AutoCTS++: Zero-shot Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting

Wu, Xinle; Yang, Bin; Guo, Chenjuan; Hu, Jilin; Jensen, Christian S.

Published in:
The VLDB Journal

DOI (link to publication from Publisher):
10.1007/s00778-024-00872-x

Publication date:
2024

Noname manuscript No.
(will be inserted by the editor)

AutoCTS++: Zero-shot Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting

Xinle Wu[1*] · Xingjian Wu[2*] · Bin Yang[2†] · Lekui Zhou[3] · Chenjuan Guo[2] · Xiangfei Qiu[2] · Jilin Hu[2] · Zhenli Sheng[3] · Christian S. Jensen[1]

Received: date / Accepted: date

Abstract Sensors in cyber-physical systems often capture interconnected processes and thus emit correlated time series (CTS), the forecasting of which enables important applications. Recent deep learning based forecasting methods show strong capabilities at capturing both the temporal dynamics of time series and the spatial correlations among time series, thus achieving impressive accuracy. In particular, automated CTS forecasting, where a deep learning architecture is configured automatically, enables forecasting accuracy that surpasses what has been achieved by manual approaches. However, automated CTS forecasting remains in its infancy, as existing proposals are only able to find optimal architectures for predefined hyperparameters and for specific datasets and forecasting settings (e.g., short vs. long term forecasting). These limitations hinder real-world industrial application, where forecasting faces diverse datasets and forecasting settings.

We propose AutoCTS++, a zero-shot, joint search framework, to efficiently configure effective CTS forecasting models (including both neural architectures and hyperparameters), even when facing unseen datasets and foreacsting settings. Specifically, we propose an architecture-hyperparameter joint search space by encoding candidate architecture and accompanying hyperparameters into a graph representation. We then introduce a zero-shot Task-aware Architecture-Hyperparameter Comparator (T-AHC) to rank architecture-hyperparameter pairs according to different tasks (i.e., datasets and forecasting settings). We propose zero-shot means to train T-AHC, enabling it to rank architecture-hyperparameter pairs given unseen datasets and forecasting settings. A final forecasting model is then selected from the top-ranked pairs. Extensive experiments involving multiple benchmark datasets and forecasting settings demonstrate that AutoCTS++ is able to efficiently devise forecasting models for unseen datasets and forecasting settings that are capable of outperforming existing manually designed and automated models.

Keywords Correlated time series · Task-aware architecture-hyperparameter comparator · Joint search · Zero-shot

1 Introduction

Many systems, including societal infrastructures such as transportation systems, electricity grids, and sewage systems [6, 49, 52, 72], include cyber-physical components that encompass multiple sensors that each emit a time series, resulting in multiple time series that are often correlated [7, 10, 32, 33, 46, 71, 79]. For example, inductive-loop detectors in a vehicular transportation system measure the time-varying traffic volume at different road locations, and measurements along the same or nearby roads often correlate. The forecasting of future values from correlated time series often has important applications [5,22,51,68,76]. For example, accurate forecasting of traffic volumes can facilitate the prediction of congestion and near-future travel times, in turn enabling, e.g., more effective vehicle routing [21,50,70].

The key to successful correlated time series forecasting is the ability to capture both the temporal dependencies among historical values of each time series

∗ Contribute equally
† Corresponding author
1 Department of Computer Science, Aalborg University, Aalborg, Denmark
2 School of Data Science & Engineering, East China Normal University, Shanghai, China
3 Algorithm Innovation Lab, Huawei Cloud Computing Technologies, Hangzhou, China

and the spatial correlations across different time series. Leveraging the powerful feature extraction capabilities of deep learning models, different neural architectures, called ST-blocks, have been proposed to capture spatio-temporal (ST) dependencies to enable accurate forecasting. Traditionally, human experts have designed manually ST-blocks and have chosen accompanying hyperparameter settings. However, this is a resource-intensive endeavor for which human expertise is ill-suited.

A more recent approach is to automate the design of effective ST-blocks [48, 65]. Figure 1(a) outlines a typical automated framework. A search space, represented as a supernet, contains a massive number of possible ST-blocks, any subnet of which is a candidate ST-block. Nodes in the supernet and subnets represent latent representations, and the directed edges between them represent different operators (e.g., convolution, graph convolution, and Transformer). In a supernet, the transformation from node h_i to node h_j is a weighted sum of all candidate operators, while in a subnet only one operator between each node pair is kept. The goal is to learn the operator-associated weights $\{\alpha_i\}$, upon which an optimal subnet is obtained by picking the operator with the highest weight between every two nodes.

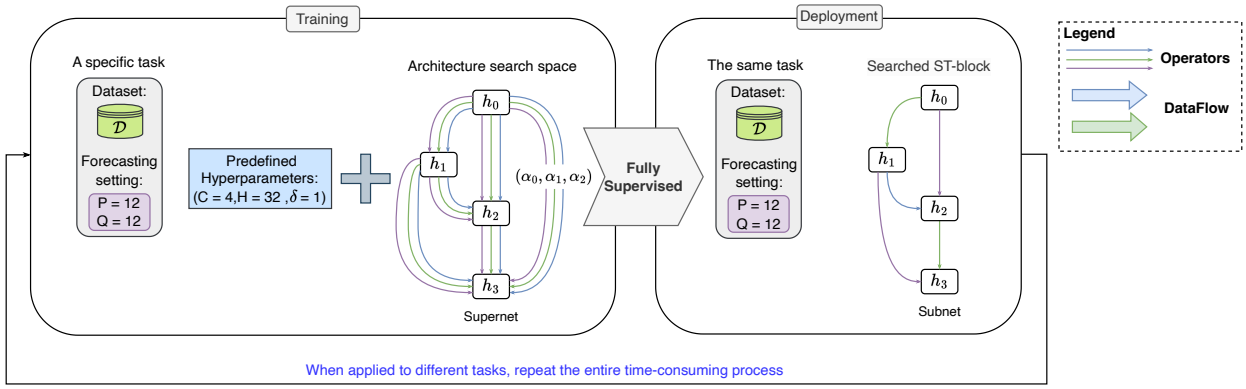
Although existing automated CTS forecasting methods achieve design automation and are capable of better performance than manually designed models, they suffer from two main limitations.

(1) Lack of support for joint architecture-and-hyperparameter search. When training supernets, existing automated CTS forecasting methods rely on predefined hyperparameters, including architectural hyperparameters (e.g., the number of latent representations, i.e., nodes, in an ST-block and the size of a latent representation) and training hyperparameters (e.g., the dropout rate). Depending on the chosen hyperparameter settings, the same architecture can yield markedly different performance. In spite of this, existing solutions rely on an expert to choose hyperparameters settings, which may well lead to the choice of a suboptimal architecture and which renders the framework only "semi-automated." Alternatively, it is possible to combine a hyperparameter optimization method (e.g., grid search or Bayesian optimization) sequentially with an existing automated architecture search method [48, 65] to achieve a two-step, automated approach. However, the running time would be excessive as existing automated architecture search needs to be performed each time a hyperparameter set is sampled. Rather, an efficient, joint architecture and hyperparameter search scheme is called for.

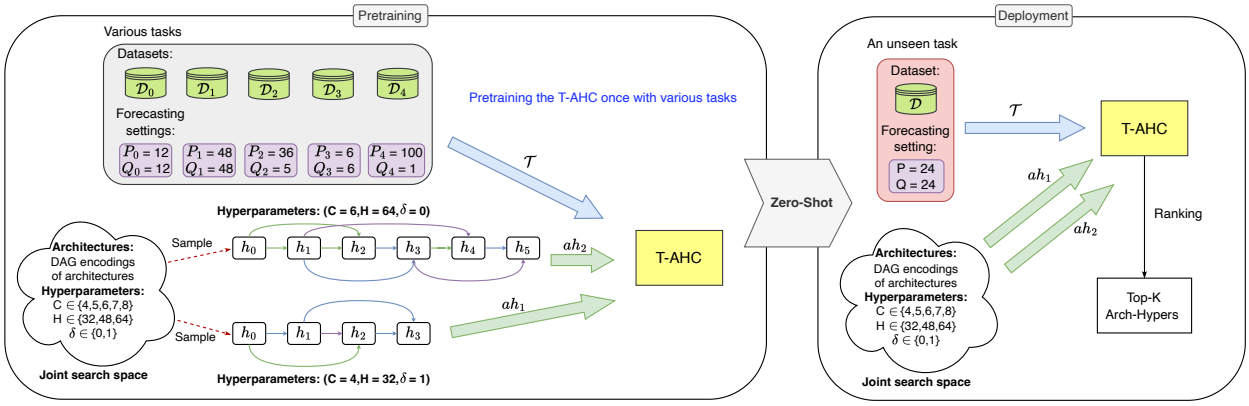
(2) Lack of support for zero-shot search ability for unseen tasks. Existing automated CTS forecasting methods are often fully supervised and task-specific—they search for an optimal forecasting model for a given task, where a task is defined by a specific dataset and a specific forecasting setting (e.g., taking P historical time steps into account to forecast Q future time steps). This fully-supervised approach hinders application in real-world industrial settings, where forecasting must be made for diverse tasks, e.g., involving many different datasets and forecasting settings with many P and Q values. It is expensive and unattractive to perform a new search from scratch whenever receiving a new task, as each search takes many (from a few to hundreds) GPU hours. This calls for zero-shot search approaches that are able to efficiently identify competitive forecasting models for diverse tasks, including unseen datasets and forecasting settings.

To eliminate the above two limitations, we propose AutoCTS++, a zero-shot, neural architecture and hyperparameter joint search framework. First, we design a joint search space that contains a wide variety of architecture-hyperparameter (arch-hyper) pairs, and then solve the problem of finding the optimal arch-hyper in this joint search space, thus addressing the first limitation. For example, with existing supernet based methods, the number C of nodes in an ST-block must be set before searching. Thus, the example in Figure 1(a) can only search for ST-blocks with $C = 4$ nodes. In contrast, our joint search space considers multiple values of C, e.g., {4, 5, 6, 7, 8} in Figure 1(b), which allows searching for ST-blocks with different numbers of nodes. The example shows two arch-hyper pairs, where pair ah_1 has $C = 4$ ST-blocks and pair ah_2 has $C = 6$ ST-blocks.

Second, we propose a novel Task-aware Architecture Hyperparameter Comparator (T-AHC) that is able to rank candidate arch-hypers from the joint search space for an unseen task, where a task is specified by a dataset and a forecasting setting given by P and Q values. Given a CTS forecasting task and two candidate arch-hypers, T-AHC estimates a binary value, indicating which arch-hyper has a better accuracy for the task. Thus, T-AHC is able to estimate a ranking of candidate arch-hypers for the task, facilitating the selection of optimal arch-hypers as final forecasting models. To ensure that T-AHC is able to return reliable rankings for unseen tasks, we pre-train T-AHC on a variety of tasks, including multiple datasets and different values for P and Q, with the aim to capture hidden relationships between tasks and model performance. This enables zero-shot joint search on unseen tasks, thus significantly enhancing the efficiency and effectiveness of automated



(a) Existing Supernet-based Framework



(b) The AutoCTS++ Framework

Fig. 1 Comparsion between the existing supernet-based framework and the proposed AutoCTS++ framework. A task includes CTS dataset $\mathcal{D}$ and P and Q which are the input and output lengths. Hyperparameter C indicates the number of nodes in an ST-block, H is the size of the hidden representations, and δ indicates whether to use dropout during training.

forecasting model design for unseen tasks. For example, the supernet-based framework in Figure 1(a) can only search for optimal ST-blocks for a specific task given by dataset $\mathcal{D}$ and forecasting settings $P = 12$ and $Q = 12$. In contrast, the framework in Figure 1(b) is pre-trained on a number of datasets and different P and Q values, which extracts the intrinsic characteristics of the tasks, and searches for optimal ST-blocks based on it, thus supporting zero-shot search for unseen tasks, e.g., for an unseen dataset $\mathcal{D}$ and unseen $P = 24$ and $Q = 24$ as shown in Figure 1(b).

To the best of our knowledge, this is the first study that enables zero-shot, joint search for architectures and hyperparameters for correlated time series forecasting. Specifically, we make the following contributions:

(1) We propose a novel search space for correlated time series forecasting to facilitate joint search for architectures-and-hyperparameter settings.
(2) We propose T-AHC, a zero-shot Task-aware Architecture-Hyperparameter Comparator to rank arch-hyper candidates for unseen tasks, thereby improving search efficiency for unseen tasks substantially.
(3) We report on extensive experiments covering multiple benchmark datasets and forecasting settings, finding that AutoCTS++ is able to efficiently devise high-quality CTS forecasting models for unseen tasks, outperforming strong baselines.

A preliminary proposal, called AutoCTS+ [66], covers part of the solution by proposing the joint search space for neural architectures and hyperparameters that we describe in Section 3.1. However, AutoCTS+ does not support zero-shot joint search for unseen tasks. In this

Table 1 Comparison of automated frameworks.

Search Space \ Search Mechanism	Fully-Supervised	Zero-Shot
Only Architectures	AutoCTS [65], AutoSTG [48], AutoSTG+ [31]	/
Architectures & Hyperparameters	AutoCTS+ [66]	**AutoCTS++**

paper, we propose a zero-shot Task-aware Architecture-Hyperparameter Comparator that is able to rank arch-hypers for unseen tasks (cf. Section 3.2). This includes (i) devising a task-aware architecture and hyperparameter comparator (T-AHC) that is able to rank arch-hyper pairs while taking forecasting tasks into account, (ii) proposing a task setting enrichment strategy along with a pre-training method for T-AHC, and (iii) reporting on extensive experiments on unseen tasks to offer insight into the effectiveness and efficiency of T-AHC.

Table 1 summarizes the main differences between the existing frameworks and AutoCTS++. First, no existing frameworks support zero-shot search on unseen tasks. Second, the supernet based frameworks AutoCTS [65], AutoSTG [48], and AutoSTG+ [31] consider only neural architectures, not hyperparameters. Third, the proposed AutoCTS++ supports both joint search and zero-shot search on unseen tasks, which fits real-world application needs better.

Section 2 covers basic concepts and definitions. Section 3 presents the design of AutoCTS++ and the pre-training method for T-AHC. Section 4 reports on the empirical study. Related work is covered in Section 5, and Section 6 concludes.

2 Preliminaries

2.1 Problem Settings

Correlated Time Series (CTS). We denote a correlated time series (CTS) by $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{N \times T \times F}$, where N is the number of time series, and each time series covers T time steps and has an F-dimension feature vector for each time step. The feature vectors in the i-th time series $\boldsymbol{X}^{(i)} \in \mathbb{R}^{T \times F} \subset \boldsymbol{\mathcal{X}}$, $1 \leq i \leq N$, are correlated with previous feature vectors in the same time series as well as with feature vectors in other time series. Therefore, it is natural to model a CTS as a graph $G = (V, E, A)$, where vertex set V represents the set of time series, edge set E represents correlation relationships between time series, and adjacency matrix $A \in \mathbb{R}^{N \times N}$ captures the strengths of the relationships between time series. A is usually predefined based on the distances of the sensors that generate the time series, or learned adaptively. Univariate and multivariate time series can be considered as special cases of correlated time series: when $N = 1$ and $C = 1$, a time series is univariate, and when $N = 1$ and $C > 1$, a time series is multivariate.

Correlated Time Series Forecasting. We consider multi-step and single-step correlated time series forecasting, both of which have important real-world applications [3, 35, 55, 67]. Given the feature vectors of the past P time steps of $\boldsymbol{\mathcal{X}}$, the goal of multi-step CTS forecasting is to predict the feature vectors of the Q future time steps, with $Q > 1$; and the goal of single-step CTS forecasting is to predict the vector at the Q-th future time step, where $Q \geq 1$. Formally, we define multi-step CTS forecasting as follows:

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

and we define single-step CTS forecasting as follows:

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

where $\boldsymbol{X}_t \in \mathbb{R}^{N \times F}$ denotes the feature vectors of all time series at time step t, $\hat{\boldsymbol{X}}$ represents the predicted feature vectors, and $\mathcal{F}$ is a CTS forecasting model.

CTS Forecasting Task. A CTS forecasting task is defined by a specific CTS dataset $\mathcal{D}$ and a specific forecasting setting with P and Q. We formalized a task as follows:

$$\mathcal{T} = (\mathcal{D}, P, Q, M), \quad (3)$$

where M indicates whether the CTS forecasting task is single-step (i.e., Q-th step) or multi-step (i.e., the next Q steps).

Problem Definition. The goal is to automatically build an optimal ST-block $\mathcal{F}^*$ from a predefined joint architecture-hyperparameter search space $\mathcal{S}$ for task $\mathcal{T}$ that minimizes the forecasting error on a validation dataset $\mathcal{D}_{val}$. The objective function is stated as follows:

$$\mathcal{F}^* = argmin_{\mathcal{F} \in \mathcal{S}} \ ErrorMetric_{\mathcal{T}}(\mathcal{F}, \mathcal{D}_{val}) \quad (4)$$

2.2 Neural Forecasting Models

As indicated in Figure 2, the common framework of manually designed neural CTS forecasting models has three components: an input module, an ST-backbone, and an output module. The input and output modules usually consist of one or two fully-connected layers that encode an input time series and decode extracted spatiotemporal features to forecasting values, respectively.

