Optimizing Time Series Forecasting Architectures: A Hierarchical Neural Architecture Search Approach

Anonymous authors
Paper under double-blind review

Abstract

The rapid development of time series forecasting research has brought many deep learning-based modules to this field. However, despite the increasing number of new forecasting architectures, it is still unclear if we have leveraged the full potential of these existing modules within a properly designed architecture. In this work, we propose a novel hierarchical neural architecture search space for time series forecasting tasks. With the design of a hierarchical search space, we incorporate many architecture types designed for forecasting tasks and allow for the efficient combination of different forecasting architecture modules. Results on long-term time series forecasting tasks show that our approach can search for lightweight, high-performing forecasting architectures across different forecasting tasks.

1 Introduction

Time series forecasting techniques are widely applied in different fields, e.g., energy consumption (Trindade, 2015), business (Makridakis et al., 2022), or traffic planning (Lana et al., 2018). However, unlike computer vision (CV) and natural language processing (NLP) tasks that are dominated by the CNN (He et al., 2016; Liu et al., 2018; Zoph et al., 2018) and Transformer (Brown et al., 2020; Devlin et al., 2019; Liu et al., 2021; Vaswani et al., 2017) families, there is no clearly dominating architecture in time series forecasting tasks. Although there are lots of transformer-based approaches applied to forecasting models (Ansari et al., 2024; Das et al., 2023; Wu et al., 2021; Liu et al., 2022; Zhou et al., 2022), many of them might even be outperformed by a simple linear baseline (Zeng et al., 2023).

The success of transformer models in CV and NLP tasks is based on their ability to capture long-term dependencies with the help of tokens with fruitful semantic information, i.e., each word embedding already contains lots of information, while an image patch with many pixels can already tell us a lot of information. All this information relaxes the requirement for the models to grab the temporal information within the input series. However, this information is usually crucial in time series forecasting tasks, given that a single value at each time step only contains a limited amount of information. The strong ability of the transformer family to capture long-term dependencies might not compensate for its poor ability to build temporal connections within the input series. This finding is evident by Zeng et al. (2023), where a simple linear layer could outperform many state-of-the-art transformer models.

Some recent works show the efficiency of transformers (Liu et al., 2023; Nie et al., 2023) on long-time forecasting tasks. However, their approach could still not overcome the temporal dependencies issues for transformers, e.g., PatchTST (Nie et al., 2023) augments the information within each token by constructing a patch with the data from multiple time steps. While iTransformer (Liu et al., 2023) simply encodes the entire input sequence into a token and tries to construct the connections among different variables.

On the other hand, many architectures were constructed for mining the local dependencies, such as CNNs (Bai et al., 2018; Luo and Wang, 2024) and RNNs (Hewamalage et al., 2021; Hochreiter et al., 2001). These architectures might not work well on long-term dependencies due to the limited receptive field (for CNN), latent bottlenecks (Didolkar et al., 2022), or vanishing gradients (for RNN families). Recent work such as ModernTCN (Luo and Wang, 2024) showed that an increased convolutional kernel size and improved microarchitecture could lead to a more accurate model. However, this approach results in a huge memory and computation consumption. Here we provide another perspective: one could combine these architectures with other operations, such as transformer layers, to develop a new architecture that combines the best of two worlds (Didolkar et al., 2022; Lai et al., 2018; Lim et al., 2021).

Nevertheless, it is still unclear (i) which type of architecture we would like to construct, given the variability of different forecasting models (Deng et al., 2022; Oreshkin et al., 2020; Salinas et al., 2020; Zeng et al., 2023) and (ii) how to connect different operations to form a new architecture. Designing a new architecture from scratch for each task might take a lot of human expert efforts and tedious trial-and-error. Neural architecture search (NAS) is a technique that automatically searches for the optimal architecture given a new task.

Previous NAS frameworks mainly focused on single network backbones types such as CNN or Transformer networks (Chen et al., 2021b; Liu et al., 2018; Zoph et al., 2018). It is still unclear how to optimize the forecasting architectures due to their internal complexity. For instance, an encoder-decoder architecture (Wu et al., 2021; Zhou et al., 2021) might work well on some tasks, while the other tasks might prefer encoder-only architectures (Liu et al., 2023) or even MLP-only architectures (Oreshkin et al., 2020; Zeng et al., 2023). This provides another challenge for designing a search space for time series forecasting tasks. In this work, we will address this challenge by designing a unified search space for time series forecasting tasks.

Our contributions are summarized as follows:

1. We design a hierarchical search space that contains most forecasting architecture design decisions and allows any sort of architecture layers to be combined to form new architectures.
2. We show that by applying DARTS-PT (Wang et al., 2021), a differentiable neural architecture search approach, to our search space. The resulting architectures, dubbed DARTS-TS, are comparable to the state-of-the-art models with much less computational resource requirements.
3. We provide an analysis of our search space, showing that our search space has different properties compared to the existing CNN based NAS search space and provides further challenges for the NAS research.

2 Related Work

Although many different deep learning architectures are proposed to solve time series tasks, there is little work that applies neural architecture search to search for a new architecture with the existing frameworks. In this section, we will provide a brief overview of deep learning-based forecasting frameworks and neural architecture search techniques.

2.1 Deep Learning-based Time Series Forecasting

Time series forecasting aims to predict the future values of target variables given their historical data. Because of its importance, much work has been investigated for a more accurate forecasting model in this research field. Previous work mainly focused on traditional statistical local approaches that train an individual model for each series (Athanasopoulos, 2021; Box et al., 2015). However, these approaches might not fit well in the era of big data, where a dataset could contain thousands of series. On the other side, the machine learning-based model trains a single global model across all the series and uses this model to predict all the series in the dataset (Godahewa et al., 2021; Makridakis et al., 2020; 2022). More recently, deep learning-based forecasting models (Bai et al., 2018; Hewamalage et al., 2021; Lai et al., 2018; Salinas et al., 2020; Shi et al., 2015; Wen et al., 2017), or even the zero-shot foundation models (Ansari et al., 2024; Das et al., 2023), have gradually become mainstream in this research field.

Time series forecasting models need to work with sequential inputs (Alexandrov et al., 2020; Beitner, 2020). Overall, these networks can be categorized into two families: *Seq Net* and *Flat Net* (Deng et al., 2022). Given a batch of sequences with shape $[B, L, N]$, where B is the batch size, L is the sequence length, and N is the number of time series variables. A *Seq Net*, such as RNNs (Cho et al., 2014; Hochreiter and Schmidhuber, 1997), TCNs (Bai et al., 2018; Oord et al., 2016), and Transformers (Li et al., 2019; Liu et al., 2022; Vaswani et al., 2017; Wu et al., 2021; Zhou et al., 2021; 2022) computes the correlations across different time steps without breaking the structure of the input sequence. On the opposite, a *Flat Net*, such as MLP (Zeng et al., 2023) and N-BEATS (Oreshkin et al., 2020), decomposes the variables into independent singe-variable series: $[B \times N, L]$.[1]. After that, this variable is passed to another network. This strategy was previously introduced such that the time series can be handled by the machine learning framework designed for tabular datasets such as MLP layers (Zeng et al., 2023) and LightGBM (Ke et al., 2017; Makridakis et al., 2022). Some recent works, such as PatchTST (Nie et al., 2023), release the correlation between different variables within a multi-variable series and consider it as a collection of independent series and make predictions for each series independently, which also belongs to this type of architecture. However, these approaches might be too expensive for datasets with many variables since a forward pass is required for each series. Hence, in this work, we mainly focus on the MLP-based *Flat Net* to only search for lightweight architectures.

Diving deeper into the *Seq Net* architecture families, we find another two main branches: encoder-decoder architectures and encoder-only architectures[2]. Encoder-decoder architectures (Sutskever et al., 2014; Vaswani et al., 2017) maintain two individual networks that embed the information from the past and future correspondingly. These architectures have shown great success in time series forecasting tasks (Wu et al., 2021; Zhou et al., 2021; 2022). On the other hand, encoder-only architectures only apply a *Seq* encoder that maps the past information into a latent feature map and utilizes another linear layer to provide the prediction with the latent feature map (Liu et al., 2023; Nie et al., 2023; Salinas et al., 2020).

Many *Seq Net* models, especially the transformer family (Lim et al., 2021; Wu et al., 2021; Zhou et al., 2021; 2022), are designed for solving series input. However, a study by Zeng et al. (2023) showed that these transformers might even be outperformed by a linear model. This inspires us to seek other uncovered modules that might perform well within a properly designed architecture, esp. architectures with more than one type of operation.

Many forecasting architectures are homogenous and only contain one type of operation layer (Bai et al., 2018; Li et al., 2019; Liu et al., 2022; Luo and Wang, 2024; Oord et al., 2016) and stack this layer repeatedly to construct a new architecture. However, some recent work has also shown the efficiency of combining architecture from different model families. LSTNet (Lai et al., 2018) stacks an RNN on top of a CNN layer. ConvLSTM (Shi et al., 2015) constructs a convolutionary operation within an LSTM cell. Temporal fusion transformer (Lim et al., 2021) stacks an explainable multi-head attention layer on top of an LSTM encoder-decoder model. All these works suggest the efficiency of combining operations from different architecture families. However, it is often tedious to find these combinations manually. Neural architecture search (NAS) is a technique that automatically searches for the optimal architecture for a given task. In the following section, we will briefly overview the NAS framework.

2.2 Neural Architecture Search

Previous NAS research mainly considered network training as a black-box process and trained every configuration from scratch until convergence (Deng et al., 2022; Jin et al., 2019; White et al., 2021; Zimmer et al., 2021; Zoph and Le, 2017). However, training a network is very expensive and requires lots of resources. To overcome this problem, the One-Shot NAS (Pham et al., 2018; Zoph et al., 2018) approach defines a supernetwork where all the child architectures' weights are inherited from the supernet. DARTS (Liu et al., 2019a) further relaxes the discrete operation search space to continuous parameters and optimizes these values with model weights jointly with gradient descent. Finally, the optimal operations and paths are selected based on the architecture parameter values.

DARTS might be unstable during the search process: the architectures suggested by the DARTS might be dominated by the skip connections (Chu et al., 2020; Jiang et al., 2023; Wang et al., 2021; Zela et al., 2020). Robust DARTS (Zela et al., 2020) showed that this instability is due to the high validation loss curvature in the search space, while operation-level early stopping DARTS found that the instability is due

[1]Howeve, in practice, we would still like to preserve the correlations between multi-variant series with, for instance, batch normalization (Ioffe and Szegedy, 2015). In this case, the input series becomes $[B, N, L]$

[2]Sometimes we might also have decoder-only architectures; however, for the sake of simplicity, we consider both as part of encoder-only architectures.

to the case that the network weights are overfitted to the training sets. DARTS-PT (Wang et al., 2021) provides a perturbation-based approach to measure the importance of each operation and use it to replace the architecture parameter-based approach introduced in Liu et al. (2019a).

ONE-NAS (Lyu et al., 2023) is an online architecture search framework that applies evolutionary algorithms to search for the optimal RNN networks on online forecasting tasks. On the other hand, SNAS4MTF (Chen et al., 2021a) proposes to use architecture search to form an end-to-end forecasting architecture framework. Furthermore, Auto-PyTorch TS (Deng et al., 2022) provides a uniform search space that includes many forecasting modules. The optimizers can freely assemble these modules to form new architectures. However, these works still focus on homogeneous architecture designs, where the type of *Seq* decoders is restricted by the decision of *Seq* Encoders. For instance, an RNN encoder only allows RNN decoders (*Seq* decoder) or MLP decoders (*Flat* decoder). As the current neural architecture tends to focus more on the one-shot weight-sharing approaches (Jiang et al., 2023; Liu et al., 2019a; Wang et al., 2021; Zoph et al., 2018), in this work, we will show how to design a general one-shot model for time series forecasting that fits most of the forecasting models, making automated deep learning for time series forecasting substantially faster than before.

Evaluating an architecture could take lots of resources. One-shot NAS approaches alleviate this by constructing a supernet where the weights of all the candidate architectures are inherited from this super network. However, training a supernet still requires lots of resources. Zero-Cost (ZC) proxies (Chen et al., 2022; Krishnakumar et al., 2022) try to solve this problem by estimating the performance of the target model without updating the model weights, and hence could greatly accelerate the searching process. However, nearly all the existing zero-cost proxies (Turner, 2019; Lee et al., 2019; Tanaka et al., 2020; Wang et al., 2019; Abdelfattah et al., 2021; Lin et al., 2021; Ning et al., 2021) are developed on the computer vision NAS tasks that only contain one architecture type (convolutional operations). It is still unclear if the ZC proxies can be generalized to compare the performances between different operation types.

