AutoCTS: Automated Correlated Time Series Forecasting

Wu, Xinle; Zhang, Dalin; Guo, Chenjuan; He, Chaoyang; Yang, Bin; Jensen, Christian S.

Published in:
Proceedings of the VLDB Endowment

DOI (link to publication from Publisher):
10.14778/3503585.3503604

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

AutoCTS: Automated Correlated Time Series Forecasting

Xinle Wu[1], Dalin Zhang[1], Chenjuan Guo[1], Chaoyang He[2], Bin Yang[1*], Christian S. Jensen[1]
[1]Aalborg University, Denmark [2]University of Southern California, USA
[1]{xinlewu, dalinz, cguo, byang, csj}@cs.aau.dk [2]chaoyang.he@usc.edu

ABSTRACT

Correlated time series (CTS) forecasting plays an essential role in many cyber-physical systems, where multiple sensors emit time series that capture interconnected processes. Solutions based on deep learning that deliver state-of-the-art CTS forecasting performance employ a variety of spatio-temporal (ST) blocks that are able to model temporal dependencies and spatial correlations among time series. However, two challenges remain. First, ST-blocks are designed manually, which is time consuming and costly. Second, existing forecasting models simply stack the same ST-blocks multiple times, which limits the model potential. To address these challenges, we propose *AutoCTS* that is able to automatically identify highly competitive ST-blocks as well as forecasting models with heterogeneous ST-blocks connected using diverse topologies, as opposed to the same ST-blocks connected using simple stacking. Specifically, we design both a micro and a macro search space to model possible architectures of ST-blocks and the connections among heterogeneous ST-blocks, and we provide a search strategy that is able to jointly explore the search spaces to identify optimal forecasting models. Extensive experiments on eight commonly used CTS forecasting benchmark datasets justify our design choices and demonstrate that *AutoCTS* is capable of automatically discovering forecasting models that outperform state-of-the-art human-designed models.

PVLDB Reference Format:
Xinle Wu, Dalin Zhang, Chenjuan Guo, Chaoyang He, Bin Yang, Christian S. Jensen. AutoCTS: Automated Correlated Time Series Forecasting. PVLDB, 15(4): 971-983, 2022.
doi:10.14778/3503585.3503604

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/WXL520/AutoCTS.

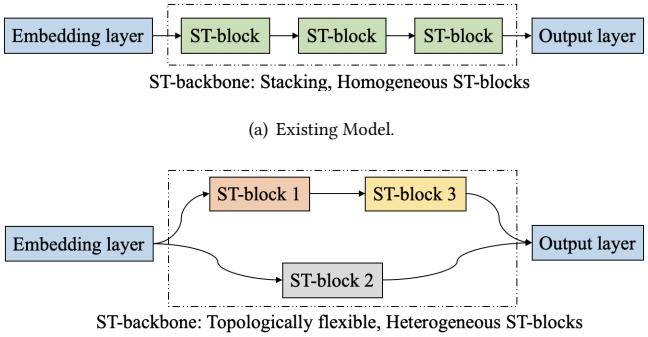
1 INTRODUCTION

We are witnessing continued developments in sensor technologies in cyber-physical systems (CPS), where sensors produce correlated time series [5, 15]. For example, in transportation, traffic sensors embedded in roads emit multiple traffic time series that record traffic flows at their locations across time. Since the traffic on a road is often correlated with the traffic on nearby roads, the traffic time series are often correlated [35]. Forecasting on correlated time series plays an essential role in ensuring effective operation of CPSs, such as identifying trends, predicting future behavior [7], and detecting outliers [2]. For instance, traffic time series forecasting can improve vehicle routing in transportation systems [13, 29, 36, 50].

By considering both *temporal dependencies* in time series and *spatial correlations* among different time series, recent deep learning models demonstrate impressive performance on correlated time series forecasting. Temporal dependencies capture how historical values influence future values. We use "spatial correlations" because the correlations among time series are often due to the proximity of the locations in which the sensors that generate the time series are deployed, but correlations may also be due to other factors. More specifically, correlated time series are modeled as a spatio-temporal (ST) graph, where nodes represent time series, and edges represent spatial correlations between pairs of time series [6, 44, 48].

Based on the above ST-graph modeling, different models are proposed to enable forecasting. Figure 1(a) summarizes existing forecasting models, which often include (1) an *embedding layer* that transforms the input time series data, (2) a *ST-backbone* that consists of a stack of multiple *ST-blocks* that are able to extract appropriate spatio-temporal features from the embedded time series data, and (3) an *output layer* that produces a final forecasting based on the features extracted by the ST-backbone.

Different studies propose unique ST-blocks that are responsible for the capture of both the temporal dependencies and spatial correlations [7, 9, 11, 14, 16, 24, 28, 45, 46, 51, 52]. For example, STGCN [51] employs a "sandwich" ST-block that includes two temporal convolutions, which model temporal dependencies, with one graph convolution in-between, which captures spatial correlations; and Graph Wavenet [45] uses a simpler ST-block that first employs gated temporal convolution to model temporal dependencies and then uses graph convolution to capture spatial correlations.



(a) Existing Model.

(b) Example from the Proposed Model Space.

Figure 1: Existing vs. Proposed CTS Forecasting Models, with Different ST-blocks Colored Differently.

*: Corresponding author.

Proceedings of the VLDB Endowment, Vol. 15, No. 4 ISSN 2150-8097.
doi:10.14778/3503585.3503604

Although various forecasting models have been proposed to improve the state-of-the-art accuracy, two major limitations remain.
Manually Designed ST-blocks. Existing studies rely on human expertise to design effective ST-blocks, which is both inefficient, often taking weeks or months, and costly. In addition, as a large number of operators exist that are able to capture temporal dependencies (i.e., T-operators) and spatial correlations (i.e., S-operators), the search space for designing ST-blocks is very large. Thus, it is almost impossible for humans to be able to identify an optimal combination of T/S-operators, thus potentially missing ST-blocks with high effectiveness. Further, rapid developments in machine learning are likely to lead to the invention of new, competitive operators. In order to benefit from new operators, it is necessary to reiterate the inefficient and costly manual design process whenever a new operator becomes available. Next, since time series from different domains have different characteristics, it is very difficult to design an ST-block that works well on substantially different datasets. To contend with these limitations, an automated design process that is able to identify optimal ST-blocks from a configurable search space of T/S-operators for specific datasets is called for.
ST-backbone with Stacking, Homogeneous ST-blocks. Existing forecasting models often have an ST-backbone that stacks the same ST-blocks sequentially multiple times to achieve "deeper" models that are better at capturing complex dependencies and correlations, as shown in Figure 1(a). We hypothesize that ST-backbones with *heterogeneous* ST-blocks connected by more *flexible topologies*, as exemplified in Figure 1(b), hold the potential to yield higher accuracy. Here, different ST-blocks, instead of same ST-blocks, can be connected using arbitrary topologies rather than by sequential stacking only. Intuitively, different ST-blocks may extract distinct features, which may enable more diverse and thus potentially better representations of time series. Supporting multiple topologies offers added flexibility, thus contributing further to enabling diverse models, which may enhance accuracy and stability. However, considering topologically flexible, heterogeneous ST-blocks increases the search space when designing forecasting models, thus rendering manual design more difficult and time-consuming. This naturally calls for an automated design approach.

Although Neural Architecture Search (NAS), a technology that automatically learns neural architectures [10], is able to outperform human-designed architectures on various tasks in computer vision (CV) [30, 37] and natural language processing (NLP) [42], existing NAS methods fail to offer automated solutions capable of solving the aforementioned two limitations. First, no well-defined search space exists for correlated time series forecasting, as existing NAS methods often focus on CV and NLP. Directly using the search space designed for other domains fails to identify ST-blocks with high potential for capturing both temporal dependencies and spatial correlations. Directly using all existing S/T-operators in the literature yields an extremely large search space, thus making it very difficult and time-consuming to identify promising ST-blocks. Second, most existing NAS methods focus on identifying an optimal cell, e.g., an ST-block in our setting, while assuming a fixed topology, e.g., stacking the same ST-blocks as shown in Figure 1(a), for connecting multiple instances of the same cell to derive the final model [10, 30, 37]. This fails to address the second limitation of manually designed forecasting models.

We propose *AutoCTS* that is able to not only automatically design ST-blocks but also ST-backbones with complex topologies that connect heterogeneous ST-blocks, thus addressing the two limitations. We first design a micro search space targeting ST-blocks that models operators and how different operators are connected using a graph. To enable effective and efficient search, we judiciously select a compact set of T-operators that model temporal dependencies and S-operators that model spatial correlations based on a thorough analysis of existing, manually designed ST-blocks. This enables us to automatically identify highly competitive ST-blocks in the proposed micro search space, thus addressing the first limitation. Next, we propose a macro search space, along with a joint search strategy that allows searches for an optimal topology among heterogeneous ST-blocks. This addresses the second limitation.

To the best of our knowledge, this is the first study that systematically investigates automated correlated time series forecasting by exploring jointly the neural architectures of ST-blocks and ST-backbones. The study makes three contributions. First, we carefully design a micro search space for correlated time series forecasting, including both T-operators and S-operators, along with a search strategy that is able to identify optimal ST-blocks from the micro search space. Second, we propose a macro search space, along with a joint search strategy that searches both ST-blocks and forecasting models, while allowing flexible topologies among heterogeneous ST-blocks. Third, we conduct extensive experiments on correlated time series from different application domains to offer insight into and justify our design choices, demonstrating also that our proposal is able to outperform state-of-the-art methods.

