016 International Joint Conference on Neural Networks (IJCNN)*. 2774–2781. https://doi.org/10.1109/IJCNN.2016.7727549

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *CoRR* abs/1912.01703 (2019). arXiv:1912.01703 http://arxiv.org/abs/1912.01703

[22] Jing Peng, Meiqi Yang, Qiong Zhang, and Xiaoxiao Li. 2025. S4M: S4 for multivariate time series forecasting with Missing values. arXiv:2503.00900 [cs.LG] https://arxiv.org/abs/2503.00900

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. arXiv:1505.04597 [cs.CV] https://arxiv.org/abs/1505.04597

[24] Siyuan Shan, Yang Li, and Junier B Oliva. 2023. NRTSI: Non-recurrent time series imputation. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1–5.

[25] Li Shen, Yuning Wei, Yangzhu Wang, Xuyi Fan, and Minghao Zhao. 2024. Siamese-like Time-series Forecasting with Prior Anomaly Detection and Inner Reconstruction. In *2024 International Joint Conference on Neural Networks (IJCNN)*. 1–9. https://doi.org/10.1109/IJCNN60899.2024.10650080

[26] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. 2021. CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation. arXiv:2107.03502 [cs.LG]

[27] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[28] Huiqiang Wang, Jian Peng, Feihu Huang, Jince Wang, Junhui Chen, and Yifei Xiao. 2023. Micn: Multi-scale local and global context modeling for long-term series forecasting. In *The eleventh international conference on learning representations*.

[29] Xu Wang, Hongbo Zhang, Pengkun Wang, Yudong Zhang, Binwu Wang, Zhengyang Zhou, and Yang Wang. 2023. An Observed Value Consistent Diffusion Model for Imputing Missing Values in Multivariate Time Series. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (<conf-loc>, <city>Long Beach</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 2409–2418. https://doi.org/10.1145/3580305.3599257

[30] Pengshuai Yao, Mengna Liu, Xu Cheng, Fan Shi, Huan Li, Xiufeng Liu, and Shengyong Chen. 2024. An End-to-End Model for Time Series Classification In the Presence of Missing Values. arXiv:2408.05849 [cs.LG] https://arxiv.org/abs/2408.05849

[31] Xiuwen Yi, Yu Zheng, Junbo Zhang, and Tianrui Li. 2016. ST-MVL: filling missing values in geo-sensory time series data. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (New York, New York, USA) *(IJCAI'16)*. AAAI Press, 2704–2710.

[32] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2019. St-unet: A spatio-temporal u-network for graph-structured time series modeling. *arXiv preprint arXiv:1903.05631* (2019).

[33] F Yu. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* (2015).

[34] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. 2016. Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/85422afb467e9456013a2a51d4dff702-Paper.pdf

[35] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are Transformers Effective for Time Series Forecasting? *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 9 (Jun. 2023), 11121–11128. https://doi.org/10.1609/aaai.v37i9.26317

[36] Yitian Zhang, Liheng Ma, Soumyasundar Pal, Yingxue Zhang, and Mark Coates. 2024. Multi-resolution Time-Series Transformer for Long-term Forecasting. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 4222–4230.

[37] Yu Zheng, Xiuwen Yi, Ming Li, Ruiyuan Li, Zhangqing Shan, Eric Chang, and Tianrui Li. 2015. Forecasting Fine-Grained Air Quality Based on Big Data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 2267–2276. https://doi.org/10.1145/2783258.2788573

7 REPRODUCIBILITY

Here we provide details regarding the datasets and code.

7.1 Data

The following publicly available datasets were utilized for our experiments.

- AirQuality
- MetrLA
- PemsBay

These datasets can be accessed and loaded directly using the torch-spatiotemporal library available here https://torch-spatiotemporal.readthedocs.io/en/latest/modules/datasets.html

7.2 Code

The source code for our proposed model, training and evaluation procedures is publicly available here. https://github.com/willtryagain/pguts

7.3 Steps to Reproduce Results

First, create the necessary Python environment by following the instructions in the `README.md` file located in the code repository: https://github.com/willtryagain/pguts. Once the environment is active, use the following steps to reproduce the key results:

(1) **Main Imputation Results (Figure 5, Table 2):** Execute the imputation script:

```
python -m experiments.run_imputation \
  --config imputation/<dataset>/pguts.yaml \
  --dataset-name <dataset>
```

Replace `<dataset>` with the target dataset name. To experiment with different temporal pooling factors, modify the `factor_t` list within the specified `pguts.yaml` configuration file (e.g., `factor_t: [3]` or `factor_t: [3, 6]`).

(2) **Forecasting Results (Table 5):** Run the forecasting script:

```
python -m experiments.run_imputation \
    --config forecasting/pguts.yaml \
    --dataset-name bay_fc
```

(3) **Ablation Studies (Analysis for Table 6):** The following modifications were made to the codebase or configuration files to evaluate individual components. After each modification, re-run the relevant imputation experiment as described in step 1.

- **To remove Space-Time TE:** In the relevant `.yaml` configuration file, set the parameter `use_pg_enc = False`.
- **To remove TE_0:** In the file `code/layers/pguts_branch.py`, comment out the (approximately line 242):
  `% h = self.enc_init_t(h) % Original line`
- **To remove TE_1 (Temporal Embeddings in Pooling):** In `code/layers/pguts_branch.py`, modify the pooling call (line 243) to skip,
  Replace:
  `h, time_pool_data_list = P(h, self.TP)`
  With:
  `h, time_pool_data_list = P(h, self.TP, skip_te=True)`
- **To remove TE_2 :** In `code/layers/pguts_branch.py`, modify the unpooling call (line 283) to skip: Replace:
  `temp = U(temp, 1, time_pool_data_list, self.TU)`

  With:

```
temp = U(temp, 1, time_pool_data_list, \
         self.TU, skip_te=True)
```

(4) **Results with Increasing Missing Data (Table 3):** Train the model using the imputation script as described in step 1 (`experiments.run_imputation`). To test with increasing percentages of missing data (e.g., 5%, 10%, 15%), modify the experiment configuration `config/inference.yaml`. Set the exp_name after checking the searching for the checkpoint directory found in log directory structured similarly to: `log/<dataset>/PGUTS/<exp_name>`. Execute the inference script:

```
python -m  experiments.run_inference  \
--config inference.yaml
```