3 Problem Setting

Time series forecasting tasks aim to predict the values of the target variables for a number of iterations after a certain time step with the observed same variables and several other feature variables. Formally, given a dataset that is composed of multiple series $\mathcal{D} = \{\mathcal{D}_i\}_{i=1}^{N}$, where each series is composed of the past observed targets $\mathbf{y}_{i,1:T_i}$, past observed features $\mathbf{x}_{i,1:T_i}$ and known future features $\hat{\mathbf{x}}_{i,T_i+1:T_i+H}$. Given a required forecasting horizon H, the model is asked to predict the target variables $\hat{\mathbf{y}}_{i,T_i+1:T_i+H}$ with all the available information:

$$\hat{\mathbf{y}}_{i,T_i+1:T_i+H} = f(\mathbf{y}_{i,1:T_i}, \mathbf{x}_{i,1:T_i}, \hat{\mathbf{x}}_{i,T_i+1:T_i+H}; \theta) \tag{1}$$

Two model families can mainly handle multi-horizontal forecasting tasks: the auto-regressive approach and the non-auto-regressive approach. Auto-regressive approaches (Box et al., 2015; Salinas et al., 2020) only predict one step within one forward pass and iteratively use the predicted value as the known feature that can be further fed to the model. On the other hand, non-auto-regressive models (Nie et al., 2023; Zeng et al., 2023; Zhou et al., 2021) directly generate multiple forecasting values within one forward pass. In this work, we mainly focus on searching for non-auto-regressive architectures.

Neural architecture search aims at finding the optimal architecture α_* for a given task:

$$\min_{\alpha} \mathcal{L}_{val}(\theta^*, \alpha) \quad \text{s.t.} \quad \theta^* \in \arg\min_{\theta} \mathcal{L}_{train}(\theta, \alpha) \tag{2}$$

The search space in previous NAS work still focused on the homogeneous search space where all the operations belong to the same architecture families. This work presents a unified heterogeneous search space containing most of the potential architectures applied for time series forecasting tasks.

4 Search Space Design

It is a common observation and a foundational assumption of NAS that no single architecture family always outperforms the others; based on the results of Deng et al. (2022), we believe that this also holds for time series forecasting tasks. We are unlikely to define a single type of model that works for all datasets. Additionally, some of the architecture might be dependent on the other decision choices. For instance, if we decide to have an encoder-only *Seq Net*, there is no need for us to search for a *Seq* decoder. Here, we propose a hierarchical search space that incorporates most of the forecasting architecture families described in Section 2.1. We will start from the most basic operation level and gradually decrease the granularity until our search space contains all the required components.

4.1 Operation Level

The first level, the operation level, describes the operations that can be used in the network. As shown in Figure 1, our search space follows the DARTS (Liu et al., 2019a) search space, a cell-based architecture where each cell is a directed acyclic graph that contains N nodes, including N_{in} input nodes. Each node represents a latent feature map, and the edges that connect the nodes are the operations applied to the latent feature maps. DARTS defines two types of cells: normal cells and reduction cells. These two cell types share the same form of input feature maps. Therefore, they can be easily concatenated to form an architecture. However, this is not the case for forecasting tasks. *Seq Net* and *Flat Net* transform the input features differently. We cannot easily connect a *Seq* cell after a *Flat* cell and vice versa. Hence, we provide a search space for each of the model families. Detailed information about each operation can be found in the appendix A.

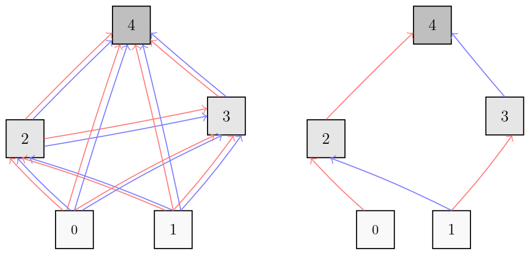


Figure 1: Opeation level Search Space. The nodes 0 and 1 are input nodes that receive the network inputs or the outputs from the other cells. Node 4 is the output node. Each colored edge represents an operation. The Search space (Left) as is a fully connected directed acyclic graph. Once we have finished the search, we get the final architecture (Right).

For the *Seq Net* families, we consider the following operations: 1. MLPMixer (Chen et al., 2023), an all-MLP architecture that applies a linear layer to feature and time dimensions, respectively, 2. LSTM (Hochreiter and Schmidhuber, 1997), 3. GRU (Cho et al., 2014), 4. Transformer (Vaswani et al., 2017), 5. TCN (Bai et al., 2018), 6. SepTCN (Luo and Wang, 2024), and 7. skip connections. These operations are spread to encoder and decoder architectures, which will be discussed in the following section.

For the *Flat Net* families, we have: 1. Linear (Zeng et al., 2023), a single linear layer that encodes the past information to the future variable, 2. NBEATs (Oreshkin et al., 2020), a repeatedly stacked MLP block, where each block contains a set of fully connected (FC) backbone layers, a forecast, and a backcast head, and 3. skip connections. There are several variants in NBEATS modules: generic model, trend model, and seasonality model. We incorporate all these modules into our search space, providing another two operations. Since the only difference between different NBeats modules is their forecast and backcast head, we ask these N-BEATs modules to share the same FC layers backbones. In total, we have five operations for each edge in the *Flat Net* cell.

Unlike the previous search space that focused on one single architecture type (Ying et al., 2019; Klyuchnikov et al., 2020; Krishnakumar et al., 2022; Mehta et al., 2022), the operations contained in DARTS-TS come from different families. Therefore, the outputs from each operation might have completely different distributions or even different output formats for the same input. The zero-cost proxies that are proven to be efficient on other benchmarks, such as *flops* or *number of parameters* (Krishnakumar et al., 2022; Wan et al., 2022), might no longer work well within this search space.

4.2 Micro Network Level

Flat operations only receives the past information $\mathbf{y}_{i,1:T_i}$. Therefore, we stack several *Flat* cells as a *Flat Net*. As shown in Figure 2a, the past targets are first transposed and then fed to the encoder layers. The transposed target, i.e., the backcast part, and a zero tensor whose length is equal to the forecasting horizon, i.e., the forecast part, are fed to the *Flat* encoders. Finally, the forecast output is fed to the forecasting head to predict the target values.[3]

Different from the *Flat Net*, we decompose the *Seq* architectures into two parts: encoders and decoders. The encoders encode the past observed values into an embedding and feed them to the decoder networks. The design of the *Seq* encoder is similar to the *Flat* Encoder, and we stack the encoder cells to form the encoder network. However, as discussed in Section 2.1, two potential ways exist to transform the encoder latent features to the forecasting heads. Therefore, we design the following two types of decoder networks: *Seq* decoder and *Flat* decoder for *Seq* encoder.

The design of *Seq* Encoder and *Seq* Decoder architecture has been widely applied in previous architecture works (Lim et al., 2021; Sutskever et al., 2014; Vaswani et al., 2017). However, architecture decoders might require different information from the encoder network. For instance, a Transformer decoder (Vaswani et al., 2017) only requires the output from the last layer of the corresponding encoder; an RNN decoder would need the hidden states from the corresponding encoder layer. We record the following information stored by each edge:

1. The output hidden feature map of this edge. It will then be contacted with the corresponding decoder feature maps and fed to the TCN decoder network.

2. The last step's feature map. This value is considered as a hidden state that can be fed to the corresponding GRU and LSTM layers to initialize their states.

3. The cell gate state that is applied to initialize the hidden cell states of the corresponding LSTM layer. If the encoder is not an LSTM network, following the idea of stitchable network (Pan et al., 2023) that uses a linear layer to stitch two networks with different shapes, we use a linear layer that transforms the hidden states into the cell gate states.

Hence, we record all the related information as intermediate states during the forward pass. This information is then used to inform the decoder networks of the information provided by the encoder.

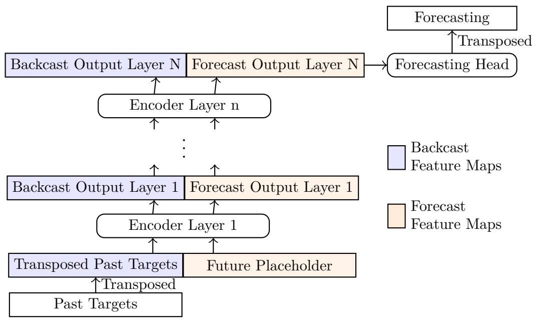
On the other hand, we might not want a complex decoder architecture since the information provided to the decoder and most of the features presented in the past are no longer available in the future. Hence, here we define another type of decoder for *Seq* network: the linear (or flat) decoder for *Seq Net.* We apply one linear layer across the time series dimension (Chen et al., 2023; Zeng et al., 2023) that maps the encoder output and the available future information to the decoder output feature. This feature is then fed to the forecasting head to generate the final prediction result. The overall *Seq Net* architecture is presented in Figure 2b.

We also consider the choice of the *Seq Net* decoder as part of the architecture search procedure. Hence, we assign another set of architecture parameters to the output of the two decoder architectures. This architecture is jointly optimized with the other operations architecture weights introduced in Section 4.1.

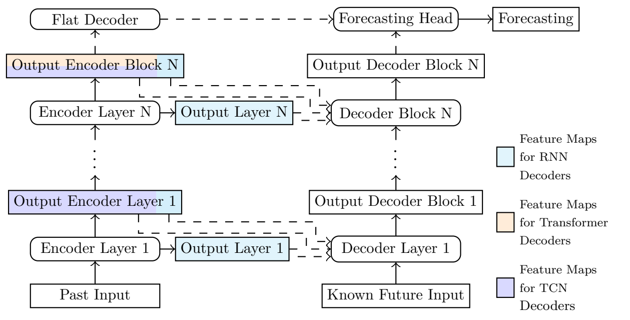
4.3 Macro Architecture Level

The *Flat Net* and *Seq Net* families defined above encode the input data from different perspectives: *Flat Net* families encode the input series along the time series dimension, while *Seq Net* families encode the input series across different variations. Hence, the two architectures can complement each other, and we concatenate the two architectures sequentially.

[3]We note that the head here does not necessarily need to be a network module since *Flat* net only predicts one variable each time.



(a) Search Space for *Flat* cells



(b) Search Space for *Seq* cells

Figure 2: Micro-level search space design for Flat and Seq cells.

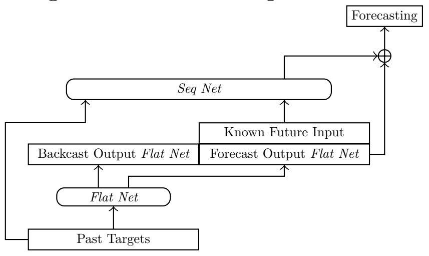


Figure 3: Marco Search Space.

As shown in Figure 3, the past target values are first fed to the *Flat Net* which results in a backcast and forecasting feature maps. The forecast feature maps are then concatenated with the known future features and fed to the *Seq* decoder. Finally, the final forecasting result is the weighted sum of both *Flat Net* and *Seq Net*[4]:

$$f_{model}(\mathbf{y}_{i,1:T_i}) = w_{seq} f_{seq}(\mathbf{y}_{i,1:T_i}, f_{flat}(\mathbf{y}_{i,1:T_i})) + w_{flat} f_{flat}(\mathbf{y}_{i,1:T_i}) \tag{3}$$

These weights are considered an architecture parameter that can be jointly learned with the other architecture parameters described in the aforementioned sections.

5 Searching for the Optimal Architectures

Here, we provide an exemplary searching strategy, i.e., DARTS (Liu et al., 2019b; Wang et al., 2021), to search within our search space. DARTS assigns a weight for each of the operations within its search space and optimizes these architecture weights jointly with the model weights using gradient descent. Some of the modules, such as Dropout (Srivastava et al., 2014) and Batch Normalization (Ioffe and Szegedy, 2015), behave differently during training and inference time. We thus switch these modules to the *eval* mode when we update the architecture weights with validation losses to simulate the evaluation process.

During the search phase, we divide the dataset into training and validation sets with the same size. The training set and validation sets are then used to optimize the architecture weights and architecture parameters, respectively. We follow the common practice for the training-validation split in time series forecasting tasks: the validation set is located at the tail of the training set. The first half of the dataset is considered as the training set that is used to optimize the weights of the supernet, while the second part of the dataset is the validation set, which is used to optimize the architecture parameters. We apply RevINV (Kim et al., 2022) during both the searching and training phases to ensure that the input features fed to the networks stay in the same distribution.

