ENHANCING TRANSFORMER EFFICIENCY FOR MULTIVARIATE TIME SERIES CLASSIFICATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The majority of current multivariate time series (MTS) classification algorithms aim to improve the predictive accuracy. However, when it comes to large-scale (either high-dimensional or long-sequential) time series (TS) datasets, it is crucial to design an efficient network architecture to reduce computational costs. In this work, we propose a mixing framework based on Transformer and Fourier transform. By pruning each module of the network separately and sequentially, we investigate the impact of each module on the predictive accuracy. We conduct comprehensive experiments on 18 benchmark MTS datasets. Ablation studies are used to evaluate the impact of each module. Through module-by-module pruning, our results demonstrate the trade-offs between efficiency and effectiveness, as well as efficiency and complexity of the network. Finally, we evaluate, via Pareto analysis, the trade-off between network efficiency and performance.

1 INTRODUCTION

Time series (TS) data is ubiquitous, occurring in healthcare (Li-wei et al., 2014), stock market (Liu & Long, 2020), astronomy (Fu, 2011), and many other areas (Gao et al., 2017; Hu et al., 2020). With the advance of sensing techniques, TS classification across wide-ranging domains has gained much interest during the past decade (Fawaz et al., 2019; Ruiz et al., 2021).

The availability of the UCR/UEA time series benchmark datasets (Ruiz et al., 2021) has led to an abundance of TS classification algorithms (Hüsken & Stagge, 2003; Zhao et al., 2017; Lines et al., 2018; Dempster et al., 2020; Zerveas et al., 2021). The classification accuracy has been the key metric used to evaluate existing methods (Lines & Bagnall, 2015). However, the high accuracy of these algorithms often comes with the cost of high computational complexity (Schäfer, 2016). From common preconceptions in natural language processing (NLP) and computer vision (CV), in order to achieve high accuracy, training top performing models with billions of parameters is a computationally intensive task, requiring days or weeks on many parallel GPUs or TPUs. However, such intensive training makes the model difficult to retrain for further improvement on performance. Likewise, for large-scale time series data with high dimensionality or long sequence length, it is challenging to maintain the balance between the predictive accuracy and training efficiency.

In this work, we propose a computationally-efficient network for MTS classification based on Transformer and Fourier transform. We use 18 benchmark MTS datasets for evaluation. Comprehensive experiments are conducted on all datasets, including ablation study of each module of the network and module-by-module pruning in terms of accuracy, training speed, and model size. Experimental results demonstrate the competitive performance of our proposed architecture compared with current state-of-the-art methods. Ablation studies identify the main contributors to the predictive performance, such as multi-head self-attention and Fourier transform. In addition, module-wise pruning of the network reveals the trade-off between model efficiency and effectiveness, as well as model efficiency and complexity. Finally, we use Pareto analysis to investigate and visualize the trade-off between efficiency and performance.

The main contributions of this paper are highlighted as follows:

(1) To the best of our knowledge, we develop the first mixing model architecture based on attention and Fourier transform for processing MTS data.

(2) Through module-by-module pruning, experimental results indicate an evident trade-off between model efficiency and its effectiveness, as well as its complexity.

(3) We propose to investigate the trade-off between efficiency and performance using Pareto analysis, which provides general guidance for researchers on how to select efficient model configurations.

The remainder of this paper is organized as follows. Section 2 describes related work of Transformer and Fourier transform on time series analysis and existing methods on model efficiency improvement. The proposed network architecture is outlined in Section 3. Section 4 discusses datasets, experiments on 18 benchmark datasets, including ablation studies, module-wise pruning and Pareto efficiency visualization. Finally, our conclusions are presented in Section 5.

2 RELATED WORK

Neural Networks for Time Series Classification. Currently, most TS classification algorithms can be divided into three categories: feature-based (Fulcher & Jones, 2014), distance-based (Abanda et al., 2019), and neural network based methods (Fawaz et al., 2019). Here, we focus only on neural network based methods. Since the advancements of deep learning, two popular frameworks, CNN and RNN, are widely applied in TS classification tasks. (Wang et al., 2017) combined Fully Convolutional Networks (FCN) and Residual Networks (ResNet) for univariate time series classification. (Lin & Runger, 2017) developed a group-constrained method, which combines a CNN with an RNN. More recent works such as InceptionTime (Fawaz et al., 2020), TapNet (Zhang et al., 2020), and TST (Zerveas et al., 2021) are proposed for TS classification. For additional deep learning methods, we refer readers to (Fawaz et al., 2019).

Fourier Transform in Time Series. The Fourier transform (FT) has been an important tool in time series analysis for decades (Bloomfield, 2004), and is widely used for applications such as anomaly detection (Ren et al., 2019), periodicity detection (Puech et al., 2019), and similarity measures (Janacek et al., 2005). The FT converts a TS from time domain to frequency domain, and uses Fourier coefficients to represent the original data. For the TS classification task, FTs have been used indirectly in disparate applications. For instance, (Geerken et al., 2005) utilizes the FT to filter noisy data for vegetation type classification, and (Samiee et al., 2014) uses the FT as a feature extraction technique to classify electroencephalography (EEG) data. However, none of the above methods apply the FT directly for TS classification, particularly in the context of neural networks. In contrast, we aim to apply the discrete FT and its inverse as modules of a deep learning framework. The unparameterized FT can reduce the computational cost of the network to some extent.

Transformer Networks for Time Series Classification. With the exemplary performance of the Transformer architecture (Vaswani et al., 2017) in NLP and CV, researchers in the time series community began exploring Transformers in TS classification in specific domains (Oh et al., 2018; Zhao et al., 2021). More recent works have generalized Transformer frameworks for MTS classification. (Zerveas et al., 2021) adopts a Transformer encoder architecture for unsupervised representation learning of MTS. (Liu et al., 2021) explored an extension of the current Transformer architecture by gating, which merges two towers for MTS classification. In contrast, we propose to generalize a mixing framework which utilizes both Transformer and FT. By replacing some self-attention sublayers with FT, the computational complexity can be reduced.

