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

Difan Deng
Institute of Artificial Intelligence
Leibniz University Hannover
d.deng@ai.uni-hannover.de

Marius Lindauer
Institute of Artificial Intelligence
Leibniz University Hannover & L3S Research Center
m.lindauer@ai.uni-hannover.de

Abstract

The rapid development of time series forecasting research has brought many deep learning-based modules in this field. However, despite the increasing amount of new forecasting architectures, it is still unclear if we have leveraged the full potential of these existing modules within an adequately designed architecture space. In this work, we propose a novel hierarchical neural architecture search (NAS) 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 specific to different forecasting tasks.

1 Introduction

Time series forecasting techniques are widely applied in different fields, e.g., energy consumption [45], business [36], or traffic planning [27]. However, unlike computer vision (CV) and natural language processing (NLP) tasks that are dominated by the CNN [19; 29; 55] and Transformer [7; 17; 34; 46] families, there is no clearly dominating architecture in time series forecasting tasks. Although there are lots of transformer-based approaches applied to forecasting models [4; 15; 49; 31; 54], many of them might even be outperformed by a simple linear baseline [52].

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. For example, a simple linear layer could outperform many state-of-the-art transformer models Zeng et al. [52].

Some recent works show the efficiency of transformers [33; 37] on long-time forecasting tasks. However, these approaches still cannot overcome the temporal dependencies issues for transformers. For example, PatchTST [37] augments the information within each token by constructing a patch with the data from multiple time steps, while iTransformer [33] simply encodes the entire input sequence into a token and tries constructing the connections among different variables.

On the other hand, many architectures were constructed for mining the local dependencies, such as CNNs [6; 35] and RNNs [20; 22]. These architectures might not work well on long-term dependencies due to the limited receptive field for CNNs, latent bottlenecks [18], or vanishing gradients in the case

NeurIPS 2024 Workshop on Time Series in the Age of Large Models.

of RNNs. Recent work such as ModernTCN [35] 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: we combine these architectures with other operations, such as transformer layers, to develop a new architecture that combines the best of two worlds [18; 26; 28].

Nevertheless, it is still unclear (i) which type of architecture we would like to construct, given the variability of different forecasting models [16; 38; 42; 52] 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 [9; 29; 55]. It is still unclear how to optimize the forecasting architectures due to their internal complexity. For instance, an encoder-decoder architecture [49; 53] might work well on some tasks, while the other tasks might prefer encoder-only architectures [33] or even MLP-only architectures [38; 52]. 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.

In this work, 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. This search space is compatible with any existing NAS search strategy and we shown that by applying DARTS to our search space, DARTS-TS, can robustly search for architectures that are comparable to the state-of-the-art models with much less computation resource requirements.

2 Search Space Design

It is a common observation and an assumption of NAS that no single architecture family always outperforms the others; based on our previous results of Deng et al. [16], we believe this also holds for time series forecasting tasks. We are unlikely to find a single architecture that is best for all datasets.

The architectures applied to time series forecasting can be classified into the *Seq Net* and the *Flat Net*. For a given input time series of size $[L, D]$, where L is the time series length and D is the time series dimension, *Seq Net*s are designed to handle this type of data format and generate another time sequence of size $[L, D_{out}]$ without destructing its format. Examples for *Seq Net* include RNN families [11; 21] and Transformers [46]. Conversely, the *Flat Net* families need to decompose the input sequence into a single vector $[L \times D]$ to fuse the information between different time steps. This type of models include MLP layers [52] and N-BEATS [38]. Further details are under the appendix A.

We propose a hierarchical search space incorporating many forecasting architecture families. We will start from the most basic operation level and gradually decrease the granularity until our search space contains all the required components.