5.1 Hierarchical Pruning of the One-Shot Model

Vanilla DARTS are shown to be unstable during the search process and might prefer to select architectures that are dominated by skip connections (Chu et al., 2020; Jiang et al., 2023; Wang et al., 2021; Zela et al.,

[4]For the sake of simplicity, we omit the feature variables $\mathbf{x}_{i,1:T_i}$ and $\hat{\mathbf{x}}_{i,T_i+1:T_i+H}$ that are fed to the f_{model} and f_{seq}

	DARTS-TS		iTransformer		ModernTCN		PatchTST		TSMixer		DLinear		TimesNet		Autoformer	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTm1	0.344	0.368	0.368	0.395	0.362	0.386	0.352	0.381	0.382	0.408	0.360	0.382	0.402	0.415	0.619	0.539
ETTm2	0.253	0.306	0.273	0.330	0.261	0.319	0.257	0.315	0.446	0.477	0.267	0.329	0.290	0.339	0.423	0.441
ETTh1	0.413	0.423	0.473	0.468	0.404	0.421	0.415	0.429	0.515	0.503	0.447	0.456	0.486	0.482	0.559	0.529
ETTh2	0.351	0.388	0.387	0.415	0.333	0.385	0.330	0.379	0.571	0.548	0.422	0.439	0.399	0.435	0.739	0.621
ECL	0.156	0.245	0.166	0.261	0.163	0.257	0.161	0.254	0.168	0.272	0.166	0.264	0.203	0.302	0.221	0.334
Exchange	0.378	0.409	0.411	0.440	0.525	0.505	0.385	0.417	0.366	0.452	0.382	0.419	0.540	0.524	1.010	0.775
Weather	0.230	0.262	0.239	0.274	0.231	0.269	0.229	0.265	0.222	0.288	0.244	0.297	0.249	0.287	0.398	0.431
Traffic	0.394	0.260	0.387	0.273	0.421	0.287	0.398	0.267	0.541	0.413	0.434	0.295	0.624	0.336	0.671	0.412

Table 1: The results of the mean performance over all the long-term forecasting datasets. The best models are marked as red while the second best model is marked with underlines.

	DARTS-TS		iTransformer		ModernTCN		PatchTST		TSMixer		DLinear		TimesNet		Autoformer	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
PEMS03	0.137	0.241	0.458	0.408	0.408	0.419	0.199	0.291	0.162	0.281	0.264	0.358	0.151	0.248	0.554	0.541
PEMS04	0.112	0.221	0.127	0.237	0.488	0.471	0.266	0.338	0.136	0.254	0.264	0.355	0.127	0.239	0.763	0.669
PEMS07	0.102	0.204	0.504	0.477	0.304	0.374	0.209	0.293	0.150	0.251	0.311	0.373	0.132	0.233	0.361	0.438
PEMS08	0.169	0.256	0.202	0.269	0.510	0.479	0.231	0.304	0.225	0.305	0.331	0.376	0.191	0.267	0.739	0.627

Table 2: The results of the mean performance over all the PEMS datasets. The best models are marked as red while the second-best model is marked with underlines.

2020). Hence, once the weights and architecture parameters are trained, as the last step, we select the optimal operations and edges using the perturbation-based approach (Wang et al., 2021).

Given that our search space is a hierarchical search space, some of the operations might be dependent on others, and we have to consider that for the pruning phase. For instance, if we select a Linear Decoder, then we do not need to further select the operations in the *Seq* decoder. However, on the other hand, this also indicates that our estimate will be biased if we select the choice of decoder before selecting any edge operations in the *Seq* Decoder. Hence, we propose pruning our network from the lowest granularity level and gradually increasing the granularity level until we prune the operations in our search space. Once all the operations are selected, we further prune the edges of our network using the same perturbation-based approach. Finally, only two edges are preserved for each node, including the cell output node.[5]

6 Experiments

In this section, we first demonstrate that DARTS-TS can identify the optimal architecture, which is comparable to many other handcrafted architectures on various datasets. We then provide a brief analysis of our search space to show the potential challenges in our search space. After that, we provide an analysis of the latency of the optimal architecture. Finally, we show some optimal architectures as examples.

6.1 Time series forecasting tasks

We evaluate our architecture search framework on the popular long-term forecasting datasets introduced by Wu et al. (2021) and Zeng et al. (2023): Weather, Traffic, Exchange Rate, Electricity (ECL), and four ETT datasets. Additionally, we evaluate our approach on the four PEMS datasets (Chen et al., 2001) that record the public traffic network data in California.

Following the experiments setup from the other works, we set the forecasting horizon H for ETT, ECL, Exchange Rate, Weather, and Traffic dataset as $\{96, 192, 336, 720\}$. For the four PEMS datasets, we set these values as $\{12, 24, 48, 96\}$. We compare our results with the following baselines: PathTST (Nie et al., 2023), ModernTCN (Luo and Wang, 2024), DLinear (Zeng et al., 2023), TSMixer (Chen et al., 2023). iTransformer (Liu et al., 2023), Autoformer (Wu et al., 2021) and TimesNet (Wu et al., 2023). For the sake of fair comparison, we follow the setup from PatchTST (Nie et al., 2023) and set the input sequence length

[5]We note that this setting is different from the traditional NAS framework, where all the edges towards the output nodes are preserved. This approach helps us to reduce the architecture size and required latency further.

for all models to 336. We ran each experiment 5 times with different seeds and recorded their mean and standard deviation, respectively [6].

The results for long-term forecasting tasks are shown in Table 1. The best results are marked in red. The full results can be found in the appendix C. Our network achieves the best or comparable results on the ECL, ETTm, Traffic, and Weather Datasets. Overall, we show that DARTS-TS automatically found architectures that are comparable to or better than many other hand-crafted architectures specifically designed for forecasting tasks with only the vanilla series modules.

While on another benchmark, the PEMS dataset, DARTS-TS outperforms all the other baselines for all the tasks, as shown in Table 2. This shows that DARTS-TS could adapt to different tasks and suggests the optimal architectures for different time series distributions.

6.2 Search Space Analysis

Unlike the search space in other tasks, which contains only a single operation type (Ying et al., 2019; Klyuchnikov et al., 2020; Krishnakumar et al., 2022; Mehta et al., 2022), our search space encompasses operations from different architectures. This differs from the previous architecture search space and may present additional challenges to the optimizers. To provide a basic understanding of our search space, we randomly sample $3,000$ configurations from our search space and evaluate them on the ECL and Traffic datasets.

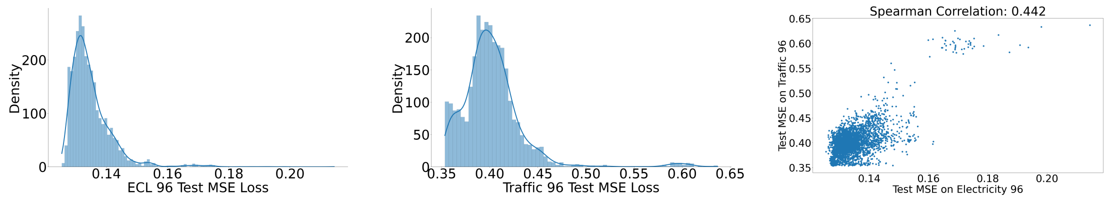


Figure 4: Random Search Evaluation Results on the ECL and Traffic datasets. (Left), MSE loss distributions on the ECL dataset. (Middle), MSE loss distributions on the Traffic dataset. (Right), loss distributions on the Traffic ECL dataset from the same architecture configurations

The random configuration performances are shown in Figure 4. For the same set of hyperparameter configurations, the MSE losses on these two tasks follow different distributions: losses on the ECL datasets are more skewed towards the lower bounds, while the losses on the Traffic datasets are more uniformly distributed. This indicates different difficulty levels for the optimizers to search for the optimal architectures on different datasets. To check whether the performance of the same hyperparameter configurations could transfer to different tasks, we plot the performance of the same configurations from the two tasks in the right part of Figure 4. The overall trend shows the consistent performance of the worst-performing configurations on both datasets. However, the performance diverges as we move towards the near-optimal configurations. Hence, the optimal configuration for one task may not work equally well for another task. There is still a need to further search for the optimal model on the target dataset. However, since the worst configurations perform consistently on the two tasks, we could incorporate the runs from other tasks as a prior (Hvarfner et al., 2022; Mallik et al., 2023).

Operations in our search space come from different architecture types. This provides further challenge for the optimizers as the performance difference among the operations in the search space can no longer be described with simple proxies. For instance, in the computer vision NAS benchmark search space, a 3×3 convolutional layer could, in most cases, achieve a better performance than a 1×1 layer, as the *number of parameters* of 3×3 convolutional layer is larger than that of 1×1 convolutional layer. However, it is questionable if a Transformer layer (with $12d^2$ parameters) will outperform an LSTM layer (with $8d^2$ parameters) since these two operations follow different computational rules. Hence, the widely applied zero-cost proxies applied in the computer vision NAS benchmarks might no longer work in our search space.

[6]Our code can be found on `https://anonymous.4open.science/r/OneShotForecastingNAS-CDD5/`.

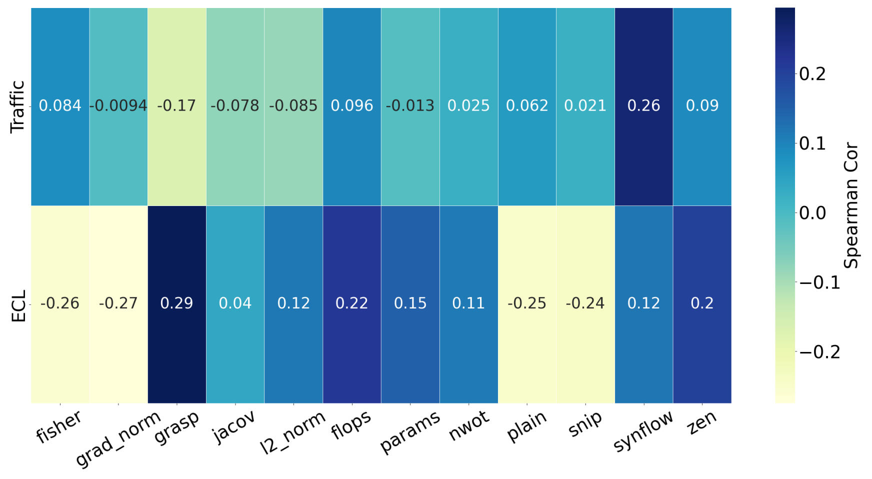


Figure 5: Spearman correlation between different ZC metrics and the evaluation test MSE losses

We computed the zero-cost (ZC) metrics for the sampled architectures. Following the settings from NAS-Bench-Suite-Zero (Krishnakumar et al., 2022), we evaluate the following zero cost proxies: fisher (Turner, 2019), flops, number of parameters (Ning et al., 2021), grad-norm, l2-norm, plain (Abdelfattah et al., 2021), grasp (Wang et al., 2019), jacov, nwot (Mellor et al., 2020), snip (Lee et al., 2019), synflow (Tanaka et al., 2020), and zen-score (Lin et al., 2021). We evaluate the ZC scores of all the sampled models and compute the Spearman correlation between the ZC scores and their test performance. The results are shown in Figure 5. Supervisingly, even though many zero-cost proxies do not limit their application to pure CNN or image classification tasks, nearly all of the zero-cost proxies fail in our search space. This indicates that the existing zero-cost proxies might not generalize well to more complex search spaces. Additionally, as shown in Figure 4, even the same configuration might perform differently on different datasets. This further shows the necessity of defining new data-dependent zero-cost proxies that work well across different architecture types.

To provide further insight into the design of ZC proxies for our search space, we check the performance of the architectures that contain at least one operation within our search space (Lopes et al., 2023). This provides a preliminary estimation of the strength of each operation on the architecture performance. The result is shown in Figure 6. Although the sequential network with sequential decoder families performs similarly to the sequential network with linear decoder on the ECL dataset, the gap becomes larger when the same architecture is evaluated on the Traffic dataset. However, the linear decoders might also lead to a much worse model, as the worst-performing linear decoders' loss is much higher than that of the sequential decoders.

Among the sequence operations, the TCN families (TCN and separated TCN models) achieve better median and 25th quantile performance on both datasets, which indicates that the dataset requires the models to focus more on local dependencies. While transformer families dominate the other benchmark, their median and 25th quantile losses are higher than those of the other operations. Although the MLP mixer encoders are quite close (and sometimes are even optimal) to the other operations, MLP Mixer decoders perform worse than the other sequential decoding operations. This highlights the importance of incorporating various encoder-decoder architectures into forecasting architecture designs.