The ST-backbone is the core component of a CTS forecasting model. It consists of B ST-blocks that can

be connected using different topologies, with sequential stacking being a simple yet effective topology and that we use in Figure 2. An ST-block captures the spatial correlations between time series and the temporal dependencies in individual time series. There are thus two categories of operators in an ST-block, S-operators (e.g., GCNs) and T-operators (e.g., Transformer), for extracting spatial and temporal features, respectively. The specific types of S/T-operators and their connections are critical to the success of a CTS forecasting model.

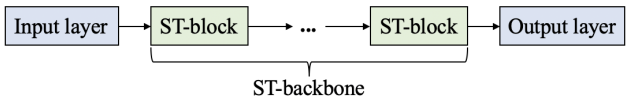


Fig. 2 An example CTS forecasting model.

2.3 Existing Automated Methods

Since the input and output modules of neural CTS forecasting models are simple, typically consisting of one or two fully-connected layers, and only involve a few design choices, such as the choice of the embedding dimension, existing automated CTS forecasting frameworks [48, 65] focus on the design of the ST-blocks. Specifically, automated frameworks typically start by designing a search space that encompasses a wide variety of ST-block architectures. The search space is represented by a directed acyclic graph (DAG) (Figure 1(a) left), dubbed a supernet, with C nodes and a number of edges. Each node h_i, $0 \leq i \leq C-1$, denotes a latent representation. Each node pair (h_i, h_j), has $|\mathcal{O}|$ directed edges from h_i to h_j, $i < j$, corresponding to $|\mathcal{O}|$ candidate S/T-operators, where $\mathcal{O}$ is a predefined candidate operator set consisting of S/T-operators, such as CNNs, GCNs, and Transformers.

The goal of existing automated frameworks is to obtain ST-blocks and a consequent CTS forecasting model by selecting operators and connections that minimize validation errors. To achieve this, a vector $\alpha^{(i,j)} \in \mathbb{R}^{|\mathcal{O}|}$ is introduced to weigh the edges between each node pair (h_i, h_j). These vectors reflect the importance of the edges and are to be learned during training. Then the transformation from node h_i to node h_j is formulated as a weighted sum of all edges, i.e., operators:

$$f^{(i,j)} = \sum_{o \in \mathcal{O}} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} exp(\alpha_{o'}^{(i,j)})} o(h_i), \tag{5}$$

where $\alpha_o^{(i,j)}$ is the weight of operator $o \in \mathcal{O}$, and $o(\cdot)$ represents the transform function of operator o. Then, the latent representation of node h_j is obtained by summing all the transformations from its predecessor nodes:

$$h_j = \sum_{i<j} f^{(i,j)} \tag{6}$$

This way, a supernet can be trained on a given task, i.e., a CTS forecasting dataset and a particular forecasting setting with P and Q, using gradient descent to learn both the neural operator parameters and architecture parameter α. After training, an optimal ST-block is derived by removing the unimportant edges from the supernet, retaining only one edge between each node pair and at most two incoming edges for each node (Figure 1(a) right).

For existing automated frameworks, hyperparameters such as the number C of nodes in an ST-block need to be predefined (e.g., C is set to 4 in Figure 1(a)). In other words, existing frameworks do not support jointly searching for architectures and hyperparameters. In addition, existing automated frameworks consume substantial memory since very large supernets must reside in memory during training. Furthermore, existing automated frameworks start from scratch for each new task, which is inefficient. We aim at enabling zero-shot search that can efficiently identify competitive forecasting models for new, unseen tasks.

3 Zero-shot Joint Search

Our framework enables zero-shot joint search for an optimal ST-block, i.e., an optimal combination of a neural architecture and a set of accompanying hyperparameters, for an unseen task. Figure 1(b) offers an overview of the proposed automated CTS forecasting framework. To support zero-shot joint search, we first design a **joint search space** (Section 3.1) containing pertinent candidate architecture-and-hyperparameter pairs, each of which is called an *arch-hyper*.

We then propose a novel search framework that leverages a **Task-aware Architecture-Hyperparameter Comparator (T-AHC)** (Section 3.2) to achieve rankings of all arch-hypers in the joint search space for a given task. Specifically, T-AHC takes the encodings of a task and two candidate arch-hypers as input and produces a binary label indicating which input arch-hyper has a higher accuracy for the specific task. To this end, we pre-train T-AHC with a large number of labeled samples of the form (t, ah_1, ah_2, y) from diverse tasks, where t denotes a task, ah_1 and ah_2 are two arch-hypers, and y is a binary label indicating wehther ah_1 is more accurate than ah_2 at task t.

After obtaining the pre-trained T-AHC, we utilize a heuristic search strategy (Section 3.3) to support a

zero-shot search for optimal arch-hypers in the joint search space for unseen tasks. Given an unseen task, AutoCTS++ first generates a representation of the task in an efficient manner, e.g., in a few minutes, then T-AHC takes the task representation and any two arch-hypers as input and outputs which arch-hyper is more accurate under the task, thus efficiently generates a ranking of candidate arch-hypers. Based on the ranking, an optimal arch-hyper is selected as the final forecasting model.

3.1 Joint Search Space

We focus on the automated design of ST-blocks that are the core components of CTS forecasting models. The joint search space considers two aspects of ST-blocks: 1) the architecture, including operators and their connections, and 2) the hyperparameters, including architecture-related structural hyperparameters (e.g., the hidden dimension) and optimization-related training hyperparameters (e.g., the dropout rate). Next, we introduce the search spaces of the architecture and hyperparameters in turn, and then show how to combine these into a joint search space. Note that AutoCTS++ adopts the joint search space from AutoCTS+ [66].

3.1.1 Architecture Search Space

In an ST-block, S/T-operators extract spatial/temporal features, and the connections between operators control the information flow.

Candidate operators. By studying manually designed CTS forecasting models and the search spaces of existing automated CTS forecasting frameworks, we identify and include two compelling candidate T-operators. The Gated Dilated Causal Convolution (GDCC) [48,65,68] can effectively capture *short-term* temporal dependencies. In contrast, the Informer (INF-T) [81], which is a variant of the Transformer, excels at learning *long-term* temporal dependencies.

We also identify and include two S-operators for extracting two sorts of spatial features. The Diffusion Graph Convolution Network (DGCN) [43], as demonstrated in many popular CTS forecasting studies [48,65, 68], is effective at capturing static spatial correlations. In addition, the Informer (INF-S) [81] is included due to its strength at discovering dynamic spatial correlations.

We also include an "identity" operator to support skip-connections between nodes. In this way, we obtain a candidate operator set O composed of the above five operators. The framework can easily accommodate additional operators. Specifically, to add a new operator, we first include the operator in the candidate operator set O. Then, we sample arch-hypers that include the new operator and use them to generate additional samples to retrain T-AHC. The samples collected before can be reused when retraining T-AHC, and T-AHC training is quite efficient (Section 3.2.4).

Topological connections. After selecting the candidate operators, we consider the possible topological connections among the operators within an ST-block. An ST-block can be represented as a directed acyclic graph (DAG) G_d (e.g., Figure 3 left) with C nodes, where each node h_i represents a feature representation and each edge represents an operator O_i. We propose the following topological connection rules to generate candidate ST-blocks: (1) There is at most one edge from node h_i to node h_j, and no edge is allowed from node h_j to node h_i, where $i < j$. This is to form the forward flow of a neural network. (2) The operator of an edge is selected from the chosen candidate operator set, O.

3.1.2 Hyperparameter Search Space

We consider two kinds of hyperparameters: structural and training hyperparameters. Table 2 summarizes the hyperparameters in the hyperparameter search space and also lists possible values. The framework can easily include additional hyperparameters as well as expanded ranges of values for existing hyperparameters.

Table 2 Hyperparameter search space.

Hyperparameters	*Possible values*
B (number of ST-blocks)	{2, 4, 6}
C (number of nodes in an ST-block)	{5, 7}
H (hidden dimension)	{32, 48, 64}
I (output dimension)	{64, 128, 256}
U (output mode)	{0, 1}
δ **(dropout)**	{0, 1}

Structural hyperparameters relate to the structure of an ST-block, including the number B of ST-blocks in a backbone, the number C of nodes in an ST-block, the hidden dimension H of S/T-operators, the output dimension I, and the output mode U of an ST-block. Larger B, C, H, and I generally result in more expressive ST-blocks, but also yield models that are more prone to overfitting on small datasets. The output mode U is a binary value indicating which node in an ST-block produces the output. We consider two alternative modes: one takes the last node h_{C-1} as the output, like AutoCTS [65], and the other takes the sum of all nodes $h_1, h_3, ..., h_{C-1}$ as the output, like Graph WaveNet [68].

Training hyperparameters include the dropout rate δ, which can be used to alleviate overfitting when train-

ing a deep CTS model. The value of δ can be 0 or 1, which means dropout is used or not used. A set of chosen hyperparameter values from the hyperparameter search space can be represented as a r-dimensional vector ($r = 6$ in this paper). For example, in Table 2, [2, 5, 32, 64, 0, 0] is a possible hyperparameter vector.

3.1.3 Encoding of the Joint Search Space

Having designed the architecture and hyperparameter search spaces, we combine them to construct a joint search space to support the search for an optimal arch-hyper. Performing a naive combination is infeasible as the two search spaces have different types of encodings (i.e., DAGs vs. vectors). We therefore choose to design the joint search space as a joint dual DAG. This encoding of the joint search space is easy to explore due to its efficient representation.

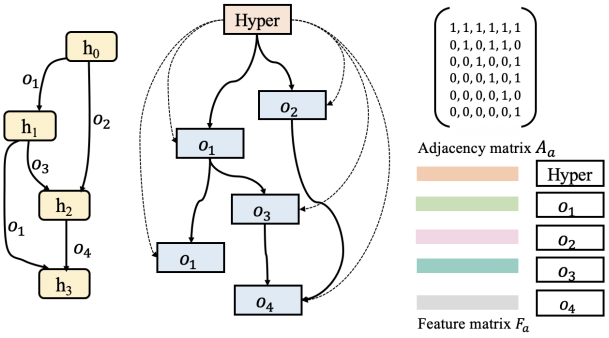


Fig. 3 Architecture DAG, Arch-hyper Graph, and its adjacency and feature matrix representations, where o_i represents the i-th operator in the candidate operator set O; the same operator o_1 is used in two different positions in the arch-hyper.

We first convert the original DAG G_d of an architecture (Figure 3 left) in the architecture search space into its dual graph G_d^* (Figure 3 middle), where nodes represent operators and edges represent information flow. This dual form facilitates learning of the representation of an arch-hyper using graph neural networks. Then, we add a new "Hyper" node that represents the hyperparameter setting of the architecture to the dual DAG (Figure 3 middle). The "Hyper" node connects to all other nodes. This way, we can use a single DAG G_a (an arch-hyper graph) to represent a complete ST-block containing both the candidate architecture and the hyperparameters, as shown in the middle of Figure 3.

We use an adjacency matrix A_a and a feature matrix F_a to encode an arch-hyper graph G_a. We consider a G_a with $n+1$ nodes ($n = 5$ in Figure 3 middle), where n nodes represent operators and one node represents the hyperparameter settings. An adjacency matrix $A_a \in \mathbb{R}^{(n+1)\times(n+1)}$ reflects the topology information of G_a, where the binary value of an entry (i, j) indicates whether there is an information flow between these two nodes. We also add self-connections to all nodes. A feature matrix $F_a \in \mathbb{R}^{(n+1)\times D}$ is also included that contains operator information of each node in an arch-hyper graph. For the "Hyper" node, the original feature is an r-dimensional vector from the hyperparameter search space. We first employ min-max normalization to normalize the original feature of the "Hyper" node and then convert the normalized feature into a D-dimensional embedding:

$$F_h = norm(H_o)W_c, \tag{7}$$

where $H_o \in \mathbb{R}^r$ is the original feature vector of the "Hyper" node, $W_c \in \mathbb{R}^{r\times D}$ is a learnable matrix, and $F_h \in \mathbb{R}^D$ is the embedding of the "Hyper" node. For the other n nodes (i.e., the operator nodes), we first embed each operator with an one-hot embedding and then introduce a learnable matrix that converts the one-hot embeddings of all operator nodes into an embedding matrix. Formally,

$$F_e = H_e W_e, \tag{8}$$

where $H_e \in \mathbb{R}^{n\times|O|}$ and $F_e \in \mathbb{R}^{n\times D}$ are the one-hot embeddings and the transformed embedding matrix of the n nodes, respectively; further, $W_e \in \mathbb{R}^{|O|\times D}$ is the learnable matrix, and $|O|$ is the number of candidate operator types in the architecture search space ($|O| = 4$ in Figure 3).

The final feature matrix $F_a \in \mathbb{R}^{(n+1)\times D}$ is the concatenation of the embeddings of the "Hyper" node and the operator nodes, i.e., $F_a = concatenate(F_h, F_e)$. This way, each arch-hyper in the joint search space can be encoded as an adjacency matrix A_a and a feature matrix F_a. The above learnable parameters W_c and W_e are learned together with the model parameters of T-AHC.

3.2 Task-aware Architecture-Hyperparameter Comparator

3.2.1 Overview

We propose a novel search framework to find the task-specific optimal arch-hyper in the joint search space. Specifically, we perform task-aware pairwise comparisons for arch-hypers in the joint search space, which is achieved by a Task-aware Architecture-Hyperparameter Comparator (T-AHC). Based on the results of pairwise comparisons, we can obtain a performance ranking of arch-hypers. The top-ranked arch-hyper is expected to have the highest accuracy on the target task and is

selected as the final CTS forecasting model. Unlike in the supernet-based search framework, where a large supernet that embeds all candidate models must reside in memory, in the proposed search framework only T-AHC needs to reside in memory during search, which renders our framework more scalable.

To rank arch-hypers and to avoid evaluating all candidate pairs, many existing studies build an accuracy estimator, which can be a neural network. The accuracy estimator needs to be trained using a large volume of $(ah,\ R(ah))$ samples, where ah is an arch-hyper and $R(ah)$ is the validation accuracy of a fully trained ah. This way, the accuracy of all candidate arch-hypers in the search space can be estimated, and top-ranked arch-hypers can be selected. However, it is computationally expensive to collect a large amount of $(ah,\ R(ah))$ samples for training an accuracy estimator, since it involves fully trainings of many ah to get their validation accuracy $R(ah)$. Further, absolute accuracy is not necessary to achieve the ranking of ah; rather, the relative comparison of two arch-hypers is sufficient to obtain their ranking.

Given these considerations, we propose to use a comparator to achieve the accuracy relation of two candidate arch-hypers. Specifically, we first design an AHC that takes the encodings of two arch-hypers ah_1 and ah_2 as input and outputs a binary value y, indicating which arch-hyper may have higher validation accuracy. Since the binary relation facilitates obtaining a linear ordering, we can use the AHC to obtain the predicted-accuracy based ranking of arch-hypers in the search space. Given a measured $(ah,\ R(ah))$ pairs, we can build $a(a-1)$ training samples for AHC in the form of $(ah_1,\ ah_2,\ y)$ by pairing every two of $(ah,\ R(ah))$ pairs, where y is a binary value indicating which arch-hyper has higher accuracy, thus alleviating the problem of requiring a large amount of training samples.

One issue with AHC is that we need to re-collect $(ah,\ R(ah))$ pairs for each target task to find the optimal arch-hyper specific to that task. This takes substantial GPU resources, making it impractical for use in many industrial settings. To address this issue, we propose to capture the similarity of different CTS forecasting tasks, where task similarity refers to the closeness of the arch-hyper performance rankings on different tasks. Intuitively, an arch-hyper that performs well on one task also performs well on similar tasks. Therefore, we propose T-AHC that uses a task embedding learning module to learn hidden representations of CTS forecasting task, and similar tasks are encouraged to have similar hidden representations. T-AHC is implemented on the basis of AHC by taking the hidden representation of the task as an additional input and outputs prediction for the current target task, which makes AHC task-aware. In order to enable the proposed task embedding learning module to measure the similarity between different CTS forecasting tasks, we pre-train T-AHC on a large number of CTS forecasting tasks to enable the task embedding learning module to output architectural performance ranking-related hidden representations for any CTS forecasting task. This way, when applied to an unseen task, we first extract the hidden representation using the proposed task embedding learning module. Together with a pair of arch-hypers, T-AHC is then able to perform a pairwise comparisons on the tasks.

3.2.2 Task Embedding Learning Module

AutoCTS++ learns a low-dimensional embedding for each CTS forecasting task, allowing the similarity of different tasks to be captured in the latent space. To develop a module that is specifically tailored to the embedding of CTS forecasting tasks, we formulate two main design objectives: (i) capable of efficiently encoding both the forecasting settings, i.e., P and Q, and the CTS datasets $\mathcal{D}$ into unified embeddings, and (ii) capable of learning relationships between forecasting tasks and model performance ranking.

To achieve the first objective, we devise a task embedding module to jointly encode a dataset $\mathcal{D}$ and forecasting setting P and Q. This means that, given the same dataset $\mathcal{D}$, but different values for P and Q, the resulting embeddings should be different. To achieve this goal, we split a CTS dataset $\mathcal{D} \in \mathbb{R}^{N\times T\times F}$ into a set of multiple time series windows $\{\mathcal{D}_i\}$ using a sliding window of length $S = P + Q$. More specifically, each time series window $\mathcal{D}_i \in \mathbb{R}^{N\times S\times F}$, representing N time series, where each time series has $S = P + Q$ time steps and each time step has F features. Based on the above, to embed a forecasting task with $\mathcal{D}$, P and Q is equivalent to embed the set of time series windows $\{\mathcal{D}_i\}$.