2 PRELIMINARIES

We introduce correlated time series forecasting, cover concepts that are necessary for the paper's proposal, and formalize the problem.
Correlated Time Series. Consider N correlated multivariate time series $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times T \times F}$, where each time series covers T timestamps and each timestamp is associated with F features. For example, assuming that 100 sensors are deployed in a road network and each sensor reports both travel speed and traffic flow every 5 minutes. Then, for one day, we have correlated time series $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{100 \times 288 \times 2}$ with $N = 100$ time series covering $T = 288$ timestamps and $F = 2$ features. We use $\boldsymbol{X}^{(i)} \in \mathbb{R}^{T \times F}$ to indicate the i-th time series, where $1 \leq i \leq N$, and $\boldsymbol{X}_t \in \mathbb{R}^{N \times F}$ to indicate the features from all time series at timestamp t, where $1 \leq t \leq T$.

To model spatial correlations among different time series, we introduce a graph $G = (V, E, A)$, where each vertex in V corresponds to a time series so that $|V| = N$, edges in E represent spatial correlations between different time series, and adjacency matrix $A \in \mathbb{R}^{N \times N}$ contains edge weights that reflect the strengths of the spatial correlations between time series. The edge weights are either predefined, e.g., based on the distances between the locations of the sensors that generate the time series [28, 40, 45], or learned in a data-driven manner [1, 6, 44].
Correlated Time Series Forecasting. We consider both single-step and multi-step correlated time series forecasting. Given the past P steps, (1) for the single step forecasting, we predict the Q-th future step, where $Q \geq 1$; (2) for the multi-step forecasting, we predict a total of Q future steps, with $Q > 1$. Formally, we define the

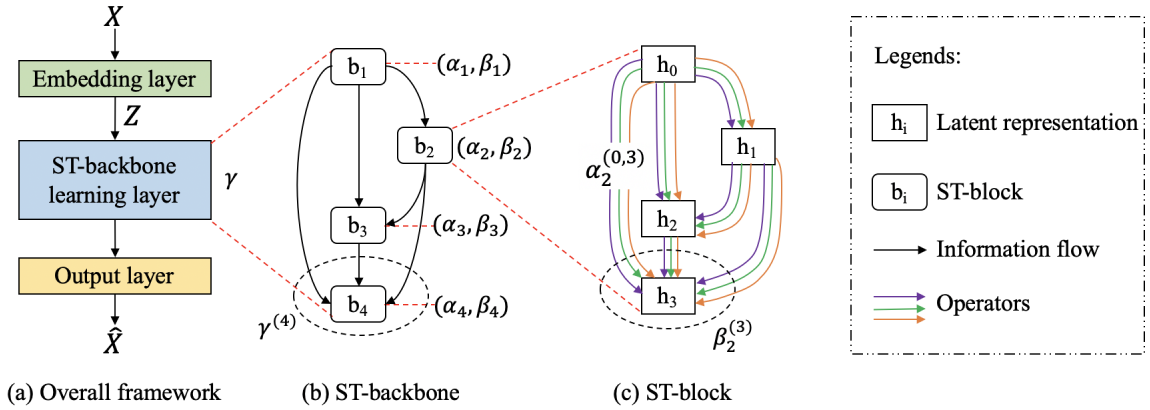


Figure 2: *AutoCTS* Overview.

single-step correlated time series forecasting problem as follows:

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

where $\mathcal{F}_w$ is a forecasting model and w is its learnable parameters; and $\hat{X}$ represents forecasted values. Likewise, the multi-step correlated time series forecasting problem is defined as follows:

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

Problem Definition. The goal of the paper is to automatically identify an accurate forecasting model $\mathcal{F}_w$. This includes the identification of (1) architecture parameters θ that describe the model $\mathcal{F}$, e.g., which operators that are used in different ST-blocks and how the different ST-blocks are connected in the ST-backbone; and (2) model parameters w that are used in the different operators, e.g., kernels in convolution operators and the projection matrices in attention operators. The objective function is show in Equation 3.

$$argmin_{\theta,w}\ ErrorMetric(\mathcal{F}_w, \mathcal{D}), \quad (3)$$

where $ErrorMetric(\mathcal{F}_w, \mathcal{D})$ returns the forecasting error of the model $\mathcal{F}_w$ that is learned on a training dataset $\mathcal{D}$.

3 AUTOMATED CTS FORECASTING

Figure 2 offers an overview of the automated CTS forecasting framework *AutoCTS*, which consists of three main components—an embedding layer, an ST-backbone learning layer, and an output layer.

The embedding layer maps the original input feature from time series X to a high-dimensional representation Z, which facilities extracting richer features from the input time series.

The ST-backbone learning layer, which is the core component of *AutoCTS*, is able to automatically design ST-backbones that encompass heterogeneous ST-blocks (as exemplified in Figure 2 (b)), where the design of the heterogeneous ST-blocks is also automated (as exemplified in Figure 2 (c)). When searching for ST-backbones, we use parameter γ to parameterize the connections among different ST-blocks. For example, $\gamma^{(4)}$ controls how the three connections from ST-blocks b_1, b_2, and b_3 connect to ST-block b_4. When searching for ST-blocks, we search both (1) the operators between two representations, parameterized by α, and (2) the different possible connections among different hidden representations, parameterized by β. For example, $\alpha_2^{(0,3)}$ represents the operators between hidden representations h_0 and h_3 in ST-block b_2, $\beta_2^{(3)}$ represents, in ST-block b_2, how the hidden representations h_0, h_1, and h_2 connect to the hidden representation h_3. We use unique sets of parameters $\{\alpha_i, \beta_i\}$ such that heterogeneous ST-blocks can be identified. The automatically designed ST-backbone takes as input the high-dimensional representation Z from the embedding layer and extracts spatio-temporal features, which are fed to the output layer.

Finally, the output layer makes the forecasting $\hat{X}$. We use a loss function, e.g., mean squared error, to measure the discrepancy between the forecast w.r.t. the ground truth to enable learning.

In the following, we first identify an appropriate search granularity (in Section 3.1), then we introduce the design of a micro search space for ST-blocks (in Section 3.2) and a macro search space for ST-backbones (in Section 3.3). Finally, we present the search strategy that explores the micro and macro search spaces jointly to discover promising forecasting models (in Section 3.4).

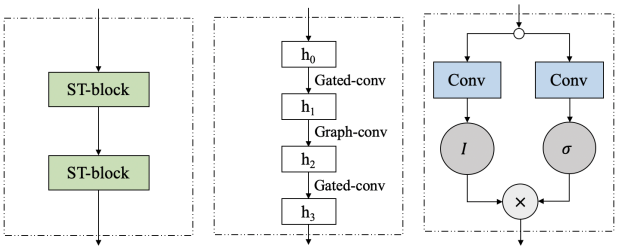
3.1 Search Granularity

The search space can be constructed from operators of different granularities. A search space based on fine-granularity operators offers more flexibility and greater opportunities for identifying promising neural architectures that cannot be identified by human experts, but it often also yields a very large search space, the search of which takes prohibitively long time and requires excessive computational resources. In contrast, a search space based on coarse-granularity operators yields a smaller search space and thus speeds up the search process, but it may also introduce human biases that may prevent the identification of high-performance architectures.

More specifically, in our problem setting, three different granularities exist. From coarse to fine, they are **ST-blocks**, **S/T operators**, and **basic computations**. We proceed to introduce the three granularities using a concrete example. Then, we discuss our design choices related to choosing the appropriate search granularity.

Figure 3 shows the neural architecture of Spatio-Temporal Graph Convolutional Networks (STGCN) [51], a human designed forecasting model. The backbone of STGCN consists of two **ST-blocks** that are stacked (cf. Figure 3(a)). An ST-block consists of three **S/T**

operators—two T-operators, i.e., gated convolutions, with an S-operator in-between, i.e., a graph convolution, (cf. Figure 3(b)). An S/T operator often consists of multiple **basic computations**. For example, Figure 3(c) shows the architecture of gated convolution, i.e., the T-operator. Here, I and σ refer to an identity and a sigmoid function, respectively; *Conv* is the convolution operator, and $\times$ is the element-wise product. These are all basic computations.



(a) ST-backbone (b) ST-block (c) Gated Convolution

Figure 3: Human Designed Forecasting Model: STGCN.

Based on the above, the coarsest search granularity is to use existing, manually-designed ST-blocks as atomic search units in the search space for finding novel ST-backbones. For example, instead of using a stacking structure with homogeneous ST-blocks as shown in Figure 3(a), it is possible to search for an ST-backbone with a more flexible structure with many different human designed ST-blocks as shown in Figure 1(b). However, since human designed ST-blocks may contain human biases already, only searching the different connections among them may limit the opportunities for finding novel and high-performance backbones.

The next granularity is that of using, human designed S/T-operators as atomic search units to search for novel ST-blocks. Since S/T-operators are at a finer granularity than ST-blocks, this granularity offers greater opportunities for discovering more powerful forecasting models that go beyond existing human designed models. In addition, whenever a new S/T-operator is designed, the new S/T-operator can be easily included in the search space. We consider this as an appropriate granularity.