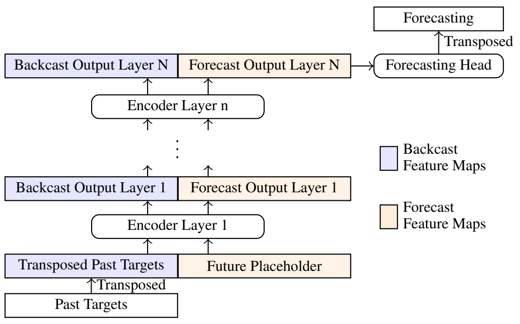
2.1 Operation Level

The first level, the operation level, describes the operations that can be used in the network. Our search space follows the DARTS [30] search space, a cell-based architecture where each cell contains multiple nodes. Each node represents a latent feature map, and the edges that connect the nodes are the operations applied to the latent feature maps. Since *Seq Net* and *Flat Net* have different output formats, and we cannot easily connect the two types of networks to each other, we design a search space for each family individually.

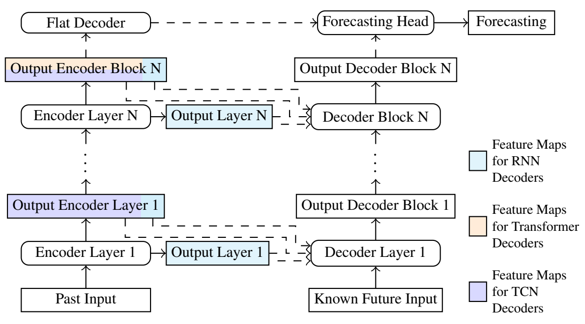
2.2 Micro Network Level

The second level, the micro-network level, connects different cells to form the *Flat Net* and *Seq Net* backbones.

Flat operations only receive the past information. Therefore, we stack several *Flat* cells as a *Flat Net*. As shown in Figure 1a, 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



(a) Search Space for *Flat* cells



(b) Search Space for *Seq* cells

Figure 1: Micro-level search space design for Flat and Seq cells.

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.[1]

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, two potential ways exist to transform the encoder latent features into 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 [28; 44; 46]. However, architecture decoders might require information different from that of the encoder network. For instance, a Transformer decoder [46] 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 hidden states 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 [39] 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.

Additionally, we define the linear (or flat) decoder for *Seq Net*. We apply one linear layer across the time series dimension [10; 52] 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 designed is presented in Figure 1b. Similar to the operations in each edge, only one decoder will be presented in the final architecture. Hence, we assign another set of architecture parameters to the output of the two decoder architectures. This architecture is jointly optimized with the other architecture weights introduced in Section 2.1.

2.3 Macro Architecture Level

The *Flat Net* and *Seq Net* families defined above encode the input data from different perspectives. Hence, the two architectures can complement each other, and we concatenate the two architectures sequentially.

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

	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.

As shown in Figure 2, 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*. These weights are considered an architecture parameter that can be jointly learned with the other architecture parameters described in the aforementioned sections.

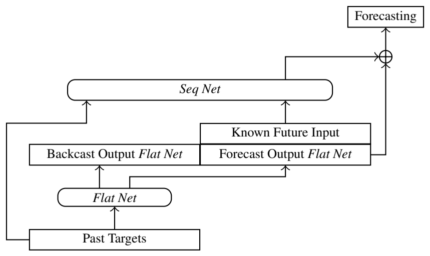


Figure 2: Architecture for Marco Search Space.

3 Searching for the Optimal Architectures

We search for the optimal architecture with DARTS [32; 47]. 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 [43] and Batch Normalization [23], 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. Further details can be found in the appendix.

4 Experiments

We evaluate our architecture search framework on the popular long-term forecasting datasets introduced by Wu et al. [49] and Zeng et al. [52]: Weather, Traffic, Exchange Rate, Electricity (ECL), and four ETT datasets. Following the experimental setup from [37; 49; 52], we set the forecasting horizon H for ETT, ECL, Exchange Rate, Weather, and Traffic dataset as $\{96, 192, 336, 720\}$. We compare our results with the following baselines: PathTST [37], ModernTCN [35], DLinear [52], TSMixer [10]. iTransformer [33], Autoformer [49] and TimesNet [50]. We fixed the input sequence of all the models to 336.

The results for long-term forecasting tasks are shown in Table 1. The best results are marked as red. The full results can be found in the appendix. 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.

5 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 the other frameworks. For instance, iTransformer [33] 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 [37] and then we can search for the optimal architecture jointly with the other *Seq Net*. Hence in future work, our approach can be easily extended to other frameworks to allow for searching for novel architectures.

