Fully Automated Correlated Time Series Forecasting in Minutes

Xinle Wu[1], Xingjian Wu[2], Dalin Zhang[1], Miao Zhang[3], Chenjuan Guo[2], Bin Yang[2*], Christian S. Jensen[1]

[1]Aalborg University, Denmark [2]East China Normal University, China
[3]Harbin Institute of Technology, Shenzhen, China
[1]{xinlewu, dalinz, csj}@cs.aau.dk [2]{xjwu, cjguo, byang }@dase.ecnu.edu.cn [3]{zhangmiao@hit.edu.cn}

ABSTRACT

Societal and industrial infrastructures and systems increasingly leverage sensors that emit correlated time series. Forecasting of future values of such time series based on recorded historical values has important benefits. Automatically designed models achieve higher accuracy than manually designed models. Given a forecasting task, which includes a dataset and a forecasting horizon, automated design methods automatically search for an optimal forecasting model for the task in a manually designed search space, and then train the identified model using the dataset to enable the forecasting. Existing automated methods face three challenges. First, the search space is constructed by human experts, rending the methods only semi-automated and yielding search spaces prone to subjective biases. Second, it is time consuming to search for an optimal model. Third, training the identified model for a new task is also costly. These challenges limit the practicability of automated methods in real-world settings. To contend with the challenges, we propose a fully automated and highly efficient correlated time series forecasting framework where the search and training can be done in minutes. The framework includes a data-driven, iterative strategy to automatically prune a large search space to obtain a high-quality search space for a new forecasting task. It includes a zero-shot search strategy to efficiently identify the optimal model in the customized search space. And it includes a fast parameter adaptation strategy to accelerate the training of the identified model. Experiments on seven benchmark datasets offer evidence that the framework is capable of state-of-the-art accuracy and is much more efficient than existing methods.

PVLDB Reference Format:
Xinle Wu, Xingjian Wu, Dalin Zhang, Miao Zhang, Chenjuan Guo, Bin Yang, Christian S. Jensen. Fully Automated Correlated Time Series Forecasting in Minutes. PVLDB, 18(2): 144 - 157, 2024.
doi:10.14778/3705829.3705835
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ccloud0525/FACTS.

1 INTRODUCTION

Many important societal and industrial infrastructures, including intelligent transportation systems, power grids, patient monitoring systems, and industrial control systems [3, 4, 6, 24, 25, 35, 37, 39, 42, 54, 56, 57, 62], involve sensors that record values that vary over time, resulting in multiple time series that are often correlated, known as correlated time series (CTS). Forecasting future CTS values based on historical values has important applications [2, 17, 53, 60]. For example, forecasting the future electricity consumption of users in an area based on their historical electricity consumption can help balance supply and demand in a power grid.

Methods employing deep learning achieve state-of-the-art performance at CTS forecasting. Most of their model architectures are designed manually by human experts [1, 7, 10–13, 21, 23, 27, 36, 40, 45, 48, 49, 52, 53]. The core components of such models are Spatio-Temporal blocks (ST-blocks), which are constructed by Spatial/Temporal (S/T) operators, such as convolution, graph convolution, and transformer, and capture both temporal dependencies among historical values and spatial correlations across time series. Although achieving promising results, even human experts struggle to design optimal ST-blocks for new tasks.

As a more promising alternative approach, automated CTS forecasting aims to automatically identify optimal ST-blocks for different tasks and then uses them for forecasting [38, 50, 51, 55]. Figure 1 illustrates the pipeline of automated methods, which includes three phases: search space design, search for an optimal ST-block, and training the identified ST-block. A search space is first designed from S/T operators commonly used in existing manually designed models. These S/T operators are then combined using topological connection rules to obtain a set of ST-blocks that then form the search space. Next, search strategies, such as gradient-based, comparator-based, or random search, are applied to the search space to find an optimal ST-block for a given task. Finally, the identified ST-block is trained to enable the forecasting task. Despite achieving better performance than manually designed models, this approach still suffers from three major limitations that makes it challenging to use in practice.

(1) Manually designed search space. The search space is still designed manually, which may yield suboptimal performance and also violates the goal of AutoML, namely to automate the entire process [18, 28, 34, 43, 66]. Unlike search spaces constructed with a few homogeneous operators in computer vision and natural language processing, the many heterogeneous S/T operators and topological connections possible when forming ST-blocks yield a *general search space* that is difficult to explore.

The state-of-the-art is to prune manually the general search space into a smaller, easier-to-explore search space. However, the pruned search spaces may be suboptimal for unseen tasks because of the manual pruning that relies on heuristic rules and statistical results from a few seen tasks.

*: Corresponding author.

Proceedings of the VLDB Endowment, Vol. 18, No. 2 ISSN 2150-8097.
doi:10.14778/3705829.3705835

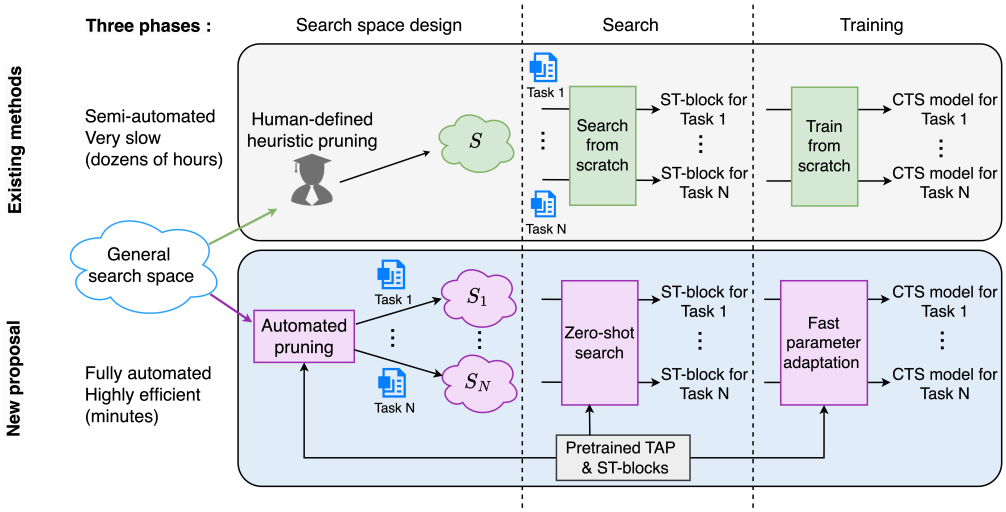


Figure 1: Existing automated methods v.s. the proposed method.

(2) High search cost. Existing automated methods employ search strategies such as gradient-based [30, 31, 50], comparator-based [5, 51], or random search [55] to explore a search space to find the optimal ST-block, which are all very time-consuming. Gradient-based methods train a supernet that represents the search space, which is much larger and harder to train than a single ST-block. Comparator-based methods train a large number of ST-blocks and obtain their validation accuracy to train a comparator. Random search methods sample a large number of ST-blocks and train them to obtain the optimal one. As a result, existing automated frameworks spending dozens of GPU hours searching for a high-performance ST-block. This high search cost makes these methods unattractive in real-world scenarios.

(3) High training cost. Having found a high-performance ST-block, existing methods train it from scratch on unseen CTS forecasting tasks. Thus usually takes hours [30, 31, 55], depending on the scale of the tasks.

To address the above limitations, we propose FACTS, a Fully Automated and highly efficient CTS forecasting framework. (1) We propose an **automated pruning strategy** to generate a high-quality search space for unseen forecasting tasks. Specifically, we construct a general search space using S/T operators and topological rules commonly used in existing CTS forecasting models. This search space is expected to contain optimal ST-blocks on unseen tasks. We then propose a strategy that partitions the general search space into disjoint subspaces that are distinguishable in terms of quality. Next, we iteratively prune the search space by removing low-quality subspaces in multiple passes on specific tasks, thereby generating customized high-quality search spaces for different unseen tasks, as shown in Figure 1 (lower left).

(2) We propose a **zero-shot search** strategy that can find the optimal ST-block for unseen CTS forecasting tasks in minutes. Specifically, we build a task-aware architecture predictor (TAP) that takes as input the architecture of an ST-block and the task feature of a task and then predicts the accuracy of the ST-block on the task. In the pretraining phase, we collect training samples from numerous and diverse CTS forecasting tasks to pretrain TAP, enabling it to predict the accuracy of ST-blocks on an unseen task without having to be trained on that task. In the zero-shot search phase, we first employ the pretrained TAP to assist search space pruning on the target task, then traverse the ST-blocks in the pruned search space and pick the one with the highest prediction accuracy for deployment. Figure 1 (lower middle) shows that FACTS searches for a high-performance ST-block for each unseen task. Since the search on unseen tasks does not involve model training, it can be completed in minutes, which addresses the second limitation.

(3) We propose a **fast parameter adaptation** strategy to accelerate the training of identified ST-blocks. Specifically, we introduce learnable coefficients to linearly combine the parameter weights of pretrained ST-blocks and use these as initial weights in the identified ST-block. This provides a good optimization starting point, allowing the training on the target task to complete quickly, thereby addressing the third limitation. Figure 1 (lower right) shows that FACTS inherits the parameter weights from pretrained ST-blocks to train identified ST-blocks, resulting in trained CTS models for forecasting.

Our contributions are summarized as the follows.

(1) We propose FACTS, a fully automated CTS forecasting framework. In particular, we propose an automated search space pruning strategy to automatically generate high-quality search spaces for unseen CTS forecasting tasks.
(2) We propose a zero-shot search strategy to search for an optimal ST-block on arbitrary unseen CTS forecasting tasks in minutes.
(3) We propose a fast parameter adaptation strategy to accelerate the training of identified ST-blocks, reducing the training time by up to 66% on unseen CTS forecasting tasks.
(4) Extensive experiments on seven benchmark datasets show that the proposed framework is capable of state-of-the-art forecast accuracy while taking less time than existing manual and automated methods.

2 PRELIMINARIES

2.1 Problem Setting

Correlated Time Series (CTS). A correlated time series (CTS) $\boldsymbol{\mathcal{X}}$ consists of N times series that each contains T timestamps and has an F-dimensional feature vector for each timestamp. Thus, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times T \times F}$. Time series are correlated when the values in each time series not only depend on its historical values but also depend on values in other time series. Correlations between time series can be captured using a graph $G = (V, E, A)$. where V is a set of vertices representing time series, E is a set of edges, representing the correlations between two time series, such correlations are often derived from the physical distances between the sensors that produce the time series, but they can also be learned adaptively. A graph G can be captured using an adjacency matrix A.