The finest search granularity is to use basic operations as atomic search units to search for novel S/T operators. However, this leads to a much larger search space than when using S/T operators as the search space unit, incurs excessive computational costs, and requires a very large dataset to enable effective training [31].

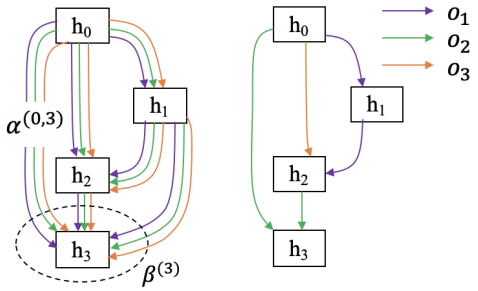
To find highly competitive forecasting models without requiring high computational and memory costs, we choose to use S/T operators as the atomic search units in a so-called micro search space to discover novel ST-blocks. Next, in the macro search space, we use the automatically learned ST-blocks as atomic search units to identify novel ST-backbones with flexible structures.

3.2 Micro Search Space

The micro search space defines the possible architectures of the ST-blocks that can be discovered. We first introduce the design of the micro search space and then explain how to reduce the size of the micro search space to speed up search.

3.2.1 Micro-DAG. We assume that an ST-block includes M latent representations. The first latent representation is the output representation from the embedding layer or the output representation of another ST-block. In addition, we consider a set O of operators, e.g., including multiple S/T operators, that are able to transform one latent representation to a new latent representation.

We represent the micro search space as a directed acyclic graph, denoted as micro-DAG (see Figure 4). The micro-DAG has M nodes h_i, $0 \leq i \leq M-1$, that each denotes a latent representation. Node h_0 denotes the representation returned by the embedding layer. For each node pair (h_i, h_j), we have $|O|$ edges, where each edge corresponds to an operator from operator set O. In Figure 4(a), as $O = \{o_1, o_2, o_3\}$ includes three operators, each node pair is associated with three edges. In addition, we only include edges from node h_i to h_j if $i < j$. This makes the graph a DAG, which simulates the forward flow when training a neural network.



(a) Micro-DAG (b) Derived ST-block

Figure 4: Micro Search Space.

Figure 4(b) shows a derived ST-block, which is a subgraph of the micro-DAG. Specifically, the derived ST-block only retains one edge, i.e., one operator, between each node pair (h_i, h_j). In addition, for each node, it preserves at most two incoming edges. This enables relatively complex internal topologies for ST-blocks and avoids introducing too many parameters.

The micro-DAG represents all possible architectures of an ST-block with M latent representations. This design yields to $|O|^{\frac{M(M-1)}{2}}$ possible ST-blocks. This is because a micro-DAG with M nodes has $\frac{M(M-1)}{2}$ node pairs (h_i, h_j), where $i < j$, and because each node pair can be connected by an operator from O. In Section 3.2.3, we discuss how to select a compact operator set O, thus reducing the size of the micro search space without comprising effectiveness.

3.2.2 Parameterizing ST-blocks. In order to derive an optimal ST-block, we introduce two sets of architecture parameters α and β, where α parameterizes node pair and β parameterizes nodes.

First, we parameterize each node pair (h_i, h_j) with vector $\alpha^{(i,j)} \in \mathbb{R}^{|O|}$ to indicate the weights over all operators in O. Then transformation $f^{(i,j)}$ from node h_i to node h_j is formulated as a weighted sum of all operators.

$$f^{(i,j)} = \sum_{o \in O} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} exp(\alpha_{o'}^{(i,j)})} o(h_i), \tag{4}$$

where $\alpha_o^{(i,j)}$ represents the weight of operator $o \in O$, which is to be learned, and $o(h_i)$ is the representation after applying operator o to representation h_i.

Next, we parameterize each node based on its incoming edges. We use another architecture parameter $\beta^{(j)} \in \mathbb{R}^j$ to assign weights to the incoming edge groups at node h_j, where each incoming edge group represents a hidden representation from a node h_i, where $0 \le i < j-1$. For example, in Figure 4, node h_3 has three incoming edge groups from h_0, h_1, and h_2, respectively. We then apply a softmax function to normalize the β parameter. If $SoftMax(\beta^{(3)}) = (0.3, 0.3, 0.4)$, it means that the weights of the representations from h_0, h_1, and h_2 are 0.3, 0.3, and 0.4, respectively. Therefore, for each node h_j, we can compute its representation as the weighted sum of all transformations of its predecessor nodes.

$$h_j = \sum_{i<j} \frac{exp(\beta^{(j)}[i])}{\sum_{i<j} exp(\beta^{(j)}[i])} f^{(i,j)} \tag{5}$$

$$= \sum_{i<j} \frac{exp(\beta^{(j)}[i])}{\sum_{i<j} exp(\beta^{(j)}[i])} \sum_{o \in O} \frac{exp(\alpha_o^{i,j})}{\sum_{o' \in O} exp(\alpha_{o'}^{i,j})} o(h_i), \tag{6}$$

where $\beta^{(j)}[i]$ is the architecture parameter value for weighting the transformation from h_i to h_j, which is to be learned.

In this way, given the first node h_0, we are able to compute the representations of the remaining nodes in the micro-DAG. We use the representation of the last node h_{M-1} as the output of the micro-DAG, and we thus feed h_{M-1} to the output layer. This gives a forecasting model that we can train using classic back propagation. The training enables us to identify the most appropriate α and β.

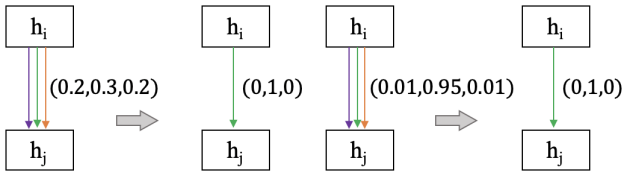
After training, we derive the final ST-block. For each node pair (h_i, h_j), we compute a weight $w_o^{(i,j)}$ using Eq. 7 for each operator o, and retain the operator with the largest $w_o^{(i,j)}$, i.e., $argmax_{o \in O} w_o^{(i,j)}$.

$$w_o^{(i,j)} = \frac{exp(\beta^{(j)}[i])}{\sum_{i<j} exp(\beta^{(j)}[i])} \frac{exp(\alpha_o^{i,j})}{\sum_{o' \in O} exp(\alpha_{o'}^{i,j})} \tag{7}$$

Next, for each node h_j, we preserve two operators. One is the operator from node h_{j-1}, i.e., its immediate predecessor node. The other one is the operator with the largest $w_o^{(i,j)}$ among the remaining operators, i.e., $argmax_{0 \le i \le j-2} w_o^{(i,j)}$. For example, in Figure 4(b), h_{i-1} always connects to h_i, where $1 \le i \le 3$. For h_3, assuming $argmax_{0 \le i \le 1} w_o^{(i,j)} = 0$, then h_0 is connected to h_3 .

Reducing the gap between the micro-DAG and the derived ST-block. Due to how we reduce a micro-DAG to an ST-Block, there may be a large gap between the micro-DAG and the derived ST-Block, which may make the derived ST-block suboptimal. In other words, although we have learned an effective micro-DAG, the derived ST-block may not perform as well as the micro-DAG since the derived ST-Block can be very different from the micro-DAG.

Figure 5(a) shows an example. After training the micro-DAG, we get weight vector $\langle 0.2, 0.3, 0.2 \rangle$. To derive the ST-block, we retain the operator with the largest weight, e.g., the second operator. However, the other two operators have relatively high weights as well, meaning that they also contribute significantly to the transformation from h_i to h_j. In contrast, in the derived ST-block, only the second operator contributes to the transformation. This represents a big gap, which makes the derived ST-block may not be optimal.



(a) Big gap, softmax (b) Small gap, softmax with τ

Figure 5: Gap between Micro-DAG and ST-Block.

To reduce the discrepancy between the derived ST-block and the extended micro-DAG, we introduce a temperature parameter τ to the $SoftMax$ function when normalizing parameter α, where we replace $\frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} exp(\alpha_{o'}^{(i,j)})}$ by $\frac{exp(\alpha_o^{(i,j)}/\tau)}{\sum_{o' \in O} exp(\alpha_{o'}^{(i,j)}/\tau)}$. Thus, h_j is computed as follows.

$$h_j = \sum_{i<j} \frac{exp(\beta^{(j)}[i])}{\sum_{i<j} exp(\beta^{(j)}[i])} \sum_{o \in O} \frac{exp(\alpha_o^{i,j}/\tau)}{\sum_{o' \in O} exp(\alpha_{o'}^{i,j}/\tau)} o(h_i),$$

When $\tau \to 0$, the output of $SoftMax$ is getting closer to a one-hot vector. In this way, for each node pair (h_i, h_j), the operator with largest $\alpha_o^{i,j}$ is dominant, thus making it very close to the derived ST-block that only retains the operator with the largest weight. Figure 5(b) shows an example with a small gap when using the temperature parameter τ. In the micro-DAG, the 2nd operator plays a dominant role for the transformation from h_i to h_j, and the other two operators contribute only slightly. Thus, when ignoring the other two operators in the derived ST-block, the gap is insignificant. This ensures that the micro-DAG with optimal architecture parameters α and β is able to derive a high-performance ST-block.

In practice, if we set the temperature τ to be very small from the beginning, the training process can be unstable. Therefore, we set the initial value of τ to be relatively large, and perform exponential annealing on τ to reduce it gradually as training epochs increase.