Acknowledgements

Difan Deng and Marius Lindauer were supported by the Federal Ministry of Education and Research (BMBF) under the project AI service center KISSKI (grantno.01IS22093C).

References

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

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

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

[4] 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.

[5] J. Ba, J. Kiros, and G. Hinton. Layer normalization, 2016.

[6] 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.

[7] 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 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)*, pages 1877–1901. Curran Associates, 2020.

[8] 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`.

[9] 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 [14], pages 12270–12280.

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

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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 [2]. Published online: iclr.cc.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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 [2]. URL `https://openreview.net/forum?id=0EXmFzUn5I`. Published online: `iclr.cc`.

[32] 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]*, 2019.

[33] 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.

[34] 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 [14], pages 10012–10022.

[35] 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`.

[36] 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`.

[37] 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 [3]. URL `https://openreview.net/pdf?id=Jbdc0vTOcol`. Published online: `iclr.cc`.

[38] 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 [1]. Published online: `iclr.cc`.

[39] 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`.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, et al. PyTorch: An imperative style, high-performance deep learning library. In 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)*, pages 8024–8035. Curran Associates, 2019.

[41] 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.

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

[43] 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.

[44] 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.

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

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In 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)*. Curran Associates, Inc., 2017.

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

[48] 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.

[49] H. Wu, J. Xu, J. Wang, and M. Long. Autoformer: Decomposition transformers with Auto-Correlation for long-term series forecasting. In 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)*. Curran Associates, 2021.

[50] 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 [3]. URL `https://openreview.net/pdf?id=ju_Uqw384Oq`. Published online: `iclr.cc`.

[51] 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 [1]. URL `https://openreview.net/forum?id=H1gDNyrKDS`. Published online: `iclr.cc`.

[52] 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`.

[53] 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 Q. Yang, K. Leyton-Brown, and Mausam, editors, *Proceedings of the Thirty-Fifth Conference on Artificial Intelligence (AAAI'21)*. Association for the Advancement of Artificial Intelligence, AAAI Press, 2021. URL `https://doi.org/10.1609/aaai.v35i12.17325`.

[54] 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`.

[55] 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 Details

In section 2.1, 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 [10], 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. [10], 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 [5] instead of BatchNorm [23] 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 [21] 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 2.2, 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 [13] 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 [46] 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 [40] with the hidden size to be $4 \times d_{model}$.
- TCN [6] 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 [41], where features with similar levels should be gathered together.
- SepTCN [35] 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 [52]. 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 [52]. If the operation is not the output layer, we attach an activation and normalization layer to introduce some non-linearity.
- NBEATS [38] 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* [2] 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 [28; 48] 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.

B Optimization Details

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 training set that is used to optimize the weights of the supernet, while the second part of the data set is the validation set, which is used to optimize the architecture parameters. We apply RevINV [25] during both the searching and training phases to ensure that the input features fed to the networks stay in the same distribution.

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

B.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 [12; 24; 47; 51]. 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 [47].

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.[3]

C Experiment Details

Besides the benchmarks introduced in Section 4, we also evaluate our approach on the four PEMS datasets [8] that record the public traffic network data in California. We set the forecasting horizon for these benchmarks as $\{12, 24, 48, 96\}$..We show detailed information on the dataset that we applied in Table 2. 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.

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 2: 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

[3]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.

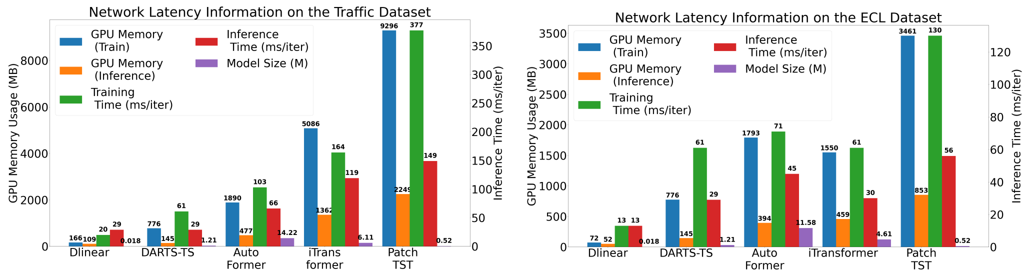


Figure 3: Latency information of different networks on the Traffic (left) and ECL (right) dataset. We set the look-back window size and the forecast horizon as 96. The batch size for both training and inference is set as 32

preserve most of the searching hyperparameters from Liu et al. [29]. 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 2.3. 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.