Correlated Time Series Forecasting. We consider multi-step and single-step CTS forecasting, both of which have important applications. Given the feature values of a CTS $\boldsymbol{\mathcal{X}}$ in the past P time steps, the goal of multi-step forecasting is to predict the feature values in Q ($Q > 1$) future time steps, formulated as follows.

$$\{\hat{\boldsymbol{X}}_{t+P+1}, \hat{\boldsymbol{X}}_{t+P+2}, ..., \hat{\boldsymbol{X}}_{t+P+Q}\} = \mathcal{F}(\boldsymbol{X}_{t+1}, \boldsymbol{X}_{t+2}, ..., \boldsymbol{X}_{t+P}; G), \quad (1)$$

where $\boldsymbol{X}_t \in \mathbb{R}^{N \times F}$ denotes the feature values of a CTS at time step t, $\hat{\boldsymbol{X}}$ denotes forecasted feature values, and $\mathcal{F}$ is a forecasting model. The goal of single-step forecasting is to forecast the feature values in the Q-th ($Q \geq 1$) future time step, formulated as follows.

$$\hat{\boldsymbol{X}}_{t+P+Q} = \mathcal{F}(\boldsymbol{X}_{t+1}, \boldsymbol{X}_{t+2}, \ldots, \boldsymbol{X}_{t+P}; G), \quad (2)$$

CTS Forecasting Task. We define a CTS forecasting task as $\mathcal{T} = (\mathcal{D}, P, Q, tag)$, where $\mathcal{D}$ represents a CTS dataset, P and Q represent the input and forecasting lengths, respectively, and tag indicates whether the CTS forecasting task is single-step or multi-step.

2.2 Existing Automated Forecasting Methods

Existing methods focus on the automated design of ST-blocks, which form the backbone of CTS forecasting models. Specifically, automated CTS forecasting includes three phases: search space design, search for an optimal ST-block, and training of the identified ST-block, as shown in Figure 1. First, a search space is designed, which is a set of ST-blocks composed of **S/T operators** and **topological connection rules** used to assemble the operators. Figure 2 shows an example of a search space and two ST-blocks in it, which feature different S/T operator combinations and topologies. Next, they employ search strategies such as gradient-based, comparator-based to explore the search space for the optimal ST-block. Finally, they train the identified ST-block for CTS forecasting.

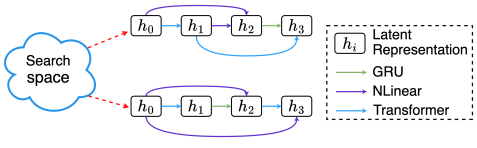


Figure 2: Example of a search space and ST-blocks.

2.2.1 Search Space Design. Existing automated CTS forecasting methods employ commonly used S/T-operators to build ST-blocks. We use O to denote the S/T-operator set. It includes 9 T-operators: 1D-CNN [44], LSTM [19], GRU [8], NLinear [61], GDCC [53], Inception [52], Transformer [47], Informer [64], and Convformer [16]. Further, it includes 5 S-operators: DGCN [53], Mix-hop [52], Spatial-Transformer [47], Spatial-Informer [64], Masked-Transformer [20]. Finally, it includes a skip operator. Thus, $|O| = 15$. These operators are used to design a **general search space** that is expected to contain optimal ST-blocks for arbitrary unseen CTS forecasting tasks, but is too large to explore to find the optimal ST-block. Existing automated methods then prune the general search space manually, in two ways: by reducing the S/T operator set [30, 31, 38]or by removing bad ST-blocks based on sampling [41, 55].

Methods in the first category [30, 38, 50] compare the accuracy and efficiency of individual operators and select a subset of high-performance operators from the full operator set O, thereby pruning the general search space. However, the pruned search space may be suboptimal. First, it compares the accuracy of S/T-operators on a few seen CTS forecasting tasks, which may not generalize to unseen tasks. For example, as shown in Table 1, NLinear performs better than a 1D-CNN on the ETTH dataset, but worse than the 1D-CNN on the PEMS04 and PEMS08 datasets. Therefore, it is unwise to remove the NLinear operator from the search space based on the results on PEMS04 and PEMS08, as it may exhibit high performance on other datasets. Second, S/T operators with high performance in isolation may not form high-performance ST-blocks. For example, as shown in Table 1, Convformer (ConvF) has higher accuracy than Informer (INF) on multiple datasets, but when we replace INF with ConvF in multiple ST-blocks, the accuracy decreases—see ST-block (INF) and ST-block (CovF) in Table 1 for an example, where ST-block (CovF) is a variant of ST-block (INF) with CovF replacing INF.

Methods in the second category [41, 55] adopt heuristic pruning strategies to divide architecture designs into disjoint angles and remove low-performance designs based on the accuracy of sampled ST-blocks, which is also performed on a few seen tasks and may not generalize to unseen tasks.

To sum up, existing methods prune the general search space based on **human designed heuristics** to generate a task-agnostic and suboptimal search space. Instead, FACTS **automatically** prunes the general search space for the target task to generate a task-aware search space, which is expected to offer a higher potential for finding high-performance ST-blocks for the specific target task.

Table 1: MAE of S/T operators.

	1D-CNN	NLinear	ConvF	InF	ST-block (INF)	ST-block (ConvF)
ETTh1	0.535	0.377	0.514	0.635	0.203	0.224
PEMS04	31.556	36.484	26.092	29.084	18.954	19.104
PEMS08	26.257	30.448	20.709	23.693	14.694	14.870

2.2.2 Search Strategy. Existing search strategies include gradient-based [30, 31, 38], comparator-based [5, 51], and random search [55]. The gradient-based search strategy models the search space as a supernet, which is time-consuming to train. The comparator-based search strategy trains a comparator to identify the better of two ST-blocks, which requires training a large number of ST-blocks

and is therefore also costly. The random search strategy samples a number of ST-blocks at random and trains them to obtain the one with the highest accuracy, thus having the highest search cost.

2.2.3 Model Training. Existing automated methods train an identified ST-block from scratch [14, 31, 38, 50, 51, 55], which typically takes hours, depending on the scale of the forecasting task.

3 METHODOLOGY

The proposed framework aims to find an ST-block that enables optimal prediction accuracy on unseen CTS forecasting tasks and to use it to obtain results in minutes. Figure 3 illustrates the framework. It includes an **automated pruning strategy** (Section 3.1) to prune the general search space to obtain a high-quality and relatively small search space suitable for unseen CTS forecasting tasks. Specifically, we first partition the general search space into disjoint subspaces, named *combs*. We define the quality of a *comb* to be a function of the performance of all ST-blocks in it and measure the quality using the error empirical distribution function (EDF) [41]. We then train a *comb* predictor to predict the EDF of *combs* and prune the search space by iteratively removing *combs* with low EDF.

To accelerate the search for optimal ST-blocks, the framework includes a highly efficient **zero-shot search strategy** (Section 3.2), which consists of pretraining and zero-shot search phases. Pretraining is a one-time task, and zero-shot search can be completed on unseen tasks in minutes. In the pretraining phase, we perform search space pruning on numerous and diverse CTS forecasting tasks, during which we collect training samples to pretrain a task-aware architecture predictor (TAP). In the zero-shot search phase, we first perform automated search space pruning on an unseen task, where we use the pseudo-EDF generated by the pretrained TAP to replace the real EDF. With the pruning completed, we use the pretrained TAP to predict the accuracy of all ST-blocks in the pruned search space on the unseen task and return the ST-block with the best predicted performance as the optimal ST-block.

Finally, we train the identified ST-block on the unseen CTS forecasting task to enable forecasting. To accelerate the training, we propose a **fast parameter adaptation strategy** (Section 3.3) that introduces learnable coefficients to linearly combine the parameter weights of pretrained ST-blocks and serve as the initial parameter weights of the identified ST-block, which accelerates convergence and reduces the training time by up to 66%.

3.1 Automated Search Space Pruning

We propose an automated strategy to prune the general search space into a high-quality and relatively small search space for a specific CTS forecasting task. To achieve this, we first partition the general search space into subspaces that are distinguishable in quality and then remove lower-quality subspaces iteratively. Next, we introduce the search space partitioning method, the EDF metric used to evaluate the quality of search spaces, and the iterative search strategy.

3.1.1 Search Space Partitioning. Before introducing the search space partitioning method, we first introduce the general search space and define the notion of a *comb*.

The **general search space** is a collection of a large number of ST-blocks, each of which is constructed by S/T-operators combined using topological connection rules. We use the S/T-operators introduced in Section 2.2.1 and follow commonly used topological connection rules [33, 38, 50] to assemble them into ST-blocks.

Definition 3.1. A *comb* is a search space containing ST-blocks with the same S/T-operator combinations. It is represented as $comb = [n_{o_1}, n_{o_2}, ..., n_{o_L}]$, where n_{o_i} is the number of occurrences of S/T-operator o_i in each ST-block in the *comb* and L is the number of S/T-operators used for constructing the search space. Note that $0 \leq n_{o_i} \leq n_o$, where n_o is the number of S/T-operators contained in an ST-block.

For example, consider a search space constructed with S/T operators *skip*, *CNN*, *GCN*, Informer-Temporal (*INF-T*), Informer-Spatial(*INF-S*), and *GRU*. Then $comb = [0, 2, 2, 0, 1, 2]$ is a search space in which the ST-blocks contain 0 *skip*, 2 *CNN*, 2 *GCN*, 0 *INF-T*, 1 *INF-S* and 2 *GRU*, but with different topological connections.

Table 2: Comparison of different *combs*.

	skip	*CNN*	*GCN*	*INF-T*	*INF-S*	*GRU*	*EDF*
$comb_1$	0	2	2	0	1	2	0.16
$comb_2$	1	1	2	1	1	1	0.22
$comb_3$	1	2	1	2	0	1	0.54
$comb_4$	1	2	1	2	1	0	0.48
$comb_5$	1	2	2	2	0	0	0.69

The partitioning method is based on the observation that the combination of S/T-operators largely determines the accuracy of the ST-blocks they compose, resulting in different *combs* being distinguishable in quality, where the quality of a *comb* refers to the overall performance of the ST-blocks in it and is measured using EDF (see Section 3.1.2). To illustrate this phenomenon, we select 5 *combs* and randomly sample 200 ST-blocks in each and calculate its EDF. As we can see in Table 2, the EDF of the different *combs* are clearly distinguishable. In particular, $comb_1$ and $comb_2$ have significantly lower quality, indicating that we can remove the two *combs* to prune the search space, resulting in a smaller and higher-quality pruned search space. We thus propose to partition a search space into *combs* and prune it by removing low-quality *combs*.

3.1.2 Search Space Quality Evaluation. The quality of a search space $\mathcal{S}$ aims to capture the overall performance of the ST-blocks in it. We thus define the quality of $\mathcal{S}$ as a function of the performance of all ST-blocks in it.

$$quality(\mathcal{S}) = g(p(b_1), p(b_2), ..., p(b_n)), \tag{3}$$

where $p(b_i)$ is the performance of ST-block b_i, which is evaluated using validation accuracy, and g is a function that aggregates the performance of the constituent blocks.