We choose to use TS2Vec [77] to provide generic embeddings of time series data. Given a time series window, TS2Vec performs contrastive learning in a hierarchical manner over augmented context views of the time series window and captures rich semantic information of the time series window. More specifically, given a time series window $\mathcal{D}_i \in \mathbb{R}^{N\times S\times F}$, TS2Vec encodes the features into an F'-dimensional hidden space, as shown below.

$$E_i = TS2Vec(\mathcal{D}_i), \tag{9}$$

where $E_i \in \mathbb{R}^{N\times S\times F'}$. A simple multi-layer perceptron can also embed time series into a hidden space. But

this method ignores the semantic information inherent in CTS datasets thus falling short in accuracy. We show this in the ablation studies covered in Section 4.2.3.

Based on the above, time series forecasting tasks are encoded into embeddings $\{E_i\}$. Tasks involving the same dataset $\mathcal{D}$ but different forecasting settings, i.e., different P and Q values, have different time series windows $\{\mathcal{D}_i\}$. Thus, we are able to generate different embeddings, which achieves the first objective where we consider $\mathcal{D}$, P, and Q together to jointly encode a CTS forecasting task.

To achieve the second objective, if the arch-hyper rankings of two CTS forecasting tasks are similar, the two tasks should have similar embeddings. To this end, we employ a two-stacked Set-Transformer to further encode the embeddings $\{E_i\}$ returned by TS2Vec into a vector. A Set-Transformer [38] is an attention-based structure that is able to model complex interactions among elements in a set. We employ a two stacked Set-Transformer to learn ranking-aware representations for CTS forecasting tasks. The two Set-Transformer layers, which we call IntraSetPool and InterSetPool, are stacked attention-based blocks. Each of the two layers can be seen as a parameterized pooling operation. Specifically, we first compute the mean over the N time series for each time series window, as shown in Equation 10. Then, we use IntraSetPool to encode the embeddings of each time series window along the time dimension, as shown in Equation 11, whereas InterSetPool aggregates all embeddings of the time series windows to a vector, as shown in Equation 12. This procedure is also shown in Figure 4.

$$\{\hat{E}_i\} = \mathit{Mean}(\{E_i\}) \tag{10}$$

$$\{\tilde{E}_i\} = \mathit{IntraSetPool}(\{\hat{E}_i\}) \tag{11}$$

$$E' = \mathit{InterSetPool}(\{\tilde{E}_i\}), \tag{12}$$

where $\hat{E}_i \in \mathbb{R}^{S \times F'}$, $\tilde{E}_i \in \mathbb{R}^{F'_1}$, and $E' \in \mathbb{R}^{F'_2}$. The F'_1 and F'_2 are the hidden dimensions resulting from using the Set-Transformer. By utilizing the two-stacked Set-Transformer, we are able to preserve and extract more useful information compared to using other downsampling methods. Note that the learnable parameters in Set-Transformer are optimized end-to-end, which enables Set-Transformer to generate architecture performance-aware task representations.

3.2.3 The Architecture of T-AHC

Figure 4 shows the proposed T-AHC. The input to T-AHC is a tuple (t_i, ah_1, ah_2), where t_i is the preliminary embedding of task $\mathcal{T}_i$ generated by TS2Vec, while each of ah_1 and ah_2 is encoded by the adjacency matrix and feature matrix of its joint graph, as described in Section 3.1. Given the preliminary task embedding E, we can obtain E_2 encoded by the task embedding learning module mentioned above. Considering the powerful ability of Graph Isomorphism Networks (GINs) [69] to distinguish any two graphs, we leverage GINs to encode the arch-hyper as a compact continuous embedding. Given an adjacency matrix A_a and the feature matrix F_a, the corresponding GIN can be expressed recursively as follows:

$$\mathit{GIN}(A_a, F_a) = H^{(L_n)} \tag{13}$$

$$H^{(k)} = \mathit{MLP}^{(k)}((1+\epsilon^{(k)}) \cdot H^{(k-1)} + A_a H^{(k-1)}), \quad k = 1, 2, ..., L_n, \tag{14}$$

where L_n is the number of GIN layers, $H^{(0)} = F_a$, ϵ is a trainable bias, and MLP is a multi-layer perceptron. To simplify the representation of an arch-hyper, we use the latent representation of the "Hyper" node from $H^{(L)}$ as the representation of the entire arch-hyper, denoted as l_a, since the "Hyper" node connects to all other nodes in the arch-hyper graph.

We encode two input arch-hypers ah_1 and ah_2 as l_a and l'_a, respectively, using the same GIN and then concatenate them in the feature dimension:

$$l_a = \mathit{GIN}(A_a, F_a), l'_a = \mathit{GIN}(A'_a, F'_a) \tag{15}$$

$$L_a = \mathit{concatenate}(l_a, l'_a). \tag{16}$$

Then, we use two fully-connected (FC) layers to separately extract deeper information from task embedding E' and arch-hyper-pair embedding L_a and then concatenate them in the feature dimension:

$$L'_a = FC_L(L_a, w_l) \tag{17}$$

$$\tilde{E}' = FC_E(E', w_e) \tag{18}$$

$$O = \mathit{concatenate}(L'_a, \tilde{E}'), \tag{19}$$

where w_l and w_e are the parameter matrices of the fully-connected layers FC_L and FC_E.

Finally, we feed O into a classifier that is composed of a two-stacked fully-connected layer and a Sigmoid function $\sigma(\cdot)$. We use 0.5 as the threshold to force the output of the classifier to only be 0 (less than the threshold) or 1 (larger than the threshold). Formally,

$$O' = FC_O(O, w_o) \tag{20}$$

$$\mathit{output} = \mathbb{1}(\sigma(FC_{O'}(O', w_{o'})) \geq 0.5), \tag{21}$$

where w_o and $w_{o'}$ are the parameter matrices of the two-layer fully-connected layer, and $\mathbb{1}(\cdot)$ is an indicator

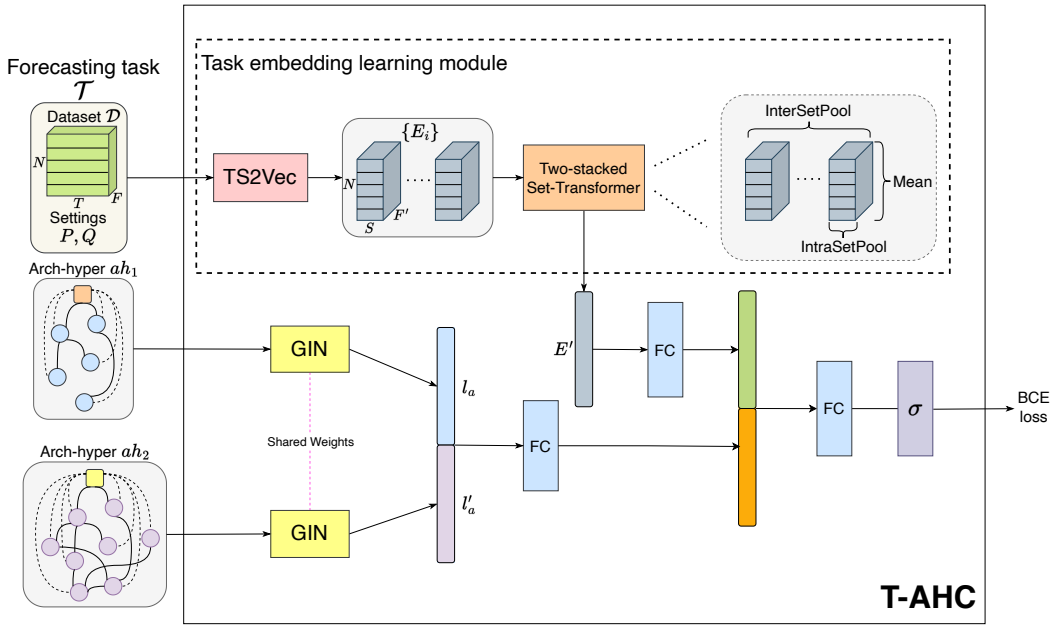


Fig. 4 Task-aware Architecture-Hyperparameter Comparator.

function with $\mathbb{1}(X) = 1$ if X is true, and $\mathbb{1}(X) = 0$, otherwise. We say that $R(ah) \geq R'(ah)$ holds if the output of T-AHC is 1, and that $R(ah) < R'(ah)$ holds if the output of T-AHC is 0. We optimize W_c, W_e, and the parameters of T-AHC using Binary Cross-Entropy (BCE) loss.

3.2.4 Pre-training a T-AHC

Enriching Tasks. Pre-training a T-AHC to enable it to generalize to return reliable arch-hyper rankings across different CTS forecasting tasks requires a large volume of training samples of the form (t_i, ah_1, ah_2, y). For a particular task $\mathcal{T}_i$, we need to train and evaluate both ah_1 and ah_2 for the task. We need to consider sufficient tasks and collect sufficient training samples for each task to adequately pre-train a T-AHC. This poses two challenges. First, although multiple correlated time series data sets exist, we may still lack sufficient amounts of correlated time series. Second, for each task, obtaining a sample (t_i, ah_1, ah_2, y) requires significant time, as we need train and evaluate ah_1 and ah_2.

To address the first challenge, we propose to split commonly used CTS datasets to generate a large number of CTS forecasting tasks such that they provide sufficient samples to pre-train a T-AHC. In order to ensure the quality of the subsets under specific task settings, we adopt some guidelines. First, we take into account the correlation between the lengths of P and Q and the size of the CTS dataset. For instance, a dataset with few time steps is not suitable for long-term forecasting as neural networks are unable to adequately capture the patterns with only a few samples. Thus, such datasets should be associated with smaller P and Q values. We consider the inherent temporal-wise continuity and spatial-wise correlation present in CTS datasets. Specifically, to keep the temporal-wise information, we derive temporally continuous subsets from the original CTS datasets. Second, we randomly sample variables and reconstruct adjacency matrices for the sampled variables, which helps to preserve spatial-wise information. Figure 5 provides a visual illustration of our approach.

To address the second challenge, we choose to use the early-validation metric $R'(\cdot)$ [66] to approximate the true label y with $\mathbb{1}(R'(ah_1) \geq R'(ah_2))$:

$$R' = ErrorMetric(\mathcal{F}(ah)_k, \mathcal{D}_{val}), \tag{22}$$

where $\mathcal{F}(ah)_k$ is the CTS forecasting model under arch-hyper ah with only k epochs of training. We find that the early-validation metric can approximate the true label well with $k = 5$ training epochs.

Selecting Shared Samples. After generating a large number of CTS forecasting tasks, we collect $2L$ samples of the form: $(t, ah, R'(ah))$ from each task, where t is the preliminary representation generated by TS2Vec. Considering that two arch-hypers may have different performance rankings on two different tasks, depending on the similarity of the two tasks, we sample the same L arch-hypers for all tasks to emphasize this point, while additionally sampling L different arch-hypers for

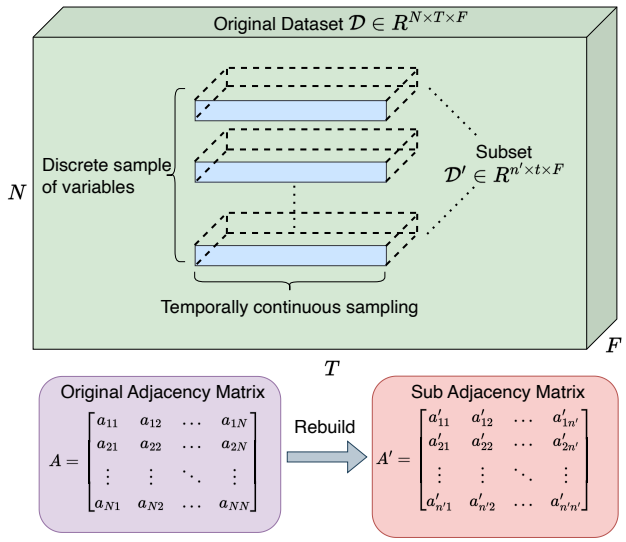


Fig. 5 Method for creating subsets.

each task to increase the diversity of training samples. Note that the L shared arch-hypers consist of relatively easy knowledge. This is because T-AHC can focus on the pairwise comparison results of the same arch-hyper pairs on different tasks and can learn directly the similarity of different tasks reflected in the L arch-hypers' rankings.

However, the L randomly sampled arch-hypers contain more challenging knowledge because the arch-hyper pairs and tasks are completely different, making it hard for T-AHC to capture the hidden relationships. Moreover, we hope that T-AHC has the generalization ability to generate a reliable ranking of all arch-hypers in the joint search space on an unseen task. To facilitate this learning process on all $2L$ samples, we employ a data-level curriculum learning approach [4]. This approach enhances the training set by gradually adding samples from the L random ones to the L shared ones. As a result, the difficulty of the training increases over time, enabling T-AHC to adapt adequately. In the experiment, we offer evidence of the effectiveness of the L shared samples (see the ablation studies Section 4.2.3). To avoid overfitting, we also adopt the dynamic pairing method used in recent studies [9, 19]. This method dynamically generates pairs in batches and shuffles them in every epoch. The detailed training process is shown in Algorithm 1.

Algorithm 1 Pre-training Algorithm

Input: n tasks $\{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_n\}$, $\mathcal{T}_i = (\mathcal{D}_i, P_i, Q_i)$, where $\mathcal{D}_i$ is the CTS dataset and P_i and Q_i are the task settings, joint search space Ω, T-AHC $\mathcal{N}$
Output: pre-trained T-AHC $\mathcal{N}^*$

Obtain shared samples $S_0 = \{ah \in \Omega\}, |S_0| = L$
for i = 1, ..., n **do**
  $\{E_i\} = TS2Vec(\mathcal{D}_i, P_i, Q_i)$
  Randomly sample $S_i = \{ah \in \Omega\}, |S_i| = L$
  Generate $S'_{z_i} = \{(\{E_i\}, ah, R'(ah)) | ah \in S_0\}, |S'_{z_i}| = L$
  Generate $S_{z_i} = \{(\{E_i\}, ah, R'(ah)) | ah \in S_i\}, |S_{z_i}| = L$
end for
Initialize $\Delta = 0$
for t = 1, ..., k_t **do**
  for i = 1, ..., n **do**
    $\tilde{S_{z_i}} = \{s | s \in S_{z_i}\}, |\tilde{S_{z_i}}| = \Delta$
    Curriculum setting: $C_t^{(i)} = S'_{z_i} \cup \tilde{S_{z_i}}, |C_t^{(i)}| = L + \Delta$
    Dynamically pairing: $P_t^{(i)} = \{(E^{(i)}, ah_1, ah_2, y) | ah_1, ah_2 \in C_t^{(i)}\}, |P_t^{(i)}| = L + \Delta$
  end for
  $P_t = \bigcup_{i=1}^{n} P_t^{(i)}$
  Train $\mathcal{N}$ with P_t
  Increase Δ
end for
return pre-trained T-AHC $\mathcal{N}^*$

3.3 Search Strategy

Once a pre-trained T-AHC is obtained, AutoCTS++ enables a zero-shot search for the optimal arch-hyper for unseen tasks. TS2Vec is capable of generating a preliminary embedding of an unseen task efficiently. Specifically, we remove the arch-hypers that do not contain either spatial or temporal operators since existing studies [67, 68] show that considering only temporal or spatial dependencies yields poor forecasting performance. Next, we adopt a heuristic approach, specifically an evolutionary algorithm [23] based on genetic algorithm, to find the best arch-hyper in the joint search space. We first sample K_s arch-hypers, which are paired up to produce $K_s(K_s - 1)/2$ comparison pairs of the form (t, ah_1, ah_2). Then the descending ranking of the K_s arch-hypers can be obtained easily based on the comparative performance determined by the pre-trained T-AHC. Then, we select the top k_p from the K_s arch-hypers in descending order as the initial population. Each arch-hyper has crossover and mutation probabilities p_1 and p_2, respectively, when generating new offspring in each evolution step. The offspring are added to the population, and the learned T-AHC is used to compare arch-hypers in the population and to remove inferior arch-hypers to maintain the population size at k_p. We then use the pre-trained T-AHC to heuristically search and rank the top-K arch-hypers from the search space. As the T-AHC does not guarantee transitivity, we use a Round-Robin algorithm to identify the top-K arch-hypers as the algorithm also does not rely on transitivity. More specifically, for each arch-hyper ah, we

count the number of wins (i.e., ah having a larger accuracy) against each of the remaining arch-hypers. Then, we choose the arch-hypers with the top-k largest numbers of wins. The process is outlined in Algorithm 2.

Algorithm 2 Zero-shot Search Algorithm

Input: joint search space Ω,pre-trained T-AHC $\mathcal{N}^*$, task $\mathcal{T} = (\mathcal{D}, P, Q)$, where $\mathcal{D}$ is the target CTS dataset and P and Q are the task settings
Output: optimal arch-hyper ah^*

1: Split $\mathcal{D}$ into $\mathcal{D}_{train}$, $\mathcal{D}_{val}$
2: $\{E_i\} = TS2Vec(\mathcal{D}, P, Q)$
3: Heuristic search and rank $ah \in \Omega$ with the pre-trained T-AHC: $\mathcal{N}^*(\{E_i\}, ah_1, ah_2)$
4: Train the top-K $ah \in \Omega$ on $\mathcal{D}_{train}$
5: **return** The ah^* that yields the highest validation accuracy on $\mathcal{D}_{val}$

4 Experimental Study

We report on comprehensive experiments on seven CTS datasets while using diverse forecasting settings. The results offer evidence that the proposed framework successfully eliminates the two main limitations of previous proposals, as intended. First, joint search facilitates the identification of forecasting models with higher forecasting accuracy than does architecture-only search. Second, zero-shot search enables efficient identification of competitive forecasting models for unseen tasks, with a latency of only a few minutes.