3.2.3 Reducing Operator Set O. Rather than using all S/T operators in the literature, we propose two principles to select a compact set of S/T operators to construct O with the aim of achieving high search efficiency without compromising accuracy. First, selecting S/T operators that capture different perspectives is purposeful. To this end, we categorize existing S/T operators according to their characteristics. Second, for each category of S/T operators, we choose the most effective variant. This helps reduce operator set O without losing promising operators.

We categorize commonly used S/T operators for correlated time series forecasting in Table 1. Specifically, we categorize the T-operators into three families—the Convolutional Neural Network (CNN) family, the Recurrent Neural Network (RNN) family, and the Attention family; and we categorize the S-operators into two families—the Graph Convolution Network (GCN) family and the Attention family.

For all equations in Table 1, $Z \in \mathbb{R}^{N \times T \times D}$ denotes the input tensor and $H \in \mathbb{R}^{N \times T' \times D'}$ denotes the output tensor after applying an S/T operator to Z. Here, N represents the number of time series or nodes in the graph, T and T' represent the number of

Table 1: Categorization of S/T Operators for Correlated Time Series Forecasting.

	Family	Operator	Literature	Equation
T-Operators	**CNN**	1D Convolution	[14]	$H^{(i)} = Z^{(i)} * W$ (8)
		Gated Dilated Causal Convolution (GDCC)	[9, 17, 51]	$H^{(i)} = (Z^{(i)} * W_1) \odot \sigma(Z^{(i)} * W_2)$ (9)
	RNN	Long Short Term Memory (LSTM)	[23, 38]	$H_t^{(i)} = LSTM(Z_t^{(i)}, H_{t-1}^{(i)})$ (10)
		Gated Recurrent Unit (GRU)	[1, 4, 28]	$H_t^{(i)} = GRU(Z_t^{(i)}, H_{t-1}^{(i)})$ (11)
	Attention	Transformer	[34, 46]	$H^{(i)} = SoftMax(\frac{(Z^{(i)}W_Q)(Z^{(i)}W_K)^\top}{\sqrt{D'}})(Z^{(i)}W_V)$ (12)
		Informer (INF-T)	[53]	$H^{(i)} = SoftMax(\frac{smp(Z^{(i)}W_Q)(Z^{(i)}W_K)^\top}{\sqrt{D'}})(Z^{(i)}W_V)$ (13)
S-Operators	**GCN**	Chebyshev GCN	[9, 11, 14, 17, 51]	$H_t = \sum_{k=0}^{K-1} W_k T_k(\tilde{L}) Z_t$ (14)
		Diffusion GCN (DGCN)	[28, 33, 45]	$H_t = \sum_{k=0}^{K} (D_O^{-1}A)^k Z_t W_1^k + (D_I^{-1}A^\top)^k Z_t W_2^k$ (15)
	Attention	Transformer	[34, 46]	$H_t = SoftMax(\frac{(Z_tW_Q)(Z_tW_K)^\top}{\sqrt{D'}})(Z_tW_V)$ (16)
		Informer (INF-S)	None	$H_t = SoftMax(\frac{smp(Z_tW_Q)(Z_tW_K)^\top}{\sqrt{D'}})(Z_tW_V)$ (17)

timestamps, and D and D' represent the number of features. We use $Z^{(i)} \in \mathbb{R}^{T\times D}, H^{(i)} \in \mathbb{R}^{T\times D'}$ to represent the input and output of the i-th time series and $Z_t \in \mathbb{R}^{N\times D}, H_t \in \mathbb{R}^{N\times D'}$ to represent the input and output of the t-th timestamp. We use A to represent the adjacency matrix; W denotes convolution kernels; W_Q, W_K, and W_V represent projection matrices used in computing attention scores; D_O and D_I represent the diagonal in-degree and out-degree matrices, respectively; and $T_k(\tilde{L})$ is the Chebyshev polynomial of the adjacency matrix. Finally, $*$ is the convolution operator, σ represents the sigmoid function, $smp(\cdot)$ is a sampling function used in Informer, and $\odot$ is the element-wise product.

Applying Principle 1: We analyze the different perspectives of different families for T-operators and S-operators, respectively. For T-operators, we consider two perspectives—(i) the ability of modeling long-term temporal dependencies and (ii) efficiency. Since short-term temporal dependencies can be relatively easily captured by all families, we do not consider it as a perspective. Figure 6 shows the CNN, RNN, and Attention families w.r.t. the two perspectives.

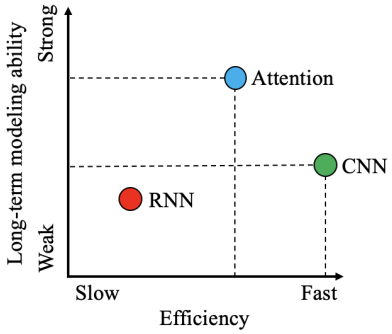


Figure 6: Comparison among Different T-operator Families.

For the CNN family, the core operation is to convolve multiple kernels, e.g., matrices or vectors, with different parts of Z to extract meaningful features. The kernel size is often set to be small, which leads to a small receptive field that considers only local features and limits its ability of modeling long-term temporal dependencies. To model long-term temporal dependencies, it is possible to stack multiple CNN layers to expand the receptive filed [20, 44, 45]. Since convolutions at different parts of the input tensors are independent, CNNs can be easily parallelized and thus being very efficient.

For the RNN family, the core operation is to compute a hidden state $H_t = f(Z_t, H_{t-1})$ for each timestamp t. Since H_t is computed based on H_{t-1}, i.e., the hidden state from the previous timestamp $t-1$, such recursive computations cannot be parallelized. Thus, RNN is inefficient. Some recent studies have shown that CNN is able to outperform RNN in capturing long-term dependencies [12, 18, 32]. This explains why RNN is placed lower than CNN in Figure 6.

For the Attention family, the core operation is, for each timestamp, to compute an attention score with each other timestamp. Then, a weighted sum based on the attention scores can be computed. Since the attention scores are computed w.r.t. all timestamps, this enables the Attention family to be very good at capturing long-term temporal dependencies. In addition, since the attention score computations for different timestamps are independent, they can be easily parallelized and thus being efficient. However, the CNN family has better efficiency than the Attention family in practice.

To conclude the discussion on the T-operators, we disregard the RNN family from our search space because CNN and Attention are more efficient and capture long-term temporal dependencies better. We keep both the CNN and Attention families because one is not better than the other one on both perspectives. We provide empirical evidence to justify this design choice in Section 4.

We proceed to analyze three different perspectives for S-operators—(i) whether an adjacency matrix is required, (ii) the ability of capturing time-varying spatial correlations, and (iii) efficiency.

The GCN family relies on an adjacency matrix that indicates the neighboring nodes of each node. The adjacency matrix is often constructed based on meta information, e.g., the distances among the

sensors which produce the time series or is learned from data [1, 45]. Given an adjacency matrix, for each node, graph convolution convolves features of neigboring nodes using a learnable kernel such that the features from the neighbors are aggregated. The aggregations per node are independent and thus can be done in parallel. As a result, GCN is efficient. The spatial correlations between two time series can be different across time, e.g., the correlations on two roads' traffic time series in peak vs. offpeak hours. However, the adjacency matrix is often constructed based on distance that does not change across time, this makes GCN fail to capture dynamic spatial correlations.

The Attention family computes attention scores w.r.t. all other nodes for each node. Thus, it does not require an adjacency matrix that indicates the neighboring relationships. Next, the attention scores can be computed based on hidden representations at different timestamps, and thus it is able to capture time-varying dependencies. In terms of efficiency, attention scores at different nodes are independent and thus are parallelizable and efficient.

Table 2 summarizes the GCN vs. Attention families w.r.t. the perspectives of interest. We observe that the two families complement each other and thus we keep both families in our search space.

Table 2: Comparison among Different S-operator Families.

Perspectives	*GCN*	*Attention*
Needs predefined adjacency matrix	Yes	No
Captures time-varying spatial correlations	No	Yes
Efficiency	Fastest	Fast

Applying Principle 2: After determining the relevant S/T operator families, we apply the second principle to choose the most effective variant for each family. To do so, we consider two scenarios. If there exist studies that compare the different variants in the same experimental setting, we then directly choose the most effective variant. If such studies do not exist, we conduct experiments to identify the most effective variant.

For the CNN family, we consider 1D Convolution and Gated Dilated Causal Convolution (GDCC). The computations of both operators are shown in Equations 8 and 9 in Table 1, which clearly indicates that GDCC is an enhanced version of 1D convolution. In addition, a recent paper [8] has shown strong empirical evidence that GDCC is more effective than 1D convolution. Thus, for the CNN family, we include only GDCC into operator set $\mathcal{O}$.

For the temporal Attention family, we have two candidates—Transformer [41] and its more efficient variant, Informer [53]. Informer improves the attention mechanism in Transformer by only sampling a subset of timestamps to calculate the attention score with all the other timestamps, denoted by $smp(\cdot)$ in Equation 13. In addition, Informer has been shown to be able to also achieve more accurate forecasting than Transformer on time series forecasting tasks [53]. Thus, we include only Informer, denoted by INF-T as it concerns temporal dependencies, into $\mathcal{O}$.