Since it is impractical to train all ST-blocks to obtain their accuracy, we use uniform sampling to approximate. A simple metric to evaluate the quality of a search space is the average accuracy of the sampled ST-blocks, but this does not accurately reflect the quality of the search space. To illustrate this, we select *combs* S_1 and S_2 and randomly sample 200 ST-blocks in each and train these ST-blocks

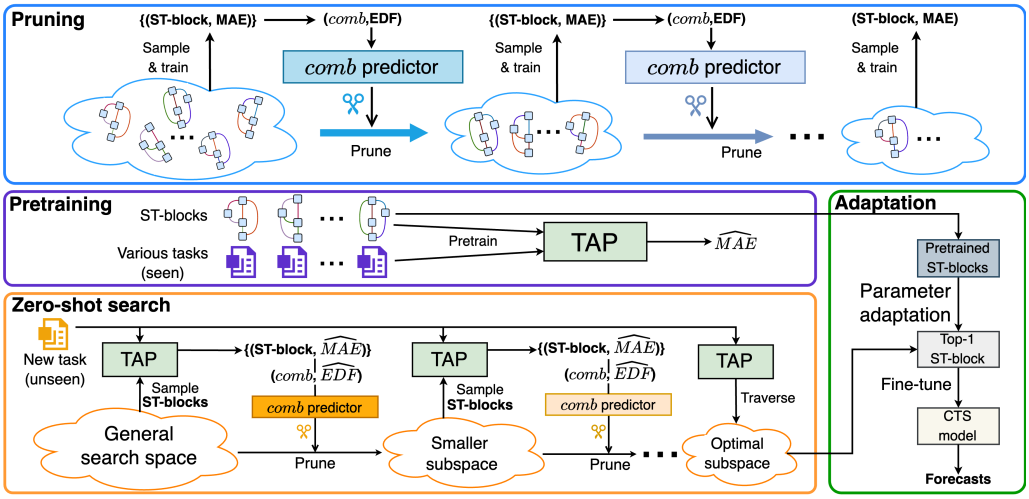


Figure 3: The FACTS framework.

to obtain their validation accuracy. We show in Figure 4 the MAE of ST-blocks sampled from the two *combs*, where a lower MAE corresponds to a higher accuracy. We see that S_2 has many more high-accuracy ST-blocks than S_1, indicating that S_2 is of higher quality than S_1, as our goal is to find a single optimal ST-block in the search space. However, the average accuracy of sampled ST-blocks in S_1 is higher than that in S_2. Thus, the average accuracy of sampled ST-blocks fails to reflect the quality of a search space.

Instead, we use the error empirical distribution function (EDF) metric [41] to quantify the quality of a search space. EDF captures the quality of a search space by randomly sampling a set of ST-blocks and calculating the proportion of ST-blocks whose MAE error is below a given threshold e:

$$F(e) = \frac{1}{n_e} \sum_{j=1}^{n_e} \mathbf{1}\left[e_j < e\right], \tag{4}$$

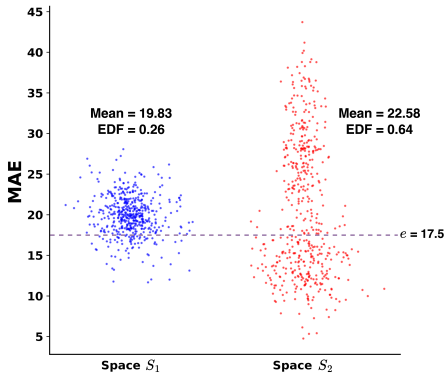


Figure 4: Search space quality metrics.

where n_e is the number of sampled ST-blocks and e_j is the validation MAE error of the j-th ST-block. In the context of CTS forecasting, the intuition is that comparing the proportion of high-accuracy ST-blocks is more robust than using the average metric, as it reduces the impact of outliers. In the case of Figure 4, the EDF of S_2 is significantly higher than that of S_1, reflecting the true quality difference between the two search spaces.

With the *comb*-based search space partitioning method, and using the EDF metric to evaluate the quality of a *comb*, a naive idea to reduce the search space is to calculate the EDF of all *combs* and remove low-quality *combs*. However, the number of *combs* is huge, and uniformly sampling ST-blocks in each *comb* to calculate its EDF is too costly. Instead, we only sample K *combs* and calculate their EDF values, and use the collected (*comb*, EDF) samples to train a *comb* predictor to predict the EDF of all remaining *combs*. For efficiency, we employ a Gradient Boosting Decision Tree (GBDT) [15] model as the *comb* predictor.

3.1.3 Iterative Search Space Pruning. Pruning the search space only once does not guarantee a high-quality pruned search space. This is because the threshold e in Equation 4 needs to be set manually, and an inappropriate e may cause EDF to fail to accurately compare the quality of different subspaces. To illustrate this, we select 3 *combs* and randomly sample 5 ST-blocks in each. We then train these ST-blocks to obtain their validation MAE error. As shown in Table 3, when e is set to the relatively large value of 16.00, we can easily identify $comb_3$ as the worst *comb*, but $comb_1$ and $comb_2$ have the same EDF, although ST-blocks in $comb_1$ perform significantly better than those in $comb_2$. This demonstrates that a large e may cause EDF to be unable to distinguish between good and relatively good *combs*,

Table 3: EDF values of different combinations.

	MAE	*MAE*	*MAE*	*MAE*	*MAE*	*EDF* (e = 16.00)	*EDF* (e = 14.70)
$comb_1$	14.65	14.72	14.73	14.75	17.36	0.8	0.2
$comb_2$	14.85	14.87	14.89	14.93	18.50	0.8	0.0
$comb_3$	15.38	15.54	15.88	16.75	20.33	0.6	0.0

which may cause us to remove high-quality subspaces, thereby reducing the potential of the pruned search space. In contrast, when e is set to the relatively small value of 14.70, the EDF labels of both $comb_2$ and $comb_3$ are 0, although the sampled ST-blocks from $comb_2$ have significantly higher accuracy than those from $comb_3$. Further, since we need to collect (*comb*, EDF) samples to train a *comb* predictor to predict the EDF of all remaining *combs*, too many 0 labels will cause the training samples to be unbalanced, resulting in an inaccurate *comb* predictor.

To solve this problem, we iteratively prune the search space, reducing e gradually. The intuition is that a larger e in the initial stage allows us to easily identify and remove bad *combs*. As e decreases, we can gradually distinguish good and relatively good *combs*, and since the quality of the remaining *combs* gradually becomes higher, there will not be a large number of (*comb*, EDF) samples with EDF values equal to 0. Specifically, we first divide the general search space into M_0 different *combs*. In the i-th pruning stage, we randomly sample c *combs*, and for each *comb*, we randomly sample r ST-blocks to calculate its EDF. Then, we train a *comb* predictor using the (*comb*, EDF) samples collected from iterations 0 to i and use it to predict the EDF values of all remaining *combs*. We end by pruning the search space by removing the 50% of *combs* with the lowest EDF values. The pruning ends when the number of remaining *combs* is below M. We set the initial threshold e_0 to be the median accuracy of the ST-blocks sampled in the 0-th iteration, and we set the final threshold e to be the $\lfloor M/M_0 \rfloor$-th best accuracy of the ST-blocks sampled in the 0-th iteration. We then reduce the threshold linearly during the iterative pruning. The complete pruning procedure is shown in Algorithm 1.

Algorithm 1 Search Algorithm

Input: CTS dataset $\mathcal{D}$, general search space S_0
Output: a pruned search space S_{n_p}

```
Split D into D_train, D_val, and D_test
Split S_0 into M_0 combs and calculate the iteration number n_p with n_p = ⌈log_2 M_0/M⌉
Randomly sample c combs to form a comb set C; then randomly sample r ST-blocks from each comb to form an ST-block set B
Train ST-blocks in B on D_train and obtain their validation accuracy on D_val
Set the initial threshold e_0 to the median accuracy of the ST-blocks in B and set the final threshold e to the ⌊M/M_0⌋-th best accuracy of the ST-blocks in B
for i = 0, ..., n_p-1 do
    Calculate the current threshold with e_i = e_0 − ((e_0 − e)/(n_p − 1)) · i
    Calculate EDF values for combs in C using Equation 4
    Train a GBDT model with (comb, EDF) pairs in C
    Use the GBDT model to predict EDF values for combs in S_i
    Remove the 50% of the combs with the lowest EDF values; the remaining combs form a new sub-search space S_{i+1}
    if i = n_p-1
        break
    Randomly sample c combs from S_{i+1} to form a comb set C_i, then randomly sample r ST-blocks from each comb in C_i to form an ST-block set B_i
    Train ST-blocks in B_i on D_train and obtain their validation accuracy on D_val
    C ← C ∪ C_i
end for
return The search space S_{n_p} formed by the remaining combs
```

3.2 Zero-Shot Search

We pretrain a TAP on numerous and diverse CTS forecasting tasks and then perform zero-shot search on unseen tasks to find optimal ST-blocks in minutes. We first introduce the structure of the TAP and then describe how to pretrain it and perform zero-shot search.

3.2.1 Task-aware Architecture Predictor. We propose a Task-aware Architecture Predictor (TAP) that takes the architecture of an ST-block and a task as input and outputs the prediction accuracy of the ST-block. Figure 5 shows the structure of the TAP, which consists of an Architecture Feature Learning (AFL) module, a Task Feature Learning (TFL) module, and an MLP regressor.

AFL extracts features of the architecture of an ST-block. We regard the architecture of an ST-block as a directed acyclic graph (DAG) and represent it by an adjacency matrix A_a and a feature matrix F_a. We then use a graph convolution network (GCN) followed by a single-layer MLP to encode A_a and F_a as a feature vector H_a, formulated as follows.

$$H_a = MLP_1(GCN(A_a, F_a)) \tag{5}$$

TFL extracts features of a CTS forecasting task. We consider both semantic features and statistical features to learn an informative vector representation of a task. We use a 2-layer Set-Transformer [29] to extract the semantic feature vector of a task. The first layer, named *IntraSetPool*, consists of a Set Attention Block (SAB) [29] for capturing the relationship between different samples of a single time series and a Multi-head Attention (PMA) [29] for pooling the samples into a single representation of the time series. The second layer, named *InterSetPool*, also consists of an SAB and a PMA, where the former is to capture the relationship between different time series and the latter is to pool these time series into a single representation of the task. We refer to Set-Transformer [29] for the detailed architecture of SAB and PMA. Formally, we have

$$\{\tilde{D_i}\} = IntraSetPool(\{D_i\}, W_1) \tag{6}$$

$$D_a = InterSetPool(\{\tilde{D_i}\}, W_2), \tag{7}$$

where $\{D_i\}$ is a CTS forecasting dataset, D_a is the learned semantic feature vector, and W_1 and W_2 are learnable parameters of the Set-Transformer. In addition, we use tsfresh [9] for extracting statistical features to construct a statistical feature vector T_a. We input the semantic and statistical feature vectors into different single-layer MLPs to achieve feature alignment, and then we concatenate the aligned feature vectors with the feature vector H_a and feed it into a two-layer MLP regressor to predict the accuracy of the ST-block on the CTS forecasting task, which is formulated as follows.

$$L_a = concatenate(MLP_2(D_a), MLP_3(T_a), H_a) \tag{8}$$

$$output = MLP_4(L_a) \tag{9}$$

Finally, we optimize the parameters of TAP using the mean squared error (MSE) loss.