4.1 Experimental Settings

4.1.1 Tasks for pre-training and testing

To pre-train T-AHC, we select eleven benchmark datasets as source datasets, create subsets from these with varying task settings based on the guidelines presented in Section 3.2.4. This allows us to generate a variety of source tasks to pre-train T-AHC. Then, we select seven benchmark datasets with several forecasting settings as target tasks to assess the ability of T-AHC to support zero-shot search on unseen tasks. Below, we describe the source and target tasks, including their datasets and forecasting settings.

Source Tasks:

- PEMS03, PEMS04, PEMS07, and PEMS08 [56]: These datasets record the traffic flow in four different regions of California, and are collected from the Caltrans Performance Measurement System (PeMS).
- METR-LA [43]: This dataset provides statistics on traffic speeds for a duration of four months. It includes data from 207 sensors located on the highways of Los Angeles County.
- ETTh1, ETTh2, ETTm1, and ETTm2 [81]: These datasets contain information on electric power deployment, featuring 7 indicators.
- Solar-Energy [35]: This dataset contains records of solar power production gathered from 137 PV plants in the state of Alabama.
- ExchangeRate [35]: The dataset encompasses the daily exchange rates of eight foreign countries.

We choose P-12/Q-12 and P-48/Q-48 as forecasting settings, i.e, we consider only multi-step forecasting. We derive 100 subsets from the source datasets based on guidelines from Section 3.2.4, thus generating 200 source tasks to pre-train T-AHC.

Target Tasks:

- Electricity [35]: This dataset comprises electricity consumption records obtained from 321 clients.
- PEMS-BAY [43]: This dataset includes data from 325 traffic sensors in the Bay Area of California over a period of 6 months.
- PEMSD7(M) [75]: This dataset contains data from 228 traffic sensor stations in California's state highway system.
- NYC-TAXI [73]: This dataset consists of taxicab records. Collected in New York City that has been categorized into 266 virtual stations.
- NYC-BIKE [73]: This dataset includes bike orders from New York City, which have been clustered into 250 virtual stations.
- Los-Loop [80]: This dataset collects traffic speed data from 207 loop detectors along the highways of Los Angeles County for a period of seven days.
- SZ-TAXI [80]: This dataset encompasses taxi trajectory data from the Luohu District in Shenzhen, China that spans 156 major roads.

To assess the generalization capability of the framework, we create 28 unseen tasks by choosing four forecasting settings for each of the seven target datasets as follows: P-12/Q-12, P-24/Q-24, P-48/Q-48 for multi-step forecasting and P-168/Q-1 (3rd) for single-step forecasting. We summarize the statistics and split ratios of datasets in Table 3.

4.1.2 Evaluation metrics

When comparing the performance of different CTS forecasting models, we follow previous studies [3, 43, 67, 68, 75] and use mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage

Table 3 Dataset statistics. Split Ratio captures the train-validation-test split. M and S refer to Multi- and Single- step forecasting.

Dataset	N	T	Split Ratio (M)	Split Ratio (S)
PEMS-BAY	325	52,116	7:1:2	6:2:2
Electricity	321	26,304	7:1:2	6:2:2
PEMSD7(M)	228	12,671	6:2:2	6:2:2
NYC-TAXI	266	4,368	6:2:2	6:2:2
NYC-BIKE	250	4,368	6:2:2	6:2:2
Los-Loop	207	2,016	7:1:2	6:2:2
SZ-TAXI	156	2,976	7:1:2	6:2:2

error (MAPE) for estimating the accuracy of multi-step forecasting. We use Root Relative Squared Error (RRSE) and Empirical Correlation Coefficient (CORR) for estimating the accuracy of single-step forecasting. For MAE, RMSE, MAPE, and RRSE, smaller values are better, while for CORR larger values are better.

4.1.3 Baselines

To enable fair comparisons in the zero-shot scenario, we choose existing CTS forecasting models as baselines, including manually designed and auto-designed ones. Specifically, we compare AutoCTS++ with five competitive manually designed CTS forecasting models and three models designed by state-of-the-art automated methods. The results of the baselines are obtained using the originally released source code.

- MTGNN: A multivariate time series forecasting model, which employs mix-hop graph convolution and dilated inception convolution to form ST-blocks [67].
- AGCRN: An adaptive graph convolutional recurrent network that employs 1D GCNs and GRUs to form ST-blocks [3].
- PDFormer: A traffic flow forecasting model based mainly on the Transformer structure [24].
- Autoformer: A transformer-based forecasting model that incorporates time series decomposition into its backbone and replaces attention with auto-correlation [64].
- FEDformer: A transformer-based forecasting model with the same backbone as Autoformer and a frequency-enhanced attention mechanism [82].
- AutoSTG+: A supernet-based automated CTS forecasting framework that employs DGCN and 1D convolution to build the search space and introduces meta learning to learn the weights of neural operators [48]. We use the optimal model built on METR-LA with P-12/Q-12.
- AutoCTS: A supernet-based automated CTS forecasting framework that focuses on selecting optimal sets of neural operators to build search space [65]. We use the optimal model built on PEMS03 with P-12/Q-12 from the case study in the original paper.
- AutoCTS+: A comparator-based automated CTS foreacasting framework, which supports joint search for architectures and hyperparameters [66]. We use the optimal model built on PEMS08 with P-48/Q-48 from the case study in the original paper.

4.1.4 Implementation details

We have implemented means of pre-training T-AHC and of training CTS forecasting models.

Setting up T-AHC. Since we allow ST-blocks to have different numbers of nodes, the adjacency matrix A_a may differ in sizes across arch-hypers. Thus, we pad the adjacency matrices with zeros so that they are all of size 14. We set the number of layers L_n of the GINs to 4, with $D = 128$ hidden units in each layer. To pre-train T-AHC, we use Adam [34] with a learning rate of 0.001 and a weight decay of 0.0005 as the optimizer. The batch size is set to 64. Moreover, we set the representation dimension F of TS2Vec to 256. The hidden dimension F_1' and F_2' of Set-Transformer are set separately to 256 and 128. We split subsets from source datasets and configure different settings to create 200 tasks. We collect about 10,000 arch-hypers from the tasks, generate pairs for each epoch dynamically and train T-AHC for 100 epochs with an early stop patience of 5 epochs. To reduce the cost of collecting arch-hypers, we use an early-validation metric [66] and set the training epochs $k = 5$. Due to the low cost of collecting arch-hypers and the T-AHC's high utilization of samples, the framework can be generalized over the large joint search space with an acceptable cost.

To make the evolutionary algorithm work well, we set the crossover and mutation probabilities p_1 and p_2 to 0.8 and 0.2, respectively, and the population size k_p to 10. Lastly, we choose the top-3 arch-hypers from the population. Being a neural network, the T-AHC does not guarantee transitivity, meaning that highly efficient sorting algorithms cannot be utilized. But the increased search time is uncritical because AutoCTS++ saves hundreds of GPU hours by supporting zero-shot search. We also show that the search time can be controlled within an acceptable scope to achieve acceptable performance; see Section 4.2.4.

Setting up CTS forecasting models. We use MAE as the training objective to train CTS forecasting models, and we use Adam with a learning rate of 0.001 and a weight decay of 0.0001 as the optimizer. The batch size is set to 64.

Reproducibility. We conduct all experiments on eight Nvidia A800 GPUs. To support reproducibility, all source code is released at:
https://github.com/decisionintelligence/AutoCTS++.

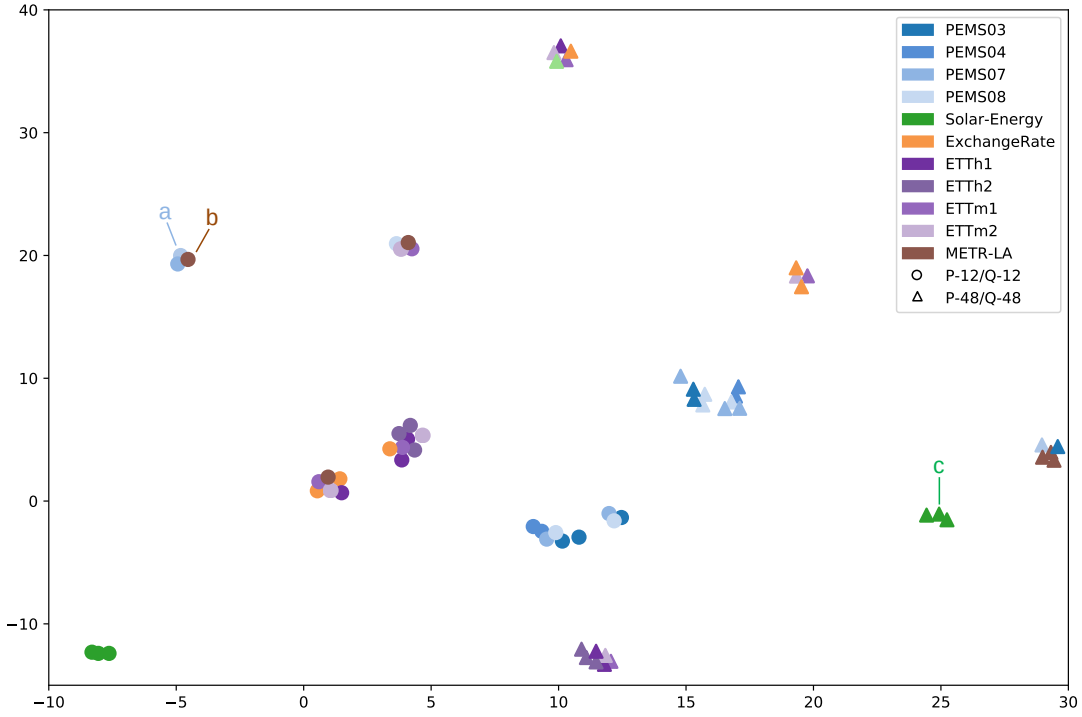


Fig. 6 Two-dimensional visualization of the embeddings of tasks. Each point represents a specific task, composed of a subset of a source dataset task settings. We use different source datasets and use circles and triangles to distinguish the forecasting settings P-12/Q-12 and P-48/Q-48. The figure shows the similarities among tasks.

4.2 Experimental Results

4.2.1 Task similarity studies

In order to discern the relationships among tasks, we randomly select three subsets from each source dataset with two different forecasting settings: P-12/Q-12 and P-48/Q-48. We then use the pre-trained T-AHC to encode the tasks and display them on a two-dimensional plane (see Figure 6). Note that the similarity of tasks is reflected in the ranking of arch-hypers. By leveraging this knowledge, T-AHC is capable of effectively evaluating task similarities by mapping them to different or the same clusters. We summarize our main observations as follows.

First, tasks with datasets from different domains often exhibit significant variations, indicated by the optimal models often changing across different domains. While tasks consist of the same dataset but different P and Q also differ. For example, with both P-12/Q-12 and P-48/Q-48 as forecasting settings, it is observed that tasks related to Solar-Energy, PEMS, and ETT belong to different clusters.

Second, the scale of a dataset affects the structures of optimal models. For example, tasks related to ExchangeRate are similar to tasks related to ETT to some extent because they both have few nodes. Their task embeddings are also similar in Figure 6. When we consider the performance of the shared arch-hypers on them, we find that arch-hypers with fewer spatial operators and more temporal operators perform better.

Third, the forecasting settings also matter. Obviously, the positions of tasks in Figure 6 with the two forecasting settings with P-12/Q-12 and P-48/Q-48, are quite dissimilar, which means that their high-dimensional coordinates are separated widely. Another example shows that tasks related to METR-LA are similar to those related to PEMS under the P-48/Q-48 setting but are dispersed into different clusters under the P-12/Q-12 setting.

To assess whether the visualized embedding similarity indeed reflects the true similarity between tasks, we select task a (a subset from PEMS08 with P-12/Q-12),

Table 4 Quantitative analysis of task similarities.

a and b		a and c		b and c	
MAE	Spear	MAE	Spear	MAE	Spear
0.0711	**0.7522**	0.2335	0.4038	0.1980	0.4927

task b (a subset from METR-LA with P-12/Q-12) and task c (a subset from Solar-Energy with P-48/Q-48) for a quantitative analysis. Specifically, we randomly sample 200 arch-hypers from the joint search space and train them on these three tasks. To quantify the similarity, we report the MAE and Spearman's rank correlation coefficient (ρ) with the arch-hypers' normalized accuracy between each pair of tasks in Table 4. The results indicate that the same arch-hyper exhibits comparable normalized accuracy between tasks a and b, while displaying a significant variation in accuracy for task c. This observation aligns with the pattern of embedding similarity illustrated in Figure 6, where the embeddings for tasks a and b are grouped closely together, in contrast to the embedding for task c, which is located at a considerable distance from the others.

To sum up, we observe intrinsic correlations between tasks, which are captured by the pre-trained T-AHC. Furthermore, by building the map between task characteristics and model rankings, AutoCTS++ is able to exploit the task characteristics to facilitate a zero-shot search for optimal arch-hypers for unseen tasks.

4.2.2 Performance comparison

Tables 5–8 report the performance of our framework and the baselines on the seven unseen CTS forecasting datasets for both multi-step and single-step forecasting. We train and test all the models five times with different random seeds and present the results as "mean±standard deviation."

To assess the generalization capability of our framework for different CTS forecasting tasks, we ensure that the seven target CTS forecasting datasets are unseen for the pre-trained T-AHC. Further, we configure three different multi-step forecasting settings: P-12/Q-12, P-24/Q-24, and P-48/Q-48, and one single-step forecasting setting: P-168/Q-1 (3rd). Among these, P-12/Q-12 and P-48/Q-48 were chosen when creating tasks for pre-training, while P-24/Q-24 and P-168/Q-1 (3rd) are new. For fair comparisons in the zero-shot scenario, we use *AutoSTG+*, *AutoCTS* and *AutoCTS+* to represent three optimal models that are obtained on METR-LA with P-12/Q-12, PEMS03 with P-12/Q-12 and PEMS08 with P-48/Q-48, respectively in past studies [65, 66]. Since the baselines do not manually tune hyperparameters under the P-24/Q-24, P-48/Q-48, and P-168/Q-1 (3rd) settings, to enable fair comparisons, we conduct grid-search for them to find the best hidden dimension H and output dimension I (2×2 times), and we also include the hyperparameter setting they use under the P-12/Q-12 setting. Since we use the same experimental settings, some of the results for MTGNN, AutoCTS, and AutoCTS+ in Tables 5–8 are obtained from the original papers. Note that PDFormer relies on a predefined adjacency matrix to generate a Laplacian matrix. As such a matrix is not available in the dataset Electricity, we use the Identity Matrix as a substitute. None of the remaining baselines and our framework require predefined adjacency matrices, as they employ self-adaptive adjacency matrices to learn correlations among time series.

We use **bold** and underline to highlight the best and the second-best results, respectively. We offer the following main observations.

First, AutoCTS++ most often achieves the best forecasting accuracy acoss all datasets and P/Q settings. In particular, it outperforms the manually and automatically designed models across different tasks. This is strong evidence of the effectiveness of AutoCTS++.

Second, AutoCTS++ achieves consistent accuracy on seven datasets with P-12/Q-12, P-24/Q-24, P-48/Q-48, and P-168/Q-1 (3rd) settings. For the baselines, although we conduct grid-search and take much time to find the best hyperparameters, this seldom yields competitive arch-hypers. In contrast, the joint search in AutoCTS++ can find high-performance arch-hypers across different P/Q settings. These findings confirm the importance of joint architecture-and-hyperparameter search as enabled by AutoCTS++.

Third, the results on the P-12/Q-12 and P-48/Q-48 tasks are generated from unseen datasets, while the results on the P-24/Q-24 and P-168/Q-1 (3rd) tasks are enabled by generalization. This is empirical evidence that the proposed framework possesses the capability of finding the most suitable combinations of architectures and hyperparameter settings for different tasks, indicating that the proposed pre-training method is effective.

We observe similar trends for single-step and multi-step forecasting. Specifically, our framework achieves the best performance in the majority of datasets and forecasting settings, indicating that the zero-shot joint search framework is successful.

4.2.3 Ablation studies

We conduct an ablation study to investigate the effectiveness of key components of the proposed zero-shot joint search framework. We report results for all

Table 5 Performance of P-12/Q-12 forecasting.