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 [49; 52] to ask the edges in the cell to focus on different levels of information.

D Model efficiency analysis

The growing demands for forecasting models have proposed 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 graph card with 11 GB GPU RAM [4]. 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 3, while having a comparable performance with iTransformer and PatchTST on the traffic dataset, our approach requires around 2x less GPU memories and is faster than the iTransformer and 3x less GPU memories 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 memories compared to a linear layer. A similar trend can be observed on the ECL dataset: DARTS-TS requires 5x less GPU memories and is 2x faster than PatchTST to achieve a better performance.

E Results on all the forecasting horizon

Table 3 and Table 4 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.

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

		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 3: 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 4: The full evaluation results on PEMS datasets. The optimal models are marked as red, and we underline the models that are not significantly

F Ablation Study

F.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 4. 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 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.

F.2 Forecasting components

We constructed a hierarchical search space in Section 2. 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*

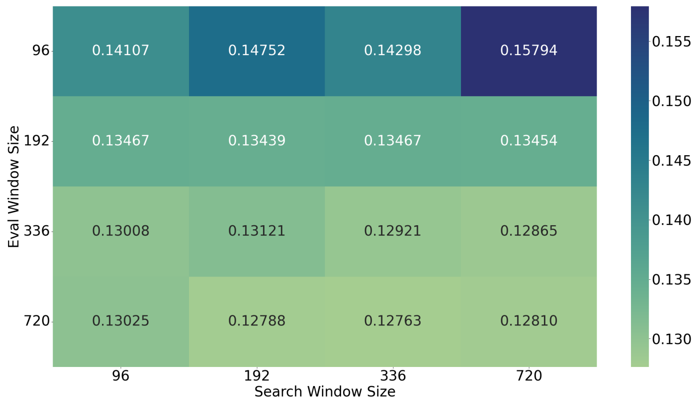


Figure 4: The impact of window size during searching and evaluations on the ECL-96 task

		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 5: Ablation over the components in DARTS-TS

- No Weights. This variation removed the weighted sum approach described in Section 2.3

The result is shown in Table 5. 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.

G Optimal Architectures

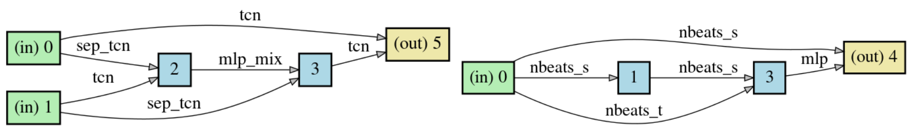


Figure 5: One optimal architecture on the ECL dataset, this is an encoder-only architecture and will be trained with a MSE loss.

In this section, we show the optimal architecture searched by our optimizers for each task. Each *Seq* cell is composed of 6 nodes, including 2 input nodes (nodes 0 and 1), 3 intermediate nodes (nodes 2, 3, and 4), and 1 output node (node 5). Each node represents an intermediate feature map. Each edge connecting the nodes represents a *Seq* network module. If multiple edges point toward the same node, we sum their output values and store the sum value within the output node. For instance, in Figure 5, we show the optimal *Seq* cell in the left. The feature map in node 0 is passed through one separable TCN layer to node 2 and one TCN layer to the final output node 5. Node 3 receives the output feature map from node 2 with an MLP-Mixer layer and 1 with a seperatable TCN layer, respectively. Since the final output, node 5, does not require the information from node 4, we can delete node 4 from the optimal architecture.

Figure 5 shows an 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 4.