3.2.2 TAP Pretraining. We pretrain TAP on numerous and diverse CTS forecasting tasks, enabling it to predict the accuracy of ST-blocks on arbitrary unseen tasks, thereby performing zero-shot search. The intuition is that we learn a combined embedding space of ST-blocks and tasks through pretraining, as well as a mapping from embeddings in the space to the accuracy of ST-blocks. When using the pretrained TAP to predict the accuracy of an ST-block on an unseen task, we first use AFL and TFL to obtain their combined embedding, and then we use the MLP regressor in the TAP to map the embedding to the accuracy of the ST-block.

We collect training samples of the form $(t, b, R(t, b))$ to pretrain the TAP, where $R(t, b)$ represents the validation accuracy of ST-block b on task t and is obtained by fully training b on t. Given a fixed training sample budget, a naive idea is to collect training samples by uniformly sampling ST-blocks from the general search space. However, the general search space is so large that it is difficult to learn the mapping from the input b and t to accuracy with limited training samples.

Considering that our goal is to find the optimal ST-block, we are more concerned with accurate prediction of high-performance ST-blocks than with accurate prediction of low-performance regions in the general search space. Therefore, we propose to collect training samples iteratively during the search space pruning process. As the search space is pruned iteratively, training samples are collected gradually from the pruned high-quality search space, so the TAP trained with these samples has more accurate prediction capabilities for high-quality regions of the search space, which is needed to identify the optimal ST-block.

Specifically, we randomly sample c *combs* from the remaining search space in the i-th pruning stage. For each *comb*, we randomly sample r ST-blocks and train the j-th ST-block on the j-th task, resulting in the training set $S_i = \bigcup_{k=1}^{c} \{(t_j, b_j, R(t_j, b_j))\}_{j=1,\cdots,r}^{b_j \in comb_k}$, which is used to calculate EDF values for the sampled *combs* using Equation 4 and is also used to pretrain the TAP. After the pruning is completed, we randomly sample z ST-blocks in the final pruned search space and train them on the r ($r < z$) tasks to construct a training set $S_e = \{(t_j, b_k, R(t_j, b_k)\}_{j=1,\cdots,r}^{k=1,\cdots,z}$. Finally, we pretrain the TAP on the training set $\bigcup_{i=1}^{p} S_i \cup S_e$, where p is the number of pruning stages.

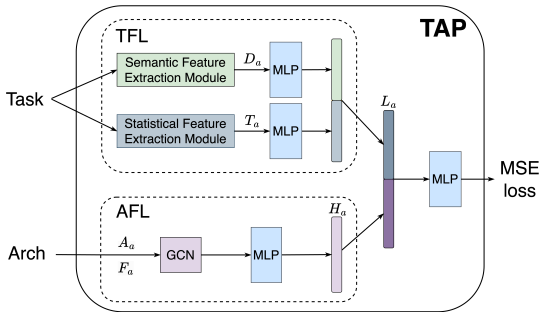


Figure 5: Task-aware Architecture Predictor.

3.2.3 Zero-shot Search on Arbitrary Unseen CTS Forecasting Tasks. After the pretraining, we obtain a TAP that can perform zero-shot search on arbitrary unseen CTS forecasting tasks. Given an unseen task t', we first perform the iterative pruning to reduce the general search space to a high-quality search space customized for task t' and then use the pretrained TAP to traverse the final pruned search space to find the optimal ST-block.

Specifically, in the i-th pruning stage, we first randomly sample c combs from the remaining search space. For each *comb*, we randomly sample r ST-blocks and train them on task t', resulting in the training set $S_i' = \bigcup_{k=1}^{c} \{(t', b_j, R'(t_j, b_j))\}_{j=1,\cdots,r}^{b_j \in comb_k}$, where $R'(t, b)$ is the prediction accuracy ($\widehat{MAE}$) of ST-block b on task t generated by the pretrained TAP. We then use S_i' to calculate pseudo-EDF ($\widehat{EDF}$) labels for the sampled *combs* using Equation 4 and train a *comb* predictor to prune the search space.

After the pruning is completed, we obtain a high-quality search space customized for task t'. We then use the pretrained TAP to predict the accuracy of all ST-blocks in the pruned search space and rank them according to prediction accuracy, where the top-1 ST-block is returned as the optimal ST-block. Since the zero-shot search phase does not involve any training, it is very fast and can be completed in minutes.

3.3 Fast Training with Parameter Adaptation

Having found a high-performance ST-block, we train its parameter weights to make forecasts on unseen tasks. Unlike existing automated methods that regard the training of different ST-blocks as independent processes and train the identified ST-block from scratch, we inherit the parameter weights of pretrained ST-blocks to **accelerate the training** of the identified ST-block.

3.3.1 Motivation for Parameter Weights Inheritance. In the pretraining phase, we train a large number of ST-blocks to generate training samples for TAP. The parameter weights of these ST-blocks contain knowledge about how to capture general spatio-temporal patterns of CTS, which is shared across ST-blocks and tasks. Inheriting this knowledge when training identified ST-blocks may facilitate fast convergence because the identified ST-blocks are trained from a good starting point rather than from scratch.

We inherit knowledge in pretrained ST-blocks by inheriting their parameter weights, unlike common transfer learning methods. Since pretrained ST-blocks differ in architecture from the identified ST-block and are trained on different tasks, it is not feasible to copy their parameter weights directly to the identified ST-block. However, the parameters of an ST-block come from multiple S/T operators that make up the ST-block, and the number of pretrained ST-blocks is sufficiently large to include all S/T operators in the search space, so we can inherit the parameter weights of each S/T operator in the identified ST-block separately.

3.3.2 Fast Parameter Adaptation. We propose a fast parameter adaptation strategy to inherit the weights of pretrained ST-blocks in a learnable manner. Then we finetune the inherited weights on the target task for a few training steps to obtain optimal weights for the identified ST-block.

Specifically, for each S/T-operator o in the identified ST-block, we first traverse the pretrained ST-blocks and select n ST-blocks that contain o and are most similar to the identified ST-block. Here, we consider two approaches to calculating the similarity between two ST-blocks. One is graph edit distance between DAGs corresponding to the two ST-blocks, which captures structural similarity. The other is the cosine similarity between the architectural feature vectors output by AFL (see Figure 5), which captures semantic similarity. We combine the two through voting.

Then, for each S/T operator o, we introduce k learnable vectors $\alpha_o = \{\alpha_o^1, \alpha_o^2, ..., \alpha_o^k\}$, where $\alpha_o^i \in \mathbb{R}^{n_o}$ and n_o is the number of parameters in o, to linearly transform the parameter weights of o that from n selected pretrained ST-blocks. Then we calculate the average of the transformed weights as the initial parameter weights of the identified ST-block, formulated as follows.

$$W_o^{init} = (\alpha_o^1 W_o^1 + \alpha_o^2 W_o^2 + ... + \alpha_o^k W_o^k)/k, \tag{10}$$

where W_o^i is the i-th parameter weight vector of o, k is the number of o contained in n selected pretrained ST-blocks, and W_o^{init} is the initial parameter weights of o of the identified ST-block. Figure 6 illustrates the proposed fast parameter adaptation strategy, where operator o_2 in the identified ST-block inherits parameter weights $W_{o_2}^1$, $W_{o_2}^2$, and $W_{o_2}^3$ from ST-blocks b_1 and b_2, and the corresponding coefficient vectors are $\alpha_{o_2}^1$, $\alpha_{o_2}^2$ and $\alpha_{o_2}^3$, respectively. We further divide W_o^i into groups based on the neural modules they belong to, with the coefficients being shared within each group to reduce the number of learnable parameters in α_o^i.

The coefficient α for all operators in the identified ST-block are learned through the following objective, enabling adaptive inheritance of knowledge from pretrained ST-blocks.

$$argmin_{\alpha} \mathbb{E}_{x \sim D} \mathcal{L}(x; W^{init}(\alpha, W)), \tag{11}$$

where W contains the parameter weights of the pretrained ST-blocks, D is the target task, and $\mathcal{L}$ is the MAE loss metric. The idea behind this optimization objective is to learn a set of coefficients to linearly transform the pretrained weights to instantiate the weights of the identified ST-block such that it achieves the highest performance on the target dataset.

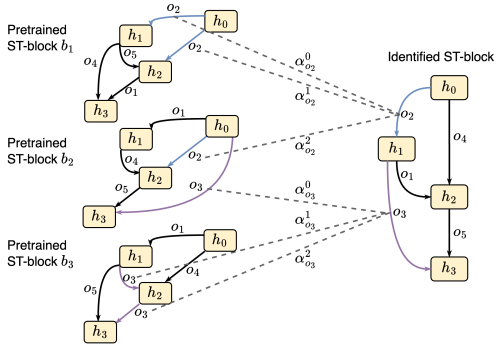


Figure 6: Inheriting parameters from pretrained ST-blocks.

After we obtain W^{init}, we further finetune it on the target dataset to get the final weights W_{final} of the identified ST-block as,

$$W^{final} = argmin_{W^{init}} \mathbb{E}_{x \sim D} \mathcal{L}(x; W^{init}) \tag{12}$$

Since we train the identified ST-block from a good starting point W^{init}, it converges much faster than when training it from scratch, thus achieving the goal of reducing the training time.

4 EXPERIMENTS

We conduct comprehensive experiments on seven public CTS forecasting datasets to assess the effectiveness and efficiency of FACTS.

4.1 Experimental Setup

4.1.1 Tasks for Pretraining and Evaluating. We summarize the datasets used for pretraining and evaluation below.

Datasets and tasks for pretraining: METR-LA [32], ETTh1 [65], ETTh2 [65], ETTm1 [65], ETTm2 [65], Solar-Energy [27], ExchangeRate [27], PEMS03 [46], PEMS04 [46], PEMS07 [46], PEMS08 [46]. We form 200 CTS forecasting tasks based on these datasets by splitting them along the temporal or spatial dimensions and considering the forecasting settings P-12/Q-12 and P-48/Q-48.