Dataset	Metric	AutoCTS++	*AutoSTG+*	*AutoCTS*	*AutoCTS+*	MTGNN	AGCRN	PDFormer	Autoformer	FEDformer
PEMS-BAY	MAE	**1.566**	1.592	1.900	1.583	1.940	1.652	1.742	1.891	1.848
		±0.009	±0.008	±0.011	±0.010	±0.008	±0.009	±0.012	±0.013	±0.020
	RMSE	**3.466**	3.620	4.350	3.526	4.490	3.815	3.920	4.335	4.248
		±0.018	±0.018	±0.023	±0.015	±0.036	±0.021	±0.017	±0.026	±0.033
	MAPE	**3.484%**	3.694%	4.510%	3.515%	4.530%	3.843%	3.947%	4.301%	4.196%
		±0.023%	±0.024%	±0.032%	±0.019%	±0.040%	±0.028%	±0.019%	±0.024%	±0.039%
Electricity	MAE	**235.353**	278.914	250.256	241.536	306.331	611.08	247.982	267.915	245.104
		±0.641	±0.730	±0.695	±0.715	±0.829	±1.740	±0.642	±0.668	±0.574
	RMSE	**2036.145**	2291.012	2244.362	2088.037	2468.959	8288.991	2178.527	2301.015	2095.128
		±5.531	±5.709	±6.820	±4.995	±7.623	±22.108	±5.012	±6.774	±4.407
	MAPE	**16.476%**	22.910%	18.045%	17.761%	24.381%	42.628%	16.864%	19.103%	16.994%
		±0.390%	±0.413%	±0.620%	±0.505%	±0.617%	±1.590%	±0.525%	±0.570%	±0.360%
PEMSD7M	MAE	**2.626**	2.655	2.674	2.701	2.643	2.697	2.631	2.648	2.633
		±0.016	±0.020	±0.013	±0.025	±0.024	±0.028	±0.015	±0.021	±0.029
	RMSE	**5.194**	5.204	5.295	5.224	5.217	5.401	5.261	5.311	5.220
		±0.014	±0.019	±0.013	±0.020	±0.028	±0.034	±0.021	±0.011	±0.017
	MAPE	**6.394%**	6.518%	6.692%	6.671%	6.523%	6.782%	6.482%	6.708%	6.510%
		±0.028%	±0.034%	±0.033%	±0.029%	±0.028%	±0.034%	±0.024%	±0.019%	±0.030%
NYC-TAXI	MAE	**5.403**	5.671	5.756	5.536	5.767	5.708	6.259	6.355	6.271
		±0.022	±0.022	±0.026	±0.021	±0.018	±0.022	±0.031	±0.023	±0.034
	RMSE	**9.513**	10.125	10.320	9.792	10.568	14.974	11.206	15.705	12.441
		±0.048	±0.056	±0.043	±0.041	±0.048	±0.051	±0.064	±0.072	±0.033
	MAPE	**39.150**	43.004	40.905	41.235	39.601%	40.327%	43.194%	46.141%	43.815%
		±0.371%	±0.322%	±0.350%	±0.373%	±0.390%	±0.357%	±0.245%	±0.377%	±0.282%
NYC-BIKE	MAE	1.978	2.019	2.037	2.026	1.997	**1.736**	2.068	1.988	2.019
		±0.010	±0.007	±0.011	±0.008	±0.005	**±0.013**	±0.011	±0.012	±0.015
	RMSE	2.954	3.204	3.114	3.102	3.035	**2.939**	3.175	3.003	3.112
		±0.022	±0.027	±0.018	±0.015	±0.018	**±0.008**	±0.014	±0.016	±0.012
	MAPE	**51.326%**	53.371%	53.059%	52.244%	51.437%	58.281%	53.861%	51.731%	52.435%
		±0.405%	±0.480%	±0.397%	±0.368%	±0.455%	±0.740%	±0.671%	±0.454%	±0.623%
Los-Loop	MAE	**3.634**	3.655	3.677	3.672	3.639	7.214	3.658	3.648	3.655
		±0.003	±0.003	±0.002	±0.002	±0.004	±0.012	±0.003	±0.002	±0.003
	RMSE	**6.904**	7.112	7.063	7.069	7.084	14.782	6.956	6.974	7.001
		±0.015	±0.016	±0.012	±0.021	±0.015	±0.194	±0.013	±0.018	±0.019
	MAPE	**10.315%**	10.695%	10.720%	10.609%	10.446%	32.641%	10.418%	10.552%	10.487%
		±0.040%	±0.037%	±0.044%	±0.035%	±0.039%	±0.041%	±0.043%	±0.048%	±0.042%
SZ-TAXI	MAE	**2.590**	3.008	3.250	3.254	3.179	3.019	2.719	2.733	2.725
		±0.007	±0.007	±0.010	±0.010	±0.009	±0.007	±0.015	±0.009	±0.007
	RMSE	**4.112**	4.237	4.544	4.557	4.474	4.229	4.364	4.301	4.278
		±0.017	±0.019	±0.017	±0.017	±0.015	±0.018	±0.022	±0.020	±0.014

datasets and settings in Tables 9–12. We again train and test all the models five times with different random seeds and present the results as "mean±standard deviation."

We compare our framework with three variants.

- **w/o TS2Vec** uses MLP to substitute TS2Vec and generates preliminary embeddings of tasks.
- **w/o Set-Transformer** uses mean-pooling to aggregate the preliminary embeddings generated by TS2Vec.
- **w/o shared samples** does not pre-train a T-AHC with shared samples but only with random samples from different tasks.

From Tables 9–12, we observe that: (1) the proposed framework consistently outperforms all variants on all evaluation metrics, indicating that each replaced component is essential to the performance of the framework; (2) *w/o Set-Transformer* is the worst, indicating that it is necessary to apply this transformer-based structure to extract and preserve the task information during pre-training; (3) *w/o shared samples* is nearly the second worst for most tasks, proving that shared samples provide more information and improve the performance of T-AHC; and (4) *w/o TS2Vec* shows extremely poor performance on some tasks, demonstrating that TS2Vec is a more powerful and robust component for encoding the tasks compared to a simple MLP.

4.2.4 Sample-limited performance study

Our framework supports zero-shot search on any unseen task by randomly sampling K_s arch-hypers from the joint search space and utilizing the task representation to rank them for top-K arch-hypers. To assess the performance of our framework in a arch-hyper sample-limited scenario, we employ five variants for K_s. Note that the main results for AutoCTS++ are produced using $K_s = 300,000$. We include the model searched by AutoCTS+ and the manually-designed PDFormer as additional baselines and perform comparisons on all seven benchmark datasets with P-24/Q-24. We train and test all the models five times with different random seeds and report means and standard deviations. We also report the search time in GPU minutes for

Table 6 Performance of P-24/Q-24 forecasting.

Dataset	Metric	AutoCTS++	*AutoSTG+*	*AutoCTS*	*AutoCTS+*	MTGNN	AGCRN	PDFormer	Autoformer	FEDformer
PEMS-BAY	MAE	**1.828**	1.923	1.911	1.836	1.874	2.039	1.843	1.879	1.903
		±0.009	±0.007	±0.012	±0.011	±0.009	±0.013	±0.012	±0.014	±0.016
	RMSE	**4.096**	4.600	4.435	4.147	4.207	4.622	4.112	4.218	4.198
		±0.010	±0.009	±0.010	±0.014	±0.014	±0.013	±0.008	±0.016	±0.13
	MAPE	**4.133%**	4.981%	4.784%	4.236%	4.387%	4.977%	4.139%	4.281%	4.191%
		±0.018%	±0.016%	±0.015%	±0.013%	±0.017%	±0.015%	±0.018%	±0.019%	±0.019%
Electricity	MAE	**184.031**	199.914	204.333	205.401	195.063	1718.216	202.571	218.910	204.332
		±0.953	±0.715	±0.669	±0.801	±1.001	±4.750	±0.685	±0.582	±0.703
	RMSE	**1537.341**	1635.016	1681.030	1784.313	1595.157	16364.798	1724.646	1891.122	1754.101
		±3.693	±3.815	±4.201	±4.566	±3.404	±35.915	±4.336	±5.017	±5.002
	MAPE	**14.532%**	15.312%	15.790%	16.085%	14.854%	58.626%	16.004%	16.077%	15.118%
		±0.280%	±0.317%	±0.320%	±0.348%	±0.314%	±3.505%	±0.290%	±0.290%	±0.304%
PEMSD7M	MAE	**3.114**	3.233	3.227	3.191	3.218	8.823	3.203	3.198	3.192
		±0.025	±0.027	±0.025	±0.028	±0.022	±0.085	±0.021	±0.019	±0.028
	RMSE	**6.134**	6.309	6.171	6.221	6.247	14.674	6.193	6.225	6.218
		±0.021	±0.022	±0.021	±0.020	±0.019	±0.018	±0.020	±0.023	±0.026
	MAPE	**7.911%**	8.546%	8.328%	8.333%	8.262%	28.915%	8.242%	8.298%	8.267%
		±0.032%	±0.034%	±0.032%	±0.037%	±0.034%	±0.460%	±0.038%	±0.048%	±0.054%
NYC-TAXI	MAE	**5.061**	5.541	6.221	5.902	5.100	6.072	5.760	5.289	5.142
		±0.020	±0.025	±0.021	±0.026	±0.022	±0.024	±0.024	±0.025	±0.024
	RMSE	**9.206**	10.011	11.834	11.106	9.347	16.772	10.506	9.877	9.458
		±0.042	±0.045	±0.039	±0.042	±0.044	±0.062	±0.036	±0.042	±0.038
	MAPE	37.709%	38.910 %	45.452%	42.283%	**35.852%**	43.417%	43.709%	37.881%	37.714%
		±0.301%	±0.335%	±0.490%	±0.383%	**±0.390%**	±0.420%	±0.431%	±0.341%	±0.387%
NYC-BIKE	MAE	1.959	2.091	2.177	2.088	2.052	**1.745**	2.049	1.980	2.143
		±0.008	±0.010	±0.012	±0.008	±0.007	**±0.012**	±0.008	±0.009	±0.011
	RMSE	**2.934**	3.241	3.440	3.232	3.146	2.941	3.147	3.000	3.301
		±0.014	±0.013	±0.014	±0.017	±0.020	±0.018	±0.022	±0.014	±0.018
	MAPE	**51.463%**	52.989%	53.915%	53.913%	53.307%	59.724%	52.648%	52.671%	53.318%
		±0.418%	±0.445%	±0.485%	±0.422%	±0.430%	±0.504%	±0.398%	±0.446%	±0.398%
Los-Loop	MAE	**3.877**	4.266	4.179	4.244	4.146	4.452	4.120	4.135	4.126
		±0.004	±0.006	±0.007	±0.008	±0.007	±0.006	±0.005	±0.010	±0.008
	RMSE	**7.513**	8.139	7.775	7.943	7.845	8.831	7.927	7.893	7.790
		±0.014	±0.013	±0.015	±0.019	±0.017	±0.016	±0.017	±0.018	±0.020
	MAPE	**11.576%**	13.284%	12.857%	13.209%	12.609%	13.720%	12.736%	12.784%	12.709%
		±0.039%	±0.041%	±0.040%	±0.040%	±0.37%	±0.042%	±0.043%	±0.044%	±0.035%
SZ-TAXI	MAE	**2.615**	3.381	3.271	3.256	3.215	2.737	3.237	3.011	2.987
		±0.007	±0.011	±0.012	±0.011	±0.011	±0.009	±0.013	±0.015	±0.018
	RMSE	**4.152**	4.770	4.559	4.547	4.528	4.342	4.542	4.390	4.388
		±0.013	±0.013	±0.016	±0.014	±0.009	±0.010	±0.014	±0.012	±0.016

all variants. For the baselines, we report the time of hyperparameter grid search. As shown in Table 13, a performance bottleneck occurs when K_s increases from 300,000 to 600,000, causing the time consumption to increase markedly. When $K_s < 300{,}000$, performance gradually decreases and fails to compete with automatically- or manually-designed baselines. Generally considering time consumption and performance, we choose $K_s = 300{,}000$ as the experimental setting in our study.

4.2.5 Efficiency study

By supporting zero-shot search for optimal arch-hypers on unseen tasks, we save tens to hundreds of GPU hours compared to the fully-supervised methods. Then we focus on the stability of the search efficiency by timing the search process on different tasks.

Specifically, we report the search time (embedding and ranking) and training time of our framework on different tasks in Figure 7. The search process consists of two phases: embedding the task and ranking arch-hypers. Although the former is correlated with the scale of the dataset, TS2Vec is capable of encoding a large-scale CTS dataset in several minutes. To enable fair comparisons, we fix the number of arch-hypers at 300,000 and report the time cost of the latter, finding that it remains stable across different tasks. Although the training time varies considerably across different tasks, the framework consistently achieves minutes-level search (i.e., embedding and ranking) time across different tasks, with dataset sizes and forecasting settings having little impact on search efficiency.

4.2.6 Case study

We show ten searched ST-blocks (arch-hypers) for different target datasets and settings in Figures 8 and 9. In Figure 8, we see the optimal arch-hypers found for the same dataset PEMS-BAY(Figures 8(a)–8(d)) change dramatically across the forecasting settings. Not only do the architectures change markedly but also do the settings of hyperparameters such as the hidden dimension H, the number of ST-blocks B, and the output mode U, exemplifying how the framework is sensitive to the forecasting settings and reacts positively to search for

Table 7 Performance of P-48/Q-48 forecasting.

Dataset	Metric	AutoCTS++	*AutoSTG+*	*AutoCTS*	*AutoCTS+*	MTGNN	AGCRN	PDFormer	Autoformer	FEDformer
PEMS-BAY	MAE	**2.057**	2.184	2.155	2.088	2.198	2.731	2.064	2.098	2.114
		±0.006	±0.005	±0.008	±0.007	±0.010	±0.006	±0.004	±0.003	±0.006
	RMSE	**4.331**	4.702	4.694	4.656	4.610	5.026	4.392	4.407	4.399
		±0.021	±0.019	±0.021	±0.022	±0.024	±0.018	±0.19	±0.20	±0.19
	MAPE	**4.920%**	5.377%	4.939%	5.012%	5.211%	6.733%	4.928%	4.975%	4.959%
		±0.026%	±0.032%	±0.0%	±0.035%	±0.036%	±0.038%	±0.029%	±0.025%	±0.026%
Electricity	MAE	**215.821**	229.435	230.258	235.367	260.799	881.452	222.223	227.015	248.105
		±0.680	±0.744	±0.601	±0.792	±0.848	±4.120	±0.698	±0.583	±0.714
	RMSE	**1839.744**	2218.553	2220.038	2293.589	1906.649	12051.841	2055.565	2191.190	2301.014
		±4.875	±5.332	±5.481	±5.665	±4.957	±38.194	±4.910	±5.326	±4.874
	MAPE	**16.681%**	17.103%	16.761%	16.739%	18.906%	45.871%	17.633%	17.602%	18.510%
		±0.382%	±0.433%	±0.458%	±0.387%	±0.397%	±2.519%	±0.421%	±0.440%	±0.525%
PEMSD7M	MAE	**3.552**	3.592	3.590	3.587	3.585	3.606	3.559	3.660	3.622
		±0.023	±0.023	±0.025	±0.023	±0.019	±0.025	±0.023	±0.025	±0.023
	RMSE	**6.801**	6.841	6.836	6.877	6.921	7.199	6.814	7.247	7.149
		±0.028	±0.030	±0.028	±0.028	±0.032	±0.035	±0.029	±0.030	±0.028
	MAPE	**9.190%**	9.274%	9.281%	9.437%	9.353%	9.579%	9.201%	9.623%	9.585%
		±0.044%	±0.047%	±0.043%	±0.049%	±0.047%	±0.046%	±0.046%	±0.044%	±0.046%
NYC-TAXI	MAE	**5.584**	5.988	6.708	5.622	6.009	6.586	5.955	5.972	5.848
		±0.026	±0.025	±0.030	±0.028	±0.029	±0.034	±0.027	±0.025	±0.024
	RMSE	**9.971**	11.334	13.199	10.174	11.419	18.049	11.863	12.301	11.874
		±0.044	±0.057	±0.065	±0.042	±0.046	±0.055	±0.042	±0.043	±0.041
	MAPE	**38.769%**	45.943%	51.636%	42.154%	46.524%	49.629%	39.576%	47.290%	46.104%
		±0.375%	±0.401%	±0.552%	±0.405%	±0.389%	±0.539%	±0.362%	±0.485%	±0.430%
NYC-BIKE	MAE	2.073	2.085	2.141	2.131	2.097	**1.713**	2.096	2.126	2.112
		±0.008	±0.006	±0.009	±0.007	±0.007	**±0.008**	±0.009	±0.010	±0.006
	RMSE	3.227	3.338	3.373	3.382	3.263	**2.900**	3.308	3.411	3.282
		±0.020	±0.023	±0.019	±0.023	±0.022	**±0.018**	±0.025	±0.026	±0.023
	MAPE	**52.110%**	53.899%	54.047%	52.661%	53.631%	59.662%	52.672%	54.381%	52.890%
		±0.408%	±0.471%	±0.486%	±0.464%	±0.470%	±0.522%	±0.405%	±0.436%	±0.462%
Los-Loop	MAE	**4.530**	4.630	4.753	4.953	4.624	8.962	4.767	4.886	4.675
		±0.006	±0.005	±0.006	±0.005	±0.008	±0.017	±0.006	±0.006	±0.005
	RMSE	**8.327**	8.557	8.715	9.116	8.549	14.956	9.003	9.056	8.878
		±0.022	±0.023	±0.020	±0.021	±0.022	±0.124	±0.025	±0.024	±0.022
	MAPE	**14.873%**	17.328%	16.228%	17.473%	15.961%	34.184%	16.028%	16.153%	16.028%
		±0.063%	±0.075%	±0.077%	±0.082%	±0.069%	±0.231%	±0.080%	±0.0721%	±0.0793%
SZ-TAXI	MAE	**2.656**	3.275	3.255	3.253	3.193	2.980	3.196	3.103	3.018
		±0.008	±0.011	±0.009	±0.010	±0.011	±0.008	±0.010	±0.013	±0.011
	RMSE	**4.193**	4.639	4.548	4.540	4.486	4.397	4.496	4.508	4.439
		±0.015	±0.018	±0.016	±0.017	±0.019	±0.014	±0.015	±0.016	±0.016

Table 8 Performance of P-168/Q-1 (3rd) forecasting.