For flat operations, MLP Flat and NBEATS-Generic achieve better performance compared to NBEATS-Seasonal and NBEATS-Trend models. However, the top-performing NBEATS-Seasonal models achieve a lower loss compared to the top-performing MLP flat layers. This might indicate that the inductive bias contained in the NBEATS seasonal models, i.e., the forecasting results are periodic, would help the model achieve better performance.

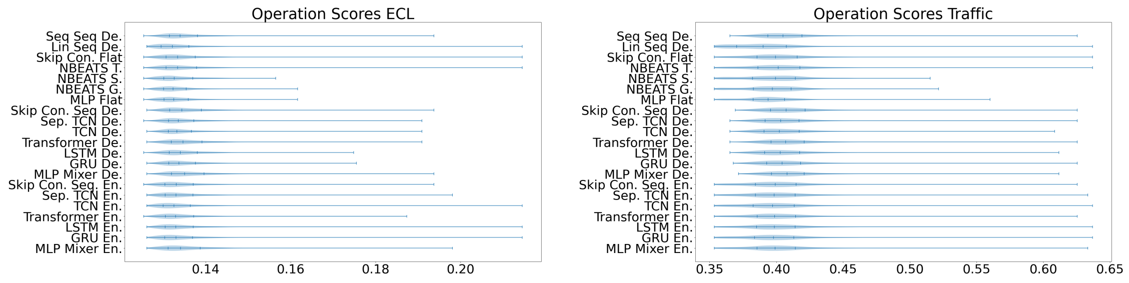


Figure 6: Performance of the randomly sampled models that contain at least one operation

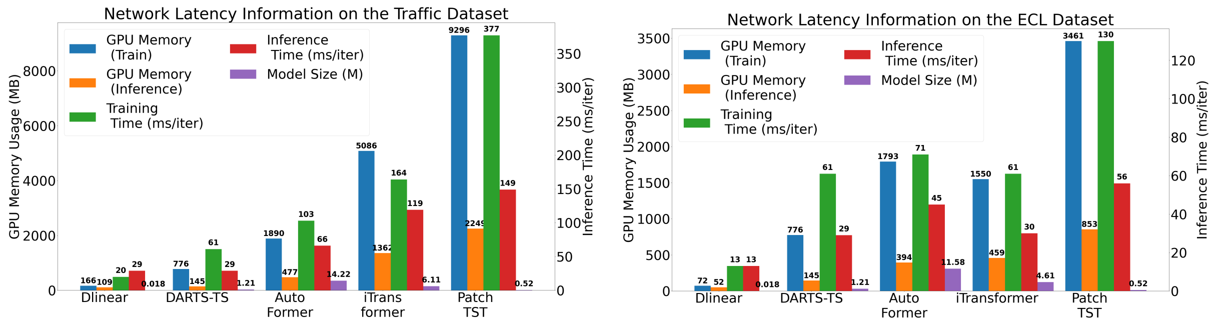


Figure 7: Latency information of different networks on the Traffic (left) and ECL (right) dataset. We set the look-back window size and the forecast horizon as 96. The batch size for both training and inference is set as 32

6.3 Model efficiency analysis

The growing demand for forecasting models has posed more challenges to the forecasting networks: the network should be fast, such that it can quickly predict the following trend. Additionally, networks need to contain fewer parameters and consume less memory so we can deploy them on embedded systems. To further show that our network could find an efficient and strong network, we ask all the networks to do a single forward pass and backpropagation with the series within the Traffic and ECL dataset, where each series contains 862 and 321 variables, respectively. We set the batch size of the series to 32 and the look-back window size to 96. The networks are then asked to predict the future series with a forecasting horizon of 96. This experiment is executed on one single Nvidia 2080 TI GPU with 11 GB GPU RAM [7]. Due to this memory constraint, some of the networks, such as ModernTCN, cannot fit into this GPU with our setup and do not appear in this comparison.

As shown in Figure 7, while having a comparable performance with iTransformer and PatchTST on the traffic dataset, our approach requires around 2x less GPU memory and is faster than the iTransformer and 3x less GPU memory and speed up compared to PatchTST during the training phases. During the test phases, the required GPU memory is even further reduced to around 300 MB, which is 4x less than the iTransformer and 6x less than the PatchTST, and only requires 3x more memory compared to a linear layer. A similar trend can be observed on the ECL dataset: DARTS-TS requires 5x less GPU memory and is 2x faster than PatchTST to achieve a better performance.

6.4 The optimal architectures

We show one of the optimal architectures found on the ECL dataset in Figure 8. Its *Seq Net* is an encoder-only architecture that is composed of MLPMixer, TCN, and Separate TCN modules. TCN modules are still

[7]However, we search the one-shot model on an Nvidia A100 GPU with 40 GB GPU RAM.

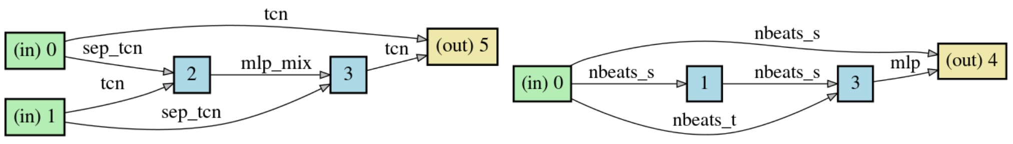


Figure 8: One optimal architecture on the ECL dataset, this is an encoder-only architecture and will be trained with an MSE loss

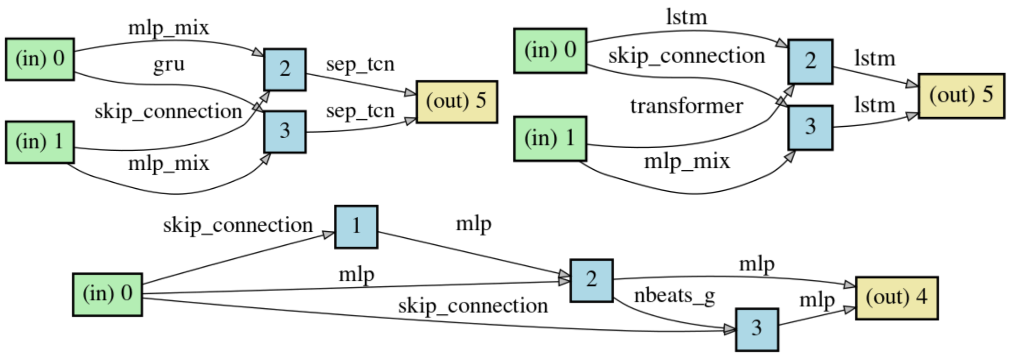


Figure 9: One optimal architecture on the ETTm2 dataset, this is an encoder-decoder architecture and will be trained with a quantile loss

preferable over the other components, showing that the ECL dataset might prefer a model that focuses on the local correlation. While the *Flat Net* contains lots of NBEATS seasonal modules, this indicates the strong seasonal and little trend signal that the dataset contains.

We present another encoder-decoder architecture in Figure 9. This architecture is optimized on the ETTm2 dataset. It applies two MLP-Mixer layers to the input node to first collect the global information from the raw input sequence and then apply two Separate TCN modules on top of that. Our optimizer also selects many LSTM modules for the decoder architectures, even if the corresponding encoder edges do not provide any hidden states. Additionally, no TCN module in the decoder layer. This is different from the optimal encoder architecture, where lots of TCN family components are selected. This indicates that the decoder networks would require modules that provide a global perspective to utilize all the information from the encoder networks, and therefore, the priority of TCN modules might decrease. More optimized architectures can be found in the appendix E.

7 Discussion and Future Work

This work proposes a general search space for time series forecasting tasks. Our search space allows the components in the search space to freely connect to each other and form a new network. This search space contains most of the forecasting architectures and can be easily extended to other frameworks. For instance, iTransformer (Liu et al., 2023) can be considered as a special case of the *Flat Net* and searched jointly with the other modules from this family. Decomposing multi-variant series into single variant series and applying a special kernel result in PatchTST (Nie et al., 2023) and then we can search for the optimal architecture jointly with the other *Seq Net*.

In Section 6, we showed that DARTS-TS can search for a lightweight architecture while keeping strong performance on various datasets. However, unlike traditional supervised problems where all the sample instances are i.i.d., time series data might have the problem of distribution shift. The optimal model searched on the validation set might no longer work well on the test set. This provides a future challenge for the AutoML forecasting frameworks from the meta-level, i.e., they need to be able to pre-determine the optimal approach to evaluate the generalization ability.

References

Proceedings of the International Conference on Learning Representations (ICLR'19), 2019. Published online: `iclr.cc`.

Proceedings of the International Conference on Learning Representations (ICLR'20), 2020. Published online: `iclr.cc`.

Proceedings of the International Conference on Learning Representations (ICLR'21), 2021. Published online: `iclr.cc`.

Proceedings of the International Conference on Learning Representations (ICLR'22), 2022. Published online: `iclr.cc`.

International Conference on Learning Representations (ICLR'23), 2023. Published online: `iclr.cc`.

M. Abdelfattah, A. Mehrotra, L. Dudziak, and N. Lane. Zero-cost proxies for lightweight NAS. In *Proceedings of the International Conference on Learning Representations (ICLR'21)* icl (2021). Published online: `iclr.cc`.

A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. Türkmen, and Y. Wang. Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, 21:116:1–116:6, 2020.

A. Ansari, L. Stella, A. Türkmen, X. Zhang, P. Mercado, H. Shen, O. Shchur, S. Rangapuram, S. Pineda-Arango, S. Kapoor, J. Zschiegner, D. Maddix, M. Mahoney, K. Torkkola, A. Wilson, M. Bohlke-Schneider, and Y. Wang. Chronos: Learning the language of time series. 2024.

R. Hyndmanand G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 3. edition, 2021.

J. Ba, J. Kiros, and G. Hinton. Layer normalization, 2016.

S. Bai, J. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv:1803.01271 [cs.LG]*, 2018.

J. Beitner. PyTorch Forecasting: Time series forecasting with PyTorch. `github.com/jdb78/pytorch-forecasting`, 2020.

G. Box, G. Jenkins, G. Reinsel, and G. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In Larochelle et al. (2020), pages 1877–1901.

C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia. Freeway performance measurement system: Mining loop detector data. *Transportation Research Record*, 2001. URL `https://doi.org/10.3141/1748-12`.

D. Chen, L. Chen, Z. Shang, Y. Zhang, B. Wen, and C. Yang. Scale-aware neural architecture search for multivariate time series forecasting, 2021a.

H. Chen, M. Lin, X. Sun, and H. Li. NAS-bench-zero: A large scale dataset for understanding zero-shot neural architecture search, 2022. URL `https://openreview.net/forum?id=hP-SILoczR`.

M. Chen, H. Peng, J. Fu, and H. Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the 24nd IEEE/CVF International Conference on Computer Vision (ICCV'21)* cvf (2021), pages 12270–12280.

S. Chen, C. Li, S. Arik, N. Yoder, and T. Pfister. TSMixer: An all-MLP architecture for time series forecasting. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=wbpxTuXgm0`.

K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734. Association for Computational Linguistics, 2014.

X. Chu, T. Zhou, B. Zhang, and J. Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, editors, *16th European Conference on Computer Vision (ECCV'20)*, pages 465–480. Springer, Springer, 2020.

J. Chung, K. Kastner, L. Dinh, K. Goel, A. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. *arXiv:1506.02216v6 [cs.LG]*, 2015.

Proceedings of the 24nd IEEE/CVF International Conference on Computer Vision (ICCV'21), 2021. cvfandieee, IEEE.

A. Das, W. Kong, R. Sen, and Y. Zhou. A decoder-only foundation model for time-series forecasting. 2023.

D. Deng, F. Karl, F. Hutter, B. Bischl, and M. Lindauer. Efficient automated deep learning for time series forecasting. In *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD*. ACM, 2022. URL `https://doi.org/10.1007/978-3-031-26409-2_40`.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186. Association for Computational Linguistics, 2019.

A. Didolkar, K. Gupta, A. Goyal, N. Gundavarapu, A. Lamb, N. Rosemary Ke, and Y. Bengio. Temporal latent bottleneck: Synthesis of fast and slow processing mechanisms in sequence learning. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS*, 2022.

R. Godahewa, C. Bergmeir, G. Webb, R. Hyndman, and P. Montero-Manso. Monash time series forecasting archive. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*. Curran Associates, 2021.

I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors. *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*, 2017. Curran Associates.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778. Computer Vision Foundation and IEEE Computer Society, IEEE, 2016.

H. Hewamalage, C. Bergmeir, and K. Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, pages 388–427, 2021.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. Based on TR FKI-207-95, TUM (1995).

S. Hochreiter, A. Younger, and P. Conwell. Learning to learn using gradient descent. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proceedings of the 11th International Conference on Artificial Neural Networks (ICANN'01)*, pages 87–94. Springer, 2001.