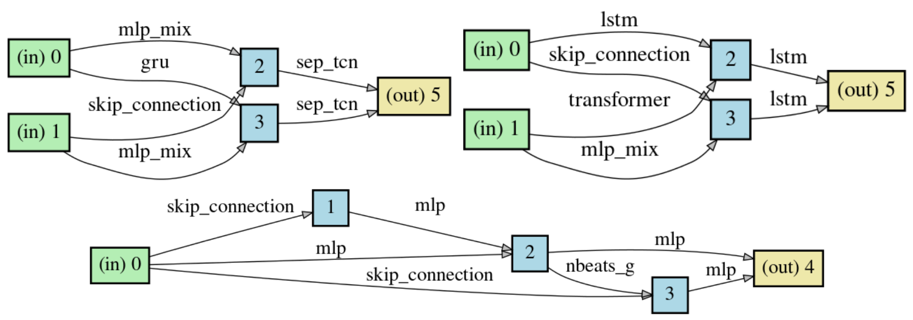


Figure 6: One optimal architecture on the ETTm2 dataset, this is an encoder-decoder architecture and will be trained with a quantile loss

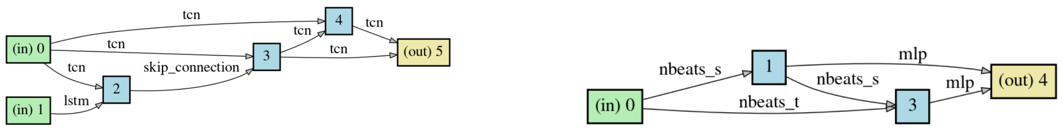


Figure 7: An optimal architecture on the ECL dataset. This is an encoder-only architecture

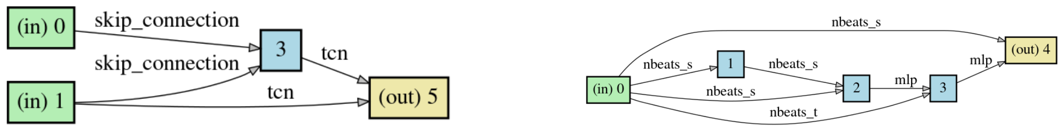


Figure 8: An optimal architecture on the Traffic dataset. This is an encoder-only architecture

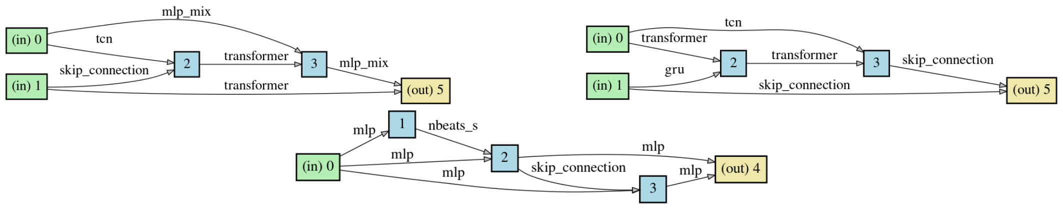


Figure 9: An optimal architecture on the ETTm1 dataset. This is an encoder-decoder architecture

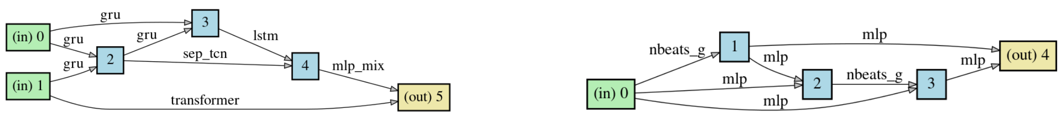


Figure 10: An optimal architecture on the ETTm2 dataset. This is an encoder-only architecture

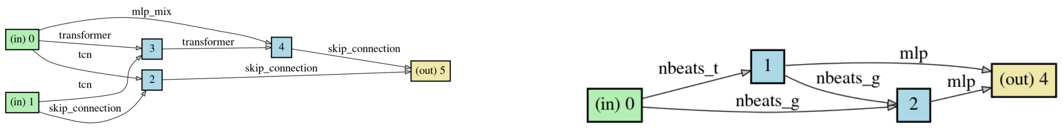


Figure 11: An optimal architecture on the Weather dataset. This is an encoder-only architecture

We also show some of the optimal architectures in Figure 8, 9, 10, and 11. We can see that no single operation dominates the optimal architecture, which shows the necessity of performing an architecture search for the optimal architecture.