Dataset	Metric	AutoCTS++	*AutoSTG+*	*AutoCTS*	*AutoCTS+*	MTGNN	AGCRN	PDFormer	Autoformer	FEDformer
PEMS-BAY	RRSE	**0.2873**	0.3095	0.2901	0.3207	0.2924	0.4719	0.2940	0.2984	0.2952
		±0.0003	±0.0004	±0.0003	±0.0004	±0.0002	±0.0004	±0.0003	±0.0003	±0.0003
	CORR	0.9308	0.9248	0.9275	0.9173	0.9262	0.8586	**0.9348**	0.9001	0.9281
		±0.0006	±0.0005	±0.0005	±0.0006	±0.0005	±0.0009	**±0.0006**	±0.0007	±0.0007
Electricity	RRSE	**0.0742**	0.0760	0.0756	0.0744	0.0745	0.1033	0.0781	0.0752	0.0749
		±0.0002	±0.0003	±0.0002	±0.0003	±0.0002	±0.0003	±0.0002	±0.0002	±0.0002
	CORR	**0.9477**	0.9430	0.9434	0.9467	0.9474	0.8854	0.9273	0.9289	0.9470
		±0.0005	±0.0006	±0.0005	±0.0006	±0.0008	±0.0012	±0.0008	±0.0008	±0.0007
PEMSD7M	RRSE	**0.2861**	0.2903	0.3154	0.3165	0.2981	0.5314	0.2884	0.2889	0.3017
		±0.0004	±0.0004	±0.0005	±0.0004	±0.0003	±0.0003	±0.0004	±0.0003	±0.0004
	CORR	**0.9292**	0.9238	0.9242	0.9234	0.9258	0.8186	0.9270	0.9267	0.9211
		±0.0009	±0.0008	±0.0007	±0.0008	±0.0008	±0.0007	±0.0009	±0.0010	±0.0007
NYC-TAXI	RRSE	**0.2138**	0.2478	0.2504	0.2218	0.2273	0.2709	0.2624	0.2232	0.2248
		±0.0004	±0.0003	±0.0004	±0.0003	±0.0004	±0.0004	±0.0004	±0.0003	±0.0003
	CORR	**0.8831**	0.8498	0.8457	0.8696	0.8565	0.8473	0.8417	0.8675	0.8647
		±0.0011	±0.0012	±0.0011	±0.0010	±0.0012	±0.0009	±0.0010	±0.0011	±0.0009
NYC-BIKE	RRSE	**0.7461**	0.7975	0.8910	0.7919	0.7783	0.8983	0.7531	0.7644	0.7582
		±0.0015	±0.0018	±0.0025	±0.0014	±0.0015	±0.0026	±0.0014	±0.0013	±0.0015
	CORR	**0.7902**	0.7583	0.7482	0.7568	0.7696	0.7534	0.7669	0.7662	0.7659
		±0.0008	±0.0008	±0.0008	±0.0010	±0.0011	±0.0010	±0.0009	±0.0010	±0.0008
Los-Loop	RRSE	**0.4183**	0.4359	0.4311	0.4360	0.4392	1.7565	0.4493	0.4475	0.4381
		±0.0006	±0.0005	±0.0006	±0.0006	±0.0006	±0.0021	±0.0007	±0.0008	±0.0006
	CORR	**0.8034**	0.7784	0.7797	0.7702	0.7695	0.0203	0.7699	0.7648	0.7738
		±0.0009	±0.0008	±0.0007	±0.0008	±0.0007	±0.0038	±0.0007	±0.0008	±0.0007
SZ-TAXI	RRSE	**0.4038**	0.4963	0.4815	0.4785	0.4878	1.2367	0.4892	0.5075	0.5484
		±0.0008	±0.0014	±0.0015	±0.0015	±0.0013	±0.0054	±0.0017	±0.0026	±0.0031
	CORR	**0.3414**	0.2138	0.2206	0.2200	0.2216	0.0345	0.2236	0.1984	0.1937
		±0.0018	±0.0016	±0.0017	±0.0015	±0.0018	±0.0016	±0.0016	±0.0015	±0.0016

Table 9 Ablation studies, P-12/Q-12 forecasting

Dataset	Metric	AutoCTS++	w/o TS2Vec	w/o Set-Transformer	w/o shared samples
PEMS-BAY	MAE	**1.566 ± 0.009**	1.592 ± 0.008	1.643 ± 0.009	1.593 ± 0.008
	RMSE	**3.466 ± 0.018**	3.502 ± 0.018	3.684 ± 0.016	3.519 ± 0.017
	MAPE	**3.484% ± 0.023%**	3.589% ± 0.022%	3.729% ± 0.024%	3.540% ± 0.021%
Electricity	MAE	**235.353 ± 0.641**	244.313 ± 0.638	242.416 ± 0.646	240.249 ± 0.690
	RMSE	**2036.145 ± 5.531**	2139.332 ± 5.602	2105.234 ± 5.517	2119.144 ± 5.622
	MAPE	**16.476% ± 0.390%**	18.544% ± 0.410%	18.341% ± 0.398%	16.906% ± 0.402%
PEMSD7M	MAE	**2.626 ± 0.016**	2.837 ± 0.020	3.203 ± 0.019	2.976 ± 0.018
	RMSE	**5.194 ± 0.014**	5.443 ± 0.017	6.193 ± 0.017	5.815 ± 0.015
	MAPE	**6.394% ± 0.028%**	6.887% ± 0.032%	8.242% ± 0.029%	7.234% ± 0.033%
NYC-TAXI	MAE	**5.403 ± 0.022**	5.786 ± 0.021	5.760 ± 0.024	5.511 ± 0.025
	RMSE	**9.513 ± 0.048**	10.583 ± 0.052	10.506 ± 0.049	9.673 ± 0.051
	MAPE	**39.150% ± 0.371%**	41.752% ± 0.385%	43.709% ± 0.381%	39.274% ± 0.379%
NYC-BIKE	MAE	**1.978 ± 0.010**	2.037 ± 0.009	2.006 ± 0.011	2.021 ± 0.011
	RMSE	**2.954 ± 0.022**	3.141 ± 0.022	3.043 ± 0.023	3.082 ± 0.024
	MAPE	**51.326% ± 0.405%**	51.563% ± 0.412%	52.065% ± 0.410%	52.884% ± 0.395%
Los-Loop	MAE	**3.634 ± 0.003**	3.845 ± 0.003	3.807 ± 0.002	3.690 ± 0.004
	RMSE	**6.904 ± 0.015**	7.400 ± 0.014	7.431 ± 0.017	7.127 ± 0.013
	MAPE	**10.315% ± 0.040%**	11.418% ± 0.048%	11.368% ± 0.047%	10.583% ± 0.043%
SZ-TAXI	MAE	**2.590 ± 0.007**	2.694 ± 0.006	2.967 ± 0.010	3.230 ± 0.013
	RMSE	**4.112 ± 0.017**	4.254 ± 0.020	4.192 ± 0.023	4.522 ± 0.019

Table 10 Ablation studies, P-24/Q-24 forecasting

Dataset	Metric	AutoCTS++	w/o TS2Vec	w/o Set-Transformer	w/o shared samples
PEMS-BAY	MAE	**1.828 ± 0.009**	1.895 ± 0.011	2.011 ± 0.010	1.855 ± 0.012
	RMSE	**4.096 ± 0.010**	4.439 ± 0.012	4.723 ± 0.011	4.202 ± 0.010
	MAPE	**4.133% ± 0.018%**	4.563% ± 0.020%	4.831% ± 0.021%	4.403% ± 0.023%
Electricity	MAE	**184.031 ± 0.953**	207.774 ± 1.031	201.941 ± 0.982	197.947 ± 1.004
	RMSE	**1537.341 ± 3.693**	1697.343 ± 3.852	1607.118 ± 3.740	1581.030 ± 3.923
	MAPE	**14.532% ± 0.280%**	16.217% ± 0.332%	15.533% ± 0.312%	14.761% ± 0.294%
PEMSD7M	MAE	**3.114 ± 0.025**	3.237 ± 0.029	3.441 ± 0.033	3.528 ± 0.038
	RMSE	**6.134 ± 0.021**	6.243 ± 0.025	6.503 ± 0.023	6.847 ± 0.026
	MAPE	**7.911% ± 0.032%**	8.387% ± 0.035%	8.872% ± 0.037%	9.062% ± 0.038%
NYC-TAXI	MAE	**5.061 ± 0.020**	5.786 ± 0.028	6.259 ± 0.024	5.760 ± 0.025
	RMSE	**9.206 ± 0.042**	10.583 ± 0.048	11.719 ± 0.045	10.506 ± 0.044
	MAPE	**37.709% ± 0.301%**	41.750% ± 0.342%	45.026% ± 0.369%	43.709% ± 0.351%
NYC-BIKE	MAE	**1.959 ± 0.008**	2.094 ± 0.010	2.106 ± 0.011	2.058 ± 0.009
	RMSE	**2.934 ± 0.014**	3.231 ± 0.015	3.231 ± 0.015	3.147 ± 0.016
	MAPE	**51.463% ± 0.418%**	53.332% ± 0.435%	55.056% ± 0.426%	52.648% ± 0.452%
Los-Loop	MAE	**3.877 ± 0.004**	4.202 ± 0.005	5.181 ± 0.005	4.146 ± 0.006
	RMSE	**7.513 ± 0.014**	7.940 ± 0.015	9.677 ± 0.016	8.187 ± 0.018
	MAPE	**11.576% ± 0.039%**	13.144% ± 0.042%	16.405% ± 0.043%	13.945% ± 0.046%
SZ-TAXI	MAE	**2.615 ± 0.007**	2.981 ± 0.008	3.237 ± 0.007	2.852 ± 0.009
	RMSE	**4.152 ± 0.013**	4.312 ± 0.012	4.542 ± 0.015	4.360 ± 0.014

an optimal arch-hyper for each specific setting. A comparison of Figures 8(a), 8(e), and 8(f) reveals that arch-hypers found for different target datasets but with the same forecasting settings mainly rely on the hidden similarity between datasets. The optimal arch-hypers found for PEMS-BAY and PEMSD7(M) are similar w.r.t. topological connections and operators between node pairs. This is because datasets contain traffic data from similar physical-world settings (both about traffic in California) so that models performing well on one can adapt to the other. In contrast, there is a gap between Electricity and PEMS-BAY that are from different domains and do not share the same well-performed models; thus the optimal arch-hypers found for these two datasets exhibit marked disparities. Observing the examples in Figure 9, we also find that some datasets with similar N and T in Table 3 (NYC-TAXI and NYC-BIKE, Los-Loop and SZ-TAXI) also share similar well-performed models so that the optimal arch-hypers found for them are similar.

To sum up, we find that tasks with different intrinsic characteristics call for different arch-hypers. The relevant characteristics include different temporal and spatial patterns in the datasets, different difficulities with forecasting settings, different trends correlated with domains, and different scales of CTS datasets such as the

Table 11 Ablation studies, P-48/Q-48 forecasting

Dataset	Metric	AutoCTS++	w/o TS2Vec	w/o Set-Transformer	w/o shared samples
PEMS-BAY	MAE	**2.057 ± 0.006**	2.140 ± 0.007	2.157 ± 0.006	2.088 ± 0.007
	RMSE	**4.331 ± 0.021**	4.707 ± 0.023	4.594 ± 0.024	4.656 ± 0.023
	MAPE	**4.920% ± 0.026%**	4.968% ± 0.028%	5.171% ± 0.026%	5.012% ± 0.030%
Electricity	MAE	**215.821 ± 0.680**	225.128 ± 0.722	230.258 ± 0.690	222.223 ± 0.732
	RMSE	**1839.744 ± 4.875**	2145.600 ± 5.023	2220.038 ± 5.115	2055.565 ± 5.012
	MAPE	**16.681% ± 0.382%**	15.495% ± 0.442%	16.761% ± 0.412%	17.633% ± 0.402%
PEMSD7M	MAE	**3.552 ± 0.023**	3.575 ± 0.025	3.607 ± 0.028	3.562 ± 0.024
	RMSE	**6.801 ± 0.028**	6.874 ± 0.030	6.885 ± 0.032	6.858 ± 0.029
	MAPE	**9.190% ± 0.044%**	9.496% ± 0.049%	9.455% ± 0.052%	9.241% ± 0.046%
NYC-TAXI	MAE	**5.584 ± 0.026**	5.874 ± 0.029	6.646 ± 0.025	5.631 ± 0.032
	RMSE	**9.971 ± 0.044**	11.183 ± 0.048	12.344 ± 0.052	10.111 ± 0.045
	MAPE	**38.769% ± 0.375%**	40.793% ± 0.385%	52.708% ± 0.402%	39.571% ± 0.421%
NYC-BIKE	MAE	**2.073 ± 0.008**	2.154 ± 0.009	2.234 ± 0.011	2.129 ± 0.009
	RMSE	**3.227 ± 0.020**	3.420 ± 0.020	3.392 ± 0.023	3.382 ± 0.021
	MAPE	**52.110% ± 0.408%**	52.295% ± 0.502%	53.003% ± 0.452%	53.775% ± 0.438%
Los-Loop	MAE	**4.530 ± 0.006**	4.659 ± 0.008	5.051 ± 0.009	4.760 ± 0.005
	RMSE	**8.327 ± 0.022**	8.566 ± 0.021	9.323 ± 0.025	8.736 ± 0.024
	MAPE	**14.873% ± 0.063%**	15.499% ± 0.069%	17.440% ± 0.082%	16.177% ± 0.068%
SZ-TAXI	MAE	**2.656 ± 0.008**	2.710 ± 0.009	3.272 ± 0.010	2.891 ± 0.012
	RMSE	**4.193 ± 0.015**	4.248 ± 0.017	4.564 ± 0.016	4.418 ± 0.017

Table 12 Ablation studies, P-168/Q-1 (3rd) forecasting

Dataset	Metric	AutoCTS++	w/o TS2Vec	w/o Set-Transformer	w/o shared samples
PEMS-BAY	RRSE	**0.2873 ± 0.0003**	0.2972 ± 0.0004	0.2953 ± 0.0003	0.3310 ± 0.0004
	CORR	**0.9308 ± 0.0006**	0.9223 ± 0.0007	0.9256 ± 0.0006	0.9052 ± 0.0005
Electricity	RRSE	**0.0742 ± 0.0002**	0.0754 ± 0.0003	0.0890 ± 0.0003	0.0851 ± 0.0003
	CORR	**0.9477 ± 0.0005**	0.9433 ± 0.0006	0.8710 ± 0.0006	0.9110 ± 0.0005
PEMSD7M	RRSE	**0.2861 ± 0.0004**	0.2913 ± 0.0005	0.3300 ± 0.0007	0.3310 ± 0.0005
	CORR	**0.9292 ± 0.0009**	0.9234 ± 0.0012	0.8950 ± 0.0010	0.8980 ± 0.0008
NYC-TAXI	RRSE	**0.2138 ± 0.0004**	0.2216 ± 0.0005	0.2314 ± 0.0004	0.2390 ± 0.0006
	CORR	**0.8831 ± 0.0011**	0.8760 ± 0.0013	0.8640 ± 0.0010	0.8620 ± 0.0011
NYC-BIKE	RRSE	**0.7461 ± 0.0015**	0.7541 ± 0.0018	0.7810 ± 0.0016	0.7520 ± 0.0017
	CORR	**0.7902 ± 0.0008**	0.7790 ± 0.0009	0.7620 ± 0.0010	0.7810 ± 0.0010
Los-Loop	RRSE	**0.4183 ± 0.0006**	0.4312 ± 0.0008	0.4580 ± 0.0007	0.4520 ± 0.0007
	CORR	**0.8034 ± 0.0009**	0.7820 ± 0.0011	0.7570 ± 0.0010	0.7600 ± 0.0009
SZ-TAXI	RRSE	**0.4038 ± 0.0008**	0.4510 ± 0.0008	0.4880 ± 0.0007	0.4910 ± 0.0008
	CORR	**0.3414 ± 0.0018**	0.2200 ± 0.0020	0.1960 ± 0.0019	0.1910 ± 0.0021

numbers of time series and their lengths. The proposed framework can successfully capture such task characteristics and is capable of supporting a zero-shot search for optimal models.