For the GCN family, we consider Chebyshev GCN [22] and Diffusion GCN [28]. Although the two variants are commonly used in the literature, there is no existing studies compare the two variants in a consistent experimental setting for CTS forecasting. We thus design an experiment to compare them. This experiment is conducted on two datasets, namely METR-LA and PEMS03 (see the deatils of the two datasets in Section 4.1). The results in Table 3 show that the diffusion GCN consistently outperforms the Chebyshev GCN. Therefore, we include only diffusion GCN into $\mathcal{O}$.

Table 3: Comparison of GCN and Attention Variants, MAE.

	DGCN	Cheby GCN	Informer	Transformer
METR-LA	**3.33**	3.42	**3.64**	3.65
PEMS03	**18.44**	21.55	23.79	**23.54**

For the spatial Attention family, only Transformer is used in the literature. However, since Informer achieves better accuracy on modeling temporal dependencies, it motivates us to consider it on modeling spatial correlations. We thus conduct an experiment to compare them. Table 3 shows that they have similar accuracy. Since Informer is more efficient than Transformer, we include Informer, denoted by INF-S as it concerns spatial correlations, into $\mathcal{O}$.

To summarize, we include GDCC, INF-T, DGCN, and INF-S as the S/T operators in our micro space. In addition, we also include two non-parametric operators, zero and identity. This yields a compact operator set $\mathcal{O}$ with 6 operators.

3.3 Macro Search Space

We design a macro search space to search for topologies among different ST-blocks. This enables *AutoCTS* to generate ST-backbones with heterogeneous ST-blocks connected by flexible topologies.

Specifically, we represent the macro search space as a macro-DAG with B nodes, where each node b_i, $1 \leq i \leq B$, represents an ST-block, and an edge (b_i, b_j) stands for information flow from node b_i to node b_j (see Figure 7(a)). Note that the predecessor of node b_1 is the embedding layer. An information flow from b_i to b_j means that the output representation of ST-block b_i is fed into as the input representation of ST-block b_j. This is different from the micro-DAG, where an edge indicates some operators that transform the representations.

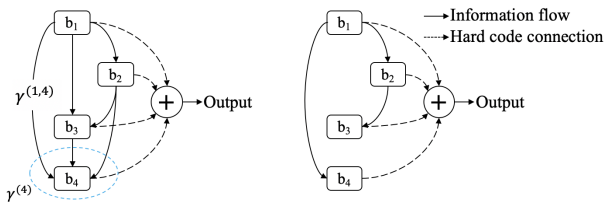
In addition to the information flows which are to be learned, we also have hard code connections from all ST-blocks to the output layer. In other words, no matter which topology is learned to connect the ST-blocks, the outputs of all ST-blocks are merged and fed to the output layer.

The final, learned ST-backbone is a subgraph of the macro-DAG, where only one incoming edge is retained for each node, i.e., each ST-block (see Figure 7(b)).

To enable the learning, we introduce the third architecture parameter γ to parameterize the information flows among ST-blocks. Let $e_{in}^{(j)}$ and $e_{out}^{(j)}$ be the input and output representations of ST-block b_j, respectively. We use a scalar-valued parameter $\gamma^{(i,j)}$ to represent the weight of edge (b_i, b_j), and calculate $e_{in}^{(j)}$ as the weighted sum of all its predecessors' outputs.

$$e_{in}^{(j)} = \sum_{i<j} \frac{exp(\gamma^{(i,j)})}{\sum_{i<j} exp(\gamma^{(i,j)})} e_{out}^{(i)} \tag{18}$$

At the end of the learning, each ST-block b_j is connected to the precedent b_i with the largest $\gamma^{(i,j)}$.



(a) Marco-DAG (b) Derived ST-backbone

Figure 7: Marco Search Space.

To enable heterogeneous ST-blocks, we allow the ST-blocks to have different micro architectures. This is achieved by using distinct micro architecture parameters for each ST-block. Specifically, we use architecture parameters α_i and β_i to parameterizing the micro search space of ST-block b_i. The joint search space, which is composed of both the micro and macro search spaces, is parameterized by $\Theta = (\{\alpha_i, \beta_i\}, \gamma)$.

3.4 Search Strategy

The goal of architecture search is to learn the architecture parameter $\Theta = (\{\alpha_i, \beta_i\}, \gamma)$ by training the macro-DAG, governed by γ, and multiple heterogeneous micro-DAGs governed by $\{\alpha_i, \beta_i\}$, in an end-to-end manner. We design a search strategy to achieve this.

The learning of *AutoCTS* follows a two-stage strategy—(i) *architecture search* and (ii) *architecture evaluation*. In the *architecture search* stage, we run *AutoCTS* on the training set to search for an optimal ST-backbone. To do this, we first divide the training data evenly into a pseudo-training data $\mathcal{D}_{train}$ and a pseudo-validation data $\mathcal{D}_{val}$, which are used to train both the architecture parameters Θ and the network weights w, e.g., kernels in CNNs and GCNs, projection matrices in Attentions. Specifically, we adopt a bi-level optimization algorithm to optimize Θ and w.

$$\min_{\Theta} \quad \mathcal{L}_{val}(w^*, \Theta) \tag{19}$$

$$s.t. \quad w^* = argmin_w \mathcal{L}_{train}(w, \Theta), \tag{20}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ denote the losses (e.g., mean absolute error or mean squared error) on the pseudo-training and pseudo-validation data, respectively. We employ first-order approximation to speed-up the architecture search [30]. The detailed training process is shown in Algorithm 1, where η and ξ are the learning rates for the two optimizers for Θ and w, respectively.

Algorithm 1 Joint Search Algorithm

Input: Correlated time series $\mathcal{X} \in R^{N \times T \times F}$, Adjacency matrix G;

1: Randomly initialize $\Theta = (\{\alpha_i, \beta_i\}, \gamma)$ and w. Split training data into pseudo train data $\mathcal{D}_{train}$ and pseudo validation data $\mathcal{D}_{val}$.
2: **While** Not exceeding the largest epoch **do**
3: Sample a mini-batch from $\mathcal{D}_{val}$.
4: Update Θ with $\Theta = \Theta - \eta \bigtriangledown_{\Theta} \mathcal{L}_{val}(w, \Theta)$
5: Sample a mini-batch from $\mathcal{D}_{train}$.
6: Update w with $w = w - \xi \bigtriangledown_{w} \mathcal{L}_{train}(w, \Theta)$.
7: **return** ST-backbone w.r.t. the learned Θ.

In the *architecture evaluation* stage, we only keep the architecture parameters Θ but discard the learned network weights w from the architecture search stage. This means that we only keep the learned neural architecture of the learned ST-backbone. Instead, we train the forecasting model with the learned ST-backbone from scratch on the original training and validation sets to obtain new network weights w'. Finally, we report the accuracy of the forecasting model with w' on the testing set.

4 EXPERIMENTS

We evaluate *AutoCTS* on both single- and multi-step time series forecasting tasks using eight correlated time series datasets from different domains to justify our design choices.

4.1 Experimental Settings

4.1.1 Datasets. To enable fair comparisons with existing studies and to facilitate reproducibility, we employ eight commonly used benchmark datasets for correlated time series forecasting, including six datasets for multi-step forecasting [1, 14, 28, 40, 45, 51] and two datasets for single-step forecasting [23, 38].

Multi-step forecasting:

- METR-LA and PEMS-BAY: Both datasets are traffic speed time series datasets, released by Li et al. [28]. The two datasets are collected from highways in the Los Angeles County and the Bay area, respectively.
- PEMS03, PEMS04, PEMS07 and PEMS08: All datasets are traffic flow time series collected from the Caltrans Performance Measurement System (PeMS), which are released by Song et al. [40].

Table 4 summarizes the statistics of the six datasets. This includes N, the number of time seriesor nodes, and T, the total number of timestamps. We adopt the same train-validation-test splits as in the original papers [28, 40], as shown in the "Split Ratio" column in Table 4. All the time series in the six dataset have a record every 5 minutes, and thus there are 12 records per hour. Following existing literature [28, 44, 45], we consider a multi-step forecasting setting where we use the recent one hour in the history (i.e., input=12 timestamps) to forecast the records in the next hour (i.e., output=12 timestamps). For each dataset, a graph is constructed where each node represents a sensor that generates a time series. The adjacency matrix represents the road network distances among the sensors [28, 40, 45, 51].

Table 4: Datasets.

Dataset	N	T	Split Ratio	Input	Output
METR-LA	207	34,272	7:1:2	12	12
PEMS-BAY	325	52,116	7:1:2	12	12
PEMS03	358	26,208	6:2:2	12	12
PEMS04	307	16,992	6:2:2	12	12
PEMS07	883	28,224	6:2:2	12	12
PEMS08	170	17,856	6:2:2	12	12
Solar-energy	137	52,560	6:2:2	168	1
Electricity	321	26,304	6:2:2	168	1

To enable direct and fair comparisons with existing studies [1, 28, 40, 44, 45], for METR-LA and PEMS-BAY, we report accuracy of

the forecasts on the 3rd, 6th, and 12th timestamps, corresponding to the next 15-min, 30-min, and 60-min, respectively; for PEMS03, PEMS04, PEMS07, and PEMS08, we report the average accuracy over all 12 future timestamps.

Single-step forecasting:

- Solar-Energy: The solar power production records collected from 137 PV plants in the Alabama State, released by Lai et al. [23]
- Electricity: The electricity consumption records collected from 321 clients, released by Lai et al. [23].