Datasets and tasks for evaluation: Electricity [27], NYC-TAXI [58], NYC-BIKE [58], SZ-TAXI [63], Los-Loop [63], PEMS-BAY [32], PEMSD7(M) [59]. We create 28 unseen tasks by considering the forecasting settings P-12/Q-12, P-24/Q-24, P-48/Q-48, and P-168/Q-1 (3rd) for each of these datasets.

4.1.2 Baselines. We compare FACTS with three competitive manually designed and three automated baselines. We reproduce the baselines based on their released code.

- MTGNN: employs mix-hop graph convolution and dilated inception convolution to build ST-blocks [52].
- AGCRN: employs 1D GCNs and GRUs to build ST-blocks [1].
- PDFormer: employs spatial and temporal transformers to build ST-blocks [20].
- AutoCTS: A gradient-based automated CTS forecasting framework with a manually designed search space [50].
- AutoCTS+: A comparator-based automated CTS foreacasting framework that jointly searches for ST-blocks and accompanying hyperparameter settings [51].
- SimpleSTG: prunes the general search space manually using sampling to remove bad design choices and employs random search to find the optimal ST-block [55].

4.1.3 Evaluation Metrics. Following previous studies [1, 32, 52, 53, 59], we use mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE) as the evaluation metrics for multi-step forecasting, and we use Root Relative Squared Error (RRSE) and Empirical Correlation Coefficient (CORR) as evaluation metrics for single-step forecasting. For MAE, RMSE, MAPE, and RRSE, lower values are better, while for CORR, higher values are better.

4.1.4 Implementation Details.

comb **predictor.** The *comb* predictor is a GBDT model based on LightGBM [22] with 100 trees and 31 leaves per tree. Standard normalization is applied to normalize the EDF labels of *combs*. We train the *comb* predictor with a learning rate of 0.05 and MSE loss.

TAP. For the structure of TAP, we set the number of layers of the GCNs to 4, with 128 hidden units in each layer. To pretrain TAP, we use the Adam [26] optimizer with a learning rate of 0.001 and a weight decay of 0.0005. The batch size is set to 64. Moreover, the hidden dimensions of the two-layer set-transformer are set to 256 and 128, respectively. We extract 128 commonly-used statistical features. We apply min-max normalization to normalize the validation MAE of ST-blocks from the same task and train TAP for 100 epochs with an early stop patience of 5 epochs.

Pruning. We randomly sample $c = 100$ *combs* at each pruning stage and collect $r = 100$ ST-blocks for each *comb*. The pruning ends when the number of *combs* in the remaining search space is less than $M = 2,000$. We then collect $z = 10,000$ ST-blocks in the pruned search space.

Parameter adaptation. We select $n = 5$ ST-blocks most similar to the identified ST-block for parameter adaptation.

CTS forecasting models. We use MAE as the training objective to train CTS forecasting models, and use Adam with a learning rate of 0.001 and a weight decay of 0.0001 as the optimizer. We set the batch size to 64. We train all CTS forecasting models for 100 epochs.

4.2 Experimental Results

4.2.1 Main Results. Tables 4 and 5 present the overall performance of FACTS and the baselines on the seven unseen CTS forecasting datasets with different forecasting settings. We run all evaluations three times with different random seeds and report the mean numbers. For ease of observation, we use bold and underline to highlight the best and second-best results, respectively. We observe that: 1) FACTS consistently outperforms the existing manual and automated methods, although we never pretrain on the P-24/Q-24 and P-168/Q-1 (3rd) forecasting settings and the seven datasets. This is evidence of the effectiveness of FACTS on arbitrary unseen CTS forecasting tasks. 2) AutoCTS+ usually achieves the second best accuracy because it supports joint search of ST-block and accompanying hyperparameter values. However, FACTS still outperforms AutoCTS+, although it does not search for hyperparameter values. Additionally, joint search can easily be integrated into FACTS, which may further improve performance.

4.2.2 Effectiveness of Automated Search Space Pruning.

Pruning improves the quality of the search space. First, we find that our pruning strategy can improve the quality of the search space. We consider 6 variants of FACTS that perform only 0 to 5 pruning stages, resulting in 6 pruned search spaces: space-0, space-1, space-2, space-3, space-4, and space-5. FACTS performs 6 pruning stages. We also consider the manually designed search spaces of AutoCTS and SimpleSTG. We randomly sample 200 ST-blocks from each of the above search spaces and train them to obtain their validation accuracy and EDF values.

We draw a scatter plot to show the results on Electricity under the P-12/Q-12 forecasting setting. Figure 7 shows that with the pruning of the search space, the sampled ST-blocks show increasingly higher accuracy, which is evidence that the proposed search space pruning strategy can remove low-quality regions of the search space, thereby improving the quality of the remaining search space. Further, the quality of the search spaces of AutoCTS and SimpleSTG are higher than that of FACTS in the early stages, but are surpassed after 4 iterations of the pruning, which indicates that the automated pruning strategy can generate search spaces that are better than the manually designed search spaces on unseen tasks.

Pruning helps search for high-performance ST-blocks. Second, we demonstrate that our pruning strategy helps search for high-performance ST-blocks. We compare FACTS with the variants space-0 to space-5. For fair comparison, we randomly sample m ST-blocks in each search space above, where m is the size of the final pruned search space of FACTS, which in this case is equal to 28, 206, 080, and we predict them using the pretrained TAP to find the optimal ST-block, and then we train it to obtain the accuracy.

We show the results on the Electricity dataset under the P-12/Q-12 forecasting setting in Table 6. We see that the identified ST-blocks show increasingly higher accuracy as the search space is pruned, which indicates that the iterative pruning strategy gradually improves the quality of the pruned search space, making it easier to find high-performance ST-blocks.

Iterative pruning v.s. one-time pruning. We offer evidence that iteratively pruning the search space yields better performance than pruning the search space only once (see Section 3.1.3). We design three variants *one-time*-1, *one-time*-2, *one-time*-3, which prune the search space only once, and sample the same number of (*comb*, EDF) pairs as FACTS to train a *comb* predictor, and prune the search space to M combs in one step. The EDF thresholds of the three variants are set to the initial, intermediate and final values of the EDF thresholds of FACTS, respectively.

Table 7 shows that although we set different EDF thresholds for the three variants, they all perform worse than FACTS. This shows the effectiveness of the proposed iterative pruning strategy.

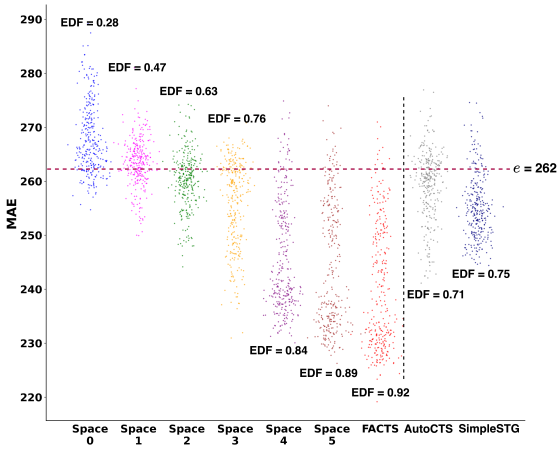


Figure 7: Quality of the search spaces during pruning.

Performance comparison between EDF and mean MAE. We compare FACTS and the variant that replaces EDF with the mean MAE metric.

The results in Table 8 show that FACTS performs considerably better than the variant on all CTS forecasting tasks, indicating that EDF is the better metric when evaluating the quality of a *comb*.

Table 4: Performance of P-12/Q-12 and P-24/Q-24 forecasting.

Data	Metric	P-12/Q-12 forecasting							P-24/Q-24 forecasting						
		FACTS	*AutoCTS*	*AutoCTS+*	*SimpleSTG*	MTGNN	AGCRN	PDFormer	FACTS	*AutoCTS*	*AutoCTS+*	*SimpleSTG*	MTGNN	AGCRN	PDFormer
PEMS-BAY	MAE	**1.517**	1.736	1.572	1.815	1.960	1.652	1.742	**1.780**	1.911	1.836	1.877	1.884	2.124	1.843
	RMSE	**3.287**	3.935	3.531	4.281	4.461	3.815	3.920	**3.986**	4.435	4.017	4.291	4.331	4.612	4.112
	MAPE	**3.494%**	3.822%	3.501%	4.316%	4.560%	3.843%	3.947%	**4.102%**	4.784%	4.236%	4.298%	4.405%	5.113%	4.139%
Electricity	MAE	**225.814**	240.65	238.513	262.815	306.331	611.08	247.982	**193.786**	204.333	205.401	207.185	211.913	1718.216	202.571
	RMSE	**1943.750**	2177.910	2106.307	2329.154	2468.959	8288.991	2178.527	**1676.211**	1681.030	1784.313	1825.164	1871.110	16364.798	1724.646
	MAPE	**15.871%**	17.275%	16.961%	21.165%	24.381%	42.628%	16.864%	**15.468%**	15.790%	16.085%	16.581%	17.136%	58.626%	15.993%
PEMSD7M	MAE	**2.551**	2.604	2.617	2.592	2.643	2.697	2.631	**3.167**	3.227	3.191	3.245	3.327	8.823	3.310
	RMSE	**4.863**	5.195	5.166	5.018	5.217	5.401	5.261	**6.078**	6.171	6.221	6.316	6.315	14.674	6.208
	MAPE	**6.296%**	6.592%	6.581%	6.471%	6.523%	6.782%	6.482%	**8.112%**	8.328%	8.323%	8.359%	8.493%	28.915%	8.251%
NYC-TAXI	MAE	**5.334**	5.576	5.536	5.625	5.847	5.818	6.259	**5.550**	5.621	5.591	5.774	5.889	5.735	5.760
	RMSE	**9.472**	9.846	9.792	10.048	11.918	13.924	11.206	**9.995**	10.834	10.127	10.830	10.710	11.512	10.560
	MAPE	**37.780%**	39.985%	41.235%	42.011%	40.271%	43.027%	43.194%	**38.338%**	38.452%	40.174%	42.915%	39.914%	44.264%	43.709%
NYC-BIKE	MAE	**1.796**	1.891	1.811	1.832	1.942	1.856	2.368	**1.842**	1.887	1.988	1.895	2.241	1.976	2.049
	RMSE	**2.732**	2.992	2.794	2.815	3.265	3.019	3.625	**2.867**	3.140	3.232	3.274	3.310	3.101	3.147
	MAPE	**49.951%**	52.389%	52.214%	52.281%	53.322%	56.128%	54.161%	**50.675%**	50.915%	51.313%	50.781%	54.015%	57.315%	52.648%
Los-Loop	MAE	**3.578**	3.677	3.652	3.668	3.640	8.912	3.980	**4.101**	4.179	4.244	4.375	4.271	4.452	4.282
	RMSE	**6.841**	7.063	7.119	7.179	7.084	15.182	7.286	**7.518**	7.775	7.943	8.143	7.905	8.831	8.014
	MAPE	**10.006%**	10.720%	10.443%	10.609%	10.212%	34.040%	10.668%	**12.279%**	12.857%	13.209%	13.591%	12.876%	13.720%	12.945%
SZ-TAXI	MAE	**3.178**	3.250	3.254	3.218	3.229	4.510	3.719	**2.892**	3.171	3.056	3.154	3.215	2.905	3.237
	RMSE	**4.217**	4.484	4.457	4.291	4.779	5.002	4.904	**4.260**	4.459	4.347	4.401	4.528	4.401	4.542

Table 5: Performance of P-48/Q-48 and P-168/Q-1(3rd) forecasting.