C. Hvarfner, D. Stoll, A. Souza, L. Nardi, M. Lindauer, and F. Hutter. πBO: Augmenting Acquisition Functions with User Beliefs for Bayesian Optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'22)* icl (2022). Published online: `iclr.cc`.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, volume 37. Omnipress, 2015.

S. Jiang, Z. Ji, G. Zhu, C. Yuan, and Y. Huang. Operation-level early stopping for robustifying differentiable NAS. In *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, 2023. URL `https://openreview.net/forum?id=yAOwkf4FyL`.

H. Jin, Q. Song, and X. Hu. Auto-Keras: An efficient neural architecture search system. In A. Teredesai, V. Kumar, Y. Li, R. Rosales, E. Terzi, and G. Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*, pages 1946–1956. ACM Press, 2019.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In Guyon et al. (2017).

T. Kim, J. Kim, Y. Tae, C. Park, J. Choi, and J. Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *Proceedings of the International Conference on Learning Representations (ICLR'22)* icl (2022). Published online: `iclr.cc`.

N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, and E. Burnaev. NAS-Bench-NLP: Neural Architecture Search benchmark for Natural Language Processing. *arXiv:2006.07116v1 [cs.LG]*, 2020.

A. Krishnakumar, C. White, A. Zela, Renbo R. Tu, M. Safari, and F. Hutter. Nas-bench-suite-zero: Accelerating research on zero cost proxies. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2022.

G. Lai, W. Chang, Y. Yang, and H. Liu. Modeling long- and short-term temporal patterns with deep neural networks. In K. Thompson, Q. Mei, B.Davison, Y. Liu, and E. Yilmaz, editors, *International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104. ACM, 2018. URL `https://doi.org/10.1145/3209978.3210006`.

I. Lana, J. Del Ser, M. Vélez, and E. Vlahogianni. Road traffic forecasting: Recent advances and new challenges. *IEEE Intell. Transp. Syst. Mag.*, 10(2):93–109, 2018.

H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors. *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, 2020. Curran Associates.

N. Lee, T. Ajanthan, and P. Torr. Snip: single-shot network pruning based on connection sensitivity. In *Proceedings of the International Conference on Learning Representations (ICLR'19)* icl (2019). Published online: `iclr.cc`.

S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In Wallach et al. (2019), pages 5244–5254.

B. Lim, S. Arık, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

M. Lin, P. Wang, Z. Sun, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin. Zen-nas: A zero-shot nas for high-performance image recognition. In *Proceedings of the 24nd IEEE/CVF International Conference on Computer Vision (ICCV'21)* cvf (2021), pages 347–356.

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*, 2018. Published online: `iclr.cc`.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'19)* icl (2019). Published online: `iclr.cc`.

S. Liu, H. Yu, C. Liao, J. Li, W. Lin, A. Liu, and S. Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *Proceedings of the International Conference on Learning Representations (ICLR'22)* icl (2022). URL `https://openreview.net/forum?id=0EXmFzUn5I`. Published online: `iclr.cc`.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.11692 [cs.CL]*, 2019b.

Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long. itransformer: Inverted transformers are effective for time series forecasting, 2023.

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the 24nd IEEE/CVF International Conference on Computer Vision (ICCV'21)* cvf (2021), pages 10012–10022.

V. Lopes, B. Degardin, and L. Alexandre. Are neural architecture search benchmarks well designed? a deeper look into operation importance. 2023. URL `https://arxiv.org/abs/2303.16938`.

D. Luo and X. Wang. ModernTCN: A modern pure convolution structure for general time series analysis. In *International Conference on Learning Representations (ICLR'24)*, 2024. URL `https://openreview.net/forum?id=vpJMJerXHU`. Published online: `iclr.cc`.

Z. Lyu, A. Ororbia, and T. Desell. Online evolutionary neural architecture search for multivariate non-stationary time series forecasting. *Appl. Soft Comput.*, 2023. URL `https://doi.org/10.1016/j.asoc.2023.110522`.

S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, pages 54–74, 2020.

S. Makridakis, E. Spiliotis, and V. Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, pages 1346–1364, 2022. URL `https://www.sciencedirect.com/science/article/pii/S0169207021001874`.

N. Mallik, E. Bergman, C. Hvarfner, D. Stoll, M. Janowski, M. Lindauer, L. Nardi, and F. Hutter. Priorband: practical hyperparameter optimization in the age of deep learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Proceedings of the 37th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, 2023.

Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter. NAS-Bench-Suite: NAS evaluation is (now) surprisingly easy. In *Proceedings of the International Conference on Learning Representations (ICLR'22)* icl (2022). Published online: `iclr.cc`.

J. Mellor, J. Turner, A. Storkey, and E. Crowley. Neural Architecture Search without training. In H. Daume III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, volume 98. Proceedings of Machine Learning Research, 2020.

Y. Nie, N. Nguyen, P. Sinthong, and J. Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations (ICLR'23)* icl (2023). URL `https://openreview.net/pdf?id=Jbdc0vTOcol`. Published online: `iclr.cc`.

X. Ning, C. Tang, W. Li, Z. Zhou, S. Liang, H. Yang, and Y. Wang. Evaluating efficient performance estimators of neural architectures. In Ranzato et al. (2021).

A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. In *the 9th ISCA Speech Synthesis Workshop*, page 125, 2016.

B. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-BEATS: neural basis expansion analysis for interpretable time series forecasting. In *Proceedings of the International Conference on Learning Representations (ICLR'20)* icl (2020). Published online: `iclr.cc`.

Z. Pan, J. Cai, and B. Zhuang. Stitchable neural networks. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'23)*, pages 16102–16112. Computer Vision Foundation and IEEE Computer Society, IEEE, 2023. URL `https://doi.org/10.1109/CVPR52729.2023.01545`.

A. Paszke, S. Gross, F. Massa, A. Lerer, et al. PyTorch: An imperative style, high-performance deep learning library. In Wallach et al. (2019), pages 8024–8035.

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient Neural Architecture Search via parameter sharing. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80. Proceedings of Machine Learning Research, 2018.

M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors. *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*, 2021. Curran Associates.

O. Ronneberger, P. Fischer, and T. Brox. U-Net convolutional networks for biomedical image segmentation. *Medical Image Computing and Computer-Assisted Intervention*, pages 234–241, 2015.

D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, pages 1181–1191, 2020.

X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pages 802–810. Curran Associates, 2015.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'14)*. Curran Associates, 2014.

H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In Larochelle et al. (2020).

A. Trindade. ElectricityLoadDiagrams20112014. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C58C86.

R. Turner. The bayes opt benchmark documentation. `https://bayesmark.readthedocs.io/en/latest/`, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In Guyon et al. (2017).

H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche Buc, E. Fox, and R. Garnett, editors. *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*, 2019. Curran Associates.

X. Wan, B. Ru, P. Esperança, and Z. Li. On redundancy and diversity in cell-based neural architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'22)* icl (2022). Published online: `iclr.cc`.

C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations* icl (2019). Published online: `iclr.cc`.

R. Wang, M. Cheng, X. Chen, X. Tang, and C. Hsieh. Rethinking architecture selection in differentiable nas. In *International Conference on Learning Representation* icl (2021). Published online: `iclr.cc`.

R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. A multi-horizon quantile recurrent forecaster. In *31st Conference on Neural Information Processing Systems, Time Series Workshop*, 2017.

C. White, W. Neiswanger, and Y. Savani. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In Yang et al. (2021), pages 10293–10301.

H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with Auto-Correlation for long-term series forecasting. In Ranzato et al. (2021).

H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long. Timesnet: Temporal 2d-variation modeling for general time series analysis. In *International Conference on Learning Representations (ICLR'23)* icl (2023). URL `https://openreview.net/pdf?id=ju_Uqw384Oq`. Published online: `iclr.cc`.

Q. Yang, K. Leyton-Brown, and Mausam, editors. *Proceedings of the Thirty-Fifth Conference on Artificial Intelligence (AAAI'21)*, 2021. Association for the Advancement of Artificial Intelligence, AAAI Press.

C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. NAS-Bench-101: Towards reproducible Neural Architecture Search. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML'19)*, volume 97, pages 7105–7114. Proceedings of Machine Learning Research, 2019.

A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. In *Proceedings of the International Conference on Learning Representations (ICLR'20)* icl (2020). URL `https://openreview.net/forum?id=H1gDNyrKDS`. Published online: `iclr.cc`.

A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting? In B. Williams, S. Bernardini, Y. Chen, and J. Neville, editors, *Proceedings of the Thirty-Seventh Conference on Artificial Intelligence (AAAI'23)*, pages 11121–11128. Association for the Advancement of Artificial Intelligence, AAAI Press, 2023. URL `https://doi.org/10.1609/aaai.v37i9.26317`.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In Yang et al. (2021). URL `https://doi.org/10.1609/aaai.v35i12.17325`.

T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning (ICML'22)*, volume 162 of *Proceedings of Machine Learning Research*. PMLR, 2022. URL `https://proceedings.mlr.press/v162/zhou22g.html`.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-Pytorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:3079–3090, 2021.

B. Zoph and Q. V. Le. Neural Architecture Search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

B. Zoph, V. Vasudevan, J. Shlens, and Q. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'18)*. Computer Vision Foundation and IEEE Computer Society, IEEE, 2018.

A Operations Deatails

In section 4.1, we briefly introduced the operations within our search space. Here, we will provide the details of these operations.

A.1 ***Seq Net***

For *Seq* net, encoders and decoders share the same operation sets. Overall, we have the following operations:

- TSMixer (Chen et al., 2023), a full MLP-based Sequential operation. Each TSMixer operation is constructed by the time and feature mixing blocks. The time mixing blocks use a fully connected (FC) layer to mix the information across different time steps. In contrast, the feature mixing block uses another set of FC layers to enhance the information within each channel. Our TSMixer encoders follow the design from Chen et al. (2023), and the size of the feature mixing layer is set as $2 \times d_{model}$. For TSMixer decoders, we first concrete the input feature map with the encoder network outputs. This concatenated feature is then provided to the time mixing modules to recover its size and return it to the forecasting horizon. Additionally, we use LayerNorm (Ba et al., 2016) instead of BatchNorm (Ioffe and Szegedy, 2015) in TSMixer to ensure that the operations within an edge generate the feature maps that follow the same distribution (since all other components in our module used LayerNorm to normalize the feature maps).

- LSTM (Hochreiter and Schmidhuber, 1997) is an RNN model. It maintains a set of cell gate states to control the amount of information passed to the next time steps. Therefore, it suffers less from the known gradient explosion problems in the RNN families. However, the introduction of the gates brings lots of additional parameters to the modules. As described in Section 4.2, the hidden states of the LSTM decoder are initialized by the last time step of the encoder feature from the corresponding layer and another feature generated with a linear embedding (or directly from the corresponding LSTM encoder).

- GRU (Chung et al., 2015) is yet another type of RNN network. It only maintains one hidden state and therefore requires much less amount of parameters and computations compared to the LSTM. The setting of the GRU Encoder/Decoder is nearly the same as the LSTM families. The only difference is that we do not maintain an additional state to initialize the GRU decoders.

- Transformer (Vaswani et al., 2017) has attracted lots of attention from different research fields and is therefore widely applied in time series forecasting tasks. Here, we use the vanilla Transformer implemented in PyTorch (Paszke et al., 2019) with the hidden size to be $4 \times d_{model}$.

- TCN (Bai et al., 2018) is a type of CNN network that can capture the local correlations among different time steps. However, the receptive field of the TCN network is restricted by its kernel size. To efficiently increase the receptive field without introducing too much computation overhead with a larger kernel, TCN implemented dilated convolution operations. Here we implement a similar approach, for edge $i < -j$ that starts from node j to node i as cell k, we set its dilation as $2^{(j+k-n_{in})}$, where n_{in} is the number of inputs of the current cell. Hence, the deeper convolutional layers will have a larger receptive field. For the TCN decoders, we concatenate the feature maps from the corresponding encoder layer with our input feature and feed them together to our TCN network. This idea is similar to U-Net (Ronneberger et al., 2015), where features with similar levels should be gathered together.

- SepTCN (Luo and Wang, 2024) is a variation of the vanilla TCN. We replace the full convolutional operations in TCN with a combination of a separated TCN model with another 1×1 linear layer.

- Skip Connection, an identity layer that passes its input to the next level. However, for the skip connection encoder, we still have a linear layer to provide initial cell gate states to the corresponding LSTM decoder layers.

Another type of *Seq* decoder is a linear decoder. We apply a linear layer that transforms the encoder out feature map with size $R^{[B,L,N]}$ to $R^{[B,H,N]}$ and feed it further to the forecasting heads.