The statistics of the two datasets are also summarized in Table 4. We also use the same train-validation-test splits as the original paper [23]. Similar to the multi-step forecasting, we consider a well-known single-step forecasting setup to enable fair comparions with existing studies. Specifically, we use the historical 168 timestamps (i.e., input=168 timestamps) to predict the value in a single future timestamp (i.e., output=1 timestamp). The single future timestamp is either 3 or 24. There is no predefined adjacency matrix for Solar-Energy and Electricity datasets.

4.1.2 Evaluation Metrics. Following the evaluation methods in previous studies [23, 28, 44, 45], we use mean absolute error (MAE), root mean squared error (RMSE), mean absolute percentage error (MAPE) to evaluate the accuracy of multi-step forecasting, and use Root Relative Squared Error (RRSE) and Empirical Correlation Coefficient (CORR) to measure the accuracy of single-step forecasting. For MAE, RMSE, MAPE, and RRSE, lower values indicate higher accuracy, while larger CORR values indicate higher accuracy.

4.1.3 Baselines. We compare *AutoCTS* with eight methods, including seven methods that are manually designed by human experts and one automated approach. The implementations of the baselines are based on the public-available code released by their authors.

- DCRNN: Diffusion convolutional recurrent neural network uses diffusion GCN with GRU to build ST-blocks, and employs an encoder-decoder architecture for multi-step forecasting [28].
- STGCN: Spatio-temporal graph convolutional network adopts a Chebyshev GCN and a gated 1D convolution to build ST-blocks [51].
- Graph WaveNet: It employs diffusion GCN and GDCC to build ST-blocks [45].
- AGCRN: Adaptive graph convolutional recurrent network combines enhanced Chebyshev GCN and GRU to build ST-blocks [1].
- LSTNet: A long- and short-term time-series network, which combines 1D convolution and GRU to extract short-term and long-term temporal dependencies [23].
- TPA-LSTM: An attention based recurrent neural network [38].
- MTGNN: A multivariate time series forecasting model with graph neural networks, which utilizes a spatial-based GCN and GDCC to construct ST-blocks [44].
- AutoSTG: A NAS based method for automated spatio-temporal graph prediction, which uses only diffusion GCN and 1D convolution as the S/T operators in the search space for only ST-blocks but not ST-backbones, and employs meta learning to learn the weights for the diffusion GCN and 1D convolution [33].

4.1.4 Implementation Details. All the model training experiments are conducted on Nvidia Quadro RTX 8000 GPUs. The source code is available at https://github.com/WXL520/AutoCTS.

Architecture Search. Following Liu et al. [30], we use the ReLU-operator-BN order for all parametric operators to improve the training stability. We vary the number of nodes in the micro-DAG M among 3, **5**, and 7, and vary the number of nodes in the macro-DAG B among 2, **4**, 6, with default values shown in bold, for all datasets. We adopt Adam [21] as the optimizer for both the architecture parameters Θ and the network weights w. For Θ, we set the initial learning rate to 3×10^{-4}, the momentum to $(0.5, 0.999)$, and the weight decay to 10^{-3}. For w, we set the initial learning rate to 10^{-3}, and the weight decay to 10^{-4}. We adopt partial channels [47] to improve the memory efficiency, where we select $1/4$ features during training. For all datasets, we set the initial temperature τ to 5.0 and use exponential annealing with a multiplicative factor of 0.9 to gradually reduce it as training evolves until it reaches 0.001.

4.2 Experimental Results

4.2.1 Multi-step Forecasting Accuracy. Tables 5 and 6 present the overall accuracy of *AutoCTS* and the baselines on multi-step forecasting datasets. We use bold to highlight the best accuracy and underline the second best accuracy. Since AutoSTG relies on additional information on the road network to enable meta learning based weight generation, and such information is unavailable on the four PEMS datasets, AutoSTG is thus unable to work on the four PEMS datasets.

Key observations are as follows. First, *AutoCTS* outperforms all manually designed models on all multi-step forecasting tasks, demonstrating that *AutoCTS* is able to produce very competitive ST-blocks and ST-backbones that outperform human designed models.

Second, when comparing to the other automated approach AutoSTG, although AutoSTG includes additional features, such as GPS coordinates of sensors, to enhance its forecasting accuracy, it is still inferior to *AutoCTS* except for the MAE at the 60-min timestamp on METR-LA. This is due to (i) AutoSTG only include diffusion GCN and 1D convolution to construct the search space, while we follow the proposed two principles to select a compact yet complementary S/T operators. (ii) AutoSTG only searches for the micro architecture of a single ST-block, and then stacks the ST-blocks to build the forecasting model. In contrast, we jointly search for both the micro architecture of ST-blocks and the macro architecture of the ST-backbone.

Third, *AutoCTS* outperforms AGCRN and DCRNN, which both employ GRU to model temporal dependencies. This justifies our design choices that disregard the RNN family in the micro search space.

Fourth, there does not exist a single manually-designed model that consistently outperforms other manually-designed models. For example, MTGNN outperforms Graph WaveNet on METR-LA, but is outperformed by Graph WaveNet on PEMS04. This suggests that the optimal neural architectures for different datasets may be different, which implies that it is beneficial to be able to automatically identify forecasting models with unique architectures for different datasets, which is what *AutoCTS* is able to offer.

4.2.2 Single-step Forecasting Accuracy. Table 8 shows the experimental results on the two single-step forecasting datasets. We observe that: (1) *AutoCTS* and MTGNN outperform LSTNet and

Table 5: Accuracy of Multi-step Forecasting, METR-LA and PEMS-BAY.

Data	Models	15 min			30 min			60 min			Parameters
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	
METR-LA	DCRNN	2.77	5.38	7.30%	3.15	6.45	8.80%	3.60	7.60	10.50%	372,353
	STGCN	2.88	5.74	7.62%	3.47	7.24	9.57%	4.59	9.40	12.70%	119,176
	Graph WaveNet	2.69	5.15	6.90%	3.07	6.22	8.37%	3.53	7.37	10.01%	309,400
	AGCRN	2.83	5.45	7.56%	3.20	6.55	8.79%	3.58	7.41	10.13%	751,650
	MTGNN	2.69	5.18	6.86%	**3.05**	6.17	8.19%	3.49	7.23	9.87%	405,452
	AutoSTG	2.70	5.16	6.91%	3.06	6.17	8.30%	**3.47**	7.27	9.87%	509,048
	AutoCTS	**2.67**	**5.11**	**6.80%**	**3.05**	**6.11**	**8.15%**	**3.47**	**7.14**	**9.81%**	358,520
PEMS-BAY	DCRNN	1.38	2.95	2.90%	1.74	3.97	3.90%	2.07	4.74	4.90%	372,353
	STGCN	1.36	2.96	2.90%	1.81	4.27	4.17%	2.49	5.69	5.79%	119,648
	Graph WaveNet	**1.30**	2.74	2.73%	1.63	3.70	3.67%	1.95	4.52	4.63%	311,760
	AGCRN	1.35	2.83	2.87%	1.69	3.81	3.84%	1.96	4.52	4.67%	752,830
	MTGNN	1.32	2.79	2.77%	1.65	3.74	3.69%	1.94	4.49	4.53%	573,484
	AutoSTG	1.31	2.76	2.73%	1.63	3.67	3.63%	1.92	4.38	4.43%	553,932
	AutoCTS	**1.30**	**2.71**	**2.69%**	**1.61**	**3.62**	**3.55%**	**1.89**	**4.32**	**4.36%**	395,984

Table 6: Accuracy of Multi-step Forecasting, PEMS03, PEMS04, PEMS07, and PEMS08.

Data	Metric	DCRNN	STGCN	Graph WaveNet	AGCRN	MTGNN	*AutoCTS*
PEMS03	MAE	18.18	17.49	14.82	15.89	15.10	**14.71**
	RMSE	30.31	30.12	25.24	28.12	25.93	**24.54**
	MAPE	18.91%	17.15%	16.16%	15.38%	15.67%	**14.39%**
PEMS04	MAE	24.70	22.70	19.16	19.83	19.32	**19.13**
	RMSE	38.12	35.55	30.46	32.26	31.57	**30.44**
	MAPE	17.12%	14.59%	13.26%	12.97%	13.52%	**12.89%**
PEMS07	MAE	25.30	25.38	21.54	21.31	22.07	**20.93**
	RMSE	38.58	38.78	34.23	35.06	35.80	**33.69**
	MAPE	11.66%	11.08%	9.22%	9.13%	9.21%	**8.90%**
PEMS08	MAE	17.86	18.02	15.13	15.95	15.71	**14.82**
	RMSE	27.83	27.83	24.07	25.22	24.62	**23.64**
	MAPE	11.45%	11.40%	10.10%	10.09%	10.03%	**9.51%**

Table 7: Search time (GPU hours) and memory (MB).