Data	Metric	P-48/Q-48 forecasting								P-168/Q-1(3rd) forecasting						
		FACTS	*AutoCTS*	*AutoCTS+*	*SimpleSTG*	MTGNN	AGCRN	PDFormer	Metric	FACTS	*AutoCTS*	*AutoCTS+*	*SimpleSTG*	MTGNN	AGCRN	PDFormer
PEMS-BAY	MAE	**1.963**	1.988	1.974	2.011	2.198	2.831	2.064	RRSE	**0.2887**	0.2901	0.2931	0.3015	0.2947	0.4719	0.2940
	RMSE	**4.175**	4.185	4.253	4.291	4.684	5.125	4.392	CORR	**0.9366**	0.9275	0.9241	0.8906	0.9116	0.8586	0.9245
	MAPE	**4.708%**	4.715%	4.856%	4.913%	5.330%	5.013%	4.928%								
Electricity	MAE	**215.821**	247.256	239.836	254.150	306.331	611.08	250.982	RRSE	**0.0692**	0.0741	0.0731	0.0815	0.0758	0.1033	0.0781
	RMSE	**1839.745**	2144.362	2038.137	2249.158	2468.959	8288.991	2218.027	CORR	**0.9785**	0.9439	0.9516	0.8911	0.9412	0.8854	0.9273
	MAPE	**16.438%**	16.845%	16.761%	16.853%	24.381%	42.628%	16.864%								
PEMSD7M	MAE	**3.454**	3.522	3.510	3.471	3.585	3.606	3.559	RRSE	**0.2867**	0.3056	0.2895	0.2981	0.3105	0.5314	0.2995
	RMSE	**6.582**	6.715	6.671	6.621	6.907	7.124	6.814	CORR	**0.9375**	0.9298	0.9338	0.9282	0.9278	0.8186	0.9336
	MAPE	**8.767%**	9.236%	9.114%	8.991%	9.322%	9.375%	9.201%								
NYC-TAXI	MAE	**5.544**	5.956	5.622	5.811	5.767	6.009	5.995	RRSE	**0.2018**	0.2324	0.2191	0.2571	0.2273	0.2709	0.2624
	RMSE	**9.931**	11.154	10.174	10.781	10.568	18.049	11.963	CORR	**0.8874**	0.8689	0.8714	0.8491	0.8697	0.8473	0.8417
	MAPE	**37.964%**	39.156%	38.235%	38.681%	39.011%	49.629%	39.894%								
NYC-BIKE	MAE	**1.990**	1.998	2.026	2.184	2.097	2.105	2.096	RRSE	**0.7478**	0.7492	0.7515	0.7590	0.7581	0.7601	0.7531
	RMSE	**2.996**	3.011	3.178	3.527	3.256	3.311	3.308	CORR	**0.7850**	0.7805	0.7773	0.7685	0.7691	0.7631	0.7669
	MAPE	**51.830%**	52.146%	52.471%	55.107%	52.737%	53.481%	52.672%								
Los-Loop	MAE	**4.520**	4.643	4.551	4.750	4.624	8.962	4.767	RRSE	**0.4258**	0.4311	0.4298	0.4307	0.4392	1.7565	0.4493
	RMSE	**8.327**	8.785	8.536	8.681	8.549	14.956	9.003	CORR	**0.7908**	0.7797	0.7825	0.7783	0.7695	0.0203	0.7699
	MAPE	**14.873%**	15.394%	15.015%	15.491%	17.417%	34.184%	15.961%								
SZ-TAXI	MAE	**2.891**	3.106	2.976	3.015	3.193	2.980	3.196	RRSE	**0.2288**	0.4815	0.4785	0.4916	0.4878	1.2367	0.4892
	RMSE	**4.394**	4.471	4.403	4.455	4.486	4.397	4.496	CORR	**0.5077**	0.2206	0.2200	0.2003	0.2216	0.0345	0.2236

Table 6: Accuracy comparison in the pruning process.

Metric	space-0	space-1	space-2	space-3	space-4	space-5	FACTS
MAE	258.540	250.515	242.858	235.870	231.231	227.410	**225.814**
RMSE	2077.255	2050.231	2016.333	1990.970	1976.011	1955.121	**1943.750**
MAPE	24.918%	22.442%	19.892%	17.932%	17.004%	16.115%	**15.871%**

Table 7: Iterative pruning v.s. one-time pruning.

variant	metric	PEMS-BAY	NYC-TAXI	Electricity
One-time-1	MAE	1.637	5.615	256.814
	RMSE	3.725	10.347	2279.015
	MAPE	3.712%	41.925%	18.517%
One-time-2	MAE	1.574	5.581	245.618
	RMSE	3.549	9.991	2185.761
	MAPE	3.682%	41.034%	16.880%
One-time-3	MAE	1.762	5.704	273.047
	RMSE	4.113	10.675	2365.225
	MAPE	4.042%	43.714%	23.027%
FACTS	MAE	**1.517**	**5.334**	**225.814**
	RMSE	**3.287**	**9.472**	**1943.750**
	MAPE	**3.494%**	**37.780%**	**15.871%**

4.2.3 Effectiveness of Zero-shot Search Strategy.

Search time comparison. We compare the search time of FACTS and the automated baselines on seven unseen datasets under the *P*-24/*Q*-24 forecasting setting and report the results in Table 9. We see that FACTS finds high-performance ST-blocks in less than 10 minutes on all CTS forecasting tasks, which is negligible compared to the baselines. This indicates that FACTS is efficient enough for deployment in practice.

Table 8: Performance comparison of EDF vs. mean MAE.

Data	Metric	P-12/Q-12		P-24/Q-24		P-48/Q-48		Metric	P-168/Q-1(3rd)	
		EDF	MAE	EDF	MAE	EDF	MAE		EDF	MAE
PEMS-BAY	MAE	**1.517**	1.572	**1.780**	1.869	**1.963**	1.976	RRSE	**0.2887**	0.2933
	RMSE	**3.287**	3.557	**3.986**	4.115	**4.175**	4.192	CORR	**0.9366**	0.9244
	MAPE	**3.494%**	3.515%	**4.102%**	4.148%	**4.708%**	4.732%			
Electricity	MAE	**225.814**	265.140	**193.786**	208.517	**215.821**	258.191	RRSE	**0.0692**	0.0744
	RMSE	**1943.750**	2237.214	**1676.211**	1715.150	**1839.745**	2025.159	CORR	**0.9785**	0.9418
	MAPE	**15.871%**	16.965%	**15.468%**	16.005%	**16.438%**	17.002%			
PEMSD7M	MAE	**2.551**	2.602	**3.167**	3.341	**3.454**	3.498	RRSE	**0.2867**	0.2893
	RMSE	**4.863**	5.025	**6.078**	6.371	**6.582**	6.628	CORR	**0.9375**	0.9326
	MAPE	**6.296%**	6.482%	**8.112%**	8.419%	**8.767%**	8.994%			
NYC-TAXI	MAE	**5.334**	5.558	**5.550**	5.701	**5.544**	5.629	RRSE	**0.2018**	0.2292
	RMSE	**9.472**	9.801	**9.995**	10.332	**9.931**	10.255	CORR	**0.8874**	0.8709
	MAPE	**37.780%**	40.112%	**38.338%**	40.156%	**37.964%**	38.414%			
NYC-BIKE	MAE	**1.796**	1.821	**1.842**	1.908	**1.990**	2.004	RRSE	**0.7478**	0.7554
	RMSE	**2.732**	2.835	**2.867**	3.185	**2.996**	3.143	CORR	**0.7850**	0.7785
	MAPE	**49.951%**	52.281%	**50.675%**	53.161%	**51.830%**	52.315%			
Los-Loop	MAE	**3.578**	3.637	**4.101**	4.298	**4.520**	4.582	RRSE	**0.4258**	0.4302
	RMSE	**6.841**	7.091	**7.518**	7.801	**8.327**	8.522	CORR	**0.7908**	0.7798
	MAPE	**10.006%**	10.285%	**12.279%**	13.055%	**14.873%**	15.296%			
SZ-TAXI	MAE	**3.178**	3.229	**2.892**	3.161	**2.891**	3.115	RRSE	**0.2288**	0.4382
	RMSE	**4.217**	4.320	**4.260**	4.398	**4.394**	4.477	CORR	**0.5077**	0.2815

Table 9: Searching time in GPU hours (h) or minutes (m).

dataset	AutoSTG	AutoCTS	AutoCTS+	SimpleSTG	FACTS
PEMS-BAY	278.1 h	405.7 h	88.6 h	324.5 h	**9.2 m**
Electricity	144.2 h	197.6 h	43.8 h	159.4 h	**7.9 m**
PEMSD7(M)	81.2 h	101.5 h	22.2 h	81.4 h	**6.5 m**
NYC-TAXI	13.3 h	20.2 h	4.8 h	16.8 h	**6.0 m**
NYC-BIKE	12.7 h	19.8 h	4.3 h	15.9 h	**5.7 m**
Los-Loop	6.8 h	11.2 h	2.5 h	7.6 h	**5.5 m**
SZ-TAXI	6.6 h	10.4 h	2.5 h	7.5 h	**5.4 m**

Ablation studies on the TFL module. We conduct ablation studies on the TFL module of TAP. We compare FACTS with the following three variants.

- **w/o task features:** removes the TFL module from TAP and keeps the rest unchanged.
- **w/o semantic features:** removes the semantic feature extraction module from TAP and keeps the rest unchanged.
- **w/o statistical features:** removes the statistical feature extraction module from TAP and keeps the rest unchanged.

We first randomly sample 100 ST-blocks from the general search space and train them to obtain validation accuracies. We then use the pretrained TAP from each variant and FACTS to predict these ST-blocks respectively to obtain the prediction accuracies. We use MAE and Spearman's rank correlation coefficient (ρ) to quantify the difference between the true validation accuracy and the prediction accuracy of the ST-blocks. We show results on three unseen datasets under the P-24/Q-24 forecasting setting in Table 10.