Model Training Efficiency. Due to the increasing size of both models and training data, many works have focused on improving model training efficiency through parameter reduction, such as DenseNet (Huang et al., 2017) and EfficientNet (Tan & Le, 2019), training speed improvement including NFNets (Brock et al., 2021) and BotNet (Srinivas et al., 2021), or both (Tan & Le, 2021). One of the most common techniques to improve network efficiency is model pruning. Early works focused on non-structured methods. For instance, (LeCun et al., 1990; Han et al., 2015) proposed to remove individual weight values. Recent works focused more on structured methods, such as channel weight pruning based on l_1 norm (Li et al., 2016).

3 METHODOLOGY

In this section, we present our network architecture, which contains all of the modules for potential model pruning. The overall model structure is illustrated in Figure 1.

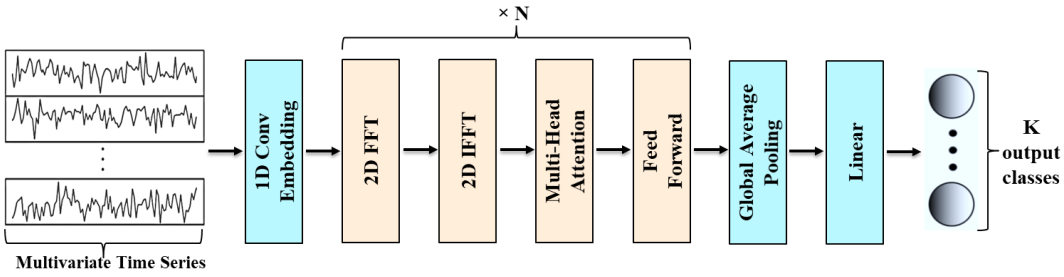


Figure 1: An overview of the full model framework. Our architecture is based on Transformer and Fourier transform. Following the sequence embedding, we apply a 2D discrete Fourier transform (particularly Fast Fourier transform) to convert the TS features from the time domain to the frequency domain, a 2D inverse discrete Fourier transform to map the features back to the time domain, and a multi-head self-attention layer. Then we employ a Global Average Pooling (GAP) layer to average the output of the MTS over the entire time dimension. Finally, a Softmax layer is used for the multi-class MTS classification task.

Input Embeddings. Input embeddings are commonly used in NLP models, which map relatively low-dimensional vectors to high-dimensional vectors to facilitate sequence modeling (Kim, 2014). Correspondingly, an embedding for TS sequence is required to capture the dependencies among different features without considering the temporal information (Song et al., 2018). Our framework employs a 1D convolutional layer to obtain the K-dimensional embeddings at each time step.

Discrete Fourier Transform. The Fourier transform decomposes a function of time into its constituent frequencies. For clarity, we first consider the 1D Discrete Fourier transform (DFT). Given a sequence of complex numbers $x(n)$ with $0 \leq n \leq N-1$, the 1D DFT is defined by

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-\frac{2\pi i}{N} kn} := \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn},\ 0 \leq k \leq N-1,$$

where $W_N = e^{-\frac{2\pi i}{N}}$. Given the DFT X_k, the original sequence can be recovered by the inverse DFT (IDFT)

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{\frac{2\pi i}{N} kn},\ 0 \leq n \leq N-1.$$

The 2D DFT is a direct extension of the 1D DFT, obtained by alternately performing the 1D DFT on the row and column dimensions. Given a 2D signal $x(m,n)$ with $0 \leq m \leq M-1, 0 \leq n \leq N-1$, the 2D DFT is given by

$$X(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m,n) \cdot e^{-2\pi j(\frac{km}{M} + \frac{ln}{N})}.$$

Similar to the 1D IDFT, the 2D DFT is invertible via the 2D IDFT,

$$x(m,n) = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X(k,l) \cdot e^{2\pi j(\frac{km}{M} + \frac{ln}{N})}.$$

To compute the DFT efficiently, the Fast Fourier Transform (FFT) algorithm takes advantage of the periodicity and symmetry properties of W_N^{kn} such that the computational complexity of the DFT reduces from $O(N^2)$ to $O(NlogN)$, regardless of dimension.

Multi-head Attention. The multi-head attention (MHA) mechanism, the major component of the Transformer architecture (Vaswani et al., 2017), allows the model to jointly attend to information from different representation subspaces at different positions. MHA is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \cdots, \text{head}_h)W^O,$$

where $Q, K, V \in \mathbb{R}^{n \times d_{model}}$ are input embedding matrices, n is the sequence length, d_{model} is the embedding dimension, and h is the number of heads. Each head i is defined as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}\left(\frac{QW_i^Q(KW_i^K)^T}{\sqrt{d_k}}\right)VW_i^V,$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}, W_i^O \in \mathbb{R}^{hdv \times d_{model}}$ are parameter matrices to be learned.

Global Average Pooling. Global average pooling involves calculating the average value of all of the elements in a feature map. It is mainly used to reduce the amount of learnable parameters.

Batch Normalization. Instead of using layer normalization in Transformer-related architectures in NLP, we consider the necessity of applying batch normalization to each yellow module shown in Figure 1. Compared to layer normalization, batch normalization can mitigate the effect of outlier values in time series data, which does not appear in text representations.

Activation Function. Using the same activation function as the Transformer architecture, we consider the necessity of applying the activation function *gelu* for each module shown in Figure 1.

Feedforward Neural Network. A position-wise feedforward neural network (FNN) is applied with two 1D convolutional layers with kernel size 1, and a *gelu* activation function in between.