A.2 *Flat Net*

We only consider the MLP families in our *Flat Net* to minimize the computational overhead when applying our approaches to problems with higher series amounts. Given an input feature series with shape $[B, N, L]$, we first concatenate it with a zero tensor with shape $[B, N, H]$ that represents the prediction results. Then this concatenated tensor is fed to the *Flat* encoder.

This architecture family includes:

- a simple Linear model (Zeng et al., 2023). This linear layer maps its input features with shape $[B, N, L + H]$ to a feature map whose size is equal to the forecasting horizon: $[B, N, H]$. Then the output feature is concatenated with the first part of the input feature maps. If the network only contains skip connections for all but the last layer, then this network becomes a DLinear model (Zeng et al., 2023). If the operation is not the output layer, we attach an activation and normalization layer to introduce some non-linearity.

- NBEATS (Oreshkin et al., 2020) modules. NBEATS is a hierarchical module where each model is composed of multiple stacks. Each stack contains multiple blocks. Each block has an FC stack with multiple FC layers, a forecasting, and a backcasting head. In our search space, each NBEATS edge corresponds to a NBEATS block. NBEATs provides three variations: generic, trend, and seasonal. We include them all in our search space. Since the only difference between these variations is their prediction heads. We ask the models to share the same FC layer backbones and only diverge at the forecasting heads.

- Skip Connection, a skip connection layer.

A.3 Forecasting Heads

We also consider the forecasting heads as part of the operations within our graph. Each of these heads is composed of one or multiple linear layers that map the *Seq Net* [8] output feature maps to the desired multiple-variable target values. These linear layers are then trained with a set of specific training loss. Let's assume that the target value is $\mathbf{y}$ and prediction value is $\hat{\mathbf{y}}$

- Quantile loss (Lim et al., 2021; Wen et al., 2017) predicts the percentiles of the target values. A quantile head can be composed of multiple heads and each of the head is asked to predict a q quantile. Given a required quantile value q, the quantile loss is computed by $\mathcal{L}_q = \max(q(\mathbf{y}-\hat{\mathbf{y}})+(1-q)(\hat{\mathbf{y}}-\mathbf{y}))$. In our network, we used the following quantile values: $\{0.1, 0.5, 0.9\}$. The final prediction is given by the 0.5 quantile values. A quantile head is then a set of linear layers whose size is the number of quantile values

- MSE loss is yet another popular choice in time series forecasting tasks. It is computed by $\mathcal{L}_{MSE} = (\hat{\mathbf{y}} - \mathbf{y})^2$. An MSE head is a single linear layer whose weights are updated with MSE loss.

- MAE loss is similar to MSE loss. However, instead of computing the mean square error from MSE, it computes the mean absolute error: $\mathcal{L}_{MAE} = |\hat{\mathbf{y}} - \mathbf{y}|$. Similar to MSE layer, an MAE head is also composed of one linear layer but its weights are updated with MAE losses.

We stack these forecasting heads on top of the *Seq Net* decoders and optimize their architecture weights with validation losses.

[8] For *Flat Net*, there is no need to have an additional head if the loss only requires one output

B Experiment Details

We show detailed information on the dataset that we applied in Table 3. Given the great discrepancy in variable size between different datasets, we divide the datasets into two groups: we assign a smaller model to the datasets with fewer variables, such as Weather, Exchange Rate, and ETTs. We then attach a normalization layer within the linear decoder. For the other datasets, we design an architecture search space with a relatively large model. Additionally, we removed the normalization layer in the linear decoder since the number of target variables is larger than our model size, and the final forecasting head does not need to recover the distribution of the target variable from the normalized feature maps.

We run our experiments on a Cluster equipped with Nvidia A100 40 GB GPUs and AMD Milan 7763 CPUs. For each dataset, we perform the search over the smallest forecasting horizons and evaluate the same optimal model on all the other forecasting horizons. Each task is repeated 5 times. The resources spent for each evaluation depend on the size of the dataset. It takes roughly 4 hours to evaluate smaller datasets such as ETThs and ExchangeRate. Other tasks might require up to 10 GPU hours. The ablation study requires another 500 GPU hours. Overall, it takes roughly 1200 GPU hours to finish the experiments.

For all the baselines, we use their official implementation from PatchTST[9], ModernTCN[10] and TSMixer[11], while for the other baselines, we take the implementation from Time-Series-Library[12].

Dataset	# Time steps	# Variables
ECL	26304	321
Traffic	17544	862
Weather	62696	21
ExchangeRate	7588	9
ETTh1	17420	7
ETTh2	17420	7
ETTm1	69680	7
ETTm2	69680	7
PEMS03	26208	358
PEMS04	16992	307
PEMS07	28224	883
PEMS08	17856	170

Table 3: Data set information

Our architecture search framework is a two-stage approach. In the first stage, we search for the optimal architecture, while in the second stage, we train the proposed network from scratch. We preserve most of the searching hyperparameters from Liu et al. (2018). However, we apply ADAM instead of SGD during the test phases to optimize the network weights.

All the look-back window sizes are set as 336 for both the searching and testing phases. We divide the datasets into two groups based on their number of variables: 1. The PEMSs, traffic and ECL dataset belongs to the big dataset 2. The remaining datasets, including Weather and ETTs, are small datasets.

Both datasets share nearly the same architecture, which is a mixed network introduced in Section 4.3. The number of *Seq* and *Flat* cells are both set as 2, and they could receive 2 and 1 input variables, respectively. However, for the big dataset, we set the number of *Seq Net* hidden dimensions as 32, while this value for the small dataset is 8. This approach is also applied to the hidden NBATS dimension of *Flat Net*: we set this value as 256 for the big datasets and 96 for the small datasets.

[9]`https://github.com/yuqinie98/PatchTST`
[10]`https://github.com/luodhhh/ModernTCN`
[11]`https://github.com/google-research/google-research/tree/master/tsmixer`
[12]`https://github.com/thuml/Time-Series-Library`

		DARTS-TS		iTransformer		ModernTCN		PatchTST		TSMixer		DLinear		TimesNet		Autoformer	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ECL	96	0.129 (0.00)	0.217 (0.00)	0.132 (0.00)	0.228 (0.00)	0.135 (0.00)	0.231 (0.00)	0.130 (0.00)	0.223 (0.00)	0.137 (0.00)	0.241 (0.00)	0.140 (0.00)	0.237 (0.00)	0.184 (0.00)	0.287 (0.00)	0.204 (0.01)	0.320 (0.01)
	192	0.147 (0.00)	0.234 (0.00)	0.155 (0.00)	0.249 (0.00)	0.149 (0.00)	0.243 (0.00)	0.148 (0.00)	0.241 (0.00)	0.156 (0.00)	0.260 (0.00)	0.153 (0.00)	0.250 (0.00)	0.194 (0.00)	0.295 (0.00)	0.212 (0.00)	0.327 (0.00)
	336	0.164 (0.00)	0.253 (0.00)	0.171 (0.00)	0.266 (0.00)	0.165 (0.00)	0.259 (0.00)	0.165 (0.00)	0.259 (0.00)	0.173 (0.00)	0.281 (0.00)	0.169 (0.00)	0.267 (0.00)	0.197 (0.00)	0.299 (0.00)	0.215 (0.01)	0.328 (0.01)
	720	0.186 (0.00)	0.275 (0.00)	0.206 (0.01)	0.299 (0.01)	0.205 (0.00)	0.295 (0.00)	0.202 (0.00)	0.292 (0.00)	0.206 (0.00)	0.308 (0.00)	0.203 (0.000)	0.301 (0.000))	0.236 (0.03)	0.329 (0.02)	0.254 (0.01)	0.360 (0.01)
ETTh1	96	0.365 (0.00)	0.385 (0.00)	0.404 (0.00)	0.419 (0.00)	0.369 (0.00)	0.394 (0.00)	0.375 (0.00)	0.400 (0.00)	0.387 (0.00)	0.413 (0.01)	0.379 (0.01)	0.403 (0.01)	0.443 (0.01)	0.453 (0.01)	0.496 (0.01)	0.492 (0.01)
	192	0.405 (0.00)	0.411 (0.00)	0.451 (0.00)	0.449 (0.00)	0.407 (0.00)	0.415 (0.00)	0.413 (0.00)	0.420 (0.00)	0.428 (0.01)	0.439 (0.01)	0.415 (0.01)	0.427 (0.01)	0.486 (0.01)	0.482 (0.01)	0.532 (0.04)	0.509 (0.02)
	336	0.437 (0.01)	0.433 (0.01)	0.471 (0.00)	0.465 (0.00)	0.392 (0.00)	0.413 (0.00)	0.427 (0.00)	0.432 (0.00)	0.505 (0.01)	0.501 (0.01)	0.470 (0.03)	0.469 (0.03)	0.482 (0.01)	0.478 (0.01)	0.544 (0.03)	0.523 (0.01)
	720	0.448 (0.01)	0.462 (0.00)	0.565 (0.02)	0.538 (0.01)	0.450 (0.00)	0.461 (0.00)	0.444 (0.00)	0.463 (0.00)	0.741 (0.09)	0.658 (0.03)	0.524 (0.02)	0.527 (0.01)	0.534 (0.03)	0.515 (0.02)	0.662 (0.14)	0.592 (0.06)
ETTh2	96	0.276 (0.00)	0.332 (0.00)	0.305 (0.00)	0.361 (0.00)	0.264 (0.00)	0.333 (0.00)	0.275 (0.00)	0.336 (0.00)	0.370 (0.01)	0.436 (0.01)	0.283 (0.00)	0.347 (0.00)	0.363 (0.03)	0.408 (0.02)	0.517 (0.05)	0.534 (0.04)
	192	0.344 (0.00)	0.377 (0.00)	0.389 (0.01)	0.411 (0.00)	0.322 (0.00)	0.377 (0.00)	0.339 (0.00)	0.379 (0.00)	0.494 (0.03)	0.510 (0.02)	0.366 (0.02)	0.403 (0.01)	0.411 (0.02)	0.437 (0.01)	0.565 (0.09)	0.559 (0.05)
	336	0.377 (0.01)	0.406 (0.00)	0.420 (0.01)	0.434 (0.01)	0.315 (0.00)	0.377 (0.00)	0.328 (0.00)	0.381 (0.00)	0.586 (0.02)	0.559 (0.01)	0.428 (0.02)	0.450 (0.01)	0.394 (0.02)	0.438 (0.01)	0.757 (0.14)	0.642 (0.07)
	720	0.406 (0.01)	0.435 (0.00)	0.436 (0.01)	0.454 (0.00)	0.429 (0.00)	0.453 (0.00)	0.378 (0.00)	0.420 (0.00)	0.837 (0.07)	0.688 (0.03)	0.610 (0.06)	0.555 (0.03)	0.429 (0.03)	0.458 (0.01)	1.115 (0.17)	0.751 (0.06)
ETTm1	96	0.283 (0.00)	0.330 (0.00)	0.305 (0.00)	0.358 (0.00)	0.296 (0.00)	0.348 (0.00)	0.290 (0.00)	0.341 (0.00)	0.308 (0.01)	0.358 (0.01)	0.301 (0.00)	0.346 (0.00)	0.330 (0.01)	0.373 (0.00)	0.497 (0.04)	0.487 (0.02)
	192	0.320 (0.00)	0.354 (0.00)	0.343 (0.00)	0.380 (0.00)	0.348 (0.00)	0.378 (0.00)	0.333 (0.00)	0.369 (0.00)	0.347 (0.01)	0.386 (0.01)	0.337 (0.00)	0.368 (0.00)	0.430 (0.05)	0.423 (0.02)	0.591 (0.03)	0.528 (0.01)
	336	0.357 (0.00)	0.377 (0.00)	0.380 (0.00)	0.402 (0.00)	0.376 (0.00)	0.395 (0.00)	0.367 (0.00)	0.391 (0.00)	0.398 (0.01)	0.420 (0.01)	0.374 (0.00)	0.392 (0.00)	0.397 (0.00)	0.416 (0.00)	0.682 (0.06)	0.561 (0.02)
	720	0.418 (0.01)	0.412 (0.00)	0.441 (0.00)	0.438 (0.00)	0.430 (0.00)	0.421 (0.00)	0.417 (0.00)	0.422 (0.00)	0.474 (0.03)	0.470 (0.02)	0.427 (0.00)	0.423 (0.00)	0.450 (0.01)	0.446 (0.01)	0.703 (0.09)	0.579 (0.03)
ETTm2	96	0.162 (0.00)	0.244 (0.00)	0.179 (0.00)	0.268 (0.00)	0.170 (0.00)	0.257 (0.00)	0.165 (0.00)	0.254 (0.00)	0.182 (0.01)	0.293 (0.01)	0.166 (0.00)	0.258 (0.00)	0.190 (0.01)	0.277 (0.00)	0.332 (0.03)	0.392 (0.02)
	192	0.223 (0.00)	0.285 (0.00)	0.242 (0.00)	0.312 (0.00)	0.225 (0.00)	0.297 (0.00)	0.222 (0.00)	0.293 (0.00)	0.284 (0.03)	0.386 (0.03)	0.224 (0.00)	0.301 (0.00)	0.247 (0.01)	0.313 (0.00)	0.380 (0.08)	0.414 (0.04)
	336	0.274 (0.00)	0.320 (0.00)	0.292 (0.00)	0.344 (0.00)	0.283 (0.00)	0.335 (0.00)	0.277 (0.00)	0.329 (0.00)	0.499 (0.04)	0.535 (0.02)	0.280 (0.00)	0.339 (0.01)	0.312 (0.02)	0.354 (0.01)	0.432 (0.05)	0.452 (0.03)
	720	0.354 (0.00)	0.373 (0.00)	0.380 (0.01)	0.397 (0.00)	0.367 (0.00)	0.386 (0.00)	0.364 (0.00)	0.383 (0.00)	0.818 (0.03)	0.695 (0.01)	0.397 (0.01)	0.416 (0.00)	0.413 (0.01)	0.411 (0.00)	0.547 (0.09)	0.506 (0.05)
Exchange	96	0.088 (0.00)	0.210 (0.00)	0.099 (0.00)	0.227 (0.00)	0.169 (0.00)	0.305 (0.00)	0.093 (0.00)	0.213 (0.00)	0.113 (0.00)	0.259 (0.01)	0.084 (0.00)	0.203 (0.00)	0.167 (0.01)	0.305 (0.01)	0.541 (0.12)	0.560 (0.07)
	192	0.186 (0.00)	0.308 (0.00)	0.202 (0.00)	0.326 (0.00)	0.276 (0.00)	0.389 (0.00)	0.192 (0.00)	0.312 (0.00)	0.237 (0.03)	0.378 (0.01)	0.164 (0.01)	0.293 (0.00)	0.309 (0.02)	0.415 (0.01)	0.956 (0.24)	0.770 (0.12)
	336	0.350 (0.01)	0.428 (0.01)	0.397 (0.01)	0.466 (0.01)	0.449 (0.00)	0.505 (0.00)	0.350 (0.00)	0.431 (0.00)	0.443 (0.08)	0.519 (0.03)	0.355 (0.01)	0.453 (0.00)	0.487 (0.02)	0.534 (0.01)	1.290 (0.23)	0.903 (0.08)
	720	0.888 (0.03)	0.689 (0.01)	0.947 (0.01)	0.740 (0.00)	1.206 (0.02)	0.821 (0.01)	0.906 (0.00)	0.713 (0.00)	0.671 (0.11)	0.653 (0.04)	0.927 (0.05)	0.727 (0.02)	1.197 (0.07)	0.842 (0.02)	1.254 (0.03)	0.866 (0.01)
Traffic	96	0.358 (0.00)	0.240 (0.01)	0.356 (0.00)	0.258 (0.00)	0.398 (0.00)	0.275 (0.00)	0.367 (0.00)	0.250 (0.00)	0.488 (0.00)	0.381 (0.00)	0.410 (0.00)	0.282 (0.00)	0.605 (0.01)	0.330 (0.00)	0.682 (0.02)	0.415 (0.02)
	192	0.386 (0.01)	0.256 (0.00)	0.376 (0.00)	0.268 (0.00)	0.412 (0.00)	0.280 (0.00)	0.385 (0.00)	0.259 (0.00)	0.524 (0.01)	0.403 (0.00)	0.423 (0.00)	0.287 (0.00)	0.616 (0.00)	0.333 (0.01)	0.669 (0.03)	0.411 (0.02)
	336	0.398 (0.01)	0.262 (0.00)	0.389 (0.00)	0.274 (0.00)	0.424 (0.00)	0.287 (0.00)	0.399 (0.00)	0.267 (0.00)	0.556 (0.01)	0.423 (0.01)	0.436 (0.000)	0.296 (0.000)	0.624 (0.01)	0.335 (0.01)	0.674 (0.03)	0.415 (0.01)
	720	0.434 (0.00)	0.284 (0.00)	0.426 (0.00)	0.293 (0.00)	0.452 (0.00)	0.305 (0.00)	0.439 (0.01)	0.292 (0.01)	0.596 (0.01)	0.444 (0.01)	0.466 (0.000)	0.315 (0.000)	0.650 (0.01)	0.346 (0.00)	0.661 (0.02)	0.406 (0.01)
Weather	96	0.148 (0.00)	0.194 (0.01)	0.163 (0.00)	0.212 (0.00)	0.152 (0.00)	0.206 (0.00)	0.151 (0.00)	0.199 (0.00)	0.146 (0.00)	0.220 (0.00)	0.174 (0.00)	0.235 (0.00)	0.165 (0.00)	0.223 (0.00)	0.295 (0.01)	0.370 (0.01)
	192	0.195 (0.00)	0.239 (0.01)	0.207 (0.00)	0.253 (0.00)	0.197 (0.00)	0.247 (0.00)	0.196 (0.00)	0.242 (0.00)	0.192 (0.00)	0.267 (0.01)	0.216 (0.00)	0.274 (0.00)	0.216 (0.00)	0.267 (0.00)	0.382 (0.04)	0.431 (0.03)
	336	0.252 (0.01)	0.282 (0.01)	0.256 (0.00)	0.291 (0.00)	0.246 (0.00)	0.285 (0.00)	0.248 (0.00)	0.283 (0.00)	0.240 (0.00)	0.304 (0.01)	0.262 (0.00)	0.314 (0.00)	0.278 (0.01)	0.309 (0.01)	0.424 (0.04)	0.445 (0.02)
	720	0.323 (0.00)	0.331 (0.01)	0.330 (0.00)	0.340 (0.00)	0.327 (0.00)	0.338 (0.00)	0.319 (0.00)	0.335 (0.00)	0.311 (0.01)	0.359 (0.01)	0.325 (0.00)	0.365 (0.00)	0.338 (0.00)	0.349 (0.00)	0.493 (0.06)	0.476 (0.03)