We observe that disabling the TFL module significantly reduces the accuracy of the pretrained TAP on unseen tasks, indicating that TFL is crucial for enabling TAP to generalize to unseen tasks. Next, enabling either the semantic feature extraction module or the statistical feature extraction module can improve the accuracy of TAP, and enabling both achieves the highest accuracy. This shows that they both help identify CTS forecasting tasks and are functionally complementary, so using both results in the best performance.

The trade-off between pretraining time and accuracy.

We study the trade-off between pretraining time and accuracy by varying the number of *combs* c sampled in each iteration. Specifically, we construct three variants of FACTS with c equal to 60, 80, and 120 and keep other settings as in FACTS. FACT uses $c = 100$.

The results under the P-12/Q-12 forecasting setting are shown in Table 12. We see that the final accuracy generally improves gradually with an increase of the pretraining time, showing that collecting more *combs* and ST-blocks to pretrain TAP can bring higher performance. However, when c exceeds 100, the performance improvement is very limited, indicating that using $c = 100$ strikes a good balance between pretraining efficiency and final accuracy.

4.2.4 Effectiveness of Fast Parameter Adaptation.

Training time comparison. We first compare the training time of the identified ST-blocks and and those of the baselines on seven unseen tasks. We also consider a variant of FACTS without fast parameter adaptation. Table 11 shows that FACTS takes the least time to train the identified ST-blocks, while still achieving the highest accuracy (see Tables 4 and 5). In particular, FACTS reduces training time by 62% to 66% compared to the variant w/o adapt on the seven unseen tasks, which is evidence of the effectiveness of the proposed fast parameter adaptation strategy. Furthermore, the total running time of FACTS, i.e., the sum of the search and training time (see Table 9) is lower than the training times of the baselines, which indicates that FACTS is efficient enough for deployment.

Ablation studies on fast parameter adaptation. We compare the trend of the validation loss across increasing training epochs for FACTS and two variants on PEMSD7(M). For the variant w/o adapt, we disable fast parameter adaptation and train the identified ST-block from scratch. For the variant simple_average, we use simple averaging instead of fast parameter adaptation to inherit the weights from pretrained ST-blocks. Figure 8 shows that the proposed fast parameter adaptation strategy contributes to the fast convergence of the ST-block without reducing the accuracy, thus indicating the proposed fast parameter adaptation strategy is effective. Next, the variant simple_average converges faster than training from scratch, but significantly more slowly than FACTS, indicating that the simple parameter inheritance strategy is not as effective as the proposed fast parameter adaptation strategy.

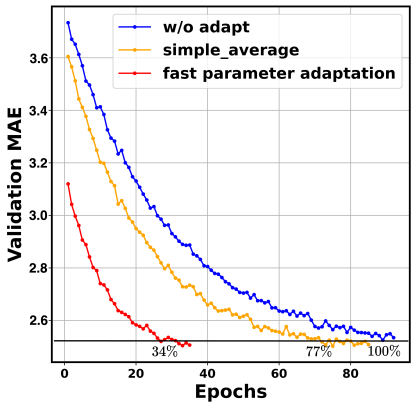


Figure 8: Efficiency comparison of training strategies.

4.2.5 Case Study. We show the ST-blocks identified on Electricity and PEMS-BAY in Figure 9. We can see that there are notable differences in the architectures of the ST-blocks. In particular, the ST-block identified on the Electricity dataset contains three NLinear, which is because Electricity exhibits strong periodicity and weak correlation between time series and because NLinear is good at capturing such patterns [61]. For comparison, the ST-block identified on the PEMS-BAY dataset contains four S-operators, i.e., two DGCN and two Spatial-Informer (INF_S). As PEMS-BAY is a traffic speed dataset with complex spatial correlations between time series, more S-operators are needed to capture the spatial correlations.

5 RELATED WORK

5.1 Manual CTS Forecasting

Manually designed CTS forecasting methods [1, 10–13, 21, 27, 45, 48, 49, 52, 53] choose or design S/T operators, assemble them into ST-blocks, and then train the ST-blocks from scratch to make forecasts. MTGNN [52] stacks mix-hop GCNs and inception convolution operators sequentially to build an ST-block. AGCRN [1] replace the

Table 10: Ablation studies on the TFL.

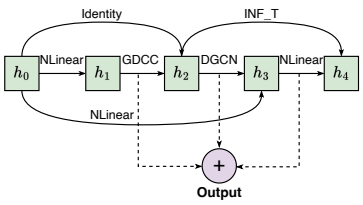
variant	metric	PEMS-BAY	NYC-TAXI	Electricity
w/o task	MAE	0.1470	0.1108	0.0903
	Spear	0.6401	0.6671	0.5915
w/o sem	MAE	0.0851	0.0703	0.0652
	Spear	0.7561	0.7255	0.6900
w/o stat	MAE	0.0688	0.0581	0.0593
	Spear	0.7641	0.7581	0.7290
FACTS	MAE	**0.0410**	**0.0328**	**0.0391**
	Spear	**0.8414**	**0.8285**	**0.8151**

Table 11: Training time comparison in GPU minutes.

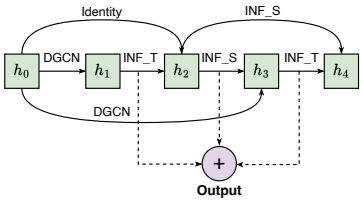
task	MTGNN	AGCRN	PDFormer	AutoCTS	AutoCTS+	SimpleSTG	w/o adapt	FACTS
PEMS-BAY	298.6	240.5	252.8	325.5	283.7	280.2	295.1	**110.3**
Electricity	240.6	193.8	261.2	273.5	255.7	260.0	271.2	**94.9**
PEMSD7(M)	46.3	39.1	50.4	62.4	53.3	51.7	47.3	**16.4**
NYC-TAXI	29.2	22.6	29.6	34.1	23.5	27.6	25.4	**9.7**
NYC-BIKE	27.2	19.9	29.0	31.2	21.0	26.2	23.7	**8.3**
Los-Loop	19.6	12.8	17.5	22.7	17.8	20.4	18.5	**6.3**
SZ-TAXI	19.3	11.9	17.1	22.9	30.2	31.0	15.8	**5.4**

Table 12: Trade-off between pretraining time and accuracy, P-12/Q-12 forecasting.

Data	Metric	c=120 **(208.9h)**	c=100 **(170.4h)**	c=80 **(139.8h)**	c=60 **(107.1h)**
PEMS-BAY	MAE	**1.509**	1.517	1.584	1.654
	RMSE	**3.287**	3.292	3.488	3.891
	MAPE	**3.488%**	3.494%	3.850%	4.491%
Electricity	MAE	**219.951**	225.814	259.963	294.065
	RMSE	**1938.105**	1943.750	2215.050	2494.129
	MAPE	**15.852%**	15.871%	18.668%	22.913%
PEMSD7M	MAE	**2.542**	2.551	2.685	2.778
	RMSE	**4.858**	4.863	5.022	5.481
	MAPE	**6.296%**	6.301%	6.645%	6.955%
NYC-TAXI	MAE	**5.327**	5.334	5.582	5.690
	RMSE	**9.460**	9.472	9.785	10.302
	MAPE	37.788%	**37.780%**	39.918%	41.051%
NYC-BIKE	MAE	**1.774**	1.796	1.953	2.079
	RMSE	**2.667**	2.732	3.003	3.184
	MAPE	**49.681%**	49.951%	51.391%	54.095%
Los-Loop	MAE	**3.563**	3.578	4.125	4.195
	RMSE	**6.829**	6.841	7.185	7.436
	MAPE	**9.993%**	10.006%	10.496%	10.920%
SZ-TAXI	MAE	**3.168**	3.178	3.219	3.552
	RMSE	**4.201**	4.217	4.294	4.785



(a) Electricity, P-12/Q-12



(b) PEMS-BAY, P-24/Q-24

Figure 9: Case studies

MLP layers in GRUs by GCNs to build an ST-block. PDFormer [20] combines semantic, geographic, and temporal self-attention operators in parallel to build an ST-block. We collect S/T operators and summarize connection rules from these proposals.

5.2 Automated CTS Forecasting

Existing automated methods manually design a search space. AutoST (a) [31] and AutoSTG [38] empirically select a few S/T operators to construct a small search space. AutoCTS [50] compares the accuracy of S/T operators on several CTS forecasting datasets and selects a few S/T operators with the highest accuracy to construct a search space. AutoST (b) [30] manually designs three S/T operators that consider the order between spatial and temporal modules to construct a small search space. SimpleSTG [55] proposes three pruning strategies, and for each strategy, it manually removes poor design choices based on the accuracy of sampled ST-blocks. Considering search efficiency, AutoST (a), AutoSTG, AutoCTS, and AutoST (b) employ a gradient-based search strategy, which requires training a supernet. SimpleSTG employs random search to find the optimal ST-block from a set of randomly sampled ST-blocks through fully training. AutoCTS+ [51] proposes a comparator to compare the accuracy of different ST-blocks, which necessitates the training of many ST-blocks to obtain their validation accuracies. Considering training efficiency, existing automated methods train identified ST-blocks from scratch.

6 CONCLUSION

We propose FACTS, an efficient and fully automated CTS forecasting framework that can find an ST-block that offers high-performance accuracy on arbitrary unseen tasks and can make forecasts in minutes. Experimental results on seven commonly used CTS forecasting datasets show that the FACTS framework is capable of state-of-the-art accuracy and efficiency. FACTS includes several hyperparameters, and methods exist that may reduce these hyperparameters, such as reinforcement learning. However, introducing the black-box optimization algorithm may also bring limitations to FACTS, such as introducing additional complexity and requiring more computing resources and time. This thus requires further research and we leave it as future work.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China (62372179), Independent Research Fund Denmark (8022- 00246B and 8048-00038B), Villum Fonden (34328 and 40567), and the Innovation Fund Denmark center, DIREC.

REFERENCES

[1] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. 2020. Adaptive Graph Convolutional Recurrent Network for Traffic Forecasting. In *NeurIPS*, Vol. 33. 17804–17815.

[2] David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. 2022. Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles. *Proc. VLDB Endow.* 15, 3 (2022), 611–623.

[3] David Campos, Bin Yang, Tung Kieu, Miao Zhang, Chenjuan Guo, and Christian S. Jensen. 2024. QCore: Data-Efficient, On-Device Continual Calibration for Quantized Models. *Proc. VLDB Endow.* 17, 11 (2024), 2708–2721.