DataSet	Search Time	Memory
METR-LA	21.43	20,109
PEMS-BAY	52.60	32,117
PEMS03	25.95	34,401
PEMS04	14.64	30,681
PEMS07	61.74	33,057
PEMS08	12.34	16,521
Solar-Energy	163.21	30,977
Electricity	145.68	36,339

TPA-LSTM. This is because LSTNet and TPA-LSTM do not explicitly model the correlations among different time series. In contrast, both *AutoCTS* and MTGNN simultaneously model the temporal and spatial dependencies. *AutoCTS* does not outperform MTGNN much for single-step forecasting when compared to multi-step forecasting. This suggests that MTGNN is already a very effective model that behaves similarly to the optimal model identified from the search space of *AutoCTS* for single-step forecasting, but is less effective for multi-step forecasting. This further justifies the needs for automated solutions for identifying specific, optimal models for different forecasting tasks. (2) *AutoCTS* achieves the best accuracy on both short-term (see Table 5 and Table 6) and long-term (see Table 8) datasets. This is because our search space contains GDCC and INF-T, which are good at modeling both short- and long-term dependencies, respectively, which enables *AutoCTS* to generate high-performance models in both cases.

4.2.3 Ablation Studies. We conduct ablation studies to justify the design choices used in *AutoCTS*. We only report results on PEMS03 and put the results on other datasets in a technical report [43]. In particular, we compare *AutoCTS* with the following variants: (1) w/o design principles: this variant does not follow the proposed two principles for selecting a compact set of S/T operators. Rather, it includes all operators in Table 1. (2) w/o temperature: it does not use the temperature parameter τ to reduce the gap between the micro-DAG and the derived ST-block. (3) w/o macro search: this variant only searches for a single optimal ST-block and then sequentially stacks ST-blocks with residual connections to build an ST-backbone.

Table 8: Accuracy of Single-step Forecasting.

Data		Solar-Energy		Electricity	
Models	Metric	3	24	3	24
LSTNet	RRSE	0.1843	0.4643	0.0864	0.1007
	CORR	0.9843	0.8870	0.9283	0.9119
TPA-LSTM	RRSE	0.1803	0.4389	0.0823	0.1006
	CORR	0.9850	0.9081	0.9439	0.9133
MTGNN	RRSE	0.1778	0.4270	0.0745	0.0953
	CORR	0.9852	0.9031	0.9474	0.9234
AutoCTS	RRSE	**0.1750**	**0.4143**	**0.0743**	**0.0947**
	CORR	**0.9855**	**0.9085**	**0.9477**	**0.9239**

(4) macro only: it employs four existing human designed ST-blocks as the atomic search units and only searches for ST-backbones. The selected ST-blocks come from STGCN [51], DCRNN [28], Graph WaveNet [45], and MTGNN [44]. We consider (1) the accuracy of the models identified by the different variants, and (2) the runtime in GPU hours that it takes to identify the models.

Table 9 shows that: (1) *AutoCTS* achieves better accuracy than its variant w/o design principles, and costs much less GPU hours for architecture search. This demonstrates the effectiveness of the proposed principles for selecting a compact and complementary S/T operators from Table 1. (2) The proposed temperature parameter helps reduce the gap and find more accurate models with similar GPU hours. (3) Disabling the macro search and search for the stacking of homogeneous ST-backbone lowers the performance without significantly decreasing the searching time. This suggests our joint search space and strategy is effective and efficient. (4) *AutoCTS* significantly outperforms the macro only variant, which justifying that S/T operators are more suitable to be used as the atomic search units in the search space than manually-designed ST-blocks. Although the macro only variant is very efficient, due to its small search space, it is unappealing as many human designed models, such as Graph WaveNet, AGCRN, and MTGNN, outperform it.

Table 9: Ablation Studies, PEMS03

Models	MAE	RMSE	MAPE	GPU hours
AutoCTS	**14.71**	**24.54**	**14.39%**	25.95
w/o design principles	15.66	25.51	15.28%	126.25
w/o temperature	14.87	24.93	14.64%	25.97
w/o macro search	15.07	25.22	14.84%	25.89
macro only	15.83	26.12	15.77%	**15.90**

4.2.4 Parameter Sensitivity Analysis. We proceed to evaluate the impact of key hyperparameters in *AutoCTS*, including M, i.e., the number of nodes in an ST-block in the micro search space, and B, i.e., the number of ST-blocks B in the macro search space, and *Edge*, i.e., the number of incoming edges per node in the derived ST-block. We use $B = 4$, $M = 5$, and $Edge = 2$ as default values. We then vary M among {3, 5, 7}, vary S among {2, 4, 6}, and vary *Edge* among {2, 3}, while keeping the rest to their default values. Due to the space limitation, we report only the results on PEMS03, and put the results on other datasets in a technical report [43].

Table 10: Impact of M and B, PEMS03.

M	MAE	RMSE	MAPE	B	MAE	RMSE	MAPE
3	14.95	25.36	15.18%	2	14.92	25.11	15.03%
5	**14.71**	**24.54**	**14.39%**	4	**14.71**	**24.54**	**14.39%**
7	14.82	25.23	14.51%	6	14.80	24.73	14.45%

Table 10 shows that *AutoCTS* achieves the best accuracy under $M = 5$ and $B = 4$. Decreasing M or B reduces the expressiveness of *AutoCTS* and thus the accuracy of the automatically identified models. A larger M or B increases the complexity of the micro and macro search space, resulting in potentially more overfitting problems when the training data is not abundant. Thus, it slightly degrades the accuracy. In addition, larger M or B may lead to models that use significantly more parameters than the baseline models.

Table 11 shows minimal accuracy improvements when *Edge* increases from 2 to 3, while the training time shows a clear increase. This suggests that using 2 edges per node yields sufficiently complex internal topologies for ST-blocks and avoids introducing too many parameters and thus maintaining good efficiency.

Table 11: Impact of *Edge*, PEMS03.

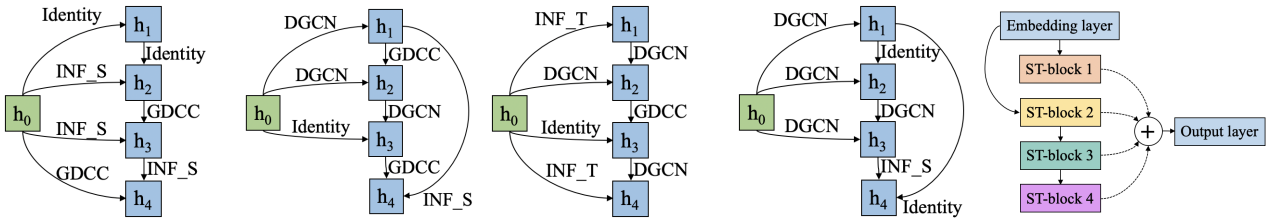
Edge	MAE	RMSE	MAPE	Training (s/epoch)
2	14.71	24.54	**14.39%**	**149.3**
3	**14.58**	**24.20**	15.40%	204.0

4.2.5 Case Study. We show the architecture of the forecasting model on PEMS03 in Figure 8. As Figures 8(a), 8(b), 8(c) and 8(d) show, each ST-block has a distinct internal architecture. In particular, the four ST-blocks contain all S/T operators in the micro search space, including 5 GDCC, 2 INF-T, 5 INF-S and 10 DGCN. This indicates the effectiveness of the proposed micro search space. The ST-backbone consists of the four heterogeneous ST-blocks, which are assembled by diverse topologies. This justifies the needs of enabling topologically flexible, heterogeneous ST-backbone, which most existing models fail to support.

4.2.6 Transferability. Since manually designed forecasting models are often applied to different datasets, it is pertinent to investigate the transferability of forecasting models learned by *AutoCTS* to assess how such models compare with traditional, manually designed models. To this end, we consider a "Transferred Model" that is identified automatically by *AutoCTS* on the PEMS03 dataset, as shown in Figure 8. We apply this model to make forecasts on datasets METR-LA and PEMS-BAY. The results are shown in Table 12, where *AutoCTS* denotes the automatically identified model on METR-LA or PEMS-BAY. The transferred model achieves competitive accuracy on METR-LA and PEMS-BAY. Although the transferred model is not as good as the model that is directly learned by *AutoCTS* on the specific dataset, it is able to outperform the baselines on most metrics, especially in the case of PEMS-BAY (cf. Table 5). This is evidence that *AutoCTS* is able to produce effective and transferable forecasting models.

4.2.7 Search Time & Memory Costs. For the architecture search phase, we consider the runtime and memory that *AutoCTS* takes. Table 7 shows that the search time varies from 12.34 to 163.21 GPU hours across datasets, depending on the number of time series/nodes, the total number of timestamps, and the length of the input time window. The searching process takes up to ca. 36 GB memory, which can fit into the memory of a single modern GPU.

The last column in Table 5 shows the number of parameters of AutoCTS and baseline models. AutoCTS often uses fewer parameters than do MTGNN, AGCRN, and AutoSTG, uses more parameters than does STGCN (whose accuracy is among the worst), and is comparable to the other methods. To understand how models identified by *AutoCTS* and baseline models compare in terms of time and space, we report the training time (seconds per epoch), inference



(a) ST-block 1 (b) ST-block 2 (c) ST-block 3 (d) ST-block 4 (e) Forecasting Model

Figure 8: The Automatically Searched Forecasting Model on the PEMS03 Dataset.

Table 12: Transferability: Transferred Model is Searched on PEMS03, *AutoCTS* is Searched on METR-LA or PEMS-BAY.

Data	Models	15 min			30 min			60 min		
		MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
METR-LA	Transferred Model	2.72	**5.11**	6.90%	3.08	**6.09**	8.28%	3.50	**7.12**	10.02%
	AutoCTS	**2.67**	**5.11**	**6.80%**	**3.05**	6.11	**8.15%**	**3.47**	7.14	**9.81%**
PEMS-BAY	Transferred Model	**1.30**	2.73	2.73%	1.62	3.63	3.63%	1.90	4.35	4.51%
	AutoCTS	**1.30**	**2.71**	**2.69%**	**1.61**	**3.62**	**3.55%**	**1.89**	**4.32**	**4.36%**

time (milliseconds per window), and total number of parameters on all datasets in a technical report [43].