Table 4: The full evaluation results on long-term forecasting datasets. We evaluate each model five times and take the mean of the final results. The optimal models are marked as red, and we underline the models that are not significantly worse than the optimal

		DARTS-TS		iTransformer		ModernTCN		PatchTST		TSMixer		DLinear		TimesNet		Autoformer	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
PEMS03	12	0.066 (0.00)	0.171 (0.00)	0.069 (0.00)	0.175 (0.00)	0.112 (0.00)	0.221 (0.00)	0.079 (0.00)	0.187 (0.00)	0.075 (0.00)	0.187 (0.00)	0.105 (0.00)	0.220 (0.00)	0.085 (0.00)	0.192 (0.00)	0.277 (0.06)	0.387 (0.04)
	24	0.097 (0.00)	0.206 (0.00)	0.098 (0.00)	0.209 (0.00)	0.173 (0.00)	0.281 (0.00)	0.124 (0.00)	0.235 (0.00)	0.113 (0.00)	0.238 (0.01)	0.182 (0.00)	0.296 (0.00)	0.110 (0.00)	0.216 (0.00)	0.422 (0.05)	0.466 (0.03)
	48	0.152 (0.01)	0.257 (0.01)	0.448 (0.57)	0.416 (0.28)	0.307 (0.00)	0.395 (0.00)	0.223 (0.00)	0.319 (0.00)	0.195 (0.02)	0.320 (0.02)	0.318 (0.00)	0.410 (0.00)	0.168 (0.01)	0.263 (0.00)	0.806 (0.08)	0.679 (0.04)
	96	0.234 (0.02)	0.331 (0.02)	1.215 (0.62)	0.831 (0.25)	1.041 (0.02)	0.779 (0.01)	0.368 (0.00)	0.425 (0.00)	0.266 (0.01)	0.380 (0.01)	0.450 (0.00)	0.507 (0.00)	0.242 (0.01)	0.321 (0.00)	0.710 (0.15)	0.634 (0.07)
PEMS04	12	0.073 (0.00)	0.176 (0.00)	0.081 (0.00)	0.188 (0.00)	0.132 (0.00)	0.245 (0.00)	0.101 (0.00)	0.209 (0.00)	0.085 (0.00)	0.195 (0.00)	0.115 (0.00)	0.228 (0.00)	0.088 (0.00)	0.197 (0.00)	0.562 (0.06)	0.577 (0.03)
	24	0.091 (0.00)	0.198 (0.00)	0.124 (0.00)	0.232 (0.00)	0.244 (0.00)	0.338 (0.00)	0.161 (0.00)	0.267 (0.00)	0.112 (0.01)	0.228 (0.01)	0.189 (0.00)	0.299 (0.00)	0.104 (0.00)	0.216 (0.00)	0.637 (0.10)	0.617 (0.05)
	48	0.120 (0.00)	0.232 (0.00)	0.135 (0.00)	0.248 (0.00)	0.452 (0.00)	0.482 (0.00)	0.294 (0.00)	0.369 (0.00)	0.159 (0.01)	0.278 (0.01)	0.323 (0.00)	0.407 (0.00)	0.138 (0.00)	0.252 (0.01)	1.002 (0.10)	0.775 (0.04)
	96	0.165 (0.00)	0.278 (0.00)	0.169 (0.00)	0.280 (0.00)	1.127 (0.00)	0.818 (0.00)	0.507 (0.00)	0.505 (0.00)	0.190 (0.01)	0.313 (0.01)	0.428 (0.00)	0.484 (0.00)	0.179 (0.00)	0.291 (0.00)	0.853 (0.24)	0.708 (0.08)
PEMS07	12	0.060 (0.00)	0.155 (0.00)	0.066 (0.00)	0.164 (0.00)	0.085 (0.00)	0.196 (0.00)	0.076 (0.00)	0.180 (0.00)	0.070 (0.00)	0.177 (0.00)	0.100 (0.00)	0.215 (0.00)	0.083 (0.00)	0.183 (0.00)	0.201 (0.02)	0.330 (0.02)
	24	0.081 (0.00)	0.180 (0.00)	0.087 (0.00)	0.190 (0.00)	0.127 (0.00)	0.245 (0.00)	0.127 (0.00)	0.234 (0.00)	0.105 (0.01)	0.221 (0.01)	0.189 (0.00)	0.302 (0.00)	0.101 (0.00)	0.204 (0.00)	0.304 (0.04)	0.402 (0.03)
	48	0.113 (0.01)	0.218 (0.01)	0.892 (0.12)	0.764 (0.08)	0.267 (0.01)	0.380 (0.01)	0.238 (0.00)	0.325 (0.00)	0.157 (0.01)	0.265 (0.00)	0.375 (0.00)	0.436 (0.00)	0.133 (0.00)	0.236 (0.00)	0.422 (0.13)	0.472 (0.08)
	96	0.156 (0.02)	0.262 (0.01)	0.972 (0.19)	0.789 (0.12)	0.736 (0.02)	0.673 (0.01)	0.394 (0.00)	0.432 (0.00)	0.268 (0.02)	0.342 (0.02)	0.579 (0.00)	0.540 (0.00)	0.211 (0.06)	0.308 (0.06)	0.519 (0.10)	0.546 (0.05)
PEMS08	12	0.074 (0.00)	0.175 (0.00)	0.089 (0.00)	0.193 (0.00)	0.125 (0.00)	0.239 (0.00)	0.091 (0.00)	0.195 (0.00)	0.095 (0.00)	0.203 (0.00)	0.112 (0.00)	0.223 (0.00)	0.110 (0.00)	0.208 (0.00)	0.467 (0.07)	0.503 (0.05)
	24	0.107 (0.01)	0.213 (0.01)	0.138 (0.00)	0.243 (0.00)	0.238 (0.00)	0.336 (0.00)	0.144 (0.00)	0.247 (0.00)	0.150 (0.01)	0.257 (0.01)	0.195 (0.00)	0.299 (0.00)	0.139 (0.00)	0.234 (0.00)	0.503 (0.07)	0.512 (0.05)
	48	0.178 (0.02)	0.277 (0.02)	0.237 (0.01)	0.277 (0.01)	0.528 (0.00)	0.534 (0.00)	0.254 (0.00)	0.332 (0.00)	0.256 (0.01)	0.344 (0.01)	0.382 (0.00)	0.431 (0.00)	0.194 (0.00)	0.277 (0.00)	0.964 (0.23)	0.729 (0.11)
	96	0.318 (0.04)	0.360 (0.04)	0.346 (0.07)	0.363 (0.05)	1.150 (0.00)	0.808 (0.00)	0.435 (0.00)	0.441 (0.00)	0.399 (0.02)	0.415 (0.02)	0.634 (0.00)	0.550 (0.01)	0.322 (0.01)	0.349 (0.01)	1.021 (0.14)	0.763 (0.06)

Table 5: The full evaluation results on PEMS datasets. The optimal models are marked as red, and we underline the models that are not significantly

Since the *Seq* encoder has 2 input nodes, instead of feeding the raw input value to both input nodes, we decompose the input nodes into trend-cyclical components by using an average moving (Wu et al., 2021; Zeng et al., 2023) to ask the edges in the cell to focus on different levels of information.