4 EXPERIMENTS

In this section, we describe benchmark MTS datasets (Ruiz et al., 2021) used for experimental evaluation, the experimental setup, and corresponding results.

4.1 DATASETS

We select a set of 18 publicly available benchmark datasets from the UCR/UEA classification archive: AtrialFibrillation (AF), BasicMotions (BM), Cricket (CR), DuckDuckGeese (DDG), Epilepsy (EP), EthanolConcentration (EC), ERing (ER), FingerMovements (FM), HandMovementDirection (HMD), Handwriting (HW), Heartbeat (HB), Libras (LIB), NATOPS (NATO), PEMSSF (PEMS), RacketSports (RS), SelfRegulationSCP1 (SRS1), SelfRegulationSCP2 (SRS2), and UWaveGestureLibrary (UW). The main characteristics of each dataset are summarised in Appendix Table 3. All of the datasets have been split into training and testing sets by default. Thus, there are no preprocessing steps for these data. The predictive performance on all datasets is evaluated in terms of accuracy.

4.2 SETUP

We set aside 20% of the default training set for the validation set, which we used to select the best collection of hyperparameters. All experiments were implemented in Pytorch (Paszke et al., 2019) on one GTX 1080 Ti GPU. We minimized the cross entropy loss with the Adam (Kingma & Ba, 2014) optimizer for training. The hyperparameter search space for each dataset is listed in Appendix Table 4. Note that the batch size choice is limited by the available GPU memory.

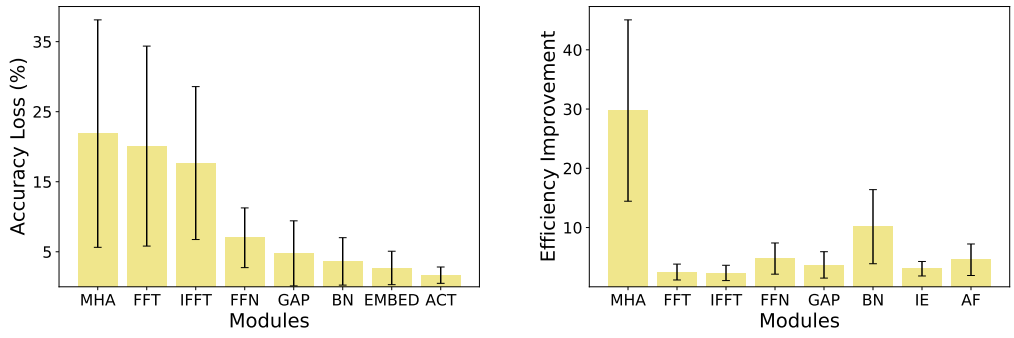
4.3 MODULE SETTINGS

Based on Section 3, we define the following eight modules of the network for further analysis: input embedding (EMBED), fast Fourier transform (FFT), inverse fast Fourier transform (IFFT), multi-head attention (MHA), feedforward neural network (FFN), global average pooling (GAP), batch normalization (BN), and activation function (ACT). The corresponding abbreviations of each module are shown in parentheses.

4.4 ABLATION STUDY

First, we conduct ablation studies to analyze the contributions of each module on the predictive performance. The contribution of each module is obtained when a module was removed from the full network while other modules remained. The fine-tuned results on 18 datasets are shown in Table 1. Starting from Column 4, the smaller the accuracy is, the larger the module's contribution is, and vice versa. The accuracy of each dataset for the unpruned model (Table 1 Column 3) is competitive with current state-of-the-art methods (Ruiz et al., 2021). Among eight modules, it can be seen that MHA and FFT contribute most to the predictive performance on 10 out of the 18 datasets and 9 out of the 18 datasets, respectively. For MTS data, the correlations between different dimensions across all time steps are important to consider. Hence, the MHA is able to catch different feature correlations, and influence the accuracy to a large extent. The FFT, as the core of signal processing and more generalized time series, extracts frequency information embedded in data, which provides a more straightforward representation compared to the original data in the time-domain. In contrast, we observe that EMBED, BN, and ACT contribute least to the predictive performance on 11 out of the 18 datasets, 5 out of the 18 datasets, and 13 out of the 18 datasets, respectively. Although these operations are important for the training of the model, they influence the testing accuracy marginally compared with MHA and FFT.

To clearly demonstrate the influence of each module on the predictive performance and efficiency of the network, the averaged testing accuracy loss and the corresponding efficiency improvement for each module (compared with the unpruned model) over all datasets are presented in Figure 2. Here, efficiency is defined as the product of training time per epoch and the amount of learnable parameters. The higher the product, the lower the efficiency is. In consideration of highly diversed datasets with respect to sequence length, number of samples, and dimensionality, the average loss in accuracy for each module demonstrates a high variance from Figure 2(a) as the performance loss extent can vary depending on dataset characteristics. The modules MHA, FFT, and IFFT demonstrate a notable influence on the model performance on average (21.9%, 20.1%, and 17.7% loss in accuracy respectively). For modules like BN, EMBED and ACT, removing them bring about minimal accuracy loss compared to other modules (3.6%, 2.7%, and 1.6% respectively). Meanwhile, comparing Figure 2(a) and Figure 2(b), the module which has larger impact on the predictive performance does not indicate that removing it can bring about more efficiency improvement. For instance, the computationally inexpensive FFT influences the predictive performance to a large extent. In contrast, although the computational cost of BN is high, its contribution to the performance is marginal.



(a) (b)

Figure 2: (a) represents the average testing accuracy loss across all datasets while removing one module at a time and other modules remain in the network. Modules MHA, FFT, and IFFT bring about larger influence on the predictive performance due to the high percentage of accuracy loss when removing them. In comparison, BN, EMBED, and ACT bring about marginal influence on the predictive performance compared with other modules. (b) represents the corresponding average efficiency improvement across all datasets when one module is removed from the network while other modules keep intact.

Dataset	Acc.	Unpruned	EMBED	FFT	IFFT	MHA	FFN	GAP	BN	ACT
AF	Mean	0.667	0.600	**0.400**	0.467	**0.400**	0.667	0.533	0.600	0.667
	Std.	0.003	0.005	0.005	0.004	0.003	0.006	0.006	0.004	0.003
BM	Mean	0.975	0.950	**0.725**	0.775	0.750	0.900	0.925	0.900	0.950
	Std.	0.008	0.010	0.012	0.009	0.012	0.010	0.014	0.009	0.011
CR	Mean	0.987	0.958	0.875	0.861	**0.833**	0.889	0.944	0.972	0.944
	Std.	0.007	0.009	0.012	0.008	0.012	0.006	0.009	0.012	0.008
DDG	Mean	0.580	0.580	0.440	0.420	**0.380**	0.520	0.560	0.560	0.580
	Std.	0.016	0.017	0.020	0.016	0.014	0.016	0.016	0.014	0.016
EP	Mean	0.986	0.978	**0.891**	0.913	0.899	0.949	0.971	0.956	0.971
	Std.	0.014	0.013	0.016	0.014	0.014	0.012	0.014	0.013	0.015
EC	Mean	0.456	0.445	0.376	0.395	**0.365**	0.418	0.441	0.445	0.452
	Std.	0.003	0.002	0.003	0.003	0.004	0.002	0.004	0.003	0.002
ER	Mean	0.963	0.956	0.896	0.889	**0.885**	0.892	0.948	0.952	0.956
	Std.	0.006	0.007	0.006	0.006	0.008	0.005	0.006	0.007	0.005
FM	Mean	0.640	0.620	**0.490**	0.520	0.500	0.600	0.590	0.610	0.620
	Std.	0.009	0.008	0.007	0.008	0.010	0.008	0.009	0.010	0.011
HMD	Mean	0.486	0.446	0.365	0.351	**0.338**	0.406	0.459	0.432	0.473
	Std.	0.018	0.016	0.020	0.017	0.018	0.019	0.018	0.016	0.020
HW	Mean	0.529	0.514	0.471	0.473	**0.468**	0.506	0.506	0.512	0.514
	Std.	0.006	0.007	0.006	0.005	0.007	0.007	0.008	0.007	0.006
HB	Mean	0.771	0.766	**0.683**	0.707	0.688	0.751	0.756	0.766	0.756
	Std.	0.014	0.015	0.014	0.017	0.015	0.016	0.014	0.015	0.016
LIB	Mean	0.917	0.906	**0.822**	0.827	0.839	0.889	0.894	0.906	0.911
	Std.	0.009	0.011	0.012	0.010	0.012	0.013	0.011	0.009	0.010
NATO	Mean	0.844	0.833	**0.728**	0.739	0.750	0.772	0.811	0.833	0.833
	Std.	0.005	0.004	0.005	0.007	0.006	0.005	0.006	0.004	0.006
PEMS	Mean	0.908	0.884	0.815	0.809	**0.803**	0.867	0.879	0.896	0.896
	Std.	0.013	0.012	0.014	0.016	0.014	0.013	0.013	0.014	0.012
RS	Mean	0.914	0.901	**0.796**	0.816	0.803	0.855	0.908	0.901	0.908
	Std.	0.021	0.020	0.020	0.018	0.019	0.021	0.020	0.021	0.019
SRS1	Mean	0.915	0.894	0.836	0.823	**0.819**	0.853	0.887	0.894	0.901
	Std.	0.005	0.007	0.006	0.006	0.005	0.007	0.006	0.005	0.005
SRS2	Mean	0.600	0.594	**0.522**	0.533	0.516	0.578	0.583	0.588	0.594
	Std.	0.002	0.003	0.002	0.001	0.004	0.002	0.003	0.003	0.002
UW	Mean	0.922	0.906	0.844	0.850	**0.841**	0.875	0.894	0.897	0.903
	Std.	0.006	0.008	0.009	0.006	0.007	0.008	0.006	0.007	0.007

Table 1: Ablation study in the testing accuracy loss on 18 datasets by removing each module at a time while leaving others the same. Each experiment is conducted 5 times with different random seeds. The results are shown in the format of mean and standard deviation. Column 2 shows the accuracy of the full model with all modules included. Columns 3 to 10 represent the accuracy when the module in that column is removed from the model. The order of modules follow the network architecture in Figure 1. Bold indicates that the module contributes most to the loss in accuracy and underlining indicates that the module contributes least to the loss in accuracy when the module is removed.

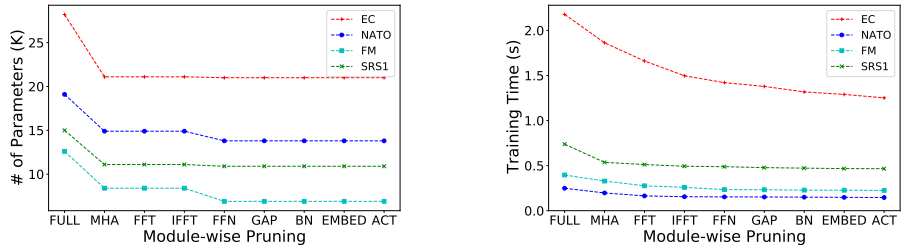
4.5 MODULE-BY-MODULE PRUNING

Next, we explore the relationship between efficiency (defined the same as Section 4.4) and effectiveness (predictive performance). Based on the contribution of each module on the performance loss shown in Figure 2(a), we perform module-by-module pruning by following the order of modules from the most significant contributor to the least significant contributor (MHA, FFT, IFFT, FFN, GAP, BN, EMBED, ACT) to accuracy. We evaluate such pruning effect in two aspects: (1) effectiveness: testing accuracy; (2) efficiency: average training time per epoch in seconds and the number of learnable parameters. The results of the testing accuracy are shown in Table 2. Due to limited space, we only show some datasets' efficiency results in Figure 3. See Appendix Table 5 for details regarding the training time and parameters of the module-wise pruned model over all datasets. We

observe that after removing the entire MHA module, the number of learnable parameters shrinks drastically, so as the accuracy (Table 2 Column 4). The representation capability of the pruned network, which has fewer parameters, is damaged since the amount of parameters is a key aspect to the network representation. Furthermore, the pace of accuracy loss and parameter reduction removal of subsequent modules slows down as FFT/IFFT has no learnable parameters. For the remaining modules, the number of parameters they carry is much fewer than the MHA module. Based on Figure 2(a), their effects on the predictive performance are moderate. Hence, the curves in Figure 3 are relatively flat following MHA. We further investigate extent of change in accuracy of module-wise pruning on all datasets, as shown in Figure 4. We notice that the performance variation in different datasets vary widely. For datasets such as AF, BM, and DDG, the model pruning has a great impact on their performance. This may be due to very limit amount of training samples. Conversely, for datasets like HB, LIB, and SRS1, the model pruning brings little effect after removing the MHA module (within 1%).

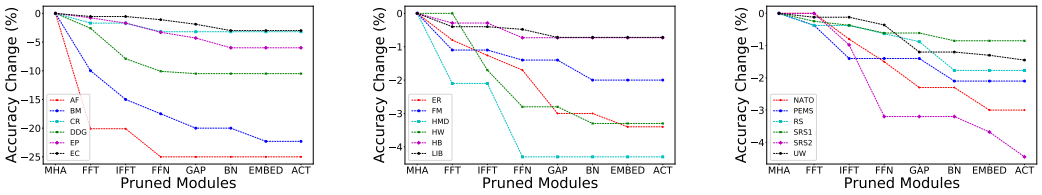
Dataset	Acc.	Unpruned	MHA	FFT	IFFT	FFN	GAP	BN	IE	AF
AF	Mean	0.667	**0.400**	0.333	0.333	0.300	0.300	0.300	0.300	0.300
	Std.	0.003	0.003	0.004	0.002	0.003	0.003	0.002	0.003	0.003
BM	Mean	0.975	**0.750**	0.675	0.650	0.638	0.625	0.625	0.613	0.613
	Std.	0.008	0.012	0.010	0.012	0.009	0.010	0.012	0.012	0.009
CR	Mean	0.987	**0.833**	0.819	0.819	0.806	0.806	0.806	0.806	0.806
	Std.	0.007	0.012	0.009	0.010	0.010	0.009	0.007	0.008	0.010
DDG	Mean	0.580	**0.380**	0.370	0.350	0.345	0.340	0.340	0.340	0.340
	Std.	0.016	0.014	0.014	0.015	0.012	0.013	0.012	0.014	0.016
EP	Mean	0.986	**0.899**	0.892	0.884	0.870	0.862	0.848	0.848	0.848
	Std.	0.014	0.014	0.011	0.012	0.014	0.013	0.012	0.010	0.009
EC	Mean	0.456	**0.365**	0.363	0.363	0.361	0.358	0.354	0.354	0.354
	Std.	0.003	0.004	0.004	0.002	0.004	0.003	0.003	0.003	0.003
ER	Mean	0.963	**0.885**	0.878	0.874	0.870	0.859	0.859	0.856	0.856
	Std.	0.006	0.008	0.006	0.007	0.007	0.008	0.007	0.008	0.008
FM	Mean	0.640	**0.500**	0.495	0.495	0.493	0.493	0.490	0.490	0.490
	Std.	0.009	0.010	0.011	0.010	0.008	0.009	0.011	0.010	0.011
HMD	Mean	0.486	**0.338**	0.331	0.331	0.324	0.324	0.324	0.324	0.324
	Std.	0.018	0.018	0.016	0.016	0.017	0.017	0.016	0.016	0.017
HW	Mean	0.529	**0.468**	0.468	0.460	0.455	0.455	0.453	0.453	0.453
	Std.	0.006	0.007	0.005	0.006	0.005	0.005	0.006	0.006	0.006
HB	Mean	0.771	**0.688**	0.686	0.686	0.683	0.683	0.683	0.683	0.683
	Std.	0.014	0.015	0.014	0.012	0.016	0.015	0.015	0.016	0.016
LIB	Mean	0.917	**0.839**	0.836	0.836	0.835	0.833	0.833	0.833	0.833
	Std.	0.009	0.012	0.011	0.012	0.008	0.009	0.010	0.009	0.010
NATO	Mean	0.844	**0.750**	0.750	0.744	0.739	0.733	0.733	0.728	0.728
	Std.	0.005	0.006	0.003	0.004	0.006	0.005	0.006	0.004	0.005
PEMS	Mean	0.908	**0.809**	0.806	0.798	0.798	0.798	0.792	0.792	0.792
	Std.	0.013	0.014	0.013	0.012	0.013	0.014	0.013	0.014	0.012
RS	Mean	0.914	**0.803**	0.800	0.800	0.798	0.796	0.789	0.789	0.789
	Std.	0.021	0.019	0.018	0.019	0.017	0.020	0.018	0.019	0.020
SRS1	Mean	0.915	**0.819**	0.817	0.816	0.814	0.814	0.812	0.812	0.812
	Std.	0.005	0.005	0.003	0.003	0.0.004	0.006	0.003	0.004	0.005
SRS2	Mean	0.600	**0.516**	0.516	0.511	0.500	0.500	0.500	0.497	0.494
	Std.	0.002	0.004	0.001	0.002	0.002	0.003	0.002	0.003	0.001
UW	Mean	0.922	**0.841**	0.840	0.840	0.838	0.831	0.831	0.830	0.829
	Std.	0.006	0.007	0.007	0.007	0.005	0.006	0.006	0.007	0.006

Table 2: Module-wise pruning results over all datasets. The results from Column 3 (MHA) to Column 10 (AF) with regard to accuracy represent that the module in that column is removed from the model architecture. Experiments are conducted 5 times with different random seeds. The accuracy results are shown in the format of mean and standard deviation. Bold represents that the module brings about much accuracy loss compared to the unpruned model. Following MHA, the accuracy decreasing trend remains stable.



(a) (b)

Figure 3: Module-wise results for changes in terms of number of parameters and training time per epoch on four datasets: EC, NATO, FM, SRS1.



(a) (b) (c)

Figure 4: Change in accuracy (%) from module-by-module pruning across all datasets. The order of datasets shown from (a) to (c) correspond to Table 1.

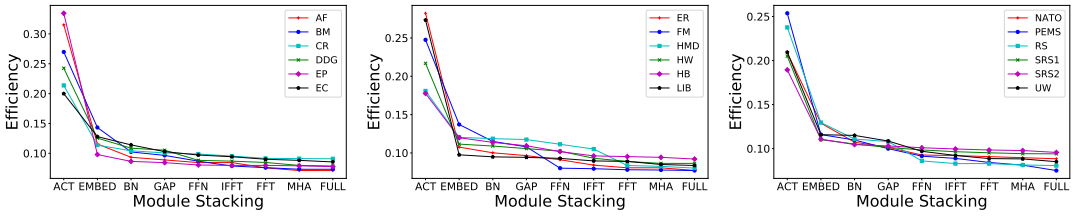
Overall, based on the above module-by-module pruning scheme, we observe that as the effectiveness (predictive performance) of the network increases, the corresponding efficiency (training speed and model size) generally decreases. The evident cost–benefit trade-off between efficiency and effectiveness provides a key question to researchers on how to find efficient model settings while maintaining the "equilibrium" between these two aspects. This problem will be discussed in Section 4.7.

4.6 EFFICIENCY VS. COMPLEXITY

Here, we explore the relationship between network efficiency and complexity. In general, the more complex a model is, the less efficient it is. The network's efficiency is defined in the same way as previous sections, in terms of the training time and the number of parameters. Meanwhile, we define the complexity of the model as the stacking of modules. Contrary to model pruning, we stack each module based on their influence on the predictive performance, from the least significant contributor to the most significant contributor (ACT, EMBED, BN, GAP, FFN, IFFT, FFT, MHA) to accuracy. Our empirical results in Figure 5 shed light on the trade-off between model efficiency and complexity. As can be seen in the Figure, as more modules are stacked over the network, the corresponding computational efficiency decreases. All datasets illustrate similar trends.

4.7 PARETO ANALYSIS FOR TRADE-OFF EXPLORATION BETWEEN EFFICIENCY AND PERFORMANCE/EFFECTIVENESS

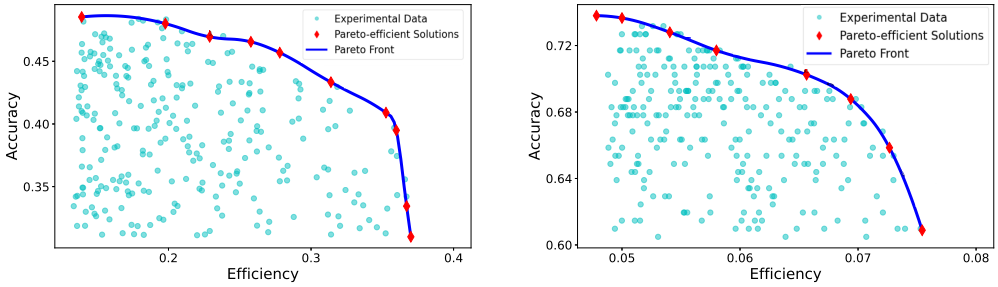
We define the model efficiency in terms of the reciprocal of the product between training time per epoch and the number of parameters. Thus, the higher the reciprocal, the higher the efficiency. To explore the relationship between model efficiency and performance, we employ Pareto analysis (Censor, 1977). Pareto efficiency represents a state for which improving the performance as measured by one criterion would worsen the performance as measured by another criterion. We choose the *FingerMovements* and *Heartbeat* datasets to obtain the Pareto frontiers, where the set of points on the front correspond to Pareto-efficient solutions. We have two objectives: (1) maximize the efficiency; (2) maximize the accuracy. Figure 6 shows the result of Pareto fronts for both datasets in blue, where the red points are Pareto-efficient solutions. The scattered cyan points are



(a) (b) (c)

Figure 5: Trade-off between network efficiency and complexity across all datasets. Due to the notable differences of dataset sizes, the computation of efficiency is normalized for each dataset. The order of datasets shown from (a) to (c) correspond to Table 1.

randomly sampled experimental data from all different configurations. The Pareto analysis provides us with a principled approach for choosing efficient network settings, while exploring the trade-off between efficiency and performance. Specifically, we can recognize that how many computational resources are required in order for a model to achieve a certain performance. Conversely, we know that how well can a model perform given a certain amount of resources.



(a) *FingerMovements*. (b) *Heartbeat*.

Figure 6: Pareto efficiency visualization of the *FingerMovements* and *Heartbeat* datasets. The scattered cyan points, the marked red points, and the blue curve represent randomly sampled experimental data, Pareto-efficient solutions, and Pareto efficient frontiers.

5 DISCUSSION

In this work, we propose an efficient mixing network based on Transformer and Fourier transform for MTS classification. Extensive experiments are conducted on 18 MTS datasets, including ablation studies on different modules of the network, module-by-module pruning evaluated in terms of the predictive performance, training speed, and the number of learnable parameters. The network achieves competitive performance compared to current best-performing methods. Ablation studies indicate that self-attention and Fourier transform are the largest contributors that influence the model performance across all datasets. Furthermore, through sequential pruning of each module, we observed the efficiency–effectiveness and the efficiency–complexity trade-offs of the network. Through Pareto analysis, we show how to choose efficient settings of the network, while investigating the performance–efficiency trade-off through visualization of the Pareto fronts. We note that for far more complex models applied on large-scale data, due to finite computational resources, it is not practical to consider all possible configurations of the model and perform experiments. In these cases, given a reasonable number of experiments, techniques like regression can be used to generate massive random model settings and corresponding model performance. Pareto analysis can then be performed to evaluate the efficiency-performance trade-off and guide researchers to adjust model settings to improve the efficiency and effectiveness.

REFERENCES

Amaia Abanda, Usue Mori, and Jose A Lozano. A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2):378–412, 2019.

Peter Bloomfield. *Fourier analysis of time series: an introduction*. John Wiley & Sons, 2004.

Andrew Brock, Soham De, Samuel L Smith, and Karen Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.

Yair Censor. Pareto optimality in multiobjective problems. *Applied Mathematics and Optimization*, 4(1):41–59, 1977.

Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, 2020.

Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

Ben D Fulcher and Nick S Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

Bin Gao, Xiaoqing Li, Wai Lok Woo, and Gui yun Tian. Physics-based image segmentation using first order statistical properties and genetic algorithm for inductive thermography imaging. *IEEE Transactions on Image Processing*, 27(5):2160–2175, 2017.

R Geerken, B Zaitchik, and JP Evans. Classifying rangeland vegetation type and coverage from ndvi time series using fourier filtered cycle similarity. *International Journal of Remote Sensing*, 26(24):5535–5554, 2005.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

Bozhen Hu, Bin Gao, Wai Lok Woo, Lingfeng Ruan, Jikun Jin, Yang Yang, and Yongjie Yu. A lightweight spatial and temporal multi-feature fusion network for defect detection. *IEEE Transactions on Image Processing*, 30:472–486, 2020.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Michael Hüsken and Peter Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

Gareth J Janacek, Anthony J Bagnall, and Michael Powell. A likelihood ratio distance measure for the similarity between the fourier transform of time series. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 737–743. Springer, 2005.

Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746–1751, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

H Lehman Li-wei, Ryan P Adams, Louis Mayaud, George B Moody, Atul Malhotra, Roger G Mark, and Shamim Nemati. A physiological time series dynamics-based approach to patient monitoring and outcome prediction. *IEEE journal of biomedical and health informatics*, 19(3):1068–1076, 2014.

Sangdi Lin and George C Runger. Gcrnn: Group-constrained convolutional recurrent neural network. *IEEE transactions on neural networks and learning systems*, 29(10):4709–4718, 2017.

Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, 2015.

Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, 12(5), 2018.

Hui Liu and Zhihao Long. An improved deep learning model for predicting stock market price time series. *Digital Signal Processing*, 102:102741, 2020.

Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated transformer networks for multivariate time series classification. *arXiv preprint arXiv:2103.14438*, 2021.

Jeeheh Oh, Jiaxuan Wang, and Jenna Wiens. Learning to exploit invariances in clinical time-series data using sequence transformer networks. In *Machine Learning for Healthcare Conference*, pp. 332–347. PMLR, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Tom Puech, Matthieu Boussard, Anthony D'Amato, and Gaëtan Millerand. A fully automated periodicity detection in time series. In *International Workshop on Advanced Analysis and Learning on Temporal Data*, pp. 43–54. Springer, 2019.

Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3009–3017, 2019.

Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, 2021.

Kaveh Samiee, Peter Kovacs, and Moncef Gabbouj. Epileptic seizure classification of eeg time-series using rational discrete short-time fourier transform. *IEEE transactions on Biomedical Engineering*, 62(2):541–552, 2014.

Patrick Schäfer. Scalable time series classification. *Data Mining and Knowledge Discovery*, 30(5): 1273–1298, 2016.

Huan Song, Deepta Rajan, Jayaraman Thiagarajan, and Andreas Spanias. Attend and diagnose: Clinical time series analysis using attention models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Aravind Srinivas, Tsung-Yi Lin, Niki Parmar, Jonathon Shlens, Pieter Abbeel, and Ashish Vaswani. Bottleneck transformers for visual recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16519–16529, 2021.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Mingxing Tan and Quoc V Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pp. 1578–1585. IEEE, 2017.

George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2114–2124, 2021.

Xuchao Zhang, Yifeng Gao, Jessica Lin, and Chang-Tien Lu. Tapnet: Multivariate time series classification with attentional prototypical network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6845–6852, 2020.

Bendong Zhao, Huanzhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169, 2017.

Yun Zhao, Qinghang Hong, Xinlu Zhang, Yu Deng, Yuqing Wang, and Linda Petzold. Bert-surv: Bert-based survival models for predicting outcomes of trauma patients. *arXiv preprint arXiv:2103.10928*, 2021.

A Appendix

A.1 Data characteristics

Dataset	Code	Train Size	Test Size	Dimensions	Length	Classes
AtrialFibrillation	AF	15	15	2	640	3
BasicMotions	BM	40	40	6	100	4
Cricket	CR	108	72	6	1197	12
DuckDuckGeese	DDG	50	50	1345	270	5
Epilepsy	EP	137	138	3	206	4
EthanolConcentration	EC	261	263	3	1751	4
ERing	ER	30	270	4	65	6
FingerMovements	FM	316	100	28	50	2
HandMovementDirection	HMD	160	74	10	400	4
Handwriting	HW	150	850	3	152	26
Heartbeat	HB	204	205	61	405	2
Libras	LIB	180	180	2	45	15
NATOPS	NATO	180	180	24	51	6
PEMS-SF	PEMS	267	173	963	144	7
RacketSports	RS	151	152	6	30	4
SelfRegulationSCP1	SRS1	268	293	6	896	2
SelfRegulationSCP2	SRS2	200	180	7	1152	2
UWaveGestureLibrary	UW	120	320	3	315	8

Table 3: Summary of the 18 UCR/UEA datasets used in experimentation.

A.2 HYPERPARAMETERS

Hyperparameters	Search Space
learning rate	[1e-3, 5e-3, 1e-4, 5e-4, 1e-5, 5e-5]
dropout rate	[0.1, 0.2, 0.3]
batch size	[8, 16, 32]
♯ of heads	[4, 8, 16]
♯ of FFT layers	[0, 1, 2, 3, 4]
♯ of IFFT layers	[0, 1, 2, 3, 4]
♯ of MHA layers	[0, 1, 2, 3, 4]
♯ of Feedforward layers	[0, 1, 2, 3, 4]

Table 4: Hyperparameter search space of the model on each dataset. If the number of layers of a module is equal to 0, then this module is removed in the pruned model.

A.3 MODULE-WISE PRUNING RESULTS (TRAINING TIME AND NUMBER OF PARAMETERS)

Dataset		Unpruned	MHA	FFT	IFFT	FFN	GAP	BN	EMBED	ACT
AF	♯ Param. (K)	7.9	3.9	3.9	3.9	3.9	3.9	3.9	3.9	3.9
	Time (s)	0.035	0.026	0.021	0.020	0.019	0.019	0.017	0.016	0.016
BM	♯ Param. (K)	3.8	2.8	2.8	2.8	2.6	2.6	2.6	2.6	2.6
	Time (s)	0.068	0.049	0.035	0.033	0.032	0.029	0.028	0.027	0.027
CR	♯ Param. (K)	115.6	86.6	86.6	86.6	86.4	86.4	86.4	86.4	86.4
	Time (s)	0.501	0.356	0.326	0.315	0.310	0.299	0.287	0.286	0.285
DDG	♯ Param. (K)	18111.8	10870.3	10870.3	10870.3	10856.8	10856.8	10856.8	10856.8	10856.8
	Time (s)	0.733	0.632	0.547	0.530	0.445	0.438	0.427	0.402	0.397
EP	♯ Param. (K)	7.2	2.6	2.6	2.6	2.5	2.5	2.5	2.5	2.5
	Time (s)	0.169	0.137	0.121	0.118	0.117	0.116	0.116	0.115	0.113
EC	♯ Param. (K)	28.2	21.1	21.1	21.1	21.0	21.0	21.0	21.0	21.0
	Time (s)	2.178	1.863	1.661	1.497	1.421	1.379	1.318	1.290	1.252
ER	♯ Param. (K)	3.7	1.8	1.8	1.8	1.7	1.7	1.7	1.7	1.7
	Time (s)	0.037	0.029	0.027	0.026	0.026	0.024	0.023	0.023	0.022
FM	♯ Param. (K)	12.6	8.4	8.4	8.4	6.9	6.9	6.9	6.9	6.9
	Time (s)	0.396	0.329	0.276	0.259	0.234	0.232	0.228	0.227	0.225
HMD	♯ Param. (K)	20.6	17.0	17.0	17.0	16.6	16.6	16.6	16.6	16.6
	Time (s)	0.258	0.208	0.205	0.203	0.197	0.186	0.148	0.146	0.142
HW	♯ Param. (K)	16.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0	12.0
	Time (s)	0.194	0.133	0.130	0.126	0.122	0.111	0.106	0.103	0.103
HB	♯ Param. (K)	83.6	68.4	68.4	68.4	67.2	67.2	67.2	67.2	67.2
	Time (s)	0.456	0.376	0.357	0.343	0.325	0.307	0.304	0.301	0.294
LIB	♯ Param. (K)	2.8	1.4	1.4	1.4	1.4	1.4	1.4	1.4	1.4
	Time (s)	0.206	0.147	0.143	0.142	0.140	0.135	0.134	0.128	0.126
NATO	♯ Param. (K)	19.1	14.9	14.9	14.9	13.8	13.8	13.8	13.8	13.8
	Time (s)	0.249	0.197	0.164	0.155	0.153	0.152	0.150	0.148	0.146
PEMS	♯ Param. (K)	9327.6	5662.8	5662.8	5662.8	5614.3	5614.3	5614.3	5614.3	5614.3
	Time (s)	2.99	2.25	2.13	1.94	1.79	1.74	1.65	1.59	1.47
RS	♯ Param. (K)	1.6	1.1	1.1	1.1	0.9	0.9	0.9	0.9	0.9
	Time (s)	0.174	0.138	0.118	0.115	0.112	0.108	0.108	0.106	0.105
SRS1	♯ Param. (K)	15.0	11.1	11,1	11.1	10.9	10.9	10.9	10.9	10.9
	Time (s)	0.738	0.536	0.512	0.493	0.488	0.478	0.472	0.467	0.466
SRS2	♯ Param. (K)	19.1	16.6	16.6	16.6	16.4	16.4	16.4	16.4	16.4
	Time (s)	0.828	0.557	0.526	0.518	0.514	0.507	0.501	0.498	0.487
UW	♯ Param. (K)	10.2	7.7	7.7	7.7	7.6	7.6	7.6	7.6	7.6
	Time (s)	0.198	0.145	0.144	0.136	0.123	0.117	0.114	0.112	0.108

Table 5: Module-by-module pruning results over all datasets. The results from Column 4 (MHA) to Column 11 (AF) with regard to number of parameters, and the average training time per epoch represent that the module in that column is removed from the model architecture. For fair comparison, the training time and the amount of parameters are reported when each module has only one layer (if exists).