5 Related Work

5.1 Manual Model Design

Many manually designed models exist for the forecasting of correlated time series [3,11–14,17,24,25,28,35,55, 59,60,67,68]. The biggest difference among these studies is that they design different ST-blocks, which determines their ability to capture temporal and spatial dependencies. Notably, Graph WaveNet [68] employs Diffusion GCNs and gated dilated causal convolutions to capture spatial and temporal dependencies, respectively, and stacks the two operators sequentially to obtain ST-blocks. AGCRN [3] employs enhanced Chebyshev GCNs and GRUs to capture spatial and temporal dependencies, respectively. MTGNN [67] uses mix-hop graph convolution and dilated inception convolution to capture spatial and temporal dependencies, respectively. SCNN [17] is an adaptive, interpretable, and scalable neural network framework designed for multivariate time series forecasting by disentangling and modeling structured components within the data. Transfomer-based models such as Informer [81], Autoformer [64], FEDformer [82], Triformer [11], and PDFormer [24] also inform the papers's proposal. Informer, Autoformer, and FEDformer propose enhanced attention mechanism to improve computational efficiency. Triformer uses path-attention to model both spatial and temporal dependencies. PDFormer proposes a mask mechanism to capture spatial dependencies. The architecture search space

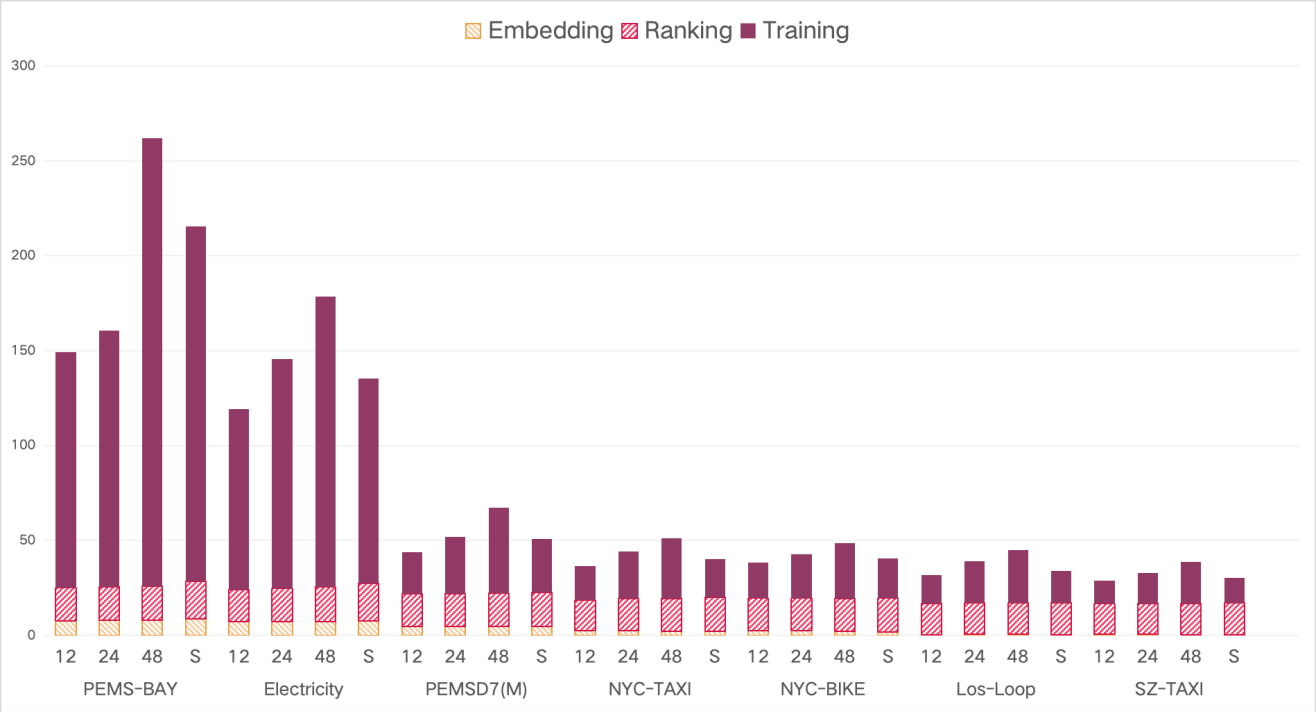


Fig. 7 Runtime of embedding, ranking, and training phases in GPU minutes for P-12/Q-12, P-24/Q-24, and P-48/Q-48 multi-step forecasting and P-168/Q-1 (3rd) single-step forecasting (S), where searching includes both the embedding and ranking phases.

we propose leverages the advances achieved by these manually designed models.

5.2 Automated Model Design

Recently, several automated frameworks have been proposed for time series forecasting.

AutoAI-TS [53] provides an automated pipeline for time series forecasting and is able to select the most appropriate forecasting model from a set of existing forecasting models for a specific dataset and forecasting setting. In contrast, the proposed AutoCTS++ framework aims at automatically designing novel CTS forecasting models. AutoST [39] divides a city's time series into grids based on longitude and latitude, and forecasts CTS data using grid-based images containing values for each time step as input. This work is not applicable to our problem because the CTS dataset we use lacks latitude and longitude information, so grid-based images cannot be constructed. AutoSTG [48], AutoSTG+ [31] and AutoCTS [65] are supernet-based methods. AutoSTG and AutoSTG+ design a search space for CTS forecasting and introduce meta-learning to learn the weights of architectures. The AutoSTG+ does better in modeling dynamical spatial correlations by means of VQDG module. AutoCTS focuses on designing a compact and effective search space for CTS forecasting and achieves good accuracy on multiple benchmark datasets. However, neither of theses proposals supports joint search. AutoCTS+ [66] is the preliminary study underlying AutoCTS++, and both share the same joint search space. As shown in Table 1, AutoCTS+ supports joint search but fails to support zero-shot search.

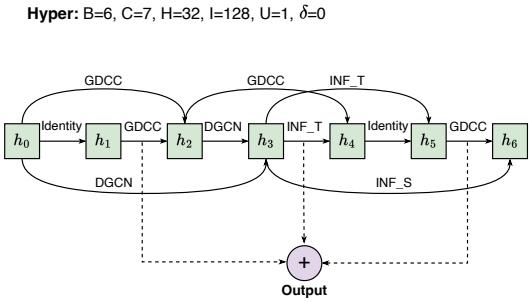
There is also a large body of studies aiming at automatically designing neural architectures for various tasks [20, 41, 42, 44, 84]. BRP-NAS [19] and CTNAS [9] perform pairwise architecture comparisons to explore the search space to identify the best architecture. However, neither methods supports joint search for architectures and hyperparameters, and thus they fail to address the first limitation. AutoHAS [18] and FBNetV3 [15] aim to automatically search for the best combination of architectures and hyperparameters. But these methods are supernet-based and are very inefficient. For example, FBNetV3 spends more than 10K GPU hours searching for the best arch-hyper, so it is unable to overcome the second limitation.

Some early studies utilize the characteristics of tasks mainly for transfer learning or Neural Architecture Search. There are two main approaches for task representation learning. The first approach involves extracting artificial or meta features from the dataset of a specific task or using a pre-trained general model to encode the entire dataset into a vector representation [29,37,40,54,

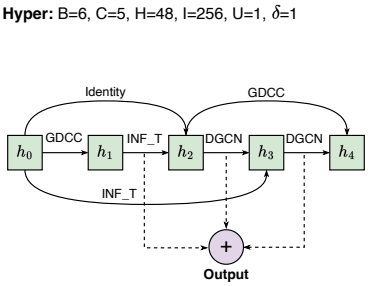
Table 13 Sample-limited performance studies, P-24/Q-24 forecasting.

Dataset	Metric	K_s=600,000	K_s=300,000	K_s=150,000	K_s=75,000	K_s=37,500	*AutoCTS+*	PDFormer
PEMS-BAY	MAE	1.825 ±0.009	1.828 ±0.009	1.838 ±0.008	1.847 ±0.010	1.868 ±0.013	1.836 ±0.011	1.843 ±0.012
	RMSE	4.086 ±0.012	4.096 ±0.010	4.122 ±0.009	4.153 ±0.011	4.195 ±0.011	4.147 ±0.014	4.112 ±0.008
	MAPE	4.120% ±0.015%	4.133% ±0.018%	4.150% ±0.022%	4.191% ±0.017%	4.257% ±0.013%	4.236% ±0.013%	4.139% ±0.018%
	TIME	48.6	26.4	17.8	13.6	10.8	540.8	476.2
Electricity	MAE	183.008 ±0.874	184.031 ±0.953	195.774 ±1.263	207.958 ±0.928	225.063 ±2.210	205.401 ±0.801	202.571 ±0.685
	RMSE	1524.825 ±4.011	1537.341 ±3.693	1635.147 ±4.191	1767.509 ±3.942	1807.854 ±5.166	1784.313 ±4.566	1724.646 ±4.336
	MAPE	14.472% ±0.352%	14.532% ±0.280%	15.356% ±0.296%	16.123% ±0.307%	17.013% ±0.427%	16.085% ±0.348%	16.004% ±0.290%
	TIME	48.1	24.9	15.6	12.0	9.9	444.8	382.9
PEMSD7M	MAE	3.105 ±0.026	3.114 ±0.025	3.164 ±0.027	3.206 ±0.023	3.228 ±0.021	3.191 ±0.028	3.203 ±0.021
	RMSE	6.037 ±0.018	6.134 ±0.021	6.176 ±0.022	6.218 ±0.020	6.245 ±0.025	6.221 ±0.020	6.193 ±0.020
	MAPE	7.891% ±0.026%	7.911% ±0.032%	8.179% ±0.036%	8.304% ±0.031%	8.423% ±0.029%	8.333% ±0.037%	8.242% ±0.038%
	TIME	42.5	19.8	13.8	9.6	7.0	120.5	98.2
NYC-TAXI	MAE	5.055 ±0.018	5.061 ±0.020	5.381 ±0.022	5.608 ±0.024	5.884 ±0.024	5.902 ±0.026	5.760 ±0.024
	RMSE	9.196 ±0.038	9.206 ±0.042	9.662 ±0.039	10.312 ±0.040	11.033 ±0.043	11.106 ±0.042	10.506 ±0.036
	MAPE	37.539% ±0.348%	37.709% ±0.301%	39.197% ±0.352%	41.647% ±0.352%	43.582% ±0.320%	42.283% ±0.383%	43.709% ±0.431%
	TIME	37.8	16.4	11.9	8.7	6.3	100.0	75.9
NYC-BIKE	MAE	1.955 ±0.010	1.959 ±0.008	2.053 ±0.011	2.079 ±0.009	2.103 ±0.0013	2.088 ±0.008	2.049 ±0.008
	RMSE	2.927 ±0.011	2.934 ±0.014	3.143 ±0.018	3.184 ±0.022	3.235 ±0.023	3.232 ±0.017	3.147 ±0.022
	MAPE	51.449% ±0.388%	51.463% ±0.418%	52.438% ±0.423%	52.975% ±0.441%	53.738% ±0.387%	53.913% ±0.422%	52.648% ±0.398%
	TIME	38.5	16.9	12.4	9.2	6.8	92.6	69.8
Los-Loop	MAE	3.871 ±0.006	3.877 ±0.004	3.985 ±0.004	4.123 ±0.006	4.269 ±0.006	4.244 ±0.008	4.120 ±0.005
	RMSE	7.466 ±0.017	7.513 ±0.014	7.785 ±0.018	7.936 ±0.017	7.987 ±0.020	7.943 ±0.019	7.927 ±0.017
	MAPE	11.498% ±0.041%	11.576% ±0.039%	12.196% ±0.040%	12.865% ±0.037%	13.114% ±0.040%	13.209% ±0.040%	12.736% ±0.043%
	TIME	36.6	14.8	9.3	7.1	5.9	68.2	45.6
SZ-TAXI	MAE	2.603 ±0.008	2.615 ±0.007	2.974 ±0.009	3.240 ±0.010	3.318 ±0.012	3.256 ±0.011	3.237 ±0.013
	RMSE	4.147 ±0.011	4.152 4.152 ±0.013±0.013	4.287 ±0.010	4.369 ±0.012	4.539 ±0.013	4.547 ±0.014	4.342 ±0.010
	TIME	36.4	14.7	9.5	7.1	5.6	64.1	39.7

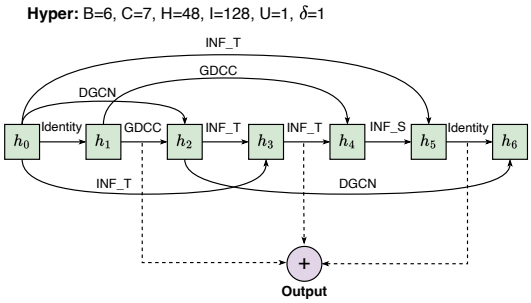
61,62,78]. The second approach involves training anchor models or fine-tuning a general model for the task and extracting gradient information from them to serve as the task representation [1, 8, 30, 36, 58]. However, when the domain shifts to CTS forecasting, these methods face challenges in terms of feasibility and efficiency—the first approach is designed for non-CTS datasets with specific expert knowledge, while the second approach takes unacceptably long time to train models for gradient information, making them unsuitable in our setting. Instead, we propose AutoCTS++ to encode CTS forecasting tasks and overcome the second limitation. Also, a line of recent studies has emerged that utilize LLMs or Foundation Models for zero-shot time series analysis [2, 26, 27, 83]. Studies based on LLMs [83] use pre-trained backbones of LLMs and fine-tune input and output layers to build a same-scale forecasting model. Another LLM-based study [26] aligns time series data with the word embedding space of LLMs and processes forecasting tasks as NLP tasks. Other studies propose time series foundation models [16,45,63] pre-trained on massive time series datasets to support zero-shot analysis on unseen tasks. However, these large models, with billions of parameters, require very substantial computational resources and runtime, while AutoCTS++ leverages a lightweight T-AHC to build a relatively



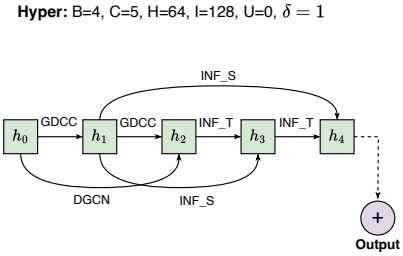
(a) PEMS-BAY dataset, P-12/Q-12



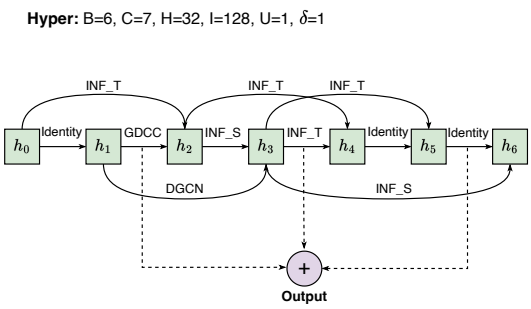
(b) PEMS-BAY dataset, P-24/Q-24



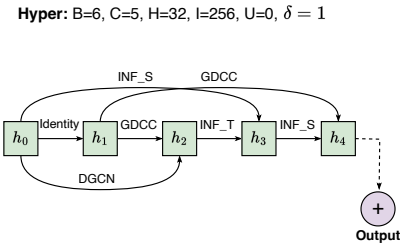
(c) PEMS-BAY dataset, P-48/Q-48



(d) PEMS-BAY dataset, P-168/Q-1 (3rd)



(e) PEMSD7(M) dataset, P-12/Q-12

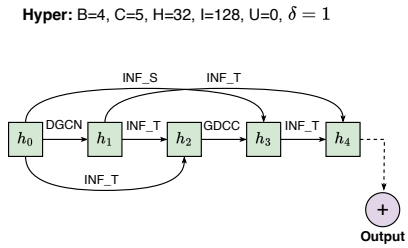


(f) Electricity dataset, P-12/Q-12

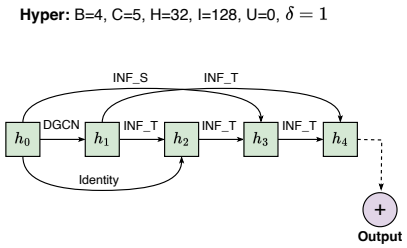
Fig. 8 Case Study of Searched ST-blocks on Different Target Datasets and P/Q Settings.

lightweight forecasting model that is tailed for a specific task. They are different technological routes.

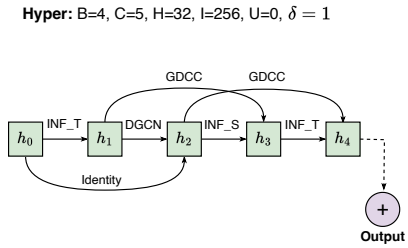
To efficiently assess model performance for new tasks, some early studies [47,57,74] propose methods of evaluating pre-trained models in transfer learning. However, LEEP [47] and NCE [57] can only evaluate classification tasks with supervised pre-trained models, while LogME [74] can evaluate models pre-trained without categorical labels for different downstream tasks. All of them evaluate a dozen pre-trained models by making inference on the target task to extract some information and calculating specially-designed ranking scores for pre-trained models, so that the computational cost increases with the number of pre-trained models and the dataset sizes. In contrast, the proposed T-AHC strategy decouples the procedures of embedding tasks and ranking arch-hypers and utilizes the task representation to quickly evaluate a large number of models from the joint search space for any target task. This arrangement aligns better our needs.



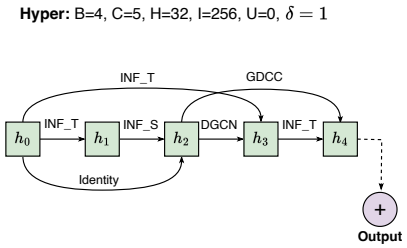
(a) NYC-TAXI dataset, P-12/Q-12



(b) NYC-BIKE dataset, P-12/Q-12



(c) Los-Loop, P-24/Q-24



(d) SZ-TAXI, P-24/Q-24

Fig. 9 Case Study of Searched ST-blocks on Different Target Datasets and P/Q Settings.

6 Conclusion

We present AutoCTS++, a zero-shot joint search strategy to automatically configure high-performance ST-blocks for CTS forecasting tasks. In particular, we first design a joint search space containing massive arch-hypers, each of which is a combination of an architecture and a hyperparameter setting. Next, we propose an T-AHC-based search strategy to explore the search space to find the best arch-hyper for unseen tasks.

In addition, we propose a pre-training method to enable our model to learn the relationship between tasks and model performance, allowing us to directly apply T-AHC strategy to unseen tasks. Comprehensive experiments on seven commonly used correlated time series forecasting datasets offer detailed evidence of the effectiveness and efficiency of the proposed framework.