[4] David Campos, Miao Zhang, Bin Yang, Tung Kieu, Chenjuan Guo, and Christian S. Jensen. 2023. LightTS: Lightweight Time Series Classification with Adaptive Ensemble Distillation. . *Proc. ACM Manag. Data 1(2): 171:1-171:27 (2023)* (2023).

[5] Yaofo Chen, Yong Guo, Qi Chen, Minli Li, Wei Zeng, Yaowei Wang, and Mingkui Tan. 2021. Contrastive neural architecture search with neural architecture comparators. In *Conference on Computer Vision and Pattern Recognition*. 9502–9511.

[6] Yunyao Cheng, Peng Chen, Chenjuan Guo, Kai Zhao, Qingsong Wen, Bin Yang, and Christian S. Jensen. 2024. Weakly Guided Adaptation for Robust Time Series Forecasting. *Proc. VLDB Endow.* (2024).

[7] Yunyao Cheng, Chenjuan Guo, Bin Yang, Haomin Yu, Kai Zhao, and Christian S. Jensen. 2024. A Memory Guided Transformer for Time Series Forecasting. *Proc. VLDB Endow.* 18 (2024).

[8] Kyunghyun Cho. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[9] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. 2018. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing* 307 (2018), 72–77.

[10] Razvan-Gabriel Cirstea, Chenjuan Guo, Bin Yang, Tung Kieu, Xuanyi Dong, and Shirui Pan. 2022. Triangular, Variable-Specific Attentions for Long Sequence Multivariate Time Series Forecasting. In *IJCAI*. 1994–2001.

[11] Razvan-Gabriel Cirstea, Bin Yang, Chenjuan Guo, Tung Kieu, and Shirui Pan. 2022. Towards Spatio-Temporal Aware Traffic Time Series Forecasting. In *ICDE*. 2900–2913.

[12] Razvan-Gabriel Cirstea, Tung Kieu, Chenjuan Guo, Bin Yang, and Sinno Jialin Pan. 2021. EnhanceNet: Plugin Neural Networks for Enhancing Correlated Time Series Forecasting. In *ICDE*. 1739–1750.

[13] Razvan-Gabriel Cirstea, Bin Yang, and Chenjuan Guo. 2019. Graph Attention Recurrent Neural Networks for Correlated Time Series Forecasting.. In *MileTS19@KDD*.

[14] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. 2020. AutoHAS: Efficient hyperparameter and architecture search. *arXiv preprint arXiv:2006.03656* (2020).

[15] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[16] Pengfei Gu, Yejia Zhang, Chaoli Wang, and Danny Z Chen. 2023. ConvFormer: Combining CNN and Transformer for Medical Image Segmentation. In *2023 IEEE 20th International Symposium on Biomedical Imaging (ISBI)*. IEEE, 1–5.

[17] Chenjuan Guo, Bin Yang, Jilin Hu, Christian S. Jensen, and Lu Chen. 2020. Context-aware, preference-based vehicle routing. *VLDB J.* 29, 5 (2020), 1149–1170.

[18] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-based systems* 212 (2021), 106622.

[19] S Hochreiter. 1997. Long Short-term Memory. *Neural Computation MIT-Press* (1997).

[20] Jiawei Jiang, Chengkai Han, Wayne Xin Zhao, and Jingyuan Wang. 2023. PDFormer: Propagation Delay-aware Dynamic Long-range Transformer for Traffic Flow Prediction. *arXiv preprint arXiv:2301.07945* (2023).

[21] Ming Jin, Yu Zheng, Yuan-Fang Li, Siheng Chen, Bin Yang, and Shirui Pan. 2023. Multivariate Time Series Forecasting With Dynamic Graph Neural ODEs. *IEEE Trans. Knowl. Data Eng.* 35, 9 (2023), 9168–9180.

[22] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[23] Duc Kieu, Tung Kieu, Peng Han, Bin Yang, Christian S. Jensen, and Bac Le. 2024. TEAM: Topological Evolution-aware Framework for Traffic Forecasting. *Proc. VLDB Endow.* 18 (2024).

[24] Tung Kieu, Bin Yang, Chenjuan Guo, Razvan-Gabriel Cirstea, Yan Zhao, Yale Song, and Christian S. Jensen. 2022. Anomaly Detection in Time Series with Robust Variational Quasi-Recurrent Autoencoders. In *ICDE*. 1342–1354.

[25] Tung Kieu, Bin Yang, Chenjuan Guo, Christian S. Jensen, Yan Zhao, Feiteng Huang, and Kai Zheng. 2022. Robust and Explainable Autoencoders for Unsupervised Time Series Outlier Detection. In *ICDE*. 3038–3050.

[26] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[27] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. 2018. Modeling long-and short-term temporal patterns with deep neural networks. In *SIGIR*. 95–104.

[28] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. 2021. Rapid neural architecture search by learning to generate graphs from datasets. *arXiv preprint arXiv:2107.00860* (2021).

[29] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*. 3744–3753.

[30] Jianxin Li, Shuai Zhang, Hui Xiong, and Haoyi Zhou. 2022. AutoST: Towards the universal modeling of spatio-temporal sequences. *Advances in Neural Information Processing Systems* 35 (2022), 20498–20510.

[31] Ting Li, Junbo Zhang, Kainan Bao, Yuxuan Liang, Yexin Li, and Yu Zheng. 2020. Autost: Efficient neural architecture search for spatio-temporal prediction. In *SIGKDD*. 794–802.

[32] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.

[33] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. In *ICLR*.

[34] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. 2021. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems* 34, 2 (2021), 550–570.

[35] Hao Miao, Ziqiao Liu, Yan Zhao, Chenjuan Guo, Bin Yang, Kai Zheng, and Christian S. Jensen. 2024. Less is More: Efficient Time Series Dataset Condensation via Two-fold Modal Matching. *Proc. VLDB Endow.* 18 (2024).

[36] Hao Miao, Jiaxing Shen, Jiannong Cao, Jiangnan Xia, and Senzhang Wang. 2022. MBA-STNet: Bayes-enhanced Discriminative Multi-task Learning for Flow Prediction. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[37] Hao Miao, Yan Zhao, Chenjuan Guo, Bin Yang, Zheng Kai, Feiteng Huang, Jiandong Xie, and Christian S. Jensen. 2024. A Unified Replay-based Continuous Learning Framework for Spatio-Temporal Prediction on Streaming Data. *ICDE* (2024).

[38] Zheyi Pan, Songyu Ke, Xiaodu Yang, Yuxuan Liang, Yong Yu, Junbo Zhang, and Yu Zheng. 2021. AutoSTG: Neural Architecture Search for Predictions of Spatio-Temporal Graph. In *Proceedings of the Web Conference 2021*. 1846–1855.

[39] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Anytime Stochastic Routing with Hybrid Learning. *Proc. VLDB Endow.* 13, 9 (2020), 1555–1567.

[40] Xiangfei Qiu, Jilin Hu, Lekui Zhou, Xingjian Wu, Junyang Du, Buang Zhang, Chenjuan Guo, Aoying Zhou, Christian S. Jensen, Zhenli Sheng, and Bin Yang. 2024. TFB: Towards Comprehensive and Fair Benchmarking of Time Series Forecasting Methods. *Proc. VLDB Endow.* 17, 9 (2024), 2363–2377.

[41] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10428–10436.

[42] Ragunathan Rajkumar, Insup Lee, Lui Sha, and John Stankovic. 2010. Cyber-physical systems: the next computing revolution. In *Design automation conference*. 731–736.

[43] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)* 54, 4 (2021), 1–34.

[44] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[45] Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. 2019. Temporal pattern attention for multivariate time series forecasting. *Machine Learning* 108, 8 (2019), 1421–1441.

[46] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. 2020. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In *AAAI*, Vol. 34. 914–921.

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[48] Senzhang Wang, Hao Miao, Hao Chen, and Zhiqiu Huang. 2020. Multi-task adversarial spatial-temporal networks for crowd flow prediction. In *International Conference on Information & Knowledge Management*. 1555–1564.

[49] Senzhang Wang, Meiyue Zhang, Hao Miao, Zhaohui Peng, and Philip S Yu. 2022. Multivariate correlation-aware spatio-temporal graph convolutional networks for multi-scale traffic prediction. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13, 3 (2022), 1–22.

[50] Xinle Wu, Dalin Zhang, Chenjuan Guo, Chaoyang He, Bin Yang, and Christian S Jensen. 2022. AutoCTS: Automated correlated time series forecasting. *Proc. VLDB Endow* 15, 4 (2022), 971–983.

[51] Xinle Wu, Dalin Zhang, Miao Zhang, Chenjuan Guo, Bin Yang, and Christian S Jensen. 2023. AutoCTS+: Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[52] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 2020. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *SIGKDD*. 753–763.
[53] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI*. 1907–1913.
[54] Ronghui Xu, Hao Miao, Senzhang Wang, Philip S Yu, and Jianxin Wang. 2024. PeFAD: A Parameter-Efficient Federated Framework for Time Series Anomaly Detection. In *SIGKDD*. 3621–3632.
[55] Zhen Xu, Yong Li, Qiang Yang, et al. 2022. Understanding and Simplifying Architecture Search in Spatio-Temporal Graph Neural Networks. *Transactions on Machine Learning Research* (2022).
[56] Sean Bin Yang, Chenjuan Guo, Jilin Hu, Bin Yang, Jian Tang, and Christian S. Jensen. 2022. Weakly-supervised Temporal Path Representation Learning with Contrastive Curriculum Learning. In *ICDE*. 2873–2885.
[57] Sean Bin Yang, Chenjuan Guo, and Bin Yang. 2022. Context-Aware Path Ranking in Road Networks. *IEEE Trans. Knowl. Data Eng.* 34, 7 (2022), 3153–3168.
[58] Junchen Ye, Leilei Sun, Bowen Du, Yanjie Fu, and Hui Xiong. 2021. Coupled layer-wise graph convolution for transportation demand prediction. In *AAAI*, Vol. 35. 4617–4625.
[59] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*. 3634–3640.
[60] Haomin Yu, Jilin Hu, Xinyuan Zhou, Chenjuan Guo, Bin Yang, and Qingyong Li. 2023. CGF: A Category Guidance Based PM2.5 Sequence Forecasting Training Framework. *IEEE Trans. Knowl. Data Eng.* 35, 10 (2023), 10125–10139.
[61] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting?. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 37. 11121–11128.
[62] Kai Zhao, Chenjuan Guo, Peng Han, Miao Zhang, Yunyao Cheng, and Bin Yang. 2024. Multiple Time Series Forecasting with Dynamic Graph Modeling. *Proc. VLDB Endow.* (2024).
[63] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2019. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems* 21, 9 (2019), 3848–3858.
[64] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 11106–11115.
[65] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *AAAI*, Vol. 35. 11106–11115.
[66] Barret Zoph and Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *ICLR*. OpenReview.net.