C Results on all the forecasting horizons

Table 4 and Table 5 show the results w.r.t. each forecasting horizon. While DARTS-TS requires much less amount of resources and parameters, it still shows comparable performances on many datasets.

D Ablation Study

D.1 The impact of window size

In our experiments, we fixed our window size to 336 and applied this window size to predict different forecasting horizons. However, since the look-back window size is an important hyperparameter in forecasting tasks, it is interesting to see if the model should always stick to the window where it is trained. To answer this, we ask the optimizer to search for an architecture that requires the input window size $\{96, 192, 336, 720\}$. We then evaluate each of these found architectures with the window sizes mentioned above.

We run this task on the ECL-96 dataset. The result is shown in Figure 10. We see that our model will perform better with the increase in search window size in general. While the search window size also influences the final evaluations, a model searched with a window size of 720 performs the worst when the architecture

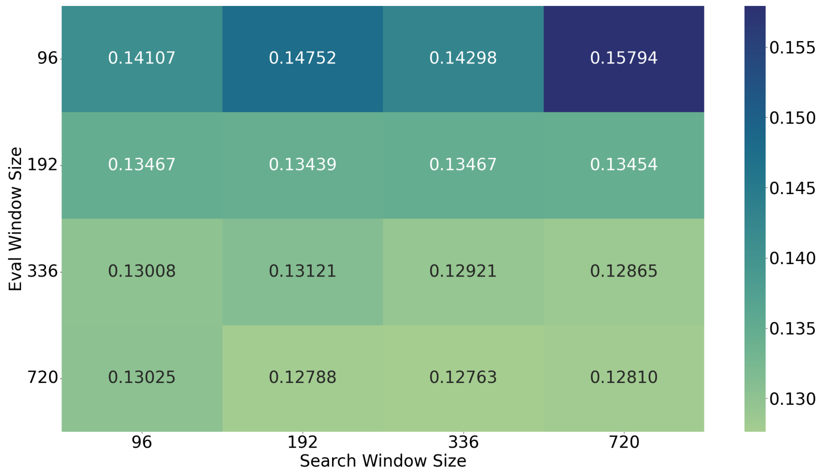


Figure 10: The impact of window size during searching and evaluations on the ECL-96 task

		DARTS-TS		Parallel		Seq Only		Flat Only		No Weights	
		MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ECL	96	0.129 (0.00)	0.217 (0.0)	0.128 (0.00)	0.219 (0.00)	0.206 (0.05)	0.305 (0.04)	0.128 (0.00)	0.221 (0.00)	0.139 (0.01)	0.234 (0.02)
ETTh1	96	0.365 (0.00)	0.385 (0.0)	0.371 (0.01)	0.392 (0.01)	0.466 (0.04)	0.463 (0.02)	0.383 (0.00)	0.406 (0.00)	0.366 (0.00)	0.385 (0.00)
ETTh2	96	0.276 (0.00)	0.332 (0.0)	0.282 (0.00)	0.341 (0.00)	0.382 (0.02)	0.421 (0.01)	0.312 (0.00)	0.361 (0.00)	0.279 (0.00)	0.338 (0.01)
ETTm1	96	0.283 (0.00)	0.330 (0.0)	0.287 (0.00)	0.338 (0.00)	0.366 (0.02)	0.402 (0.02)	0.295 (0.00)	0.345 (0.00)	0.286 (0.00)	0.334 (0.01)
ETTm2	96	0.162 (0.00)	0.244 (0.0)	0.166 (0.00)	0.249 (0.00)	0.221 (0.01)	0.301 (0.01)	0.173 (0.00)	0.255 (0.00)	0.162 (0.00)	0.246 (0.00)
Exchange	96	0.088 (0.00)	0.210 (0.0)	0.098 (0.00)	0.220 (0.00)	0.227 (0.03)	0.347 (0.02)	0.107 (0.01)	0.234 (0.01)	0.092 (0.00)	0.214 (0.00)
Traffic	96	0.358 (0.00)	0.240 (0.0)	0.370 (0.02)	0.244 (0.01)	0.572 (0.02)	0.312 (0.01)	0.359 (0.00)	0.249 (0.00)	0.452 (0.00)	0.300 (0.01)
Weather	96	0.148 (0.00)	0.194 (0.0)	0.150 (0.00)	0.195 (0.01)	0.191 (0.00)	0.244 (0.00)	0.151 (0.00)	0.201 (0.00)	0.157 (0.00)	0.209 (0.00)

Table 6: Ablation over the components in DARTS-TS

suggested by this optimizer is asked to make a prediction with a window size of 96 and vice versa. However, this gap becomes less if the search window size is closer to the eval window size. This indicates that the window size we used to search for the network should not be too far away from the actual window size used to evaluate the model.

D.2 Forecasting components

We constructed a hierarchical search space in Section 4. Here, we will show the efficiency of each component. We provide the following variations:

- Flat Only. This variation only contains the *Flat Net* in the search space.
- Seq Only. This variation only contains the *Seq Net* in the search space.
- Parallel. This variation is similar to our approach. However, the *Seq Net* receives only feature variables instead of the output of the *Flat Net*
- No Weights. This variation removed the weighted sum approach described in Section 4.3

The result is shown in Table 6. The Parallel approach is slightly worse than DARTS-TS on many datasets. However, there is a huge gap between Parallel and DARTS-TS on the Traffic dataset. While the Flat Only approach is generally worse than DARTS-TS on the ETT datasets. Overall, we show that the architectural design of DARTS-TS generally provides us with architectures that are robust across many datasets.

D.3 Separated Encoder-Decoder Search Space

In Section 4.2, we design a search space that allows the encoder-decoder pairs from the same edge to have different architecture types. This design decision enlarges the search space. Here, we demonstrate that the

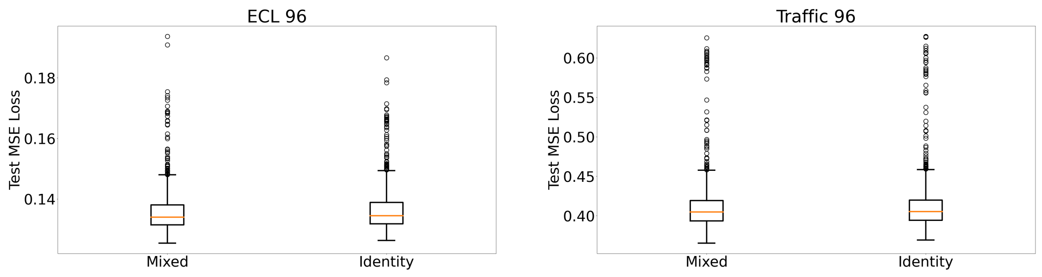


Figure 11: Performance of the architectures with mixed and identical encoder-decoder architectures

enlarged search space encompasses architectures that may outperform those with the same encoder-decoder architecture.

Hence, we check the random configurations from Section 6.2 and select all the configurations that contain *Seq* decoders. Then, for each decoder edge, instead of randomly sampling a new decoder component, we initialize the decoders with operations that are the same as those of the corresponding encoder nodes. We then reevaluate those architectures with the identical operations and compare them with the corresponding mixed architectures.

The result is shown in Figure 11. On both datasets, architectures with mixed operations achieve a lower minimum test loss (0.365 *vs* 0.369 on the Traffic dataset and 0.125 *vs* 0.126 on the ECL dataset). This shows that applying different operations to the encoders and decoders provides a potentially better model compared to the architectures that apply the same encoder and decoder operation within the search space.

E Optimal Architectures on other datasets

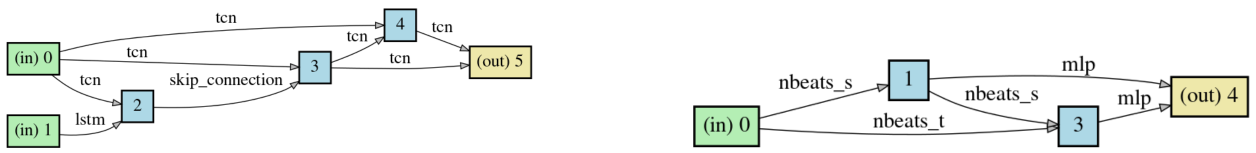


Figure 12: An optimal architecture on the ECL dataset. This is an encoder-only architecture

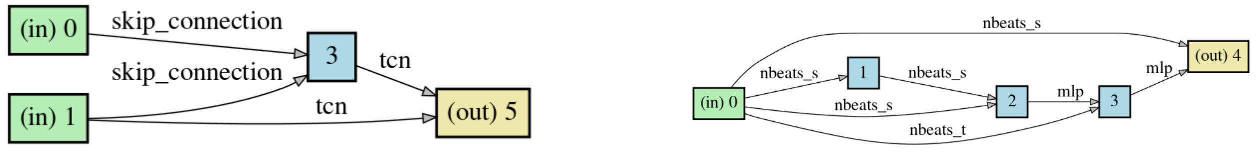


Figure 13: An optimal architecture on the Traffic dataset. This is an encoder-only architecture

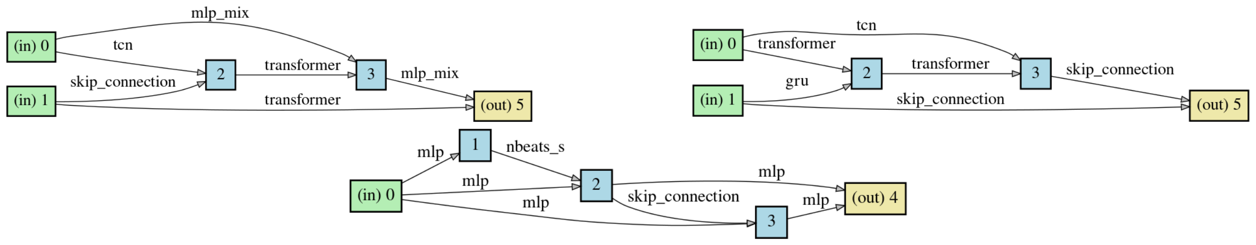


Figure 14: An optimal architecture on the ETTm1 dataset. This is an encoder-decoder architecture

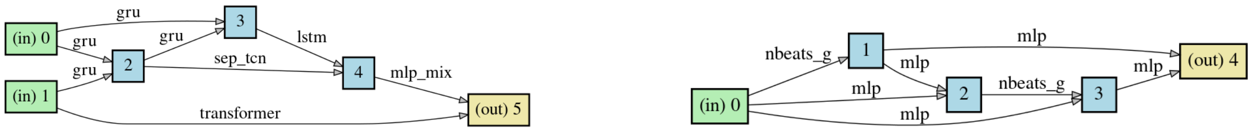


Figure 15: An optimal architecture on the ETTm2 dataset. This is an encoder-only architecture

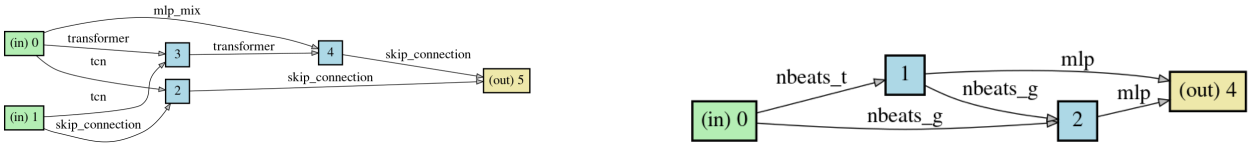


Figure 16: An optimal architecture on the Weather dataset. This is an encoder-only architecture

In Section 6.4, we showed one of the optimal architectures in the electricity dataset. In this section, we will provide more searched architectures.

Figure 12 shows yet another optimal architecture on the ECL dataset. We can see that TCN and NBEATS seasonal modules are still contained in the optimal modules, further confirming our conclusions in Section 6.

We also show some of the optimal architectures in Figure 13, 14, 15, and 16. We can see that no single operation dominates the optimal architecture, which shows the necessity of performing an architecture search for the optimal architecture.