Although the proposed framework performs efficiently and effectively on real-world benchmark datasets, it may miss some flexibility due to its manually-designed joint search space. In future work, it is of interest to explore an automated strategy to build the search space for any specific task. Another direction is to extend the proposed T-AHC strategy to other domains such as computer vision and natural language processing, exploiting the ability of the proposed joint search space, i.e., the arch-hyper graph, to represent models used in those domains.

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

References

1. Achille, A., Lam, M., Tewari, R., Ravichandran, A., Maji, S., Fowlkes, C.C., Soatto, S., Perona, P.: Task2Vec: Task embedding for meta-learning. In: International conference on computer vision, pp. 6430–6439 (2019)
2. Ansari, A.F., Stella, L., Turkmen, C., Zhang, X., Mercado, P., Shen, H., Shchur, O., Rangapuram, S.S., Arango, S.P., Kapoor, S., et al.: Chronos: Learning the language of time series. arXiv preprint arXiv:2403.07815 (2024)
3. Bai, L., Yao, L., Li, C., Wang, X., Wang, C.: Adaptive graph convolutional recurrent network for traffic forecasting. In: NeurIPS, vol. 33, pp. 17,804–17,815 (2020)
4. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: International conference on machine learning, pp. 41–48 (2009)
5. Campos, D., Kieu, T., Guo, C., Huang, F., Zheng, K., Yang, B., Jensen, C.S.: Unsupervised time series outlier detection with diversity-driven convolutional ensembles. Proc. VLDB Endow. **15**(3), 611–623 (2022)
6. Campos, D., Yang, B., Kieu, T., Zhang, M., Guo, C., Jensen, C.S.: Qcore: Data-efficient, on-device continual calibration for quantized models. Proc. VLDB Endow. (2024)
7. Campos, D., Zhang, M., Yang, B., Kieu, T., Guo, C., Jensen, C.S.: LightTS: Lightweight time series classifica-

tion with adaptive ensemble distillation. . Proc. ACM Manag. Data 1(2): 171:1-171:27 (2023)
8. Cao, K., You, J., Liu, J., Leskovec, J.: Autotransfer: AutoML with knowledge transfer - an application to graph neural networks. In: International Conference on Learning Representations (2023)
9. Chen, Y., Guo, Y., Chen, Q., Li, M., Zeng, W., Wang, Y., Tan, M.: Contrastive neural architecture search with neural architecture comparators. In: Conference on Computer Vision and Pattern Recognition, pp. 9502–9511 (2021)
10. Cheng, Y., Chen, P., Guo, C., Zhao, K., Wen, Q., Yang, B., Jensen, C.S.: Weakly guided adaptation for robust time series forecasting. Proc. VLDB Endow. (2024)
11. Cirstea, R., Guo, C., Yang, B., Kieu, T., Dong, X., Pan, S.: Triangular, variable-specific attentions for long sequence multivariate time series forecasting. In: IJCAI, pp. 1994–2001 (2022)
12. Cirstea, R., Yang, B., Guo, C., Kieu, T., Pan, S.: Towards spatio-temporal aware traffic time series forecasting. In: ICDE, pp. 2900–2913 (2022)
13. Cirstea, R.G., Kieu, T., Guo, C., Yang, B., Pan, S.J.: Enhancenet: Plugin neural networks for enhancing correlated time series forecasting. In: ICDE, pp. 1739–1750 (2021)
14. Cirstea, R.G., Yang, B., Guo, C.: Graph attention recurrent neural networks for correlated time series forecasting. In: MileTS19@KDD (2019)
15. Dai, X., Wan, A., Zhang, P., Wu, B., He, Z., Wei, Z., Chen, K., Tian, Y., Yu, M., Vajda, P., et al.: Fbnetv3: Joint architecture-recipe search using predictor pretraining. In: Conference on Computer Vision and Pattern Recognition, pp. 16,276–16,285 (2021)
16. Das, A., Kong, W., Sen, R., Zhou, Y.: A decoder-only foundation model for time-series forecasting. arXiv preprint arXiv:2310.10688 (2023)
17. Deng, J., Chen, X., Jiang, R., Yin, D., Yang, Y., Song, X., Tsang, I.W.: Disentangling structured components: Towards adaptive, interpretable and scalable time series forecasting. IEEE Transactions on Knowledge and Data Engineering (2024)
18. Dong, X., Tan, M., Yu, A.W., Peng, D., Gabrys, B., Le, Q.V.: Autohas: Efficient hyperparameter and architecture search. arXiv preprint arXiv:2006.03656 (2020)
19. Dudziak, L., Chau, T., Abdelfattah, M., Lee, R., Kim, H., Lane, N.: Prediction-based NAS using GCNs. BRP-NAS: Advances in Neural Information Processing Systems **33**, 10,480–10,490 (2020)
20. El, O.B., Milo, T., Somech, A.: Automatically generating data exploration sessions using deep reinforcement learning. pp. 1527–1537 (2020)
21. Guo, C., Xu, R., Yang, B., Yuan, Y., Kieu, T., Zhao, Y., Jensen, C.S.: Efficient stochastic routing in path-centric uncertain road networks. Proc. VLDB Endow. (2024)
22. Guo, C., Yang, B., Hu, J., Jensen, C.S., Chen, L.: Context-aware, preference-based vehicle routing. VLDB J. **29**(5), 1149–1170 (2020)
23. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: European conference on computer vision, pp. 544–560 (2020)
24. Jiang, J., Han, C., Zhao, W.X., Wang, J.: Pdformer: Propagation delay-aware dynamic long-range transformer for traffic flow prediction. arXiv preprint arXiv:2301.07945 (2023)
25. Jin, G., Liang, Y., Fang, Y., Shao, Z., Huang, J., Zhang, J., Zheng, Y.: Spatio-temporal graph neural networks for predictive learning in urban computing: A survey. IEEE Transactions on Knowledge and Data Engineering (2023)
26. Jin, M., Wang, S., Ma, L., Chu, Z., Zhang, J.Y., Shi, X., Chen, P.Y., Liang, Y., Li, Y.F., Pan, S., et al.: Time-llm: Time series forecasting by reprogramming large language models. arXiv preprint arXiv:2310.01728 (2023)
27. Jin, M., Wen, Q., Liang, Y., Zhang, C., Xue, S., Wang, X., Zhang, J., Wang, Y., Chen, H., Li, X., et al.: Large models for time series and spatio-temporal data: A survey and outlook. arXiv preprint arXiv:2310.10196 (2023)
28. Jin, M., Zheng, Y., Li, Y., Chen, S., Yang, B., Pan, S.: Multivariate time series forecasting with dynamic graph neural odes. IEEE Trans. Knowl. Data Eng. **35**(9), 9168–9180 (2023)
29. Jomaa, H.S., Schmidt-Thieme, L., Grabocka, J.: Dataset2vec: Learning dataset meta-features. Data Mining and Knowledge Discovery **35**, 964–985 (2021)
30. Karakida, R., Akaho, S., Amari, S.i.: Universal statistics of fisher information in deep neural networks: Mean field approach. In: International Conference on Artificial Intelligence and Statistics, pp. 1032–1041 (2019)
31. Ke, S., Pan, Z., He, T., Liang, Y., Zhang, J., Zheng, Y.: Autostg+: An automatic framework to discover the optimal network for spatio-temporal graph prediction. Artificial Intelligence **318**, 103,899 (2023)
32. Kieu, T., Yang, B., Guo, C., Cirstea, R., Zhao, Y., Song, Y., Jensen, C.S.: Anomaly detection in time series with robust variational quasi-recurrent autoencoders. In: ICDE, pp. 1342–1354 (2022)
33. Kieu, T., Yang, B., Guo, C., Jensen, C.S., Zhao, Y., Huang, F., Zheng, K.: Robust and explainable autoencoders for unsupervised time series outlier detection. In: ICDE, pp. 3038–3050 (2022)
34. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
35. Lai, G., Chang, W.C., Yang, Y., Liu, H.: Modeling long- and short-term temporal patterns with deep neural networks. In: SIGIR, pp. 95–104 (2018)
36. Le, C.P., Soltani, M., Dong, J., Tarokh, V.: Fisher task distance and its application in neural architecture search. IEEE Access **10**, 47,235–47,249 (2022)
37. Lee, H., Hyung, E., Hwang, S.J.: Rapid neural architecture search by learning to generate graphs from datasets. arXiv preprint arXiv:2107.00860 (2021)
38. Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., Teh, Y.W.: Set transformer: A framework for attention-based permutation-invariant neural networks. In: International conference on machine learning, pp. 3744–3753 (2019)
39. Li, T., Zhang, J., Bao, K., Liang, Y., Li, Y., Zheng, Y.: Autost: Efficient neural architecture search for spatio-temporal prediction. In: SIGKDD, pp. 794–802 (2020)
40. Li, X., Li, Z., Xie, H., Li, Q.: Merging statistical feature via adaptive gate for improved text classification. In: AAAI, vol. 35, pp. 13,288–13,296 (2021)
41. Li, Y., Chen, Z., Zha, D., Zhou, K., Jin, H., Chen, H., Hu, X.: AutoOD: Neural architecture search for outlier detection. In: ICDE, pp. 2117–2122. IEEE (2021)
42. Li, Y., Shen, Y., Zhang, W., Jiang, J., Li, Y., Ding, B., Zhou, J., Yang, Z., Wu, W., Zhang, C., Cui, B.: Volcanoml: Speeding up end-to-end automl via scalable search space decomposition. Proc. VLDB Endow. **14**(11), 2167–2176 (2021)
43. Li, Y., Yu, R., Shahabi, C., Liu, Y.: Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In: ICLR (2018)
44. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: ICLR (2018)

45. Liu, Y., Zhang, H., Li, C., Huang, X., Wang, J., Long, M.: Timer: Transformers for time series analysis at scale. arXiv preprint arXiv:2402.02368 (2024)
46. Miao, H., Zhao, Y., Guo, C., Yang, B., Kai, Z., Huang, F., Xie, J., Jensen, C.S.: A unified replay-based continuous learning framework for spatio-temporal prediction on streaming data. ICDE (2024)
47. Nguyen, C., Hassner, T., Seeger, M., Archambeau, C.: Leep: A new measure to evaluate transferability of learned representations. In: International Conference on Machine Learning, pp. 7294–7305. PMLR (2020)
48. Pan, Z., Ke, S., Yang, X., Liang, Y., Yu, Y., Zhang, J., Zheng, Y.: AutoSTG: Neural architecture search for predictions of spatio-temporal graphs. In: WWW, pp. 1846–1855 (2021)
49. Pedersen, S.A., Yang, B., Jensen, C.S.: Anytime stochastic routing with hybrid learning. Proc. VLDB Endow. **13**(9), 1555–1567 (2020)
50. Pedersen, S.A., Yang, B., Jensen, C.S.: Fast stochastic routing under time-varying uncertainty. VLDB J. **29**(4), 819–839 (2020)
51. Qiu, X., Hu, J., Zhou, L., Wu, X., Du, J., Zhang, B., Guo, C., Zhou, A., Jensen, C.S., Sheng, Z., Yang, B.: Tfb: Towards comprehensive and fair benchmarking of time series forecasting methods. Proc. VLDB Endow. **17**, 2363 – 2377 (2024)
52. Rajkumar, R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical systems: the next computing revolution. In: Design automation conference, pp. 731–736 (2010)
53. Shah, S.Y., Patel, D., Vu, L., Dang, X., Chen, B., Kirchner, P., Samulowitz, H., Wood, D., Bramble, G., Gifford, W.M., Ganapavarapu, G., Vaculín, R., Zerfos, P.: AutoAI-TS: Autoai for time series forecasting. In: SIGMOD, pp. 2584–2596 (2021)
54. Shala, G., Elsken, T., Hutter, F., Grabocka, J.: Transfer NAS with meta-learned bayesian surrogates. In: International Conference on Learning Representations (2023)
55. Shih, S.Y., Sun, F.K., Lee, H.y.: Temporal pattern attention for multivariate time series forecasting. Machine Learning **108**(8), 1421–1441 (2019)
56. Song, C., Lin, Y., Guo, S., Wan, H.: Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. In: AAAI, vol. 34, pp. 914–921 (2020)
57. Tran, A.T., Nguyen, C.V., Hassner, T.: Transferability and hardness of supervised classification tasks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 1395–1405 (2019)
58. Wang, J., Wang, K.C., Rudzicz, F., Brudno, M.: Grad2task: Improved few-shot text classification using gradients for task representation. Advances in Neural Information Processing Systems **34**, 6542–6554 (2021)
59. Wang, S., Miao, H., Chen, H., Huang, Z.: Multi-task adversarial spatial-temporal networks for crowd flow prediction. In: International Conference on Information & Knowledge Management, pp. 1555–1564 (2020)
60. Wang, S., Zhang, M., Miao, H., Peng, Z., Yu, P.S.: Multivariate correlation-aware spatio-temporal graph convolutional networks for multi-scale traffic prediction. ACM Transactions on Intelligent Systems and Technology (TIST) **13**(3), 1–22 (2022)
61. Wei, Y., Zhao, P., Huang, J.: Meta-learning hyperparameter performance prediction with neural processes. In: International Conference on Machine Learning, pp. 11,058–11,067 (2021)
62. Wong, C., Houlsby, N., Lu, Y., Gesmundo, A.: Transfer learning with neural AutoML. Advances in neural information processing systems **31** (2018)
63. Woo, G., Liu, C., Kumar, A., Xiong, C., Savarese, S., Sahoo, D.: Unified training of universal time series forecasting transformers. arXiv preprint arXiv:2402.02592 (2024)
64. Wu, H., Xu, J., Wang, J., Long, M.: Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. Advances in neural information processing systems **34**, 22,419–22,430 (2021)
65. Wu, X., Zhang, D., Guo, C., He, C., Yang, B., Jensen, C.S.: AutoCTS: Automated correlated time series forecasting. Proc. VLDB Endow **15**(4), 971–983 (2022)
66. Wu, X., Zhang, D., Zhang, M., Guo, C., Yang, B., Jensen, C.S.: AutoCTS+: Joint neural architecture and hyperparameter search for correlated time series forecasting. Proceedings of the ACM on Management of Data **1**(1), 1–26 (2023)
67. Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., Zhang, C.: Connecting the dots: Multivariate time series forecasting with graph neural networks. In: SIGKDD, pp. 753–763 (2020)
68. Wu, Z., Pan, S., Long, G., Jiang, J., Zhang, C.: Graph wavenet for deep spatial-temporal graph modeling. In: IJCAI, pp. 1907–1913 (2019)
69. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? arXiv preprint arXiv:1810.00826 (2018)
70. Yang, S.B., Guo, C., Hu, J., Tang, J., Yang, B.: Unsupervised path representation learning with curriculum negative sampling. In: IJCAI, pp. 3286–3292 (2021)
71. Yang, S.B., Guo, C., Hu, J., Yang, B., Tang, J., Jensen, C.S.: Weakly-supervised temporal path representation learning with contrastive curriculum learning. In: ICDE, pp. 2873–2885 (2022)
72. Yang, S.B., Guo, C., Yang, B.: Context-aware path ranking in road networks. IEEE Trans. Knowl. Data Eng. **34**(7), 3153–3168 (2022)
73. Ye, J., Sun, L., Du, B., Fu, Y., Xiong, H.: Coupled layer-wise graph convolution for transportation demand prediction. In: AAAI, vol. 35, pp. 4617–4625 (2021)
74. You, K., Liu, Y., Wang, J., Long, M.: Logme: Practical assessment of pre-trained models for transfer learning. In: International Conference on Machine Learning, pp. 12,133–12,143. PMLR (2021)
75. Yu, B., Yin, H., Zhu, Z.: Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting. In: IJCAI, pp. 3634–3640 (2018)
76. Yu, H., Hu, J., Zhou, X., Guo, C., Yang, B., Li, Q.: CGF: A category guidance based PM2.5 sequence forecasting training framework. IEEE Trans. Knowl. Data Eng. **35**(10), 125–139 (2023)
77. Yue, Z., Wang, Y., Duan, J., Yang, T., Huang, C., Tong, Y., Xu, B.: TS2Vec: Towards universal representation of time series. In: AAAI, vol. 36, pp. 8980–8987 (2022)
78. Zamir, A.R., Sax, A., Shen, W., Guibas, L.J., Malik, J., Savarese, S.: Taskonomy: Disentangling task transfer learning. In: Conference on computer vision and pattern recognition, pp. 3712–3722 (2018)
79. Zhao, K., Guo, C., Han, P., Zhang, M., Cheng, Y., Yang, B.: Multiple time series forecasting with dynamic graph modeling. Proc. VLDB Endow. (2024)
80. Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., Li, H.: T-GCN: A temporal graph convolutional network for traffic prediction. IEEE Transactions on Intelligent Transportation Systems **21**(9), 3848–3858 (2019)
81. Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W.: Informer: Beyond efficient transformer for

long sequence time-series forecasting. In: AAAI, vol. 35, pp. 11,106–11,115 (2021)
82. Zhou, T., Ma, Z., Wen, Q., Wang, X., Sun, L., Jin, R.: Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In: International conference on machine learning, pp. 27,268–27,286. PMLR (2022)
83. Zhou, T., Niu, P., Sun, L., Jin, R., et al.: One fits all: Power general time series analysis by pretrained lm. Advances in neural information processing systems **36** (2024)
84. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: ICLR. OpenReview.net (2017)