5 RELATED WORK

We categorize existing studies on CTS forecasting into two categories—manually designed models vs. automated designed models.

Manually Designed Models. Recent deep learning models achieve the state-of-the-art accuracy in correlated time series forecasting. Such models rely on different types of human designed ST-blocks that capture both temporal dependencies and spatial correlations. Table 13 summarizes the ST-blocks in the literature according to two dimensions—temporal dependencies modeling (including the CNN, RNN and attention families) vs. spatial correlation modeling (including the GCN and attention families).

Table 13: Categorization of Human Designed ST-blocks.

	CNN	RNN	Attention
GCN	[9, 11, 14, 17, 44, 45, 51]	[1, 4, 16, 28]	[14]
Attention	[14]	None	[46, 52]

Automatically Designed Models. Neural Architecture Search (NAS) has been employed to automatically design neural architectures for the many tasks. Existing NAS methods can be divided into evolutionary algorithm based [39], reinforcement learning based [37], performance predictor based [26] and gradient-based methods [30]. *AutoCTS* is a gradient-based method due to its high efficiency. Despite of great success in computer vision [3, 37], natural language processing [39], and AutoML systems [27, 54], little effort has been devoted to time series forecasting. AutoST [25] is proposed for spatio-temporal prediction, where the time series are from a uniform grid. The values at each timestamp are considered as an image, and then a search space that only contains convolution operators is proposed. AutoST does not apply in our setting, where time series are not necessarily from a uniform grid, making the image modeling inapplicable. AutoSTG [33] considers correlated time series forecasting. However, it differs from *AutoCTS* in the following perspective. (1) AutoSTG only considers one T-operator and one S-operator, i.e., 1D convolution and Diffusion GCN. In contrast, we propose two principles to select the most effective and efficient operators from diverse families. (2) AutoSTG only designs ST-blocks, whereas we search both ST-blocks and the ST-backbone. (3) AutoSTG relies on additional information of the graph to enable meta-learning to learn network weight w. In contrast, *AutoCTS* does not rely on such additional information but purely on the time series themselves. Thus, *AutoCTS* has a wider application scope.

6 CONCLUSION

We present *AutoCTS*, a framework that is able to automatically learn a neural network model for correlated time series forecasting. In particular, we design a micro search space with a compact set of S/T operators to find novel ST-blocks. In addition, we design a macro search space to identify the topology among heterogeneous ST-blocks to construct novel ST-backbones. Extensive experiments on eight commonly used correlated time series forecasting datasets justify the design choices of *AutoCTS*. As future work, it is of interest to include model efficiency as an additional criterion into the search strategy to automatically identify both accurate and efficient models. It is also of interest to extend *AutoCTS* to other analytics tasks, such as outlier detection [19] and trajectory analytics [49].

ACKNOWLEDGMENTS

This work was partially supported by Independent Research Fund Denmark under agreements 8022-00246B and 8048-00038B, the VILLUM FONDEN under agreements 34328 and 40567, and the Innovation Fund Denmark centre, 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] Liang-Chieh Chen, Maxwell D Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. 2018. Searching for efficient multi-scale architectures for dense image prediction. In *NeurIPS*. 8713–8724.

[4] Weiqi Chen, Ling Chen, Yu Xie, Wei Cao, Yusong Gao, and Xiaojie Feng. 2020. Multi-range attentive bicomponent graph convolutional network for traffic forecasting. In *AAAI*, Vol. 34. 3529–3536.

[5] Razvan-Gabriel Cirstea, Darius-Valer Micu, Gabriel-Marcel Muresan, Chenjuan Guo, and Bin Yang. 2018. Correlated Time Series Forecasting using Multi-Task Deep Neural Networks. In *CIKM*. 1527–1530.

[6] 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.

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

[8] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *ICML*. 933–941.

[9] Zulong Diao, Xin Wang, Dafang Zhang, Yingru Liu, Kun Xie, and Shaoyao He. 2019. Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting. In *AAAI*, Vol. 33. 890–897.

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20 (2019), 1–21.

[11] Shen Fang, Qi Zhang, Gaofeng Meng, Shiming Xiang, and Chunhong Pan. 2019. GSTNet: Global Spatial-Temporal Network for Traffic Flow Prediction. In *IJCAI*. 2286–2293.

[12] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *ICML*. 1243–1252.

[13] 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.

[14] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, Vol. 33. 922–929.

[15] Jilin Hu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. *VLDB J.* 27, 2 (2018), 179–200.

[16] Jilin Hu, Bin Yang, Chenjuan Guo, Christian S. Jensen, and Hui Xiong. 2020. Stochastic Origin-Destination Matrix Forecasting Using Dual-Stage Graph Convolutional, Recurrent Neural Networks. In *ICDE*. 1417–1428.

[17] Rongzhou Huang, Chuyin Huang, Yubao Liu, Genan Dai, and Weiyang Kong. 2020. LSGCN: Long Short-Term Traffic Prediction with Graph Convolutional Networks. In *IJCAI*. 2355–2361.

[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099* (2016).

[19] 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*.

[20] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2018. Distinguishing Trajectories from Different Drivers using Incompletely Labeled Trajectories. In *CIKM*. 863–872.

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

[22] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[23] 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.

[24] Mengzhang Li and Zhanxing Zhu. 2021. Spatial-Temporal Fusion Graph Neural Networks for Traffic Flow Forecasting. In *AAAI*, Vol. 35. 4189–4196.

[25] 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.

[26] Wei Li, Shaogang Gong, and Xiatian Zhu. 2020. Neural graph embedding for neural architecture search. In *AAAI*, Vol. 34. 4707–4714.

[27] Yang Li, Yu Shen, Wentao Zhang, Jiawei Jiang, Bolin Ding, Yaliang Li, Jingren Zhou, Zhi Yang, Wentao Wu, Ce Zhang, et al. 2021. VolcanoML: speeding up end-to-end AutoML via scalable search space decomposition. *Proc. VLDB Endow.* 14 (2021), 2167–2176.

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

[29] Huiping Liu, Cheqing Jin, Bin Yang, and Aoying Zhou. 2018. Finding Top-k Optimal Sequenced Routes. In *ICDE*. 569–580.

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

[31] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. 2019. AtomNAS: Fine-Grained End-to-End Neural Architecture Search. In *ICLR*.

[32] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. 2016. Conditional image generation with PixelCNN decoders. In *NeurIPS*. 4797–4805.

[33] 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 Graphs. In *WWW*. 1846–1855.

[34] Cheonbok Park, Chunggi Lee, Hyojin Bahng, Yunwon Tae, Seungmin Jin, Kihwan Kim, Sungahn Ko, and Jaegul Choo. 2020. ST-GRAT: A novel spatio-temporal graph attention networks for accurately forecasting dynamically changing road speed. In *CIKM*. 1215–1224.

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

[36] Simon Aagaard Pedersen, Bin Yang, and Christian S. Jensen. 2020. Fast stochastic routing under time-varying uncertainty. *Proc. VLDB Endow.* 29, 4 (2020), 819–839.

[37] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *ICML*. 4095–4104.

[38] 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.

[39] David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *ICML*. 5877–5886.

[40] 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.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*, Vol. 30.

[42] Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. 2020. Textnas: A neural architecture search space tailored for text representation. In *AAAI*, Vol. 34. 9242–9249.

[43] Xinle Wu, Dalin Zhang, Chenjuan Guo, Chaoyang He, Bin Yang, and Christian S. Jensen. 2021. AutoCTS: Automated Correlated Time Series Forecasting – Extended Version. *arXiv preprint arXiv:2112.11174* (2021).

[44] 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.

[45] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI*. 1907–1913.

[46] Mingxing Xu, Wenrui Dai, Chunmiao Liu, Xing Gao, Weiyao Lin, Guo-Jun Qi, and Hongkai Xiong. 2020. Spatial-temporal transformer networks for traffic flow forecasting. *arXiv preprint arXiv:2001.02908* (2020).

[47] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. 2019. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *ICLR*.

[48] Bin Yang, Chenjuan Guo, and Christian S. Jensen. 2013. Travel Cost Inference from Sparse, Spatio-Temporally Correlated Time Series Using Markov Models. *Proc. VLDB Endow.* 6, 9 (2013), 769–780.

[49] Sean Bin Yang, Chenjuan Guo, Jilin Hu, Jian Tang, and Bin Yang. 2021. Unsupervised Path Representation Learning with Curriculum Negative Sampling. In *IJCAI*. 3286–3292.

[50] Sean Bin Yang, Chenjuan Guo, and Bin Yang. 2020. Context-Aware Path Ranking in Road Networks. *TKDE* (2020).

[51] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In *IJCAI*. 3634–3640.

[52] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. Gman: A graph multi-attention network for traffic prediction. In *AAAI*, Vol. 34. 1234–1241.

[53] 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.

[54] Fatjon Zogaj, José Pablo Cambronero, Martin C Rinard, and Jürgen Cito. 2021. Doing more with less: characterizing dataset downsampling for AutoML. *Proc. VLDB Endow.* 14 (2021), 